Exercise 2.2.1a

For relation Accounts, the attributes are:

acctNo, type, balance

For relation Customers, the attributes are:

firstName, lastName, idNo, account

Exercise 2.2.1b

For relation Accounts, the tuples are:

(12345, savings, 12000),
(23456, checking, 1000),
(34567, savings, 25)

For relation Customers, the tuples are:

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

Exercise 2.2.1c

For relation Accounts and the first tuple, the components are:

123456 → acctNo
savings → type
12000 → balance

For relation Customers and the first tuple, the components are:

Robbie → firstName
Banks → lastName
901-222 → idNo
12345 → account

Exercise 2.2.1d

For relation Accounts, a relation schema is:

Accounts(acctNo, type, balance)

For relation Customers, a relation schema is:

Customers(firstName, lastName, idNo, account)

Exercise 2.2.1e

An example database schema is:

Accounts (
acctNo,
type,
balance
)
Customers (
firstName,

lastName,
idNo,
account
)

Exercise 2.2.1f

A suitable domain for each attribute:

acctNo → Integer
type → String
balance → Integer
firstName → String
lastName → String
idNo → String (because there is a hyphen we cannot use Integer)
account → Integer

Exercise 2.2.1g

Another equivalent way to present the Account relation:

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

Another equivalent way to present the Customers relation:

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

Exercise 2.2.2

Examples of attributes that are created for primarily serving as keys in a relation:

Universal Product Code (UPC) used widely in United States and Canada to track products in stores.

Serial Numbers on a wide variety of products to allow the manufacturer to individually track each product.

Vehicle Identification Numbers (VIN), a unique serial number used by the automotive industry to identify vehicles.

Exercise 2.2.3a

We can order the three tuples in any of 3! = 6 ways. Also, the columns can be ordered in any of 3! = 6 ways. Thus, the number of presentations is 6*6 = 36.

Exercise 2.2.3b

We can order the three tuples in any of 5! = 120 ways. Also, the columns can be ordered in any of 4! = 24 ways. Thus, the number of presentations is 120*24 = 2880

Exercise 2.2.3c

We can order the three tuples in any of m! ways. Also, the columns can be ordered in any of n! ways. Thus, the number of presentations is n!m!

Exercise 2.3.1a

```
CREATE TABLE Product (
	maker	CHAR(30),
	model	CHAR(10) PRIMARY KEY,
	type CHAR(15)
);
```

Exercise 2.3.1b

```
CREATE TABLE PC (
	model CHAR(30),
	speed DECIMAL(4,2),
	ram INTEGER,
	hd INTEGER,
	price DECIMAL(7,2)
);
```

Exercise 2.3.1c

```
CREATE TABLE Laptop (
	model CHAR(30),
	speed DECIMAL(4,2),
	ram INTEGER,
	hd INTEGER,
	screen DECIMAL(3,1),
	price DECIMAL(7,2)
);
```

Exercise 2.3.1d

```
CREATE TABLE Printer (
	model CHAR(30),
	color BOOLEAN,
	type CHAR (10),
	price DECIMAL(7,2)
);
```

Exercise 2.3.1e

```
ALTER TABLE Printer DROP color;
```

Exercise 2.3.1f

```
ALTER TABLE Laptop ADD od CHAR (10) DEFAULT ‘none’;
```

Exercise 2.3.2a

```
CREATE TABLE Classes (
	class CHAR(20),
	type CHAR(5),
	country CHAR(20),
	numGuns INTEGER,
	bore DECIMAL(3,1),
	displacement INTEGER
);
```

Exercise 2.3.2b

```
CREATE TABLE Ships (
        name CHAR(30),
        class CHAR(20),
        launched INTEGER
);
```

Exercise 2.3.2c

```
CREATE TABLE Battles (
        name CHAR(30),
        date DATE
);
```

Exercise 2.3.2d

```
CREATE TABLE Outcomes (
        ship CHAR(30),
        battle CHAR(30),
        result CHAR(10)
);
```

Exercise 2.3.2e

```
ALTER TABLE Classes DROP bore;
```

Exercise 2.3.2f

```
ALTER TABLE Ships ADD yard CHAR(30);
```

Exercise 2.4.1a

R1 := $\sigma_{speed \geqslant 3.00}$ (PC)
R2 := π_{model}(R1)

model
1005
1006
1013

Exercise 2.4.1b

R1 := $\sigma_{hd \geqslant 100}$ (Laptop)
R2 := Product $\bowtie$ (R1)
R3 := π_{maker} (R2)

maker
E
A
B
F
G

Exercise 2.4.1c

R1 := $\sigma_{maker=B}$ (Product $\bowtie$ PC)
R2 := $\sigma_{maker=B}$ (Product $\bowtie$ Laptop)
R3 := $\sigma_{maker=B}$ (Product $\bowtie$ Printer)
R4 := $\pi_{model,price}$ (R1)
R5 := $\pi_{model,price}$ (R2)
R6: = $\pi_{model,price}$ (R3)

$R7 := R4 \bigcup R5 \bigcup R6$

model	price
1004	649
1005	630
1006	1049
2007	1429

Exercise 2.4.1d

$R1 := \sigma_{color = true\ AND\ type = laser}$ (Printer)
$R2 := \pi_{model}$ (R1)

model
3003
3007

Exercise 2.4.1e

$R1 := \sigma_{type=laptop}$ (Product)
$R2 := \sigma_{type=PC}$(Product)
$R3 := \pi_{maker}$(R1)
$R4 := \pi_{maker}$(R2)
R5 := R3 – R4

maker
F
G

Exercise 2.4.1f

$R1 := \rho_{PC1}$(PC)
$R2 := \rho_{PC2}$(PC)
$R3 := R1 \bowtie_{(PC1.hd = PC2.hd\ AND\ PC1.model <> PC2.model)} R2$
$R4 := \pi_{hd}$(R3)

hd
250
80
160

Exercise 2.4.1g

$R1 := \rho_{PC1}$(PC)
$R2 := \rho_{PC2}$(PC)
$R3 := R1 \bowtie_{(PC1.speed = PC2.speed\ AND\ PC1.ram = PC2.ram\ AND\ PC1.model < PC2.model)} R2$
$R4 := \pi_{PC1.model,PC2.model}$(R3)

PC1.model	PC2.model
1004	1012

Exercise 2.4.1h

$R1 := \pi_{model}(\sigma_{speed \geq 2.80}(PC)) \bigcup \pi_{model}(\sigma_{speed \geq 2.80}(Laptop))$
$R2 := \pi_{maker,model}(R1 \bowtie Product)$
$R3 := \rho_{R3(maker2,model2)}(R2)$
$R4 := R2 \bowtie_{(maker = maker2\ AND\ model <> model2)} R3$
$R5 := \pi_{maker}(R4)$

maker
B

E

Exercise 2.4.1i

$R1 := \pi_{model,speed}(PC)$
$R2 := \pi_{model,speed}(Laptop)$
$R3 := R1 \cup R2$
$R4 := \rho_{R4(model2,speed2)}(R3)$
$R5 := \pi_{model,speed}\ (R3 \bowtie_{(speed < speed2)} R4)$
$R6 := R3 - R5$
$R7 := \pi_{maker}(R6 \bowtie Product)$

maker
B

Exercise 2.4.1j

$R1 := \pi_{maker,speed}(Product \bowtie PC)$
$R2 := \rho_{R2(maker2,speed2)}(R1)$
$R3 := \rho_{R3(maker3,speed3)}(R1)$
$R4 := R1 \bowtie_{(maker = maker2\ AND\ speed <> speed2)} R2$
$R5 := R4 \bowtie_{(maker3 = maker\ AND\ speed3 <> speed2\ AND\ speed3 <> speed)} R3$
$R6 := \pi_{maker}(R5)$

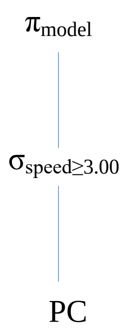
maker
A
D
E

Exercise 2.4.1k

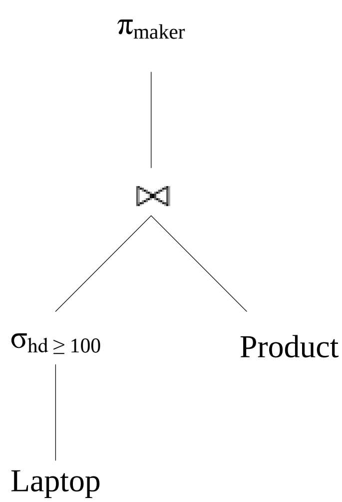
$R1 := \pi_{maker,model}(Product \bowtie PC)$
$R2 := \rho_{R2(maker2,model2)}(R1)$
$R3 := \rho_{R3(maker3,model3)}(R1)$
$R4 := \rho_{R4(maker4,model4)}(R1)$
$R5 := R1 \bowtie_{(maker = maker2\ AND\ model <> model2)} R2$
$R6 := R3 \bowtie_{(maker3 = maker\ AND\ model3 <> model2\ AND\ model3 <> model)} R5$
$R7 := R4 \bowtie_{(maker4 = maker\ AND\ (model4=model\ OR\ model4=model2\ OR\ model4=model3))} R6$
$R8 := \pi_{maker}(R7)$

maker
A
B
D
E

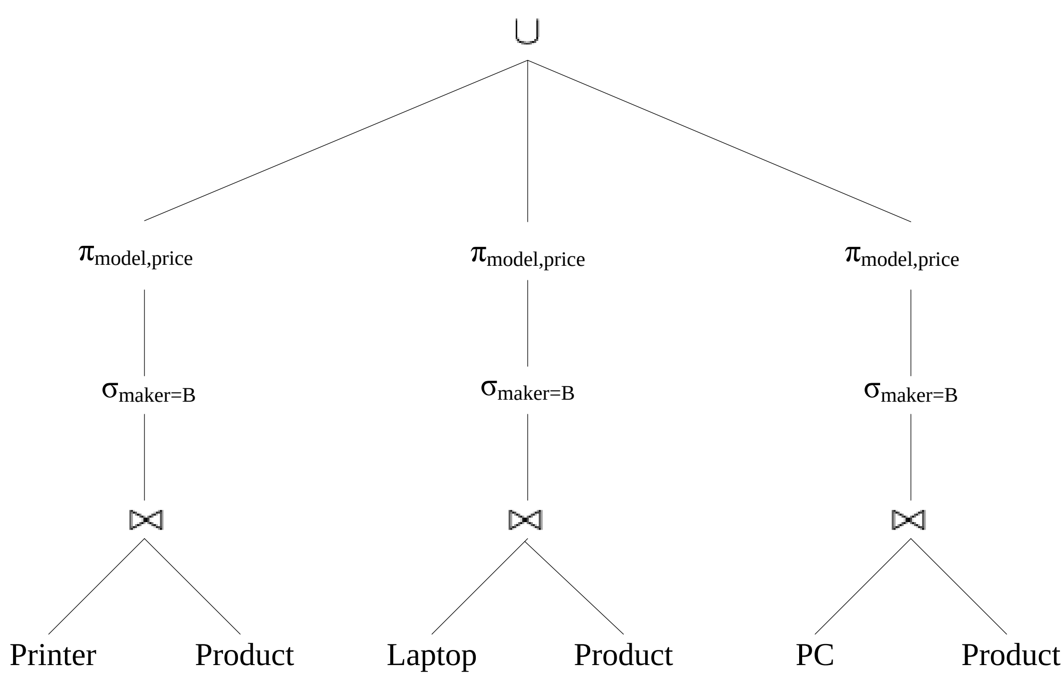
Exercise 2.4.2a



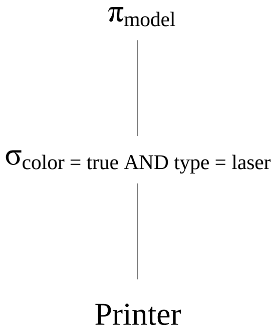
Exercise 2.4.2b



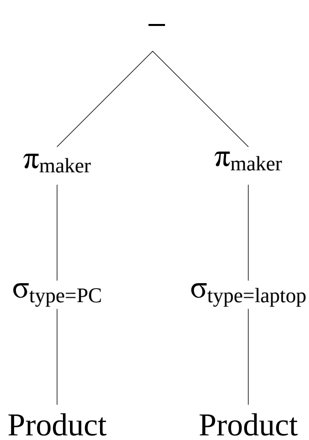
Exercise 2.4.2c



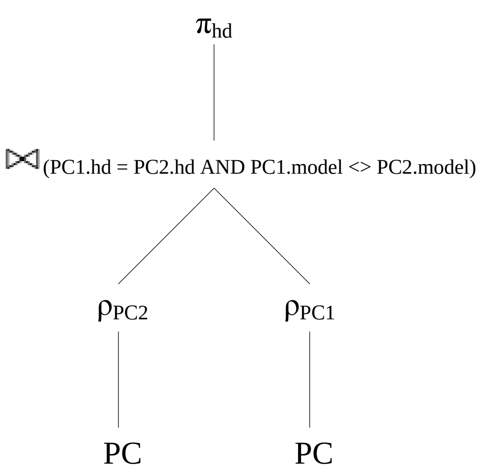
Exercise 2.4.2d



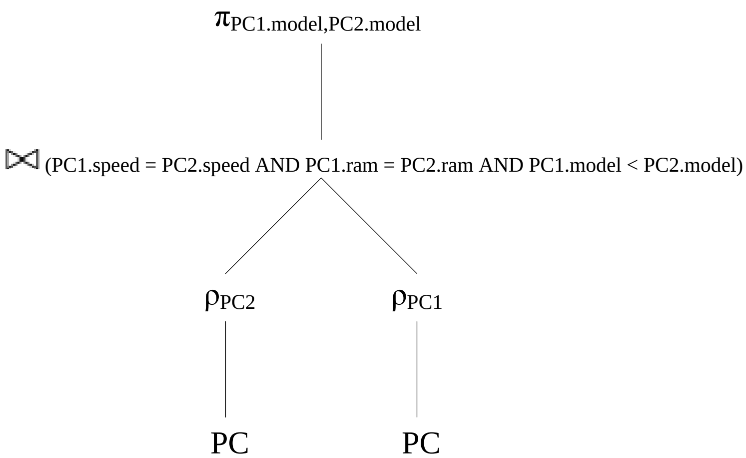
Exercise 2.4.2e



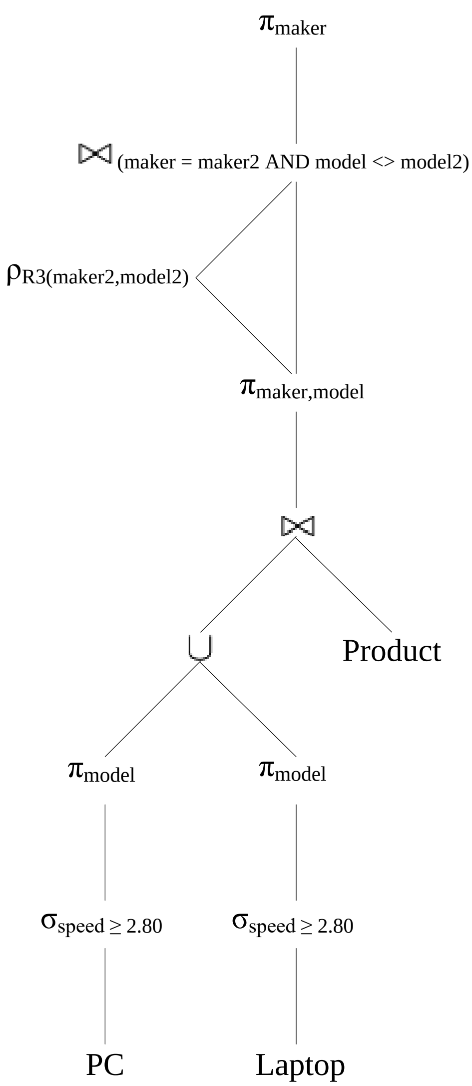
Exercise 2.4.2f



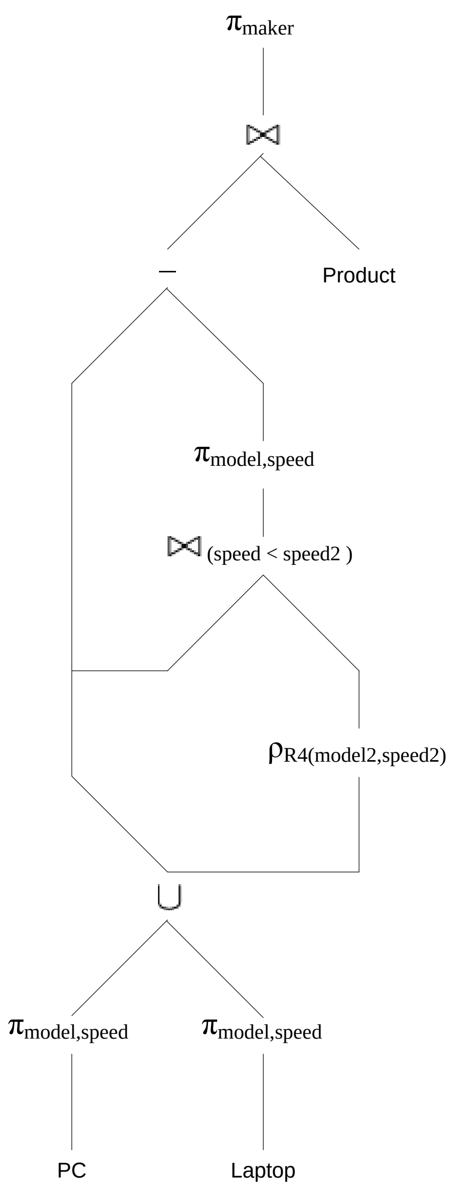
Exercise 2.4.2g



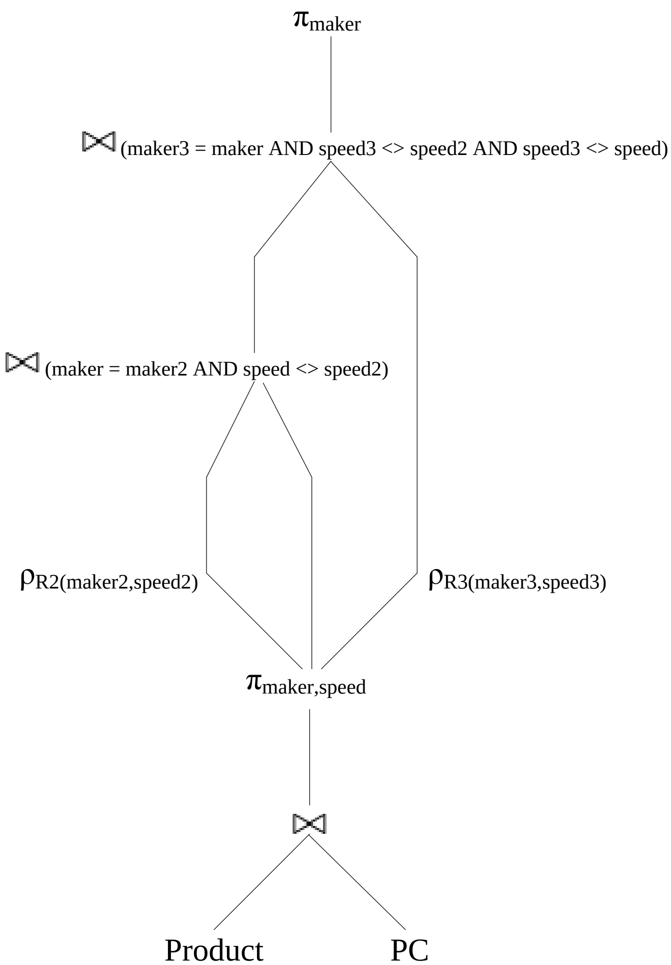
Exercise 2.4.2h



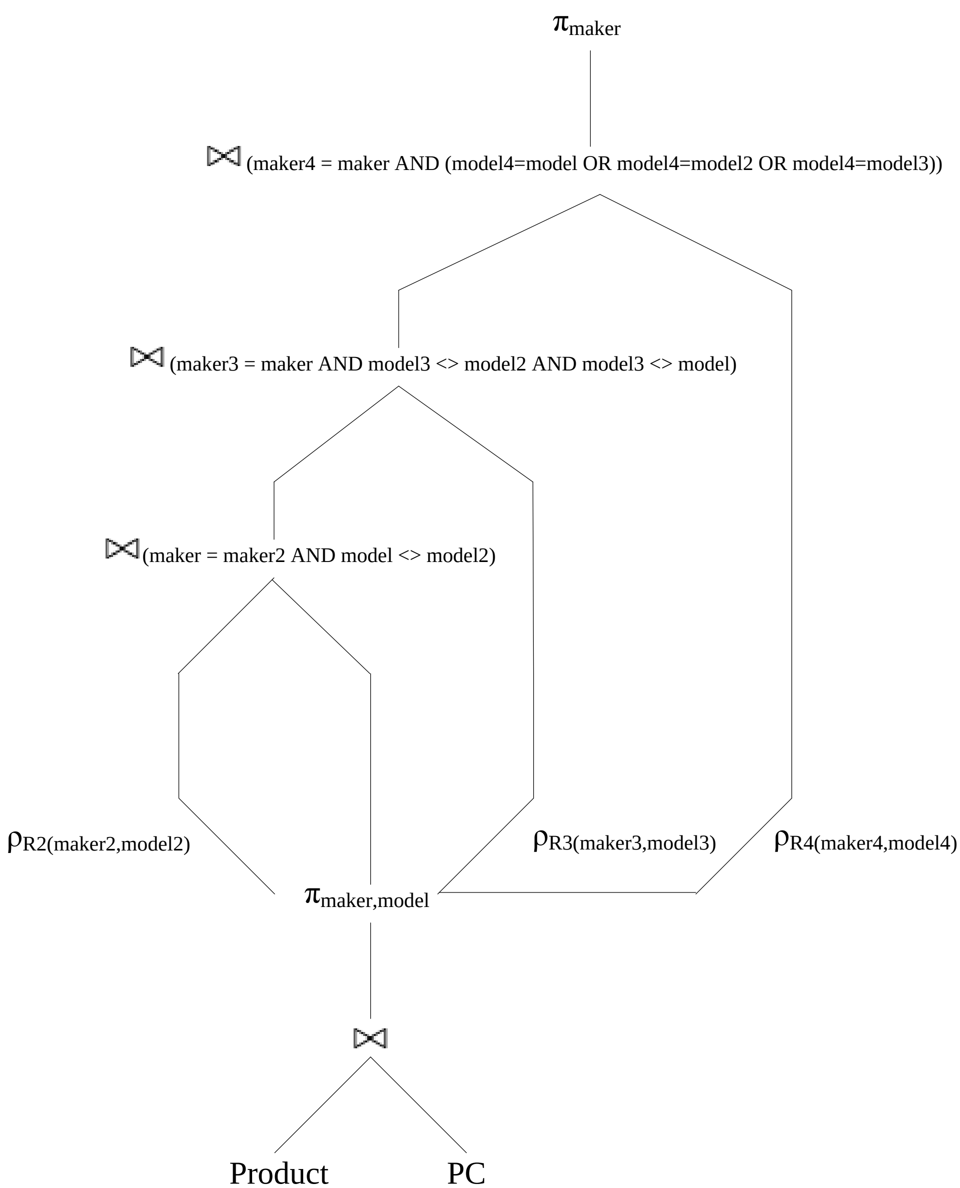
Exercise 2.4.2i



Exercise 2.4.2j



Exercise 2.4.2k



Exercise 2.4.3a

R1 := $\sigma_{bore \geq 16}$ (Classes)
R2 := $\pi_{class,country}$ (R1)

class	country
Iowa	USA
North Carolina	USA
Yamato	Japan

Exercise 2.4.3b

R1 := $\sigma_{launched < 1921}$ (Ships)
R2 := π_{name} (R1)

name

Haruna
Hiei
Kirishima
Kongo
Ramillies
Renown
Repulse
Resolution
Revenge
Royal Oak
Royal Sovereign
Tennessee

Exercise 2.4.3c

R1 := $\sigma_{\text{battle=Denmark Strait AND result=sunk}}$(Outcomes)
R2 := π_{ship} (R1)

ship
Bismarck
Hood

Exercise 2.4.3d

R1 := Classes $\bowtie$ Ships
R2 := $\sigma_{\text{launched > 1921 AND displacement > 35000}}$ (R1)
R3 := π_{name} (R2)

name
Iowa
Missouri
Musashi
New Jersey
North Carolina
Washington
Wisconsin
Yamato

Exercise 2.4.3e

R1 := $\sigma_{\text{battle=Guadalcanal}}$(Outcomes)
R2 := Ships $\bowtie_{\text{(ship=name)}}$ R1
R3 := Classes $\bowtie$ R2
R4 := $\pi_{\text{name,displacement,numGuns}}$(R3)

name	displacement	numGuns
Kirishima	32000	8
Washington	37000	9

Exercise 2.4.3f

R1 := π_{name}(Ships)
R2 := π_{ship}(Outcomes)
R3 := $\rho_{\text{R3(name)}}$(R2)
R4 := R1 $\cup$ R3

name
California
Haruna

Hiei
Iowa
Kirishima
Kongo
Missouri
Musashi
New Jersey
North Carolina
Ramillies
Renown
Repulse
Resolution
Revenge
Royal Oak
Royal Sovereign
Tennessee
Washington
Wisconsin
Yamato
Arizona
Bismarck
Duke of York
Fuso
Hood
King George V
Prince of Wales
Rodney
Scharnhorst
South Dakota
West Virginia
Yamashiro

Exercise 2.4.3g

From 2.3.2, assuming that every class has one ship named after the class.

R1 := π_{class}(Classes)
R2 := $\pi_{class}(\sigma_{name <> class}$(Ships))
R3 := R1 – R2

class
Bismarck

Exercise 2.4.3h

R1 := $\pi_{country}(\sigma_{type=bb}$(Classes))
R2 := $\pi_{country}(\sigma_{type=bc}$(Classes))
R3 := R1 ∩ R2

country
Japan
Gt. Britain

Exercise 2.4.3i

R1 := $\pi_{ship,result,date}$(Battles $\bowtie_{(battle=name)}$ Outcomes)
R2 := $\rho_{R2(ship2,result2,date2)}$(R1)
R3 := R1 $\bowtie_{(ship=ship2 \text{ AND } result=damaged \text{ AND } date < date2)}$ R2
R4 := π_{ship}(R3)

No results from sample data.

Exercise 2.4.4a

$\pi_{class,country}$

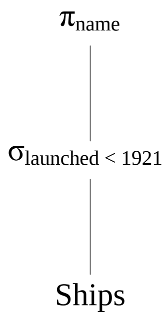
|

$\sigma_{bore \geq 16}$

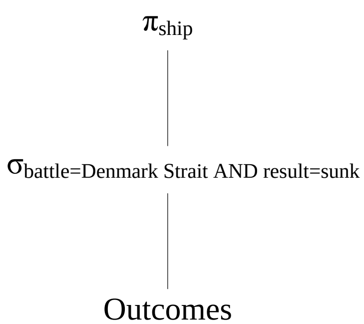
|

Classes

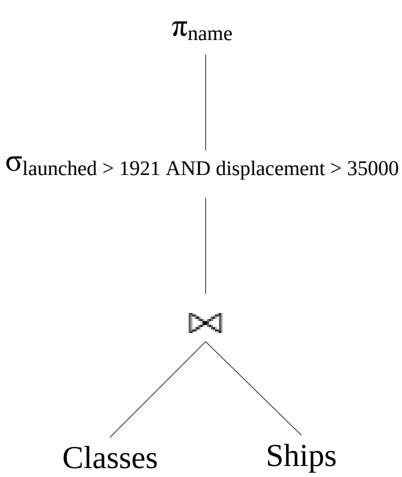
Exercise 2.4.4b



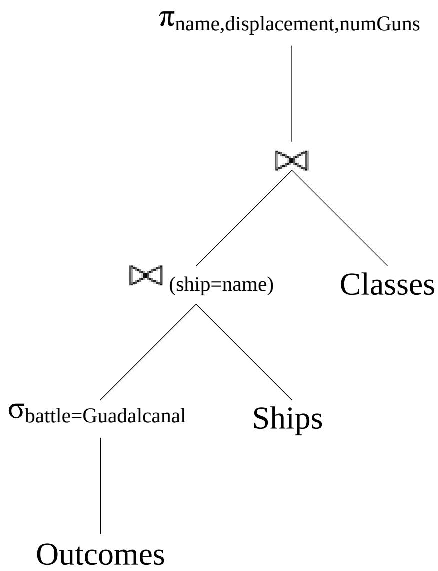
Exercise 2.4.4c



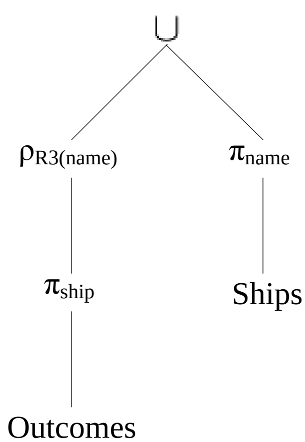
Exercise 2.4.4d



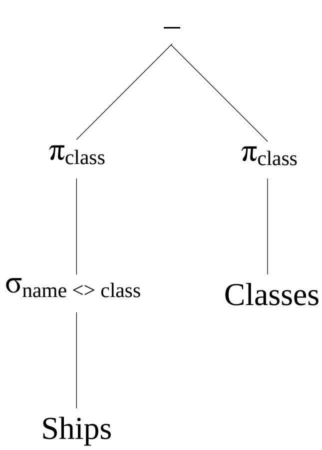
Exercise 2.4.4e



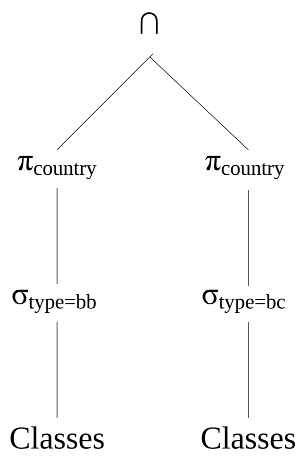
Exercise 2.4.4f



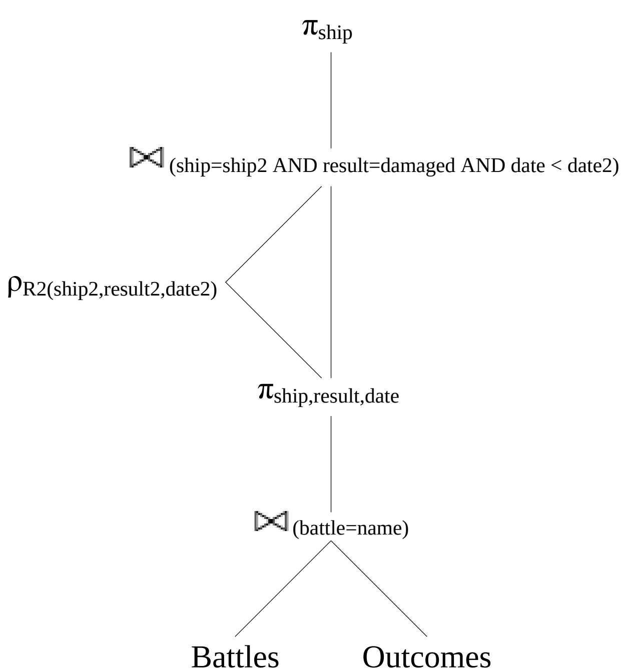
Exercise 2.4.4g



Exercise 2.4.4h



Exercise 2.4.4i



Exercise 2.4.5

The result of the natural join has only one attribute from each pair of equated attributes. On the other hand, the result of the theta-join has both columns of the attributes and their values are identical.

Exercise 2.4.6

Union

If we add a tuple to the arguments of the union operator, we will get all of the tuples of the original result and maybe the added tuple. If the added tuple is a duplicate tuple, then the set behavior will eliminate that tuple. Thus the union operator is monotone.

Intersection

If we add a tuple to the arguments of the intersection operator, we will get all of the tuples of the original result and maybe the added tuple. If the added tuple does not exist in the relation that it is added but does exist in the other relation, then the result set will include the added tuple. Thus the intersection operator is monotone.

Difference

If we add a tuple to the arguments of the difference operator, we may not get all of the tuples of the original result. Suppose we have relations R and S and we are computing R − S. Suppose also that tuple *t* is in R but not in S. The result of R − S would include tuple *t*. However, if we add tuple *t* to S, then the new result will not have tuple *t*. Thus the difference operator is not monotone.

Projection

If we add a tuple to the arguments of the projection operator, we will get all of the tuples of the original result and the projection of the added tuple. The projection operator only selects columns from the relation and does not affect the rows that are selected. Thus the projection operator is monotone.

Selection

If we add a tuple to the arguments of the selection operator, we will get all of the tuples of the original result and maybe the added tuple. If the added tuple satisfies the select condition, then it will be added to the new result. The original tuples are included in the new result because they still satisfy the select condition. Thus the selection operator is monotone.

Cartesian Product

If we add a tuple to the arguments of the Cartesian product operator, we will get all of the tuples of the original result and possibly additional tuples. The Cartesian product pairs the tuples of one relation with the tuples of another relation. Suppose that we are calculating R x S where R has m tuples and S has n tuples. If we add a tuple to R that is not already in R, then we expect the result of R x S to have $(m + 1) * n$ tuples. Thus the Cartesian product operator is monotone.

Natural Joins

If we add a tuple to the arguments of a natural join operator, we will get all of the tuples of the original result and possibly additional tuples. The new tuple can only create additional successful joins, not less. If, however, the added tuple cannot successfully join with any of the existing tuples, then we will have zero additional successful joins. Thus the natural join operator is monotone.

Theta Joins

If we add a tuple to the arguments of a theta join operator, we will get all of the tuples of the original result and possibly additional tuples. The theta join can be modeled by a Cartesian product followed by a selection on some condition. The new tuple can only create additional tuples in the result, not less. If, however, the added tuple does not satisfy the select condition, then no additional tuples will be added to the result. Thus the theta join operator is monotone.

Renaming

If we add a tuple to the arguments of a renaming operator, we will get all of the tuples of the original result and the added tuple. The renaming operator does not have any effect on whether a tuple is selected or not. In fact, the renaming operator will always return as many tuples as its argument. Thus the renaming operator is monotone.

Exercise 2.4.7a

If all the tuples of R and S are different, then the union has $n + m$ tuples, and this number is the maximum possible.

The minimum number of tuples that can appear in the result occurs if every tuple of one relation also appears in the other. Then the union has $\max(m , n)$ tuples.

Exercise 2.4.7b

If all the tuples in one relation can pair successfully with all the tuples in the other relation, then the natural join has $n * m$ tuples. This number would be the maximum possible.

The minimum number of tuples that can appear in the result occurs if none of the tuples of one relation can pair successfully with all the tuples in the other relation. Then the natural join has zero tuples.

Exercise 2.4.7c

If the condition C brings back all the tuples of R, then the cross product will contain $n * m$ tuples. This number would be the maximum possible.

The minimum number of tuples that can appear in the result occurs if the condition C brings back none of the tuples of R. Then the cross product has zero tuples.

Exercise 2.4.7d

Assuming that the list of attributes L makes the resulting relation $\pi_L(R)$ and relation S schema compatible, then the maximum possible tuples is *n*. This happens when all of the tuples of $\pi_L(R)$ are not in S.

The minimum number of tuples that can appear in the result occurs when all of the tuples in $\pi_L(R)$ appear in S. Then the difference has $\max(n - m, 0)$ tuples.

Exercise 2.4.8

Defining r as the schema of R and s as the schema of S:

1. $\pi_r(R \bowtie S)$
2. $R \bowtie \delta(\pi_{r \cap s}(S))$ where δ is the duplicate-elimination operator in Section 5.2 pg. 213
3. $R - (R - \pi_r(R \bowtie S))$

Exercise 2.4.9

Defining r as the schema of R

1. $R - \pi_r(R \bowtie S)$

Exercise 2.4.10

$\pi_{A1,A2 \cdots An}(R \bowtie S)$

Exercise 2.5.1a

$\sigma_{speed < 2.00 \text{ AND } price > 500}(PC) = \varnothing$

Model 1011 violates this constraint.

Exercise 2.5.1b

$\sigma_{screen < 15.4 \text{ AND } hd < 100 \text{ AND } price \geqslant 1000}(Laptop) = \varnothing$

Model 2004 violates the constraint.

Exercise 2.5.1c

$\pi_{maker}(\sigma_{type = laptop}(Product)) \cap \pi_{maker}(\sigma_{type = pc}(Product)) = \varnothing$

Manufacturers A,B,E violate the constraint.

Exercise 2.5.1d

This complex expression is best seen as a sequence of steps in which we define temporary relations R1 through R4 that stand for nodes of expression trees. Here is the sequence:

R1(maker, model, speed) := $\pi_{maker,model,speed}(Product \bowtie PC)$
R2(maker, speed) := $\pi_{maker,speed}(Product \bowtie Laptop)$
R3(model) := $\pi_{model}(R1 \bowtie_{R1.maker = R2.maker \text{ AND } R1.speed \leqslant R2.speed} R2)$

$R4(model) := \pi_{model}(PC)$

The constraint is $R4 \subseteq R3$
Manufacturers B,C,D violate the constraint.

Exercise 2.5.1e

$\pi_{model}(\sigma_{Laptop.ram > PC.ram \text{ AND } Laptop.price \leqslant PC.price}(PC \times Laptop)) = \varnothing$

Models 2002,2006,2008 violate the constraint.

Exercise 2.5.2a

$\pi_{class}(\sigma_{bore > 16}(Classes)) = \varnothing$

The Yamato class violates the constraint.

Exercise 2.5.2b

$\pi_{class}(\sigma_{numGuns > 9 \text{ AND } bore > 14}(Classes)) = \varnothing$

No violations to the constraint.

Exercise 2.5.2c

This complex expression is best seen as a sequence of steps in which we define temporary relations R1 through R5 that stand for nodes of expression trees. Here is the sequence:

$R1(class,name) := \pi_{class,name}(Classes \bowtie Ships)$
$R2(class2,name2) := \rho_{R2(class2,name2)}(R1)$
$R3(class3,name3) := \rho_{R3(class3,name3)}(R1)$
$R4(class,name,class2,name2) := R1 \bowtie_{(class = class2 \text{ AND } name <> name2)} R2$
$R5(class,name,class2,name2,class3,name3) := R4 \bowtie_{(class=class3 \text{ AND } name <> name3 \text{ AND } name2 <> name3)} R3$

The constraint is $R5 = \varnothing$
The Kongo, Iowa and Revenge classes violate the constraint.

Exercise 2.5.2d

$\pi_{country}(\sigma_{type = bb}(Classes)) \cap \pi_{country}(\sigma_{type = bc}(Classes)) = \varnothing$

Japan and Gt. Britain violate the constraint.

Exercise 2.5.2e

This complex expression is best seen as a sequence of steps in which we define temporary relations R1 through R5 that stand for nodes of expression trees. Here is the sequence:

$R1(ship,battle,result,class) := \pi_{ship,battle,result,class}(Outcomes \bowtie_{(ship = name)} Ships)$
$R2(ship,battle,result,numGuns) := \pi_{ship,battle,result,numGuns}(R1 \bowtie Classes)$
$R3(ship,battle) := \pi_{ship,battle}(\sigma_{numGuns < 9 \text{ AND } result = sunk}(R2))$
$R4(ship2,battle2) := \rho_{R4(ship2,battle2)}(\pi_{ship,battle}(\sigma_{numGuns > 9}(R2)))$
$R5(ship2) := \pi_{ship2}(R3 \bowtie_{(battle = battle2)} R4)$

The constraint is $R5 = \varnothing$
No violations to the constraint. Since there are some ships in the Outcomes table that are not in the Ships table, we are unable to determine the number of guns on that ship.

Exercise 2.5.3

Defining r as the schema $A_1,A_2,\cdots,A_n$ and s as the schema $B_1,B_2,\cdots,B_n$:

$\pi_r(R) \triangleright \pi_s(S) = \emptyset$ where $\triangleright$ is the antisemijoin

Exercise 2.5.4

The form of a constraint as $E_1 = E_2$ can be expressed as the other two constraints.
Using the "equating an expression to the empty set" method, we can simply say:

$E_1 - E_2 = \emptyset$

As a containment, we can simply say:

$E_1 \subseteq E_2$ AND $E_2 \subseteq E_1$

Thus, the form $E_1 = E_2$ of a constraint cannot express more than the two other forms discussed in this section.

Exercise 3.1.1

Answers for this exercise may vary because of different interpretations.

Some possible FDs:

Social Security number → name
Area code → state
Street address, city, state → zipcode

Possible keys:

{Social Security number, street address, city, state, area code, phone number}

Need street address, city, state to uniquely determine location. A person could have multiple addresses. The same is true for phones. These days, a person could have a landline and a cellular phone

Exercise 3.1.2

Answers for this exercise may vary because of different interpretations

Some possible FDs:

ID → x-position, y-position, z-position
ID → x-velocity, y-velocity, z-velocity
x-position, y-position, z-position → ID

Possible keys:

{ID}
{x-position, y-position, z-position}

The reason why the positions would be a key is no two molecules can occupy the same point.

Exercise 3.1.3a

The superkeys are any subset that contains A_1. Thus, there are $2^{(n-1)}$ such subsets, since each of the n-1 attributes A_2 through A_n may independently be chosen in or out.

Exercise 3.1.3b

The superkeys are any subset that contains A_1 or A_2. There are $2^{(n-1)}$ such subsets when considering A_1 and the n-1 attributes A_2 through A_n. There are $2^{(n-2)}$ such subsets when considering A_2 and the n-2 attributes A_3 through A_n. We do not count A_1 in these subsets because they are already counted in the first group of subsets. The total number of subsets is $2^{(n-1)} + 2^{(n-2)}$.

Exercise 3.1.3c

The superkeys are any subset that contains $\{A_1, A_2\}$ or $\{A_3, A_4\}$. There are $2^{(n-2)}$ such subsets when considering $\{A_1, A_2\}$ and the n-2 attributes A_3 through A_n. There are $2^{(n-2)} - 2^{(n-4)}$ such subsets when considering $\{A_3, A_4\}$ and attributes A_5 through A_n along with the individual attributes A_1 and A_2. We get the $2^{(n-4)}$ term because we have to discard the subsets that contain the key $\{A_1, A_2\}$ to avoid double counting. The total number of subsets is $2^{(n-2)} + 2^{(n-2)} - 2^{(n-4)}$.

Exercise 3.1.3d

The superkeys are any subset that contains $\{A_1, A_2\}$ or $\{A_1, A_3\}$. There are $2^{(n-2)}$ such subsets when considering $\{A_1, A_2\}$ and the n-2 attributes A_3 through A_n. There are $2^{(n-3)}$ such subsets when considering $\{A_1, A_3\}$ and the n-3 attributes A_4 through A_n We do not count A_2 in these subsets because they are already counted in the first group of subsets. The total number of subsets is $2^{(n-2)} + 2^{(n-3)}$.

Exercise 3.2.1a

We could try inference rules to deduce new dependencies until we are satisfied we have them all. A more systematic way is to consider the closures of all 15 nonempty sets of attributes.

For the single attributes we have $\{A\}^+ = A$, $\{B\}^+ = B$, $\{C\}^+ = ACD$, and $\{D\}^+ = AD$. Thus, the only new dependency we get with a single attribute on the left is C→A.

Now consider pairs of attributes:
$\{AB\}^+ = ABCD$, so we get new dependency AB→D. $\{AC\}^+ = ACD$, and AC→D is nontrivial. $\{AD\}^+ = AD$, so nothing new. $\{BC\}^+ = ABCD$, so we get BC→A, and BC→D. $\{BD\}^+ = ABCD$, giving us BD→A and BD→C. $\{CD\}^+ = ACD$, giving CD→A.

For the triples of attributes, $\{ACD\}^+ = ACD$, but the closures of the other sets are each ABCD. Thus, we get new dependencies ABC→D, ABD→C, and BCD→A.

Since $\{ABCD\}^+ = ABCD$, we get no new dependencies.

The collection of 11 new dependencies mentioned above are:
C→A, AB→D, AC→D, BC→A, BC→D, BD→A, BD→C, CD→A, ABC→D, ABD→C, and BCD→A.

Exercise 3.2.1b

From the analysis of closures above, we find that AB, BC, and BD are keys. All other sets either do not have ABCD as the closure or contain one of these three sets.

Exercise 3.2.1c

The superkeys are all those that contain one of those three keys. That is, a superkey that is not a key must contain B and more than one of A, C, and D. Thus, the (proper) superkeys are ABC, ABD, BCD, and ABCD.

Exercise 3.2.2a

i) For the single attributes we have $\{A\}^+$ = ABCD, $\{B\}^+$ = BCD, $\{C\}^+$ = C, and $\{D\}^+$ = D. Thus, the new dependencies are A→C and A→D.

Now consider pairs of attributes:
$\{AB\}^+$ = ABCD, $\{AC\}^+$ = ABCD, $\{AD\}^+$ = ABCD, $\{BC\}^+$ = BCD, $\{BD\}^+$ = BCD, $\{CD\}^+$ = CD. Thus the new dependencies are AB→C, AB→D, AC→B, AC→D, AD→B, AD→C, BC→D and BD→C.

For the triples of attributes, $\{BCD\}^+$ = BCD, but the closures of the other sets are each ABCD. Thus, we get new dependencies ABC→D, ABD→C, and ACD→B.

Since $\{ABCD\}^+$ = ABCD, we get no new dependencies.

The collection of 13 new dependencies mentioned above are:
A→C, A→D, AB→C, AB→D, AC→B, AC→D, AD→B, AD→C, BC→D, BD→C, ABC→D, ABD→C and ACD→B.

ii) For the single attributes we have $\{A\}^+$ = A, $\{B\}^+$ = B, $\{C\}^+$ = C, and $\{D\}^+$ = D. Thus, there are no new dependencies.

Now consider pairs of attributes:
$\{AB\}^+$ = ABCD, $\{AC\}^+$ = AC, $\{AD\}^+$ = ABCD, $\{BC\}^+$ = ABCD, $\{BD\}^+$ = BD, $\{CD\}^+$ = ABCD. Thus the new dependencies are AB→D, AD→C, BC→A and CD→B.

For the triples of attributes, all the closures of the sets are each ABCD. Thus, we get new dependencies ABC→D, ABD→C, ACD→B and BCD→A.

Since $\{ABCD\}^+$ = ABCD, we get no new dependencies.

The collection of 8 new dependencies mentioned above are:
AB→D, AD→C, BC→A, CD→B, ABC→D, ABD→C, ACD→B and BCD→A.

iii) For the single attributes we have $\{A\}^+$ = ABCD, $\{B\}^+$ = ABCD, $\{C\}^+$ = ABCD, and $\{D\}^+$ = ABCD. Thus, the new dependencies are A→C, A→D, B→D, B→A, C→A, C→B, D→B and D→C.

Since all the single attributes' closures are ABCD, any superset of the single attributes will also lead to a closure of ABCD. Knowing this, we can enumerate the rest of the new dependencies.

The collection of 24 new dependencies mentioned above are:
A→C, A→D, B→D, B→A, C→A, C→B, D→B, D→C, AB→C, AB→D, AC→B, AC→D, AD→B, AD→C, BC→A, BC→D, BD→A, BD→C, CD→A, CD→B, ABC→D, ABD→C, ACD→B and BCD→A.

Exercise 3.2.2b

i) From the analysis of closures in 3.2.2a(i), we find that the only key is A. All other sets either do not have ABCD as the closure or contain A.

ii) From the analysis of closures 3.2.2a(ii), we find that AB, AD, BC, and CD are keys. All other sets either do not have ABCD as the closure or contain one of these four sets.

iii) From the analysis of closures 3.2.2a(iii), we find that A, B, C and D are keys. All other sets either do not have ABCD as the closure or contain one of these four sets.

Exercise 3.2.2c

i) The superkeys are all those sets that contain one of the keys in 3.2.2b(i). The superkeys are AB, AC, AD, ABC, ABD, ACD, BCD and ABCD.

ii) The superkeys are all those sets that contain one of the keys in 3.2.2b(ii). The superkeys are ABC, ABD, ACD, BCD and ABCD.

iii) The superkeys are all those sets that contain one of the keys in 3.2.2b(iii). The superkeys are AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD and ABCD.

Exercise 3.2.3a

Since $A_1A_2 \cdots A_nC$ contains $A_1A_2 \cdots A_n$, then the closure of $A_1A_2 \cdots A_nC$ contains B. Thus it follows that $A_1A_2 \cdots A_nC \rightarrow B$.

Exercise 3.2.3b

From 3.2.3a, we know that $A_1A_2 \cdots A_nC \rightarrow B$. Using the concept of trivial dependencies, we can show that $A_1A_2 \cdots A_nC \rightarrow C$. Thus $A_1A_2 \cdots A_nC \rightarrow BC$.

Exercise 3.2.3c

From $A_1A_2 \cdots A_nE_1E_2 \cdots E_j$, we know that the closure contains $B_1B_2 \cdots B_m$ because of the FD $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$. The $B_1B_2 \cdots B_m$ and the $E_1E_2 \cdots E_j$ combine to form the $C_1C_2 \cdots C_k$. Thus the closure of $A_1A_2 \cdots A_nE_1E_2 \cdots E_j$ contains D as well. Thus, $A_1A_2 \cdots A_nE_1E_2 \cdots E_j \rightarrow D$.

Exercise 3.2.3d

From $A_1A_2 \cdots A_nC_1C_2 \cdots C_k$, we know that the closure contains $B_1B_2 \cdots B_m$ because of the FD $A_1A_2 \cdots A_n \rightarrow B_1B_2 \cdots B_m$. The $C_1C_2 \cdots C_k$ also tell us that the closure of $A_1A_2 \cdots A_nC_1C_2 \cdots C_k$ contains $D_1D_2 \cdots D_j$. Thus, $A_1A_2 \cdots A_nC_1C_2 \cdots C_k \rightarrow B_1B_2 \cdots B_kD_1D_2 \cdots D_j$.

Exercise 3.2.4a

If attribute A represented Social Security Number and B represented a person’s name, then we would assume A→B but B→A would not be valid because there may be many people with the same name and different Social Security Numbers.

Exercise 3.2.4b

Let attribute A represent Social Security Number, B represent gender and C represent name. Surely Social Security Number and gender can uniquely identify a person's name (i.e. $AB \rightarrow C$). A Social Security Number can also uniquely identify a person's name (i.e. $A \rightarrow C$). However, gender does not uniquely determine a name (i.e. $B \rightarrow C$ is not valid).

Exercise 3.2.4c

Let attribute A represent latitude and B represent longitude. Together, both attributes can uniquely determine C, a point on the world map (i.e. $AB \rightarrow C$). However, neither A nor B can uniquely identify a point (i.e. $A \rightarrow C$ and $B \rightarrow C$ are not valid).

Exercise 3.2.5

Given a relation with attributes $A_1A_2 \cdots A_n$, we are told that there are no functional dependencies of the form $B_1B_2 \cdots B_{n-1} \rightarrow C$ where $B_1B_2 \cdots B_{n-1}$ is n-1 of the attributes from $A_1A_2 \cdots A_n$ and C is the remaining attribute from $A_1A_2 \cdots A_n$. In this case, the set $B_1B_2 \cdots B_{n-1}$ and any subset do not functionally determine C. Thus the only functional dependencies that we can make are ones where C is on both the left and right hand sides. All of these functional dependencies would be trivial and thus the relation has no nontrivial FD's.

Exercise 3.2.6

Let's prove this by using the contrapositive. We wish to show that if X^+ is not a subset of Y^+, then it must be that X is not a subset of Y.

If X^+ is not a subset of Y^+, there must be attributes $A_1A_2 \cdots A_n$ in X^+ that are not in Y^+. If any of these attributes were originally in X, then we are done because Y does not contain any of the $A_1A_2 \cdots A_n$. However, if the $A_1A_2 \cdots A_n$ were added by the closure, then we must examine the case further. Assume that there was some FD $C_1C_2 \cdots C_m \rightarrow A_1A_2 \cdots A_j$ where $A_1A_2 \cdots A_j$ is some subset of $A_1A_2 \cdots A_n$. It must be then that $C_1C_2 \cdots C_m$ or some subset of $C_1C_2 \cdots C_m$ is in X. However, the attributes $C_1C_2 \cdots C_m$ cannot be in Y because we assumed that attributes $A_1A_2 \cdots A_n$ are only in X^+ and are not in Y^+. Thus, X is not a subset of Y.

By proving the contrapositive, we have also proved if $X \subseteq Y$, then $X^+ \subseteq Y^+$.

Exercise 3.2.7

The algorithm to find X^+ is outlined on pg. 76. Using that algorithm, we can prove that $(X^+)^+ = X^+$. We will do this by using a proof by contradiction.

Suppose that $(X^+)^+ \neq X^+$. Then for $(X^+)^+$, it must be that some FD allowed additional attributes to be added to the original set X^+. For example, $X^+ \rightarrow A$ where A is some attribute not in X^+. However, if this were the case, then X^+ would not be the closure of X. The closure of X would have to include A as well. This contradicts the fact that we were given the closure of X, X^+. Therefore, it must be that $(X^+)^+ = X^+$ or else X^+ is not the closure of X.

Exercise 3.2.8a

If all sets of attributes are closed, then there cannot be any nontrivial functional dependencies. Suppose $A_1A_2...A_n \to B$ is a nontrivial dependency. Then $\{A_1A_2...A_n\}^+$ contains B and thus $A_1A_2...A_n$ is not closed.

Exercise 3.2.8b

If the only closed sets are ø and {A,B,C,D}, then the following FDs hold:

A→B	A→C	A→D
B→A	B→C	B→D
C→A	C→B	C→D
D→A	D→B	D→C

AB→C AB→D
AC→B AC→D
AD→B AD→C
BC→A BC→D
BD→A BD→C
CD→A CD→B
ABC→D
ABD→C
ACD→B
BCD→A

Exercise 3.2.8c

If the only closed sets are ø, {A,B} and {A,B,C,D}, then the following FDs hold:

A→B
B→A

C→A	C→B	C→D
D→A	D→B	D→C

AC→B AC→D
AD→B AD→C
BC→A BC→D
BD→A BD→C
CD→A CD→B
ABC→D
ABD→C
ACD→B
BCD→A

Exercise 3.2.9

We can think of this problem as a situation where the attributes A,B,C represent cities and the functional dependencies represent one way paths between the cities. The minimal bases are the minimal number of pathways that are needed to connect the cities. We do not want to create another roadway if the two cities are already connected.

The systematic way to do this would be to check all possible sets of the pathways. However, we can simplify the situation by noting that it takes more than two pathways to visit the two other cities and come back. Also, if we find a set of pathways that is minimal, adding additional pathways will not create another minimal set.

The two sets of minimal bases that were given in example 3.11 are:

{A→B, B→C, C→A}
{A→B, B→A, B→C, C→B}

The additional sets of minimal bases are:

{C→B, B→A, A→C}
{A→B, A→C, B→A, C→A}
{A→C, B→C, C→A, C→B}

Exercise 3.2.10a

We need to compute the closures of all subsets of {ABC}, although there is no need to think about the empty set or the set of all three attributes. Here are the calculations for the remaining six sets:
$\{A\}^+$=A
$\{B\}^+$=B
$\{C\}^+$=ACE
$\{AB\}^+$=ABCDE
$\{AC\}^+$=ACE
$\{BC\}^+$=ABCDE

We ignore D and E, so a basis for the resulting functional dependencies for ABC is: C→A and AB→C. Note that BC->A is true, but follows logically from C->A, and therefore may be omitted from our list.

Exercise 3.2.10b

We need to compute the closures of all subsets of {ABC}, although there is no need to think about the empty set or the set of all three attributes. Here are the calculations for the remaining six sets:
$\{A\}^+$=AD
$\{B\}^+$=B
$\{C\}^+$=C
$\{AB\}^+$=ABDE

$\{AC\}^+$=ABCDE
$\{BC\}^+$=BC

We ignore D and E, so a basis for the resulting functional dependencies for ABC is: AC→B.

Exercise 3.2.10c

We need to compute the closures of all subsets of {ABC}, although there is no need to think about the empty set or the set of all three attributes. Here are the calculations for the remaining six sets:
$\{A\}^+$=A
$\{B\}^+$=B
$\{C\}^+$=C
$\{AB\}^+$=ABD
$\{AC\}^+$=ABCDE
$\{BC\}^+$=ABCDE

We ignore D and E, so a basis for the resulting functional dependencies for ABC is: AC→B and BC→A.

Exercise 3.2.10d

We need to compute the closures of all subsets of {ABC}, although there is no need to think about the empty set or the set of all three attributes. Here are the calculations for the remaining six sets:
$\{A\}^+$=ABCDE
$\{B\}^+$=ABCDE
$\{C\}^+$=ABCDE
$\{AB\}^+$=ABCDE
$\{AC\}^+$=ABCDE
$\{BC\}^+$=ABCDE

We ignore D and E, so a basis for the resulting functional dependencies for ABC is: A→B, B→C and C→A.

Exercise 3.2.11

For step one of Algorithm 3.7, suppose we have the FD ABC→DE. We want to use Armstrong's Axioms to show that ABC→D and ABC→E follow. Surely the functional dependencies DE→D and DE→E hold because they are trivial and follow the reflexivity property. Using the transitivity rule, we can derive the FD ABC→D from the FDs ABC→DE and DE→D. Likewise, we can do the same for ABC→DE and DE→E and derive the FD ABC→E.

For steps two through four of Algorithm 3.7, suppose we have the initial set of attributes of the closure as ABC. Suppose also that we have FDs C→D and D→E. According to Algorithm 3.7, the closure should become ABCDE. Taking the FD C→D and augmenting both sides with attributes AB we get the FD ABC→ABD. We can use the splitting method in step one to get the FD ABC→D. Since D is not in the closure, we can add attribute D. Taking the FD D→E and augmenting both sides with attributes ABC we get the FD ABCD→ABCDE. Using again the splitting method in step one we get the FD ABCD→E. Since E is not in the closure, we can add attribute E.

Given a set of FDs, we can prove that a FD F follows by taking the closure of the left side of FD F. The steps to compute the closure in Algorithm 3.7 can be mimicked by Armstrong's axioms and thus we can prove F from the given set of FDs using Armstrong's axioms.

Exercise 3.3.1a

In the solution to Exercise 3.2.1 we found that there are 14 nontrivial dependencies, including the three given ones and eleven derived dependencies. They are: C→A, C→D, D→A, AB→D, AB→ C, AC→D, BC→A, BC→D, BD→A, BD→C, CD→A, ABC→D, ABD→C, and BCD→A.

We also learned that the three keys were AB, BC, and BD. Thus, any dependency above that does not have one of these pairs on the left is a BCNF violation. These are: C→A, C→D, D→A, AC→D, and CD→A.

One choice is to decompose using the violation C→D. Using the above FDs, we get ACD and BC as decomposed relations. BC is surely in BCNF, since any two-attribute relation is. Using Algorithm 3.12 to discover the projection of FDs on relation ACD, we discover that ACD is not in BCNF since C is its only key. However, D→A is a dependency that holds in ABCD and therefore holds in ACD. We must further decompose ACD into AD and CD. Thus, the three relations of the decomposition are BC, AD, and CD.

Exercise 3.3.1b

By computing the closures of all 15 nonempty subsets of ABCD, we can find all the nontrivial FDs. They are B→C, B→D, AB→C, AB→D, BC→D, BD→C, ABC→D and ABD→C. From the closures we can also deduce that the only key is AB. Thus, any dependency above that does not contain AB on the left is a BCNF violation. These are: B→C, B→D, BC→D and BD→C.

One choice is to decompose using the violation B→C. Using the above FDs, we get BCD and AB as decomposed relations. AB is surely in BCNF, since any two-attribute relation is. Using Algorithm 3.12 to discover the projection of FDs on relation BCD, we discover that BCD is in BCNF since B is its only key and the projected FDs all have B on the left side. Thus the two relations of the decomposition are AB and BCD.

Exercise 3.3.1c

In the solution to Exercise 3.2.2(ii), we found that there are 12 nontrivial dependencies, including the four given ones and the eight derived ones. They are AB→C, BC→D, CD→A, AD→B, AB→D, AD→C, BC→A, CD→B, ABC→D, ABD→C, ACD→B and BCD→A.

We also found out that the keys are AB, AD, BC, and CD. Thus, any dependency above that does not have one of these pairs on the left is a BCNF violation. However, all of the FDs contain a key on the left so there are no BCNF violations.

No decomposition is necessary since all the FDs do not violate BCNF.

Exercise 3.3.1d

In the solution to Exercise 3.2.2(iii), we found that there are 28 nontrivial dependencies, including the four given ones and the 24 derived ones. They are A→B, B→C, C→D, D→A, A→C, A→D, B→D, B→A, C→A, C→B, D→B, D→C, AB→C, AB→D, AC→B, AC→D, AD→B, AD→C, BC→A, BC→D, BD→A, BD→C, CD→A, CD→B, ABC→D, ABD→C, ACD→B and BCD→A.

We also found out that the keys are A,B,C,D. Thus, any dependency above that does not have one of these attributes on the left is a BCNF violation. However, all of the FDs contain a key on the left so there are no BCNF violations.

No decomposition is necessary since all the FDs do not violate BCNF.

Exercise 3.3.1e

By computing the closures of all 31 nonempty subsets of ABCDE, we can find all the nontrivial FDs. They are AB→C, DE→C, B→D, AB→D, BC→D, BE→C, BE→D, ABC→D, ABD→C, ABE→C, ABE→D, ADE→C, BCE→D, BDE→C, ABCE→D, and ABDE→C. From the closures we can also deduce that the only key is ABE. Thus, any dependency above that does not contain ABE on the left is a BCNF violation. These are: AB→C, DE→C, B→D, AB→D, BC→D, BE→C, BE→D, ABC→D, ABD→C, ADE→C, BCE→D and BDE→C.

One choice is to decompose using the violation AB→C. Using the above FDs, we get ABCD and ABE as decomposed relations. Using Algorithm 3.12 to discover the projection of FDs on relation ABCD, we discover that ABCD is not in BCNF since AB is its only key and the FD B→D follows for ABCD. Using violation B→D to further decompose, we get BD and ABC as decomposed relations. BD is in BCNF because it is a two-attribute relation. Using Algorithm 3.12 again, we discover that ABC is in BCNF since AB is the only key and AB→C is the only nontrivial FD. Going back to relation ABE, following Algorithm 3.12 tells us that ABE is in BCNF because there are no keys and no nontrivial FDs. Thus the three relations of the decomposition are ABC, BD and ABE.

Exercise 3.3.1f

By computing the closures of all 31 nonempty subsets of ABCDE, we can find all the nontrivial FDs. They are: C→B, C→D, C→E, D→B, D→E, AB→C, AB→D, AB→E, AC→B, AC→D, AC→E, AD→B, AD→C, AD→E, BC→D, BC→E, BD→E, CD→B, CD→E, CE→B, CE→D, DE→B, ABC→D, ABC→E, ABD→C, ABD→E, ABE→C, ABE→D, ACD→B, ACD→E, ACE→B, ACE→D, ADE→B, ADE→C, BCD→E, BCE→D, CDE→B, ABCD→E, ABCE→D, ABDE→C and ACDE→B. From the closures we can also deduce that the keys are AB, AC and AD. Thus, any dependency above that does not contain one of the above pairs on the left is a BCNF violation. These are: C→B, C→D,

C→E, D→B, D→E, BC→D, BC→E, BD→E, CD→B, CD→E, CE→B, CE→D, DE→B, BCD→E, BCE→D and CDE→B.

One choice is to decompose using the violation D→B. Using the above FDs, we get BDE and ABC as decomposed relations. Using Algorithm 3.12 to discover the projection of FDs on relation BDE, we discover that BDE is in BCNF since D, BD, DE are the only keys and all the projected FDs contain D, BD, or DE in the left side. Going back to relation ABC, following Algorithm 3.12 tells us that ABC is not in BCNF because since AB and AC are its only keys and the FD C→B follows for ABC. Using violation C→B to further decompose, we get BC and AC as decomposed relations. Both BC and AC are in BCNF because they are two-attribute relations. Thus the three relations of the decomposition are BDE, BC and AC.

Exercise 3.3.2

Yes, we will get the same result. Both A→B and A→BC have A on the left side and part of the process of decomposition involves finding $\{A\}^+$ to form one decomposed relation and A plus the rest of the attributes not in $\{A\}^+$ as the second relation. Both cases yield the same decomposed relations.

Exercise 3.3.3

Yes, we will still get the same result. Both A→B and A→BC have A on the left side and part of the process of decomposition involves finding $\{A\}^+$ to form one decomposed relation and A plus the rest of the attributes not in $\{A\}^+$ as the second relation. Both cases yield the same decomposed relations.

Exercise 3.3.4

This is taken from Example 3.21 pg. 95.
Suppose that an instance of relation R only contains two tuples.

A	B	C
1	2	3
4	2	5

The projections of R onto the relations with schemas {A,B} and {B,C} are:

A	B
1	2
4	2

B	C
2	3
2	5

If we do a natural join on the two projections, we will get:

A	B	C

1	2	3
1	2	5
4	2	3
4	2	5

The result of the natural join is not equal to the original relation R.

Exercise 3.4.1a

This is the initial tableau:

A	B	C	D	E
a	b	c	d_1	e_1
a_1	b	c	d	e_1
a	b_1	c	d_1	e

This is the final tableau after applying FDs B→E and CE→A.

A	B	C	D	E
a	b	c	d_1	e_1
a	b	c	d	e_1
a	b_1	c	d_1	e

Since there is not an unsubscripted row, the decomposition for R is not lossless for this set of FDs.

We can use the final tableau as an instance of R as an example for why the join is not lossless. The projected relations are:

A	B	C
a	b	c
a	b_1	c

B	C	D
b	c	d_1
b	c	d
b_1	c	d_1

A	C	E
a	c	e_1
a	c	e

The joined relation is:

A	B	C	D	E
a	b	c	d_1	e_1
a	b	c	d	e_1
a	b_1	c	d_1	e_1
a	b	c	d_1	e
a	b	c	d	e
a	b_1	c	d_1	e

The joined relation has three more tuples than the original tableau.

Exercise 3.4.1b

This is the initial tableau:

A	B	C	D	E
a	b	c	d_1	e_1
a_1	b	c	d	e_1
a	b_1	c	d_1	e

This is the final tableau after applying FDs AC→E and BC→D

A	B	C	D	E
a	b	c	d	e
a_1	b	c	d	e_1
a	b_1	c	d_1	e

Since there is an unsubscripted row, the decomposition for R is lossless for this set of FDs.

Exercise 3.4.1c

This is the initial tableau:

A	B	C	D	E
a	b	c	d_1	e_1
a_1	b	c	d	e_1
a	b_1	c	d_1	e

This is the final tableau after applying FDs A→D, D→E and B→D.

A	B	C	D	E
a	b	c	d	e
a_1	b	c	d	e
a	b_1	c	d	e

Since there is an unsubscripted row, the decomposition for R is lossless for this set of FDs.

Exercise 3.4.1d

This is the initial tableau:

A	B	C	D	E
a	b	c	d_1	e_1
a_1	b	c	d	e_1
a	b_1	c	d_1	e

This is the final tableau after applying FDs A→D, CD→E and E→D

A	B	C	D	E
a	b	c	d	e
a_1	b	c	d	e
a	b_1	c	d	e

Since there is an unsubscripted row, the decomposition for R is lossless for this set of FDs.

Exercise 3.4.2

When we decompose a relation into BCNF, we will project the FDs onto the decomposed relations to get new sets of FDs. These dependencies are preserved if the union of these new sets is equivalent to the original set of FDs.

For the FDs of 3.4.1a, the dependencies are not preserved. The union of the new sets of FDs is CE→A. However, the FD B→E is not in the union and cannot be derived. Thus the two sets of FDs are not equivalent.

For the FDs of 3.4.1b, the dependencies are preserved. The union of the new sets of FDs is AC→E and BC→D. This is precisely the same as the original set of FDs and thus the two sets of FDs are equivalent.

For the FDs of 3.4.1c, the dependencies are not preserved. The union of the new sets of FDs is B→D and A→E. The FDs A→D and D→E are not in the union and cannot be derived. Thus the two sets of FDs are not equivalent.

For the FDs of 3.4.1d, the dependencies are not preserved. The union of the new sets of FDs is AC→E. However, the FDs A→D, CD→E and E→D are not in the union and cannot be derived. Thus the two sets of FDs are not equivalent.

Exercise 3.5.1a

In the solution to Exercise 3.3.1a we found that there are 14 nontrivial dependencies. They are: C→A, C→D, D→A, AB→D, AB→ C, AC→D, BC→A, BC→D, BD→A, BD→C, CD→A, ABC→D, ABD→C, and BCD→A.

We also learned that the three keys were AB, BC, and BD. Since all the attributes on the right sides of the FDs are prime, there are no 3NF violations.

Since there are no 3NF violations, it is not necessary to decompose the relation.

Exercise 3.5.1b

In the solution to Exercise 3.3.1b we found that there are 8 nontrivial dependencies. They are B→C, B→D, AB→C, AB→D, BC→D, BD→C, ABC→D and ABD→C.

We also found out that the only key is AB. FDs where the left side is not a superkey or the attributes on the right are not part of some key are 3NF violations. The 3NF violations are B→C, B→D, BC→D and BD→C.

Using algorithm 3.26, we can decompose into relations using the minimal basis B→C and B→D. The resulting decomposed relations would be BC and BD. However, none of these two sets of attributes is a superkey. Thus we add relation AB to the result. The final set of decomposed relations is BC, BD and AB.

Exercise 3.5.1c

In the solution to Exercise 3.3.1c we found that there are 12 nontrivial dependencies. They are AB→C, BC→D, CD→A, AD→B, AB→D, AD→C, BC→A, CD→B, ABC→D, ABD→C, ACD→B and BCD→A.

We also found out that the keys are AB, AD, BC, and CD. Since all the attributes on the right sides of the FDs are prime, there are no 3NF violations.

Since there are no 3NF violations, it is not necessary to decompose the relation.

Exercise 3.5.1d

In the solution to Exercise 3.3.1d we found that there are 28 nontrivial dependencies. They are A→B, B→C, C→D, D→A, A→C, A→D, B→D, B→A, C→A, C→B, D→B, D→C, AB→C, AB→D, AC→B, AC→D, AD→B, AD→C, BC→A, BC→D, BD→A, BD→C, CD→A, CD→B, ABC→D, ABD→C, ACD→B and BCD→A.

We also found out that the keys are A,B,C,D. Since all the attributes on the right sides of the FDs are prime, there are no 3NF violations.

Since there are no 3NF violations, it is not necessary to decompose the relation.

Exercise 3.5.1e

In the solution to Exercise 3.3.1e we found that there are 16 nontrivial dependencies. They are AB→C, DE→C, B→D, AB→D, BC→D, BE→C, BE→D, ABC→D, ABD→C, ABE→C, ABE→D, ADE→C, BCE→D, BDE→C, ABCE→D, and ABDE→C.

We also found out that the only key is ABE. FDs where the left side is not a superkey or the attributes on the right are not part of some key are 3NF violations. The 3NF violations are AB→C, DE→C, B→D, AB→D, BC→D, BE→C, BE→D, ABC→D, ABD→C, ADE→C, BCE→D and BDE→C.

Using algorithm 3.26, we can decompose into relations using the minimal basis AB→C, DE→C and B→D. The resulting decomposed relations would be ABC, CDE and BD. However, none of these three sets of attributes is a superkey. Thus we add relation ABE to the result. The final set of decomposed relations is ABC, CDE, BD and ABE.

Exercise 3.5.1f

In the solution to Exercise 3.3.1f we found that there are 41 nontrivial dependencies. They are: C→B, C→D, C→E, D→B, D→E, AB→C, AB→D, AB→E, AC→B, AC→D, AC→E, AD→B, AD→C, AD→E, BC→D, BC→E, BD→E, CD→B, CD→E, CE→B, CE→D, DE→B, ABC→D, ABC→E, ABD→C, ABD→E, ABE→C, ABE→D, ACD→B, ACD→E, ACE→B, ACE→D, ADE→B, ADE→C, BCD→E, BCE→D, CDE→B, ABCD→E, ABCE→D, ABDE→C and ACDE→B.

We also found out that the keys are AB, AC and AD. FDs where the left side is not a superkey or the attributes on the right are not part of some key are 3NF violations. The 3NF violations are C→E, D→E, BC→E, BD→E, CD→E and BCD→E.

Using algorithm 3.26, we can decompose into relations using the minimal basis AB→C, C→D, D→B and D→E. The resulting decomposed relations would be ABC, CD, BD and DE. Since relation ABC contains a key, we can stop with the decomposition. The final set of decomposed relations is ABC, CD, BD and DE.

Exercise 3.5.2a

The usual procedure to find the keys would be to take the closure of all 63 nonempty subsets. However, if we notice that none of the right sides of the FDs contains attributes H and S. Thus we know that attributes H and S must be part of any key. We eventually will find out that HS is the only key for the Courses relation.

Exercise 3.5.2b

The first step to verify that the given FDs are their own minimal basis is to check to see if any of the FDs can be removed. However, if we remove any one of the five FDs, the remaining four FDs do not imply the removed FD.

The second step to verify that the given FDs are their own minimal basis is to check to see if any of the left sides of an FD can have one or more attributes removed without losing the dependencies. However, this is not the case for the four FDs that contain two attributes on the left side.

Thus, the given set of FDs has been verified to be the minimal basis.

Exercise 3.5.2c

Since the only key is HS, the given set of FDs has some dependencies that violate 3NF. We also know that the given set of FDs is a minimal basis. Thus the decomposed relations are CT, HRC, HTR, HSR and CSG. Since the relation HSR contains a key, we do not need to add an additional relation. The final set of decomposed relations is CT, HRC, HTR, HSR and CSG.

None of the decomposed relations violate BCNF. This can be verified by projecting the given set of FDs onto each of the decomposed relations. All of the projections of FDs have superkeys on their left sides.

Exercise 3.5.3

The usual procedure to find the keys would be to take the closure of all 63 nonempty subsets. However, if we notice that none of the right sides of the FDs contains attributes I and S. Thus we know that attributes I and S must be part of any key. We eventually will find out that IS is the only key for the Stocks relation.

The first step to verify that the given FDs are their own minimal basis is to check to see if any of the FDs can be removed. However, if we remove any one of the four FDs, the remaining three FDs do not imply the removed FD.

The second step to verify that the given FDs are their own minimal basis is to check to see if any of the left sides of an FD can have one or more attributes removed without losing the dependencies. However, this is not the case for the one FD that contains two attributes on the left side.

Thus, the given set of FDs has been verified to be the minimal basis.

Since the only key is IS, the given set of FDs has some dependencies that violate 3NF. We also know that the given set of FDs is a minimal basis. Thus the decomposed relations are SD, IB, ISQ and BO. Since the relation ISQ contains a key, we do not need to add an additional relation. The final set of decomposed relations is SD, IB, ISQ and BO.

Exercise 3.5.4

This is the initial tableau:

A	B	C	D	E
a	b	c	d_1	e_1
a	b_2	c_2	d	e_2
a	b	c_3	d_3	e

This is the final tableau after applying FDs AB→C, C→B and A→D.

A	B	C	D	E
a	b	c	d	e_1
a	b_2	c_2	d	e_2
a	b	c	d	e

Since there is an unsubscripted row, the decomposition for R is lossless for this set of FDs.

Exercise 3.5.5

Suppose that our relation relates to the work environment and has three attributes, Name, RoomNumber and ComputerID. Suppose also that only the following FDs hold:

Name→RoomNumber
ComputerID→RoomNumber

The first FD says that a person can only be in one office at a time. The second FD says that each computer can only be associated with one office at a time (i.e. the computer is fixed to each room). None of the left sides of the FDs are keys and the respective right side attributes do not belong to any key. Using algorithm 3.20 to decompose, we take the violating FD Name→RoomNumber and use it to decompose the initial relation into two smaller relations:

R1(Name,RoomNumber)
R2(Name,ComputerID)

We do not need to further decompose the two relations so we are done. However, we have lost the FD ComputerID→RoomNumber with this decomposition. Suppose that 'John Doe' works in room number '1'. Suppose also that computer ID 'A' is located in room '1'. The original relation would contain the tuple:

{ 'John Doe', 1, 'A' }

If we follow the decomposition, we would expect the tuple above to be broken into two smaller tuples:

{ 'John Doe', 1}
{ 'John Doe', 'A' }

Suppose that 'John Doe' moves to room 2 and 'Jane Doe' moves into room 1 and that 'Jane Doe' is using computer ID 'A'. The following tuples would be in the respective relations:

{ 'John Doe', 2}
{ 'Jane Doe', 1}
{ 'John Doe', 'A' }
{ 'Jane Doe', 'A' }

If we were to join back the two relations, we would get:

{ 'John Doe', 2, 'A' }
{ 'Jane Doe', 1, 'A' }

which violates the FD ComputerID→RoomNumber

Exercise 3.6.1

Since $A \rightarrow\rightarrow B$, and all the tuples have the same value for attribute A, we can pair the B-value from any tuple with the value of the remaining attribute C from any other tuple. Thus, we know that R must have at least the nine tuples of the form (a,b,c), where b is any of b_1, b_2, or b_3, and c is any of c_1, c_2, or c_3. That is, we can derive, using the definition of a multivalued dependency, that each of the tuples (a,b_1,c_2), (a,b_1,c_3), (a,b_2,c_1), (a,b_2,c_3), (a,b_3,c_1), and (a,b_3,c_2) are also in R.

Exercise 3.6.2a

First, people have unique Social Security numbers and unique birthdates. Thus, we expect the functional dependencies ssNo→name and ssNo→birthdate hold. The same applies to children, so we expect childSSNo→childname and childSSNo→childBirthdate. Finally, an automobile has a unique brand, so we expect autoSerialNo→autoMake.

There are two multivalued dependencies that do not follow from these functional dependencies. First, the information about one child of a person is independent of other information about that person. That is, if a person with social security number *s* has a tuple with *cn,cs,cb*, then if there is any other tuple *t* for the same person, there will also be another tuple that agrees with t except that it has *cn, cs, cb* in its components for the child name, child Social Security number, and child birthdate. That is the multivalued dependency

$$ssNo\ childSSNo \to\to childName\ childBirthdate$$

Similarly, an automobile serial number and make are independent of any of the other attributes, so we expect the multivalued dependency

$$ssNo \to\to autoSerialNo\ autoMake$$

The dependencies are summarized below:

ssNo → name, birthdate
childSSNo → childName, childBirthdate
autoSerialNo → autoMake
ssNo →→ childSSNo, childName, childBirthdate
ssNo →→ autoSerialNo, autoMake

Exercise 3.6.2b

We suggest the relation schemas:

{ssNo, name, birthdate}
{ssNo, childSSNo}
{childSSNo, childName childBirthdate}
{ssNo, autoSerialNo}
{autoSerialNo, autoMake}

An initial decomposition based on the two multivalued dependencies would give us:

{ssNo, name, birthDate}
{ssNo, childSSNo, childName, childBirthdate}
{ssNo, autoSerialNo, autoMake}

Functional dependencies force us to decompose the second and third of these.

Exercise 3.6.3a

Since there are no functional dependencies, the only key is all four attributes, ABCD. Thus, each of the nontrvial multivalued dependencies A→→B and A→→C violate 4NF.

We must separate out the attributes of these dependencies, first decomposing into AB and ACD, and then decomposing the latter into AC and AD because A→→C is still a 4NF violation for ACD. The final set of relations is AB, AC, and AD.

Exercise 3.6.3b

Since there are no functional dependencies, the only key is all four attributes, ABCD. Thus each of the nontrivial multivalued dependencies A→→B and B→→CD violate 4NF.

We must separate out the attributes of these dependencies, decomposing into AB and ACD. There are no 4NF violations for the two decomposed relations so we are done. The final set of relations is AB and ACD.

Exercise 3.6.3c

From the FD B→D, we can deduce that the only key is ABC. The MVD AB→→C and the derived MVD B→→D are both 4NF violations.

We must separate out the attributes of these dependencies, first decomposing into ABC and ABD, and then decomposing the latter into AB and BD because of the 4NF violation of B→→D. There are no more 4NF violations for the three decomposed relations so we are done. Since the attributes of relation ABC are a superset of the attributes of relation AB, we can discard relation AB. The final set of relations is ABC and BD.

Exercise 3.6.3d

From the FDs A→D and AB→E, we can deduce that the only key is ABC. The MVDs A→→B, AB→→C and the derived MVDs A→→D and AB→→E all violate 4NF.

We must separate out the attributes of these dependencies, first decomposing into AB and ACDE. However, there is still a 4NF violation in the latter from the MVD A→→D because A is not a superkey. Thus we further decompose ACDE into relations AD and ACE. There are no more 4NF violations for the three decomposed relations so we are done. The final set of relations is AB, AD and ACE.

Exercise 3.6.4

We would not expect *name* to be functionally determined by the other four attributes because there could be more than one person living at the same address who starred in the same movie. For example, a husband and wife could star in a romance movie.

We would not expect *street* to be functionally determined by the other four attributes because a person could potentially own multiple houses in the same city.

We would not expect *city* to be functionally determined by the other four attributes because a person could potentially own two houses in different cities with the same street address.

We would not expect *title* to be functionally determined by the other four attributes because a person can star in more than one movie in the same year.

We would not expect *year* to be functionally determined by the other four attributes because a person could potentially star in two movies with the same title but in different years. An example would be a person appearing in the original and then the remake of a movie.

Exercise 3.7.1a

Our starting tableau is:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_2	e_2

Applying MVD A→→BC we get:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_2	e_2
a	b_1	c_1	d_2	e_2
a	b_2	c_2	d_1	e_1

Applying FD B→D we get:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_1	e_2
a	b_1	c_1	d_1	e_2
a	b_2	c_2	d_1	e_1

Applying MVD C→→E we get:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_1	e_2
a	b_1	c_1	d_1	e_2
a	b_2	c_2	d_1	e_1
a	b_1	c_1	d_1	e_2
a	b_2	c_2	d_1	e_1
a	b_1	c_1	d_1	e_1

a	b_2	c_2	d_1	e_2

Using the chase test, it appears that the FD A→D holds in the relation.

Exercise 3.7.1b

Our starting tableau is:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c	d_2	e

Applying MVD A→→BC we get:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c	d_2	e
a	b	c	d	e_1
a	b_1	c_1	d_2	e

Applying FD B→D we get:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c	d	e
a	b	c	d	e_1
a	b_1	c_1	d	e

Applying MVD C→→E we get:

A	B	C	D	E
a	b_1	c_1	d	e_1
a	b	c	d	e
a	b	c	d	e_1
a	b_1	c_1	d	e
a	b_1	c_1	d	e
a	b	c	d	e_1
a	b	c	d	e
a	b_1	c_1	d	e_1

Since a row of all unsubscripted attributes exists, then the MVD $A \twoheadrightarrow D$ holds in the relation.

Exercise 3.7.1c

Our starting tableau is:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_2	e_2

Applying MVD $A \twoheadrightarrow BC$ we get:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_2	e_2
a	b_1	c_1	d_2	e_2
a	b_2	c_2	d_1	e_1

Applying FD $B \rightarrow D$ we get:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_1	e_2
a	b_1	c_1	d_1	e_2
a	b_2	c_2	d_1	e_1

Applying MVD $C \twoheadrightarrow E$ we get:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_1	e_2
a	b_1	c_1	d_1	e_2
a	b_2	c_2	d_1	e_1
a	b_1	c_1	d_1	e_2
a	b_2	c_2	d_1	e_1
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_1	e_2

Using the chase test, it appears that the FD $A \rightarrow E$ does not hold in the relation.

Exercise 3.7.1d

Our starting tableau is:

A	B	C	D	E
a	b_1	c_1	d_1	e
a	b	c	d	e_2

Applying MVD A→→BC we get:

A	B	C	D	E
a	b_1	c_1	d_1	e
a	b	c	d	e_2
a	b_1	c_1	d	e_2
a	b	c	d_1	e

Applying FD B→D we get:

A	B	C	D	E
a	b_1	c_1	d	e
a	b	c	d	e_2
a	b_1	c_1	d	e_2
a	b	c	d	e

Applying MVD C→→E we get:

A	B	C	D	E
a	b_1	c_1	d	e
a	b	c	d	e_2
a	b_1	c_1	d	e_2
a	b	c	d	e
a	b_1	c_1	d	e_2
a	b	c	d	e
a	b_1	c_1	d	e
a	b	c	d	e_2

Since a row of all unsubscripted attributes exists, then the MVD A→→E holds in the relation.

Exercise 3.7.2

Using the list of simplifications on pg. 120, we can narrow the list of possible FDs down to A→C, A→E, C→A, C→E, AC→E, AE→C and CE→A. Similarly, we can narrow down the list of possible MVDs down to A→→C, A→→E, C→→A and C→→E.

From these lists, we will have to perform the chase to determine whether the FDs and MVDs hold. Fortunately, we only have to perform the chase once for each unique left hand side of FDs and each unique MVD. Furthermore, for the MVDs, we only need to prove one of A→→C or A→→E and one of C→→A or C→→E because of the complementation rule.

For the FDs A→C, A→E the initial tableau looks like:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_2	e_2

The final tableau looks like:

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_1	e_2
a	b_1	c_1	d_1	e_2
a	b_2	c_2	d_1	e_1

We conclude that neither A→C nor A→E hold in relation S.

For the FDs C→A, C→E the initial tableau looks like:

A	B	C	D	E
a_1	b_1	c	d_1	e_1
a_2	b_2	c	d_2	e_2

The final tableau looks like:

A	B	C	D	E
a_1	b_1	c	d_1	e_1
a_2	b_2	c	d_2	e_2
a_1	b_1	c	d_1	e_2
a_2	b_2	c	d_2	e_1

We conclude that neither C→A nor C→E hold in relation S.

For the FD $AC \rightarrow E$ the initial tableau looks like:

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b_2	c	d_2	e_2

The final tableau looks like:

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b_2	c	d_1	e_2
a	b_1	c	d_1	e_2
a	b_2	c	d_1	e_1

We conclude that $AC \rightarrow E$ does not hold in relation S.

For the FD $AE \rightarrow C$ the initial tableau looks like:

A	B	C	D	E
a	b_1	c_1	d_1	e
a	b_2	c_2	d_2	e

The final tableau looks like:

A	B	C	D	E
a	b_1	c_1	d_1	e
a	b_2	c_2	d_1	e
a	b_1	c_1	d_1	e
a	b_2	c_2	d_1	e

We conclude that the FD $AE \rightarrow C$ does not hold in relation S.

For the FD $CE \rightarrow A$ the initial tableau looks like:

A	B	C	D	E
a_1	b_1	c	d_1	e
a_2	b_2	c	d_2	e

The final tableau looks like:

A	B	C	D	E
a_1	b_1	c	d_1	e
a_2	b_2	c	d_2	e

We conclude that the FD CE→A does not hold in relation S.

For the MVD A→→C the initial tableau looks like:

A	B	C	D	E
a	b_1	c	d_1	e_1
a	b	c_2	d	e

The final tableau looks like:

A	B	C	D	E
a	b_1	c	d	e_1
a	b	c_2	d	e
a	b_1	c	d	e
a	b	c_2	d	e_1

We conclude that the MVD A→→C holds as well as the complement A→→E.

For the MVD C→→A the initial tableau looks like:

A	B	C	D	E
a	b_1	C	d_1	e_1
a_2	b	C	d	e

The final tableau looks like:

A	B	C	D	E
a	b_1	C	d_1	e_1
a_2	b	C	d	e
a	b_1	C	d_1	e
a_2	b	C	d	e_1

We conclude that the MVD C→→A holds as well as the complement C→→E.

Therefore, the only dependencies that hold are A→→C, A→→E, C→→A and C→→E.

Exercise 3.7.3a

Let W be the set of attributes not in X, Y, or Z. Consider two tuples xyzw and xy'z'w' in the relation R in question. Because $X \rightarrow\rightarrow Y$, we can swap the y's, so xy'zw and xyz'w' are in R. Because $X \rightarrow\rightarrow Z$, we can take the pair of tuples xyzw and xyz'w' and swap the z's to get xyz'w and xyzw'. Similarly, we can take the pair xy'z'w' and xy'zw and swap Z's to get xy'zw' and xy'z'w.

In conclusion, we started with tuples xyzw and xy'z'w' and showed that xyzw' and xy'z'w must also be in the relation. That is exactly the statement of the MVD $X \rightarrow\rightarrow (Y \cup Z)$.

Exercise 3.7.3b

Let W be the set of attributes not in X, Y, or Z, V be the set of attributes that Y and Z have in common, Y_1 be the set of attributes of Y not in V and Z_1 be the set of attributes of Z not in V.
Consider the two tuples xy_1vz_1w and $xy_1'v'z_1'w'$. Because $X \rightarrow\rightarrow Y$, we can swap the y's so tuples $xy_1'v'z_1w$ and $xy_1vz_1'w'$ are in R. Because $X \rightarrow\rightarrow Z$, we can take the pair xy_1vz_1w and $xy_1vz_1'w'$ and swap the z's to get $xy_1vz_1'w$ and xy_1vz_1w'. Because $X \rightarrow\rightarrow Z$, we can take the pair $xy_1'v'z_1'w'$ and $xy_1'v'z_1w$ and swap the z's to get $xy_1'v'z_1'w$ and $xy_1'v'z_1w'$. Because $X \rightarrow\rightarrow Z$, we can take the pair $xy_1'v'z_1w$ and $xy_1vz_1'w$ and swap the z's to get $xy_1v'z_1w$ and $xy_1'vz_1'w$. Because $X \rightarrow\rightarrow Z$, we can take the pair $xy_1vz_1'w'$ and $xy_1'v'z_1w$ and swap the z's to get $xy_1v'z_1w'$ and $xy_1'vz_1'w'$.

In conclusion, we started with tuples xy_1vz_1w and $xy_1'v'z_1'w'$ and showed that $xy_1v'z_1w$ and $xy_1'vz_1'w'$ must also be in the relation. That is exactly the statement of the MVD $X \rightarrow\rightarrow (Y \cap Z)$.

Exercise 3.7.3c

Let W be the set of attributes not in X, Y, or Z, V be the set of attributes that Y and Z have in common, Y_1 be the set of attributes of Y not in V and Z_1 be the set of attributes of Z not in V.
Consider the two tuples xy_1vz_1w and $xy_1'v'z_1'w'$. Because $X \rightarrow\rightarrow Y$, we can swap the y's so tuples $xy_1'v'z_1w$ and $xy_1vz_1'w'$ are in R. Because $X \rightarrow\rightarrow Z$, we can take the pair $xy_1'v'z_1w$ and xy_1vz_1w swap the z's to get $xy_1'vz_1w$ and $xy_1v'z_1w$. Because $X \rightarrow\rightarrow Z$, we can take the pair $xy_1vz_1'w'$ and $xy_1'v'z_1'w'$ and swap the z's to get $xy_1v'z_1'w'$ and $xy_1'vz_1'w'$.

In conclusion, we started with tuples xy_1vz_1w and $xy_1'v'z_1'w'$ and showed that $xy_1'vz_1w$ and $xy_1v'z_1'w'$ must also be in the relation. That is exactly the statement of the MVD $X \rightarrow\rightarrow (Y - Z)$.

Exercise 3.7.3d

Let W be the set of attributes not in X or Y, V be the set of attributes that X and Y have in common, Y_1 be the set of attributes of Y not in V and X_1 be the set of attributes of X not in V.

Consider the two tuples x_1vy_1w and $x_1vy_1'w'$. Because $X \to\to Y$, we can swap the y's so tuples $x_1vy_1'w$ and x_1vy_1w' are in R.

In conclusion, we started with tuples x_1vy_1w and $x_1vy_1'w'$ and showed that $x_1vy_1'w$ and x_1vy_1w' must also be in the relation. That is exactly the statement of the MVD $X \to\to (Y - X)$.

Exercise 3.7.4a

If we want to perform the chase test for $A \to\to B$, then an example of an initial tableau is:

A	B	C	D
a	b	c_1	d_1
a	b_2	c	d

If we apply the MVD $A \to\to BC$, we get:

A	B	C	D
a	b	c_1	d_1
a	b_2	c	d
a	b_2	c	d_1
a	b	c_1	d

The above tableau does not satisfy $A \to\to B$. Thus we would not expect $A \to\to B$ to follow from $A \to\to BC$.

Exercise 3.7.4b

If we want to perform the chase test for $A \to B$, then an example of an initial tableau is:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

If we apply the MVD $A \to\to B$, we get:

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2
a	b_2	c_1	d_1
a	b_1	c_2	d_2

The above tableau does not satisfy A→B. Thus we would not expect A→B to follow from A→→B.

Exercise 3.7.4c

If we want to perform the chase test for A→→C, then an example of an initial tableau is:

A	B	C	D
a	b_1	c	d_1
a	b	c_2	d

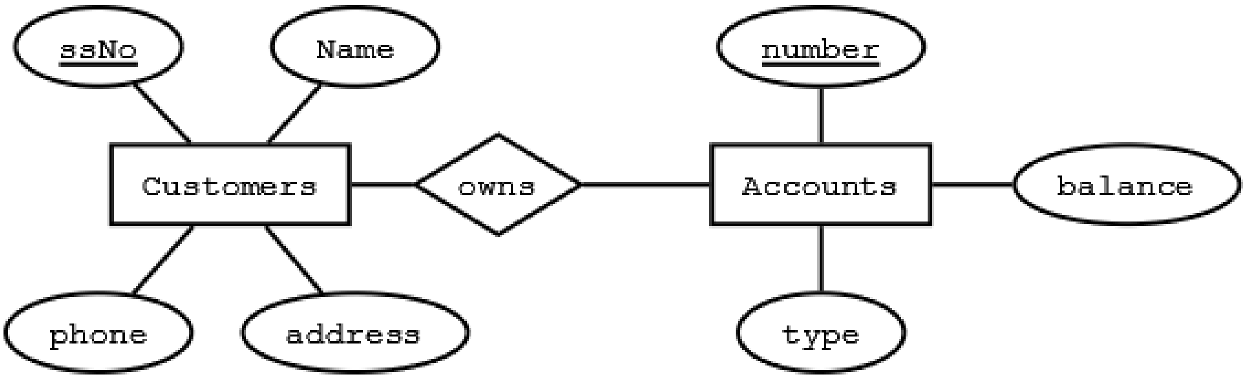
If we apply the MVD AB→→C, we get:

A	B	C	D
a	b_1	c	d_1
a	b	c_2	d

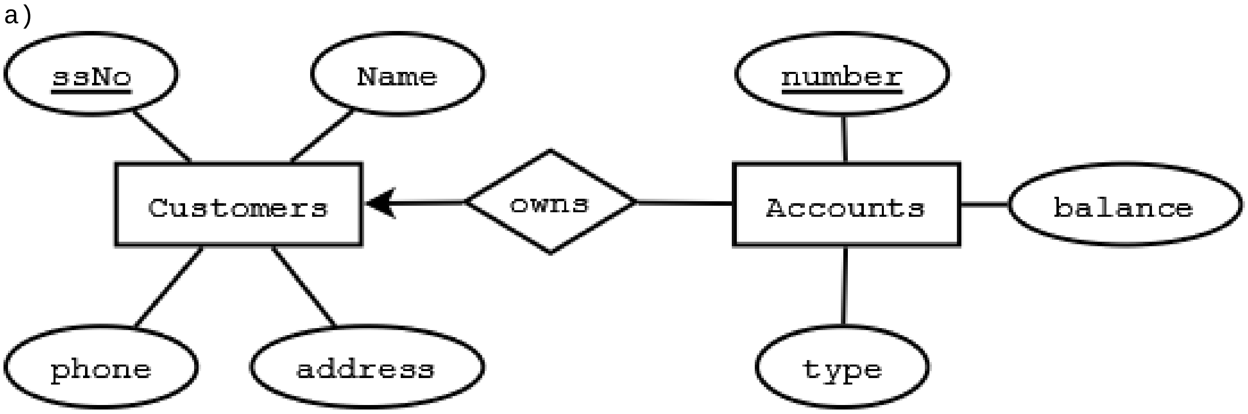
The above tableau does not satisfy A→→C. We cannot apply the MVD AB→→C because none of the tuples match in both attributes A and B. Thus we would not expect A→→C to follow from AB→→C.

Solutions
Chapter 4

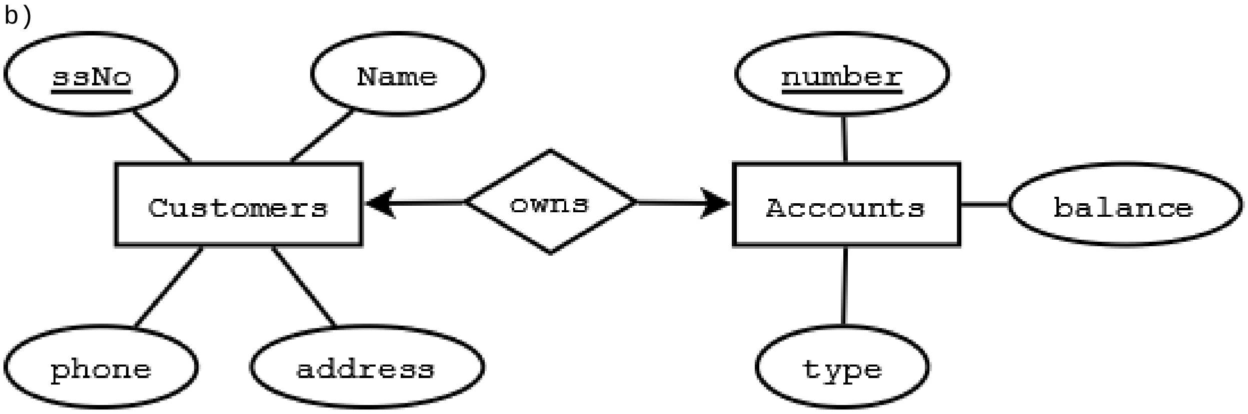
4.1.1



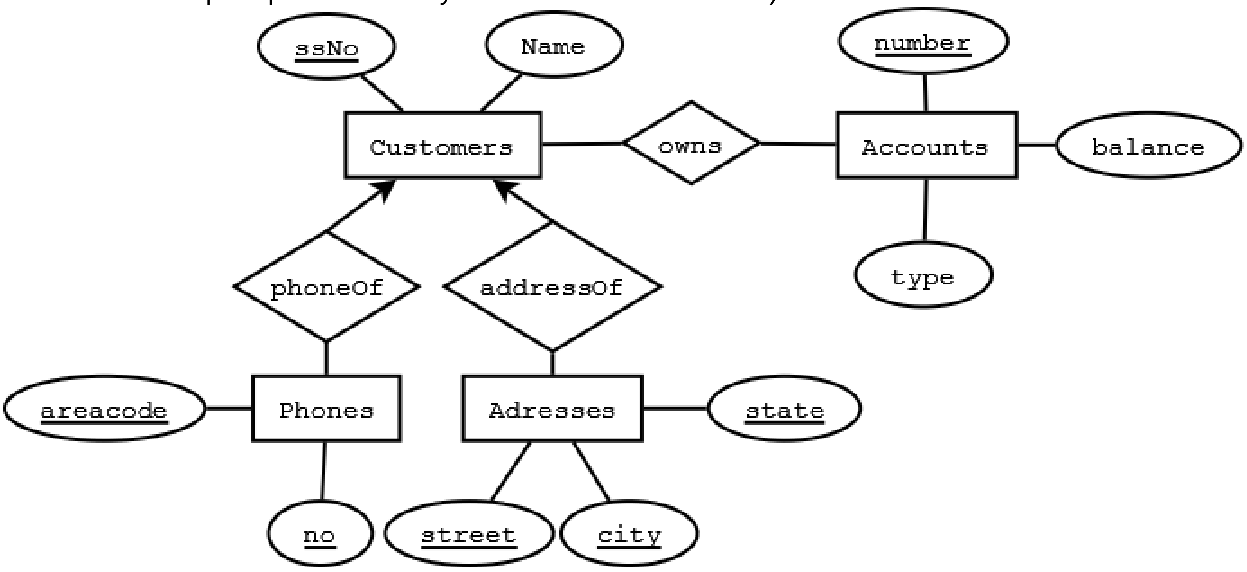
4.1.2
a)



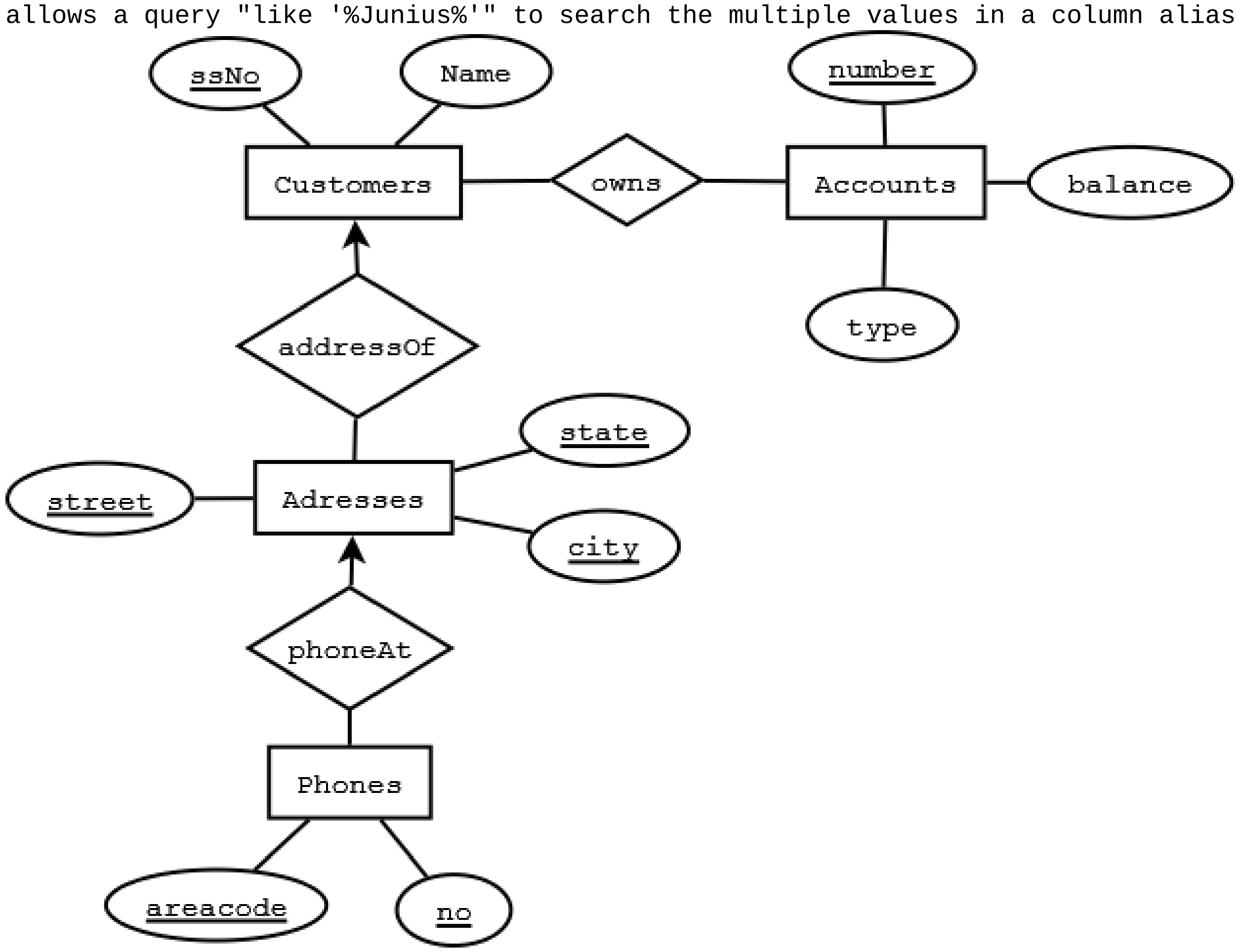
b)



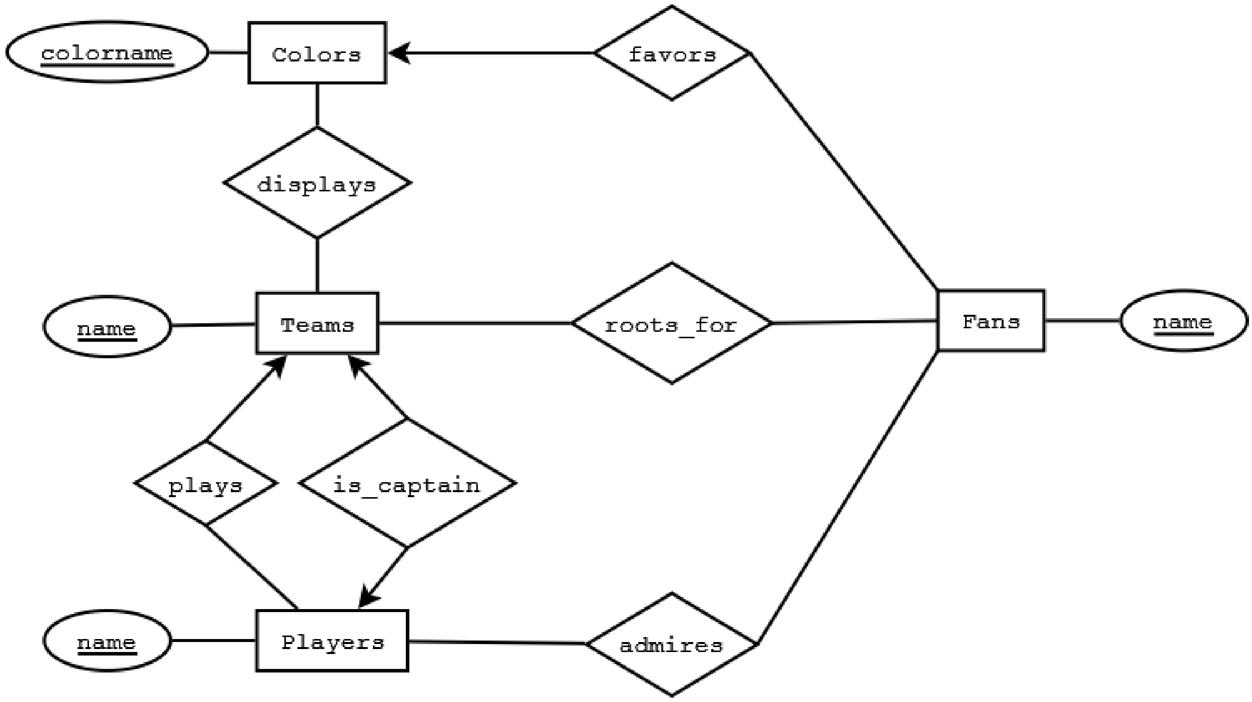
c)
In c we assume that a phone and address can only belong to a single customer (1-m relationship represented by arrow into customer).



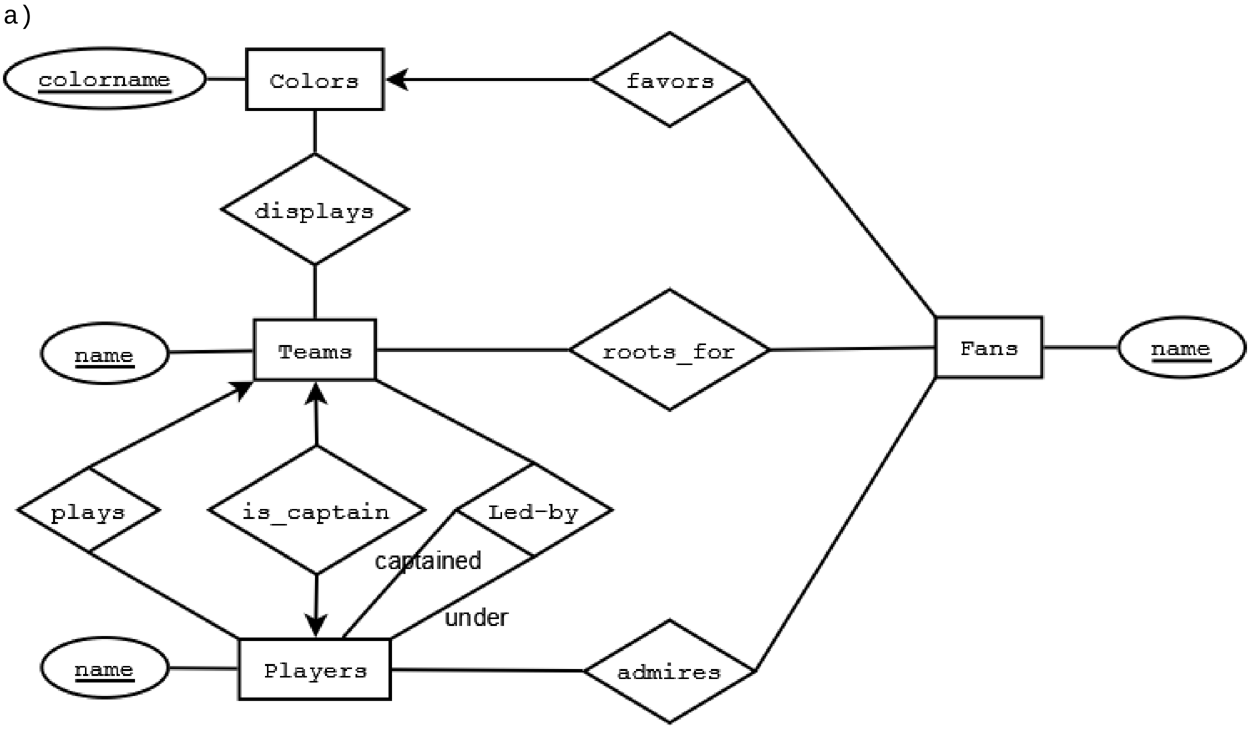
d)
In d we assume that an address can only belong to one customer and a phone can exist at only one address.
If the multiplicity of above relationships were m-to-n, the entity set becomes weak and the key ssNo of customers will be needed as part of the composite key of the entity set.
In c&d, we convert 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.



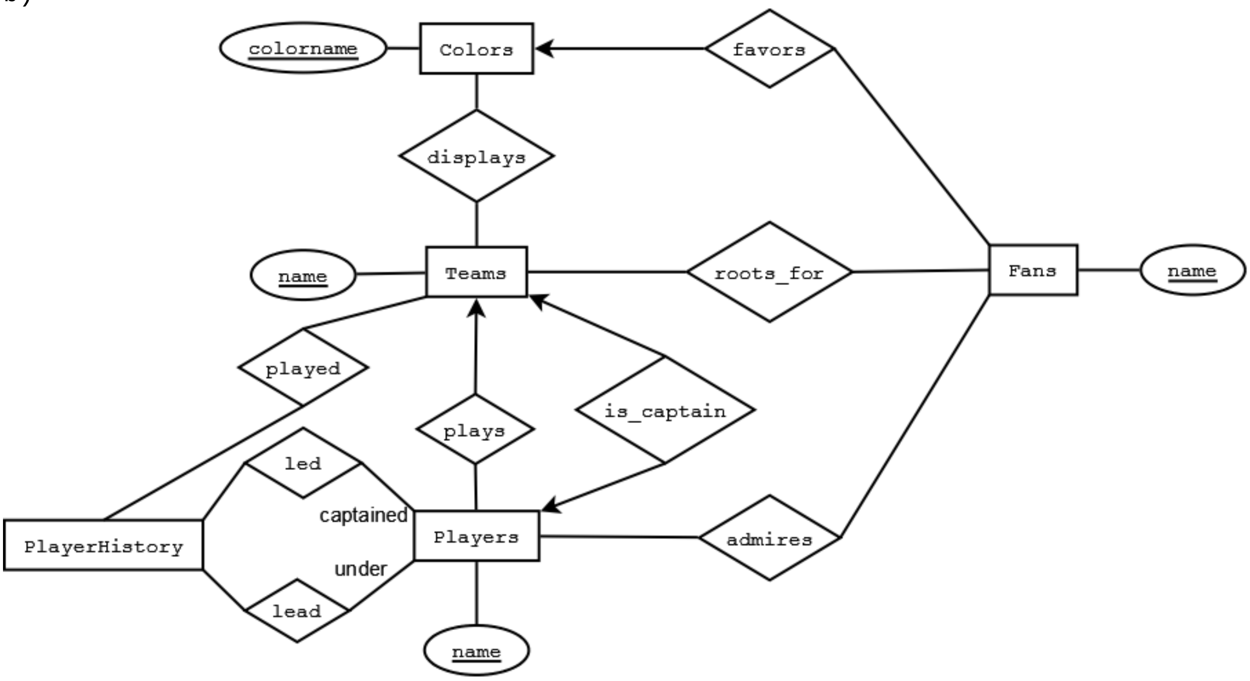
4.1.3


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

4.1.4
a)

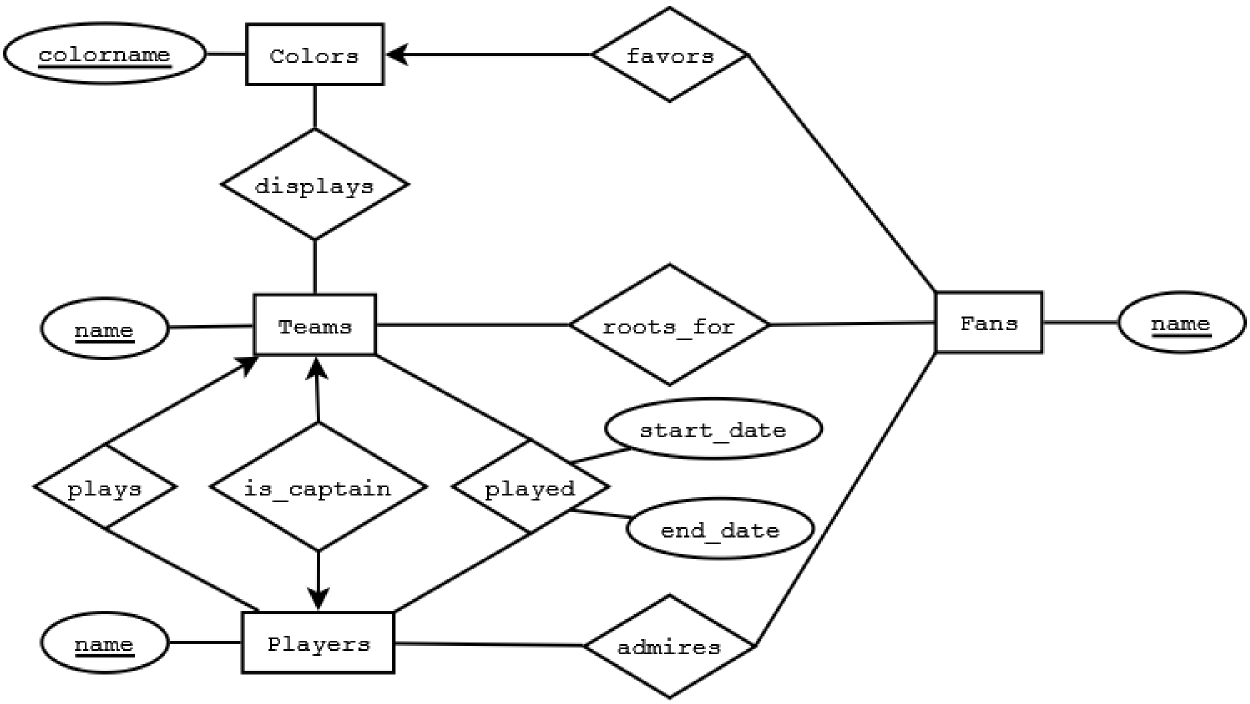

colorname
Colors
favors
displays
name
Teams
roots_for
Fans
name
plays
is_captain
Led-by
captained
under
name
Players
admires

b)

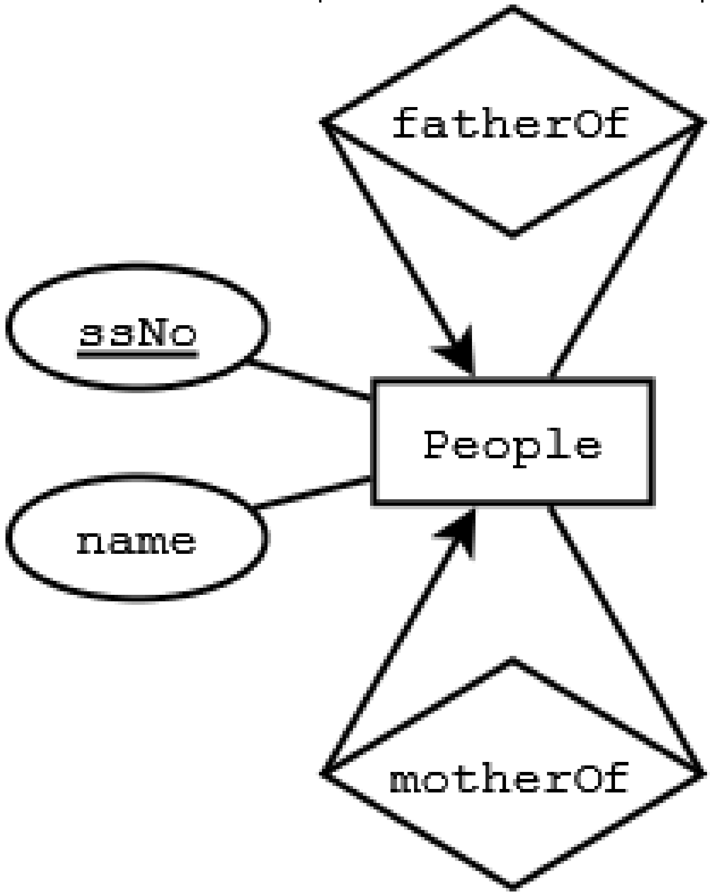


c)
The relationship "played" between Teams and Players is similar to relationship "plays" between Teams and Players.

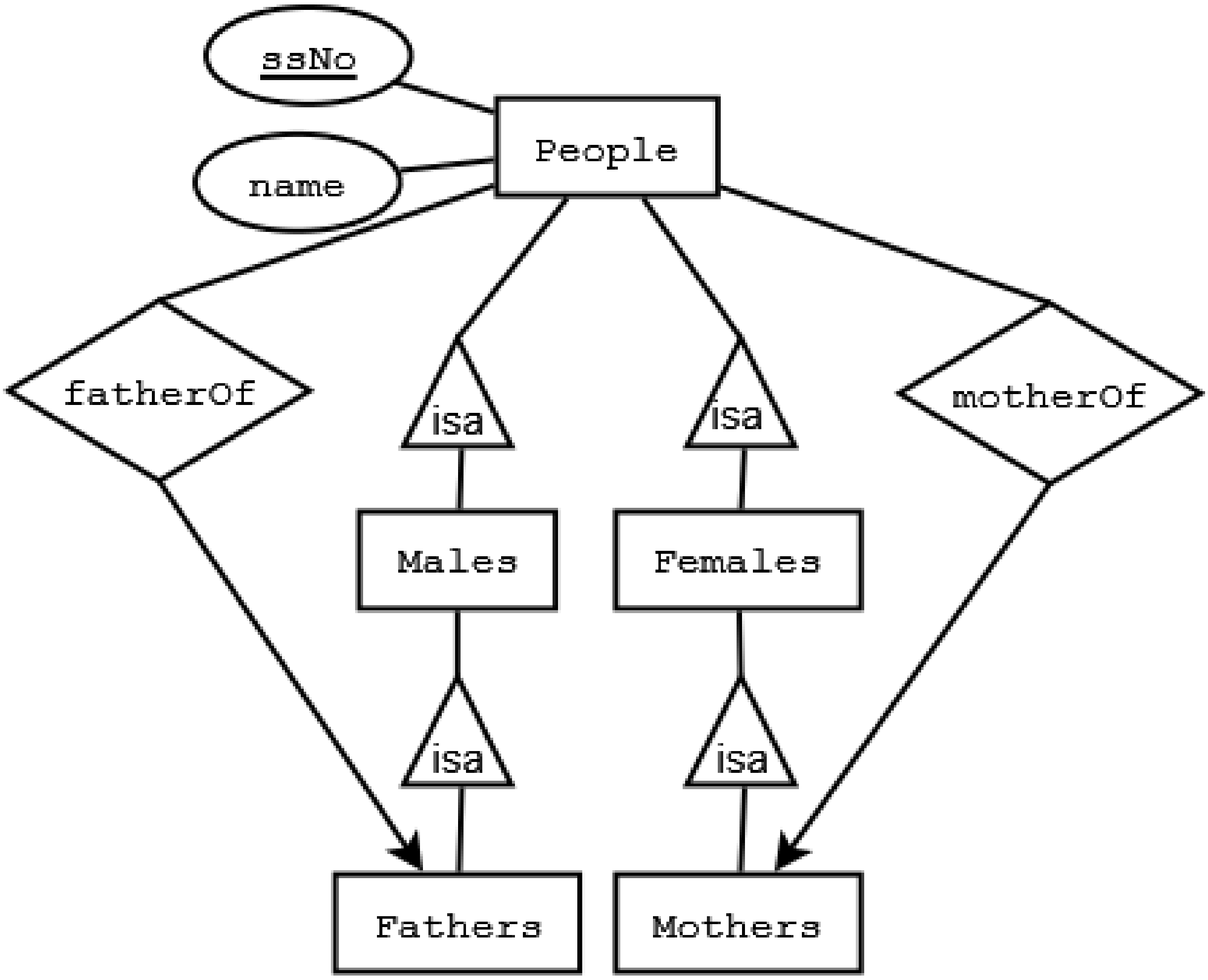
4.1.5



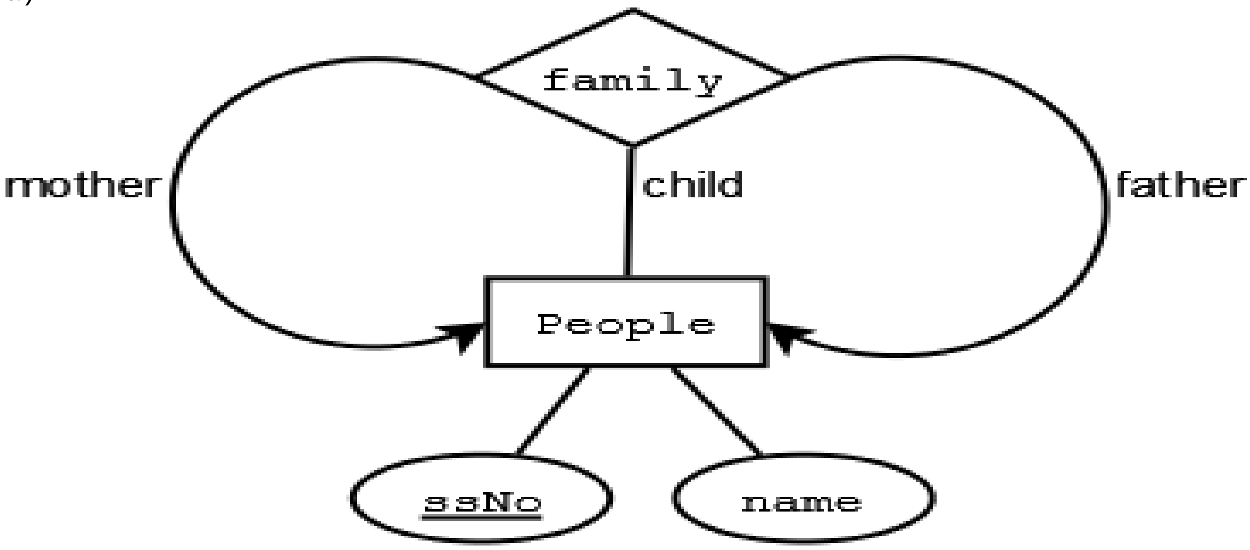
4.1.6 The information about children can be ascertained from motherOf and fatherOf relationships. Attribute ssNo is required since names are not unique.



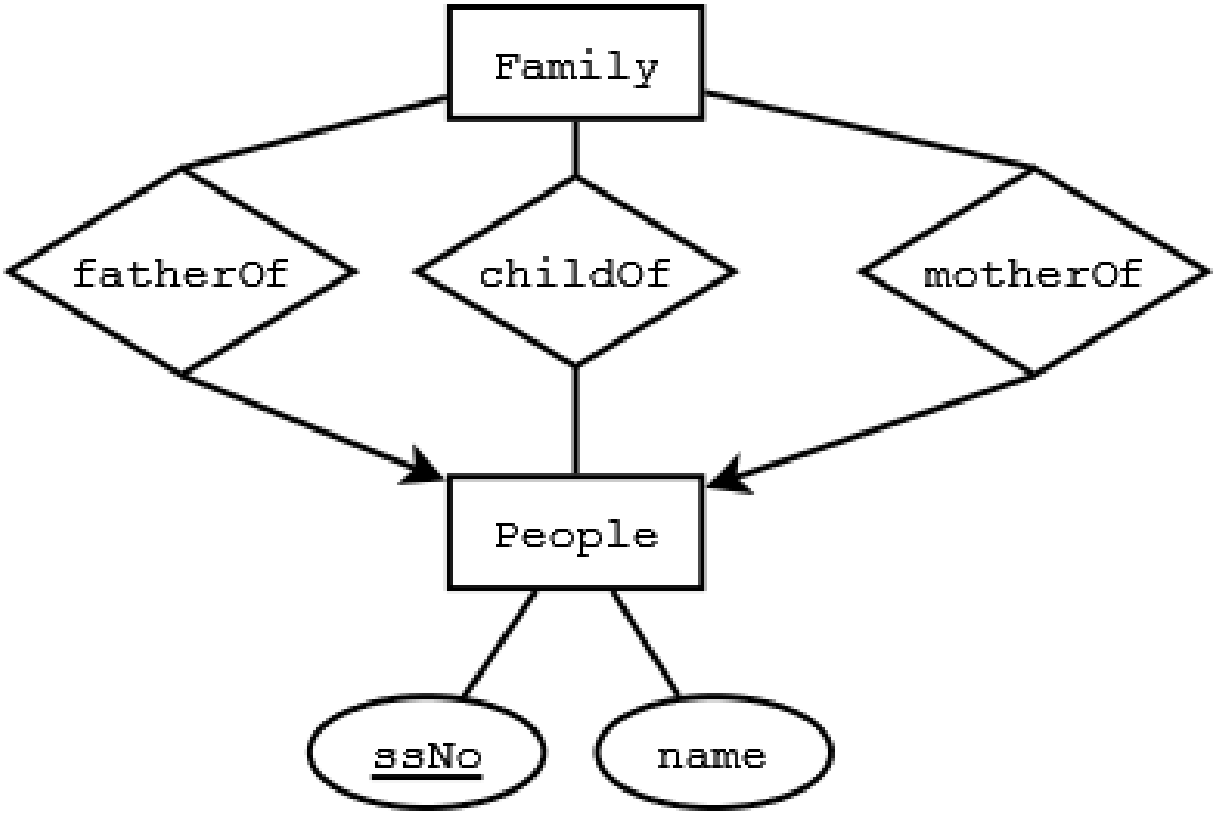
4.1.7


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

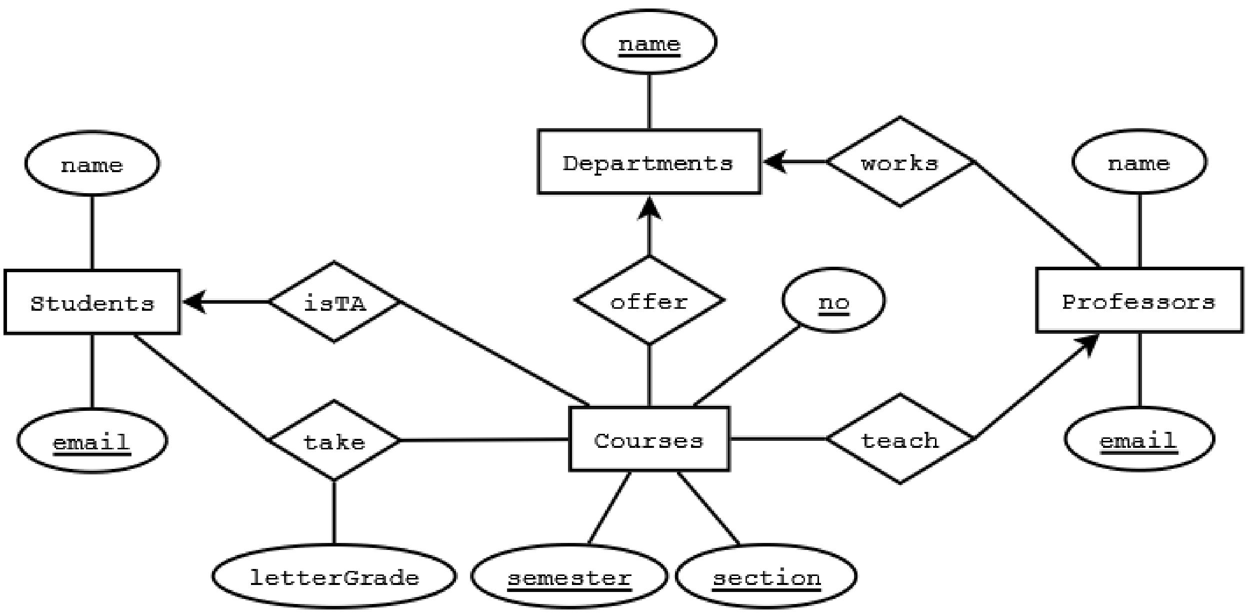
4.1.8
a)



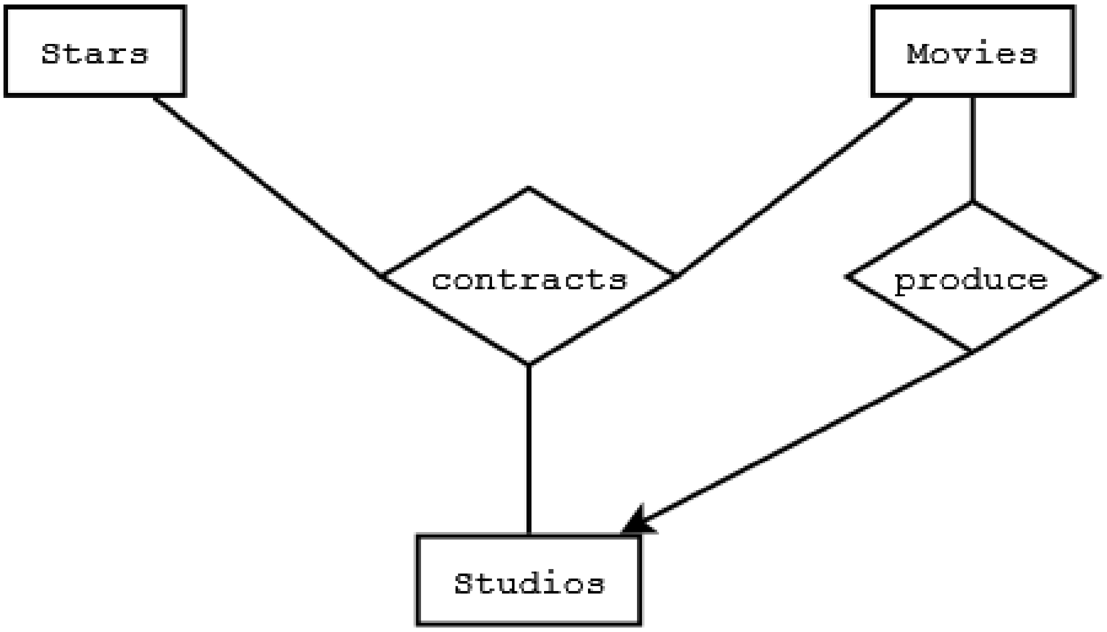
(b)

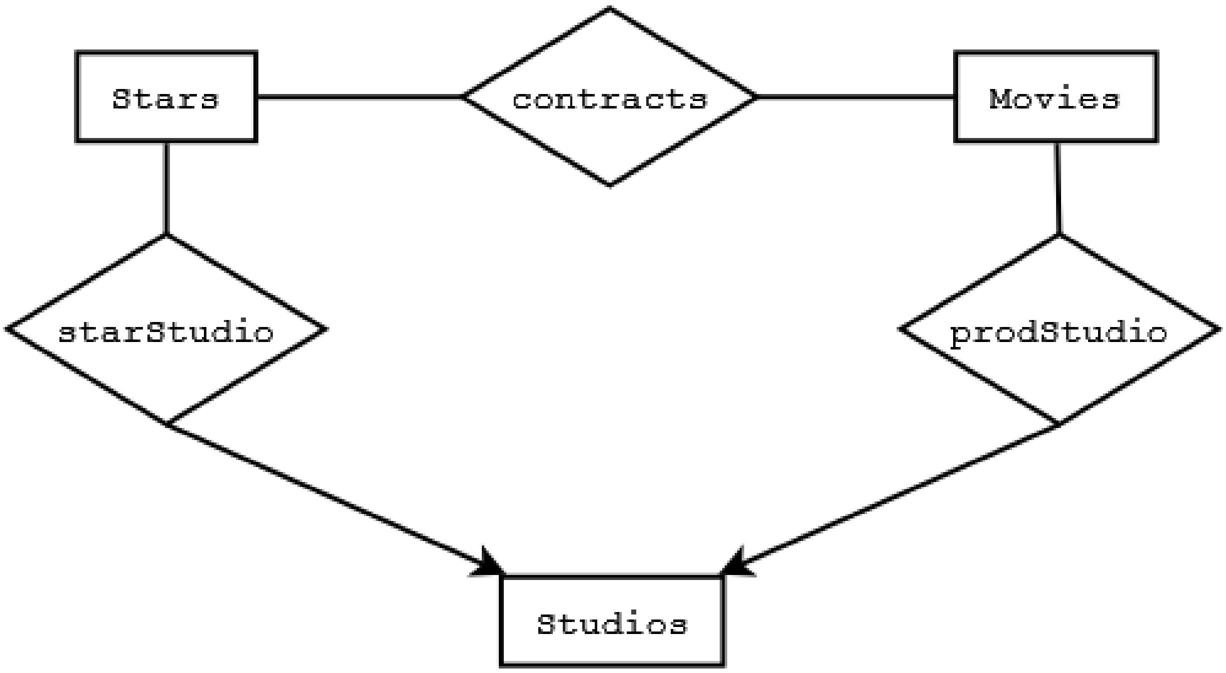


4.1.9
Assumptions
A Professor only works in at most one department.
A course has at most one TA.
A course is only taught by one professor and offered by one department.
Students and professors have been assigned unique email ids.
A course is uniquely identified by the course no, section no, and semester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

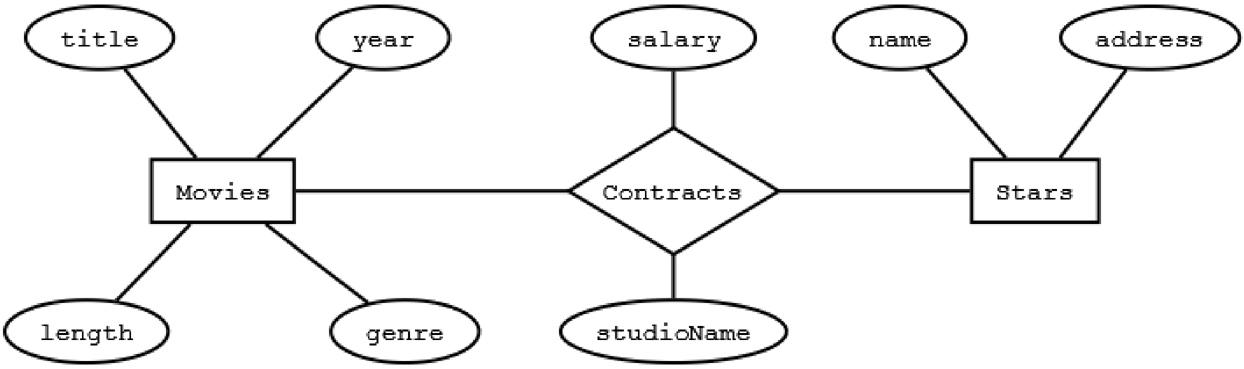




4.2.1
Redundancy: The owner address is repeated in AccSets and Addresses entity sets.
Simplicity: AccSets does not serve any useful purpose and the design can be more simply represented by creating many-to-many relationship between Customers and Accounts.

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.

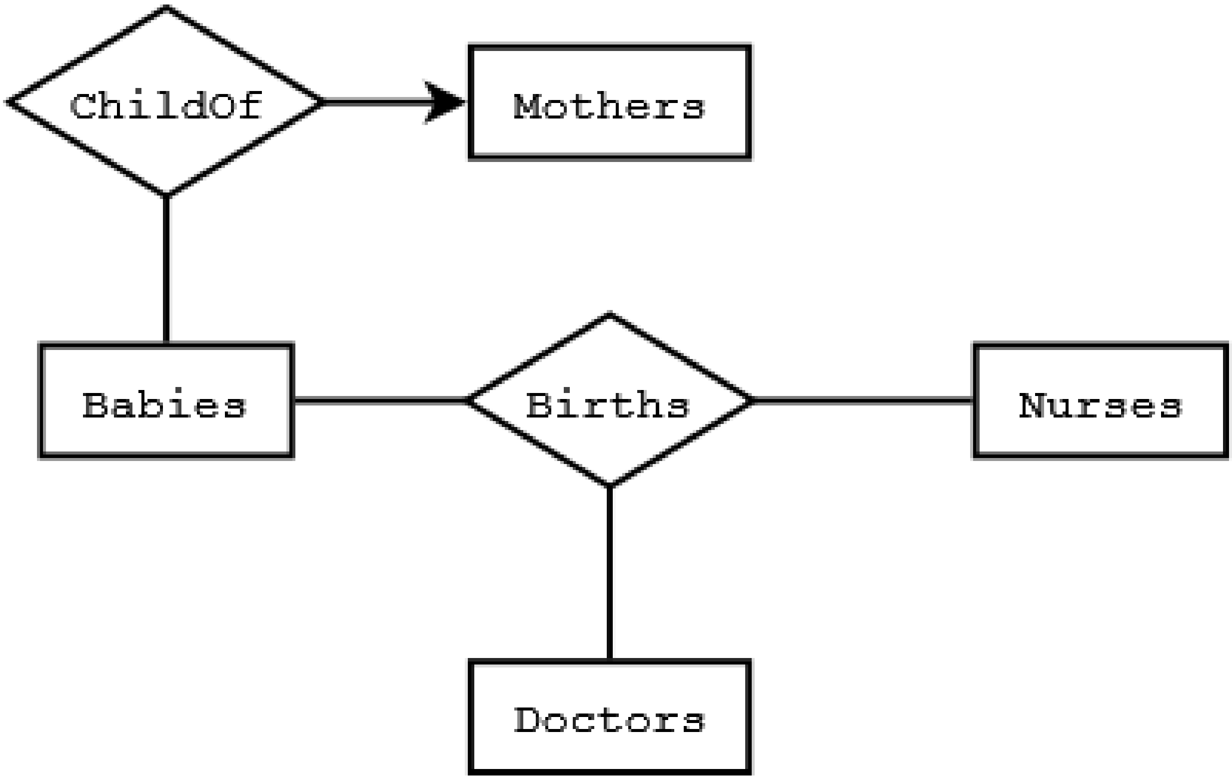
4.2.3


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%'

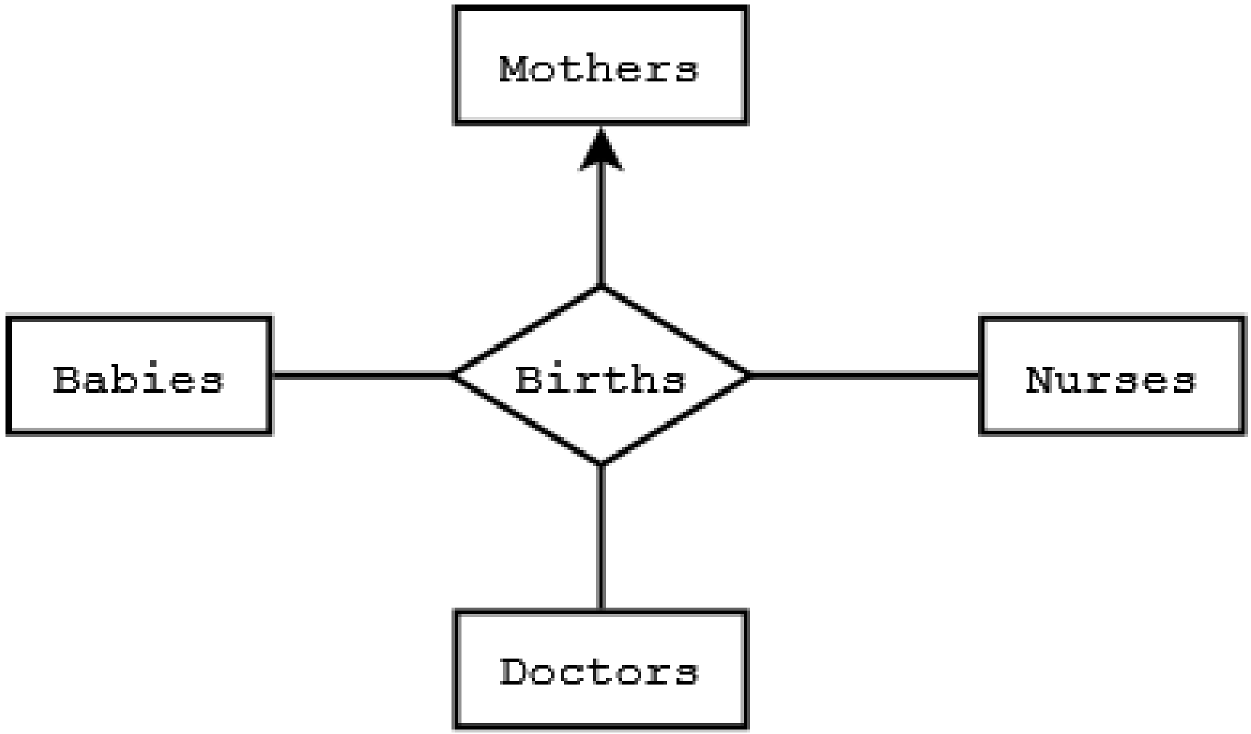
4.2.5

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.



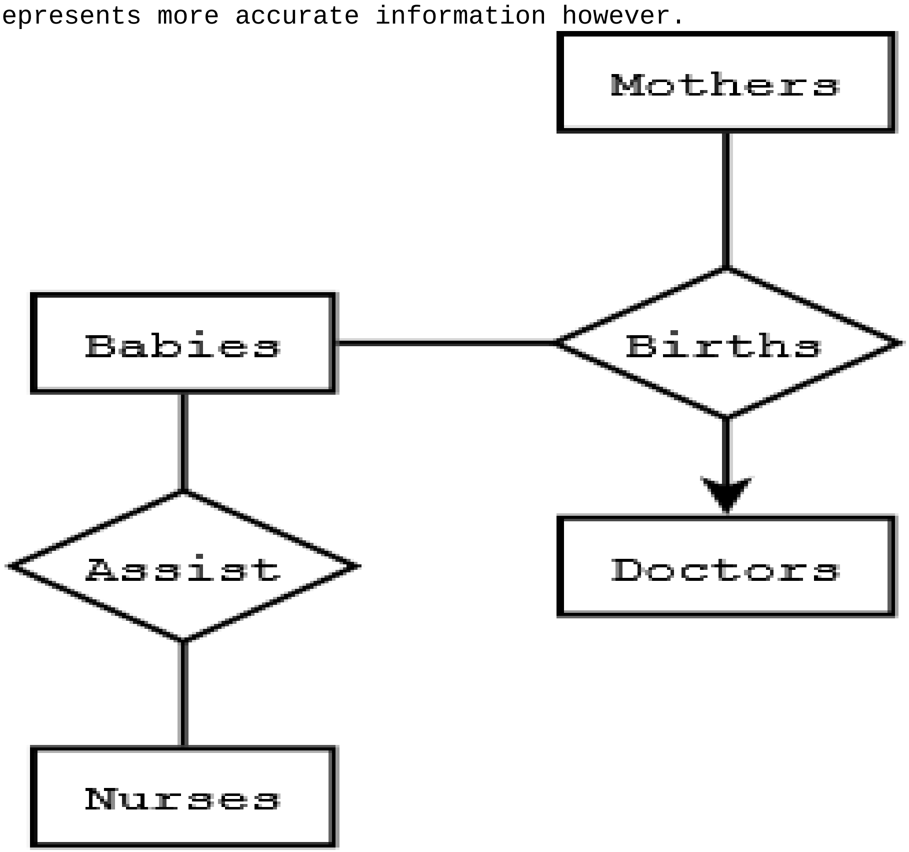
b)



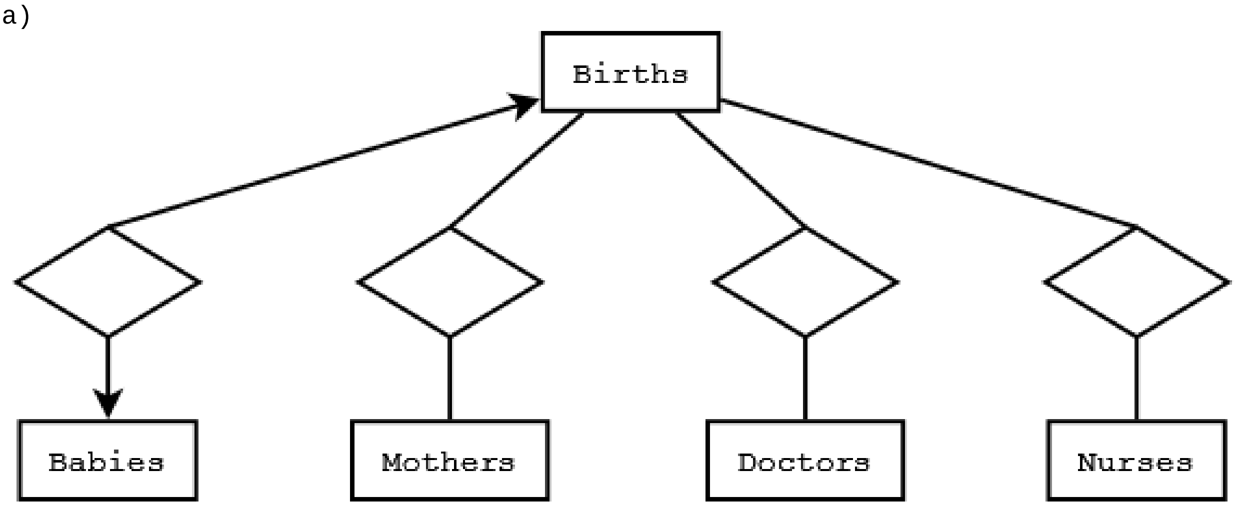
c)

Again from Augmentation rule of Functional Dependency,

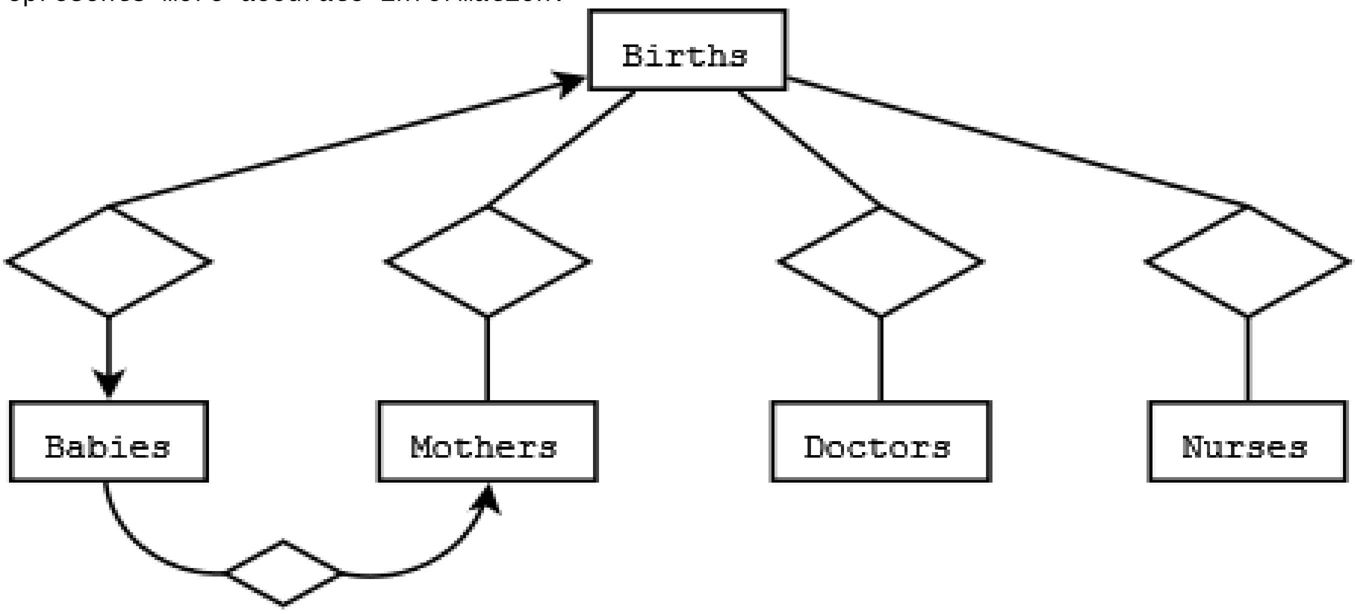
given
BM -> D
then
BMN -> D
Thus we can just add an arrow entering Doctors to fig 4.15. Below figure represents more accurate information however.



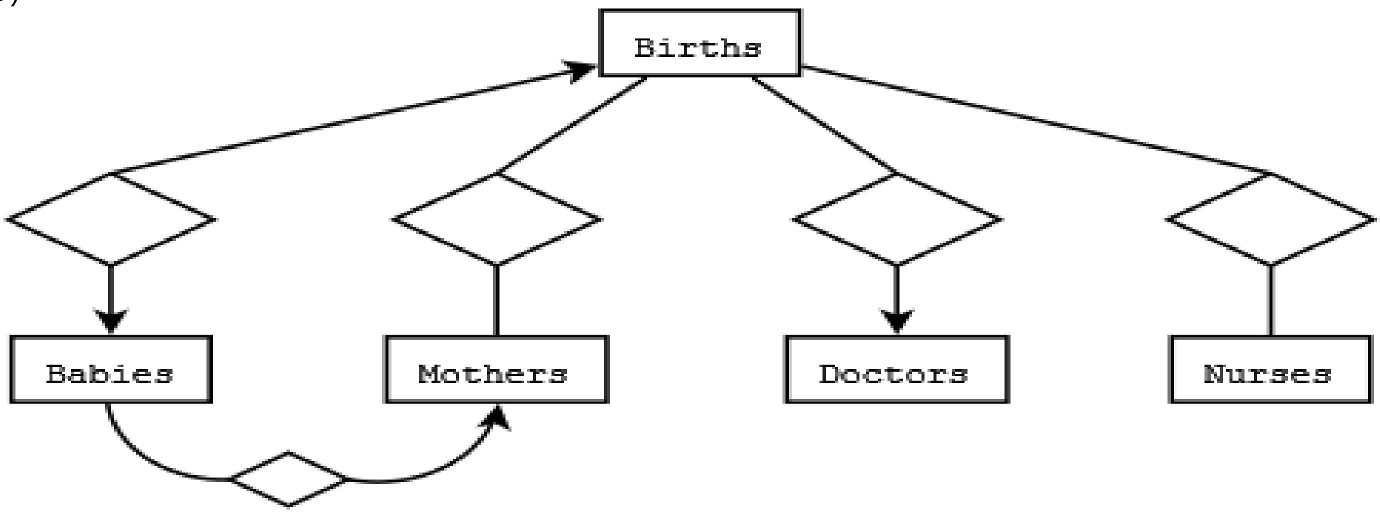
4.2.6
a)



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.

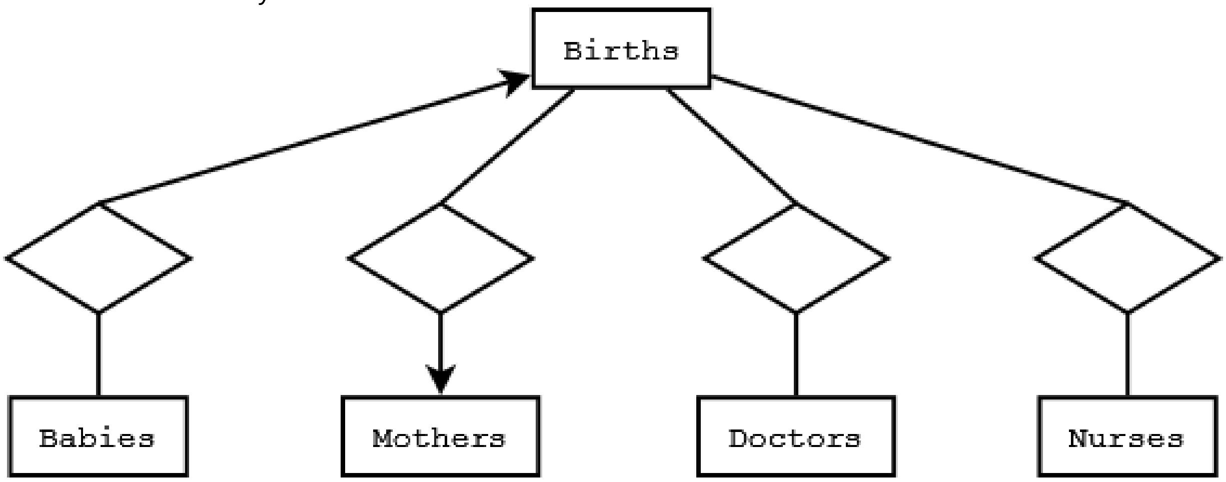


c)

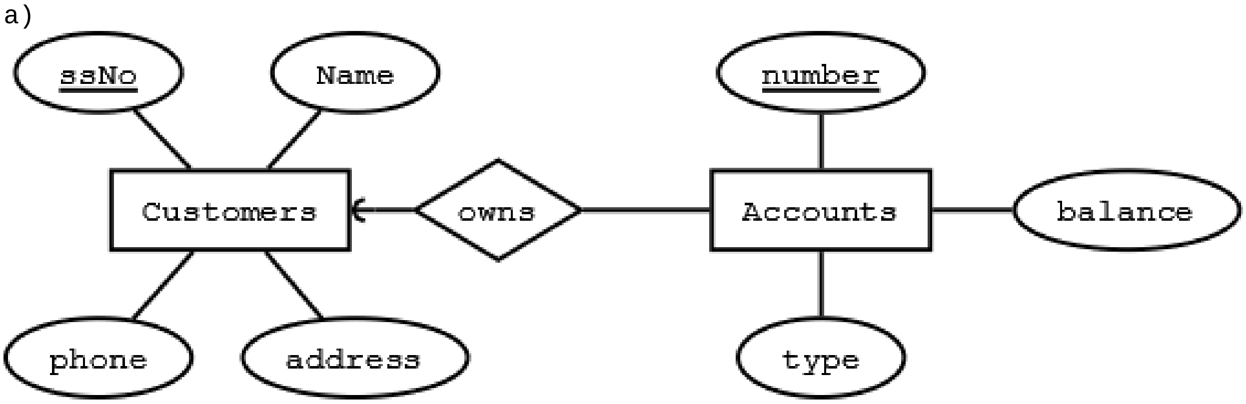


Design flaws in abc above 1. As suggested above, using Transitivity and Augmentation rules of Functional Dependency, much simpler design is possible.

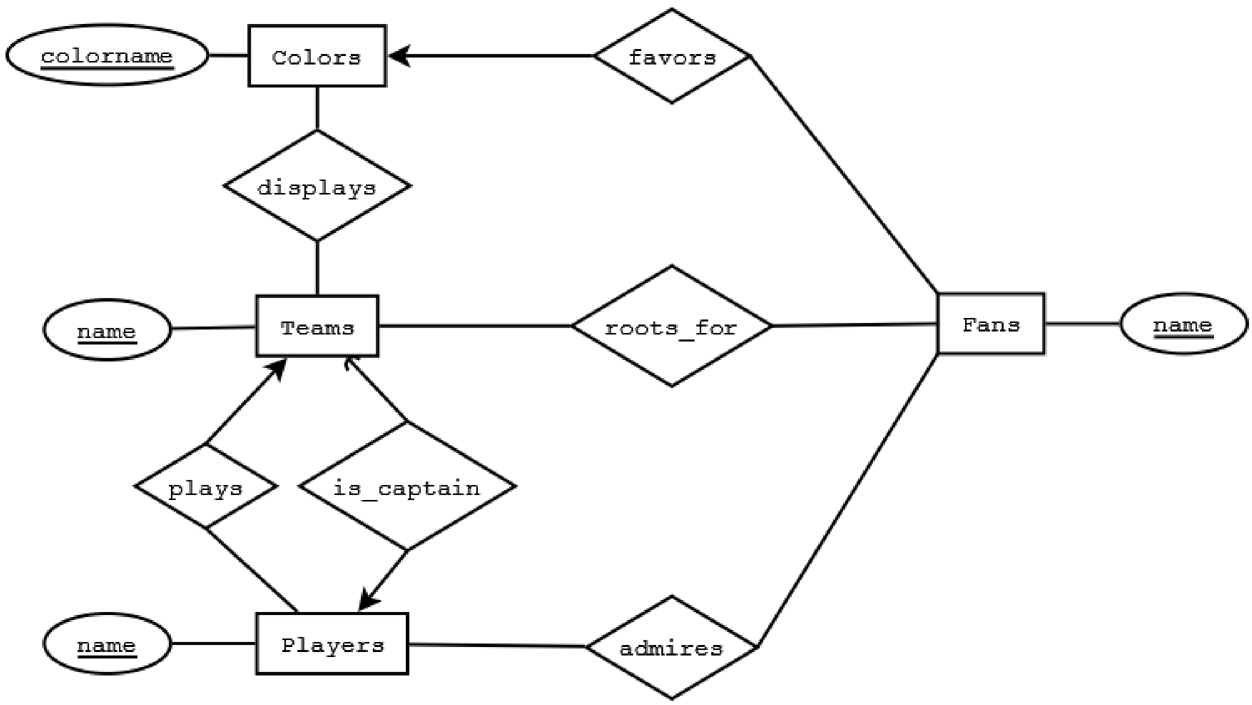
4.2.7
In below figure there exists a many-to-one relationship between Babies and Births and another many-to-one relationship between Births and Mothers. From transitivity of relationships, there is a many-to-one relationship between Babies and Mothers. Hence a baby has a unique mother while a birth can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



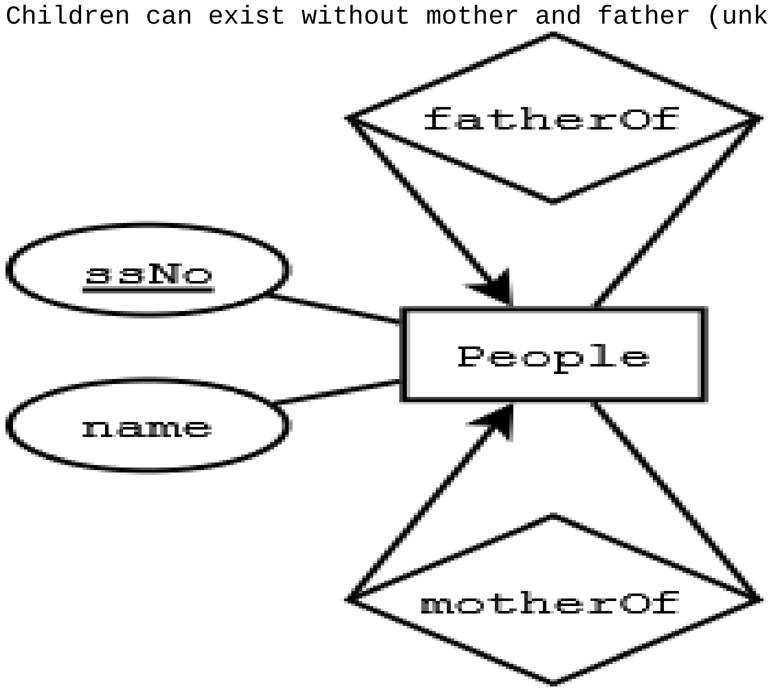
4.3.1
a)



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

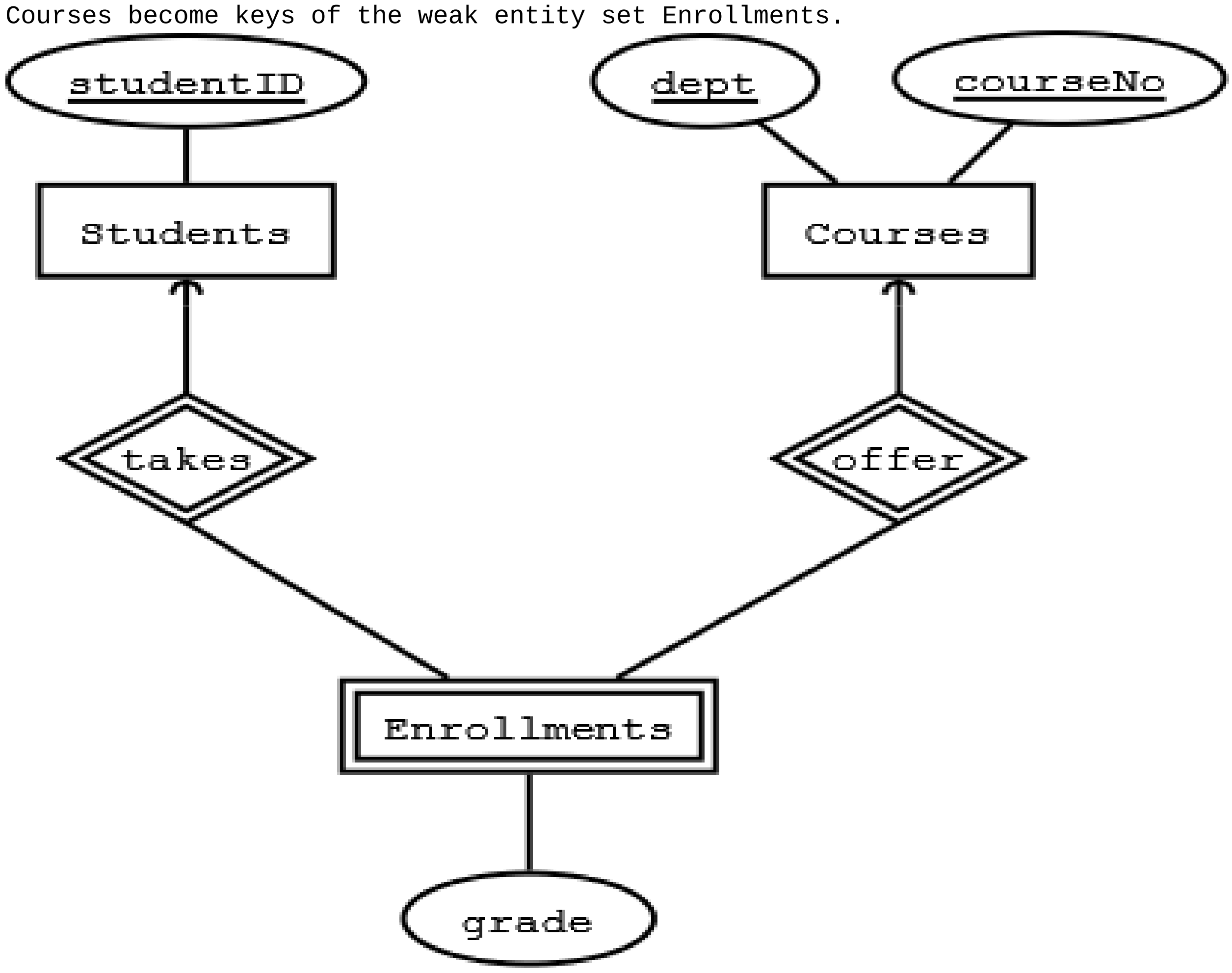
c)
The key of E2

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

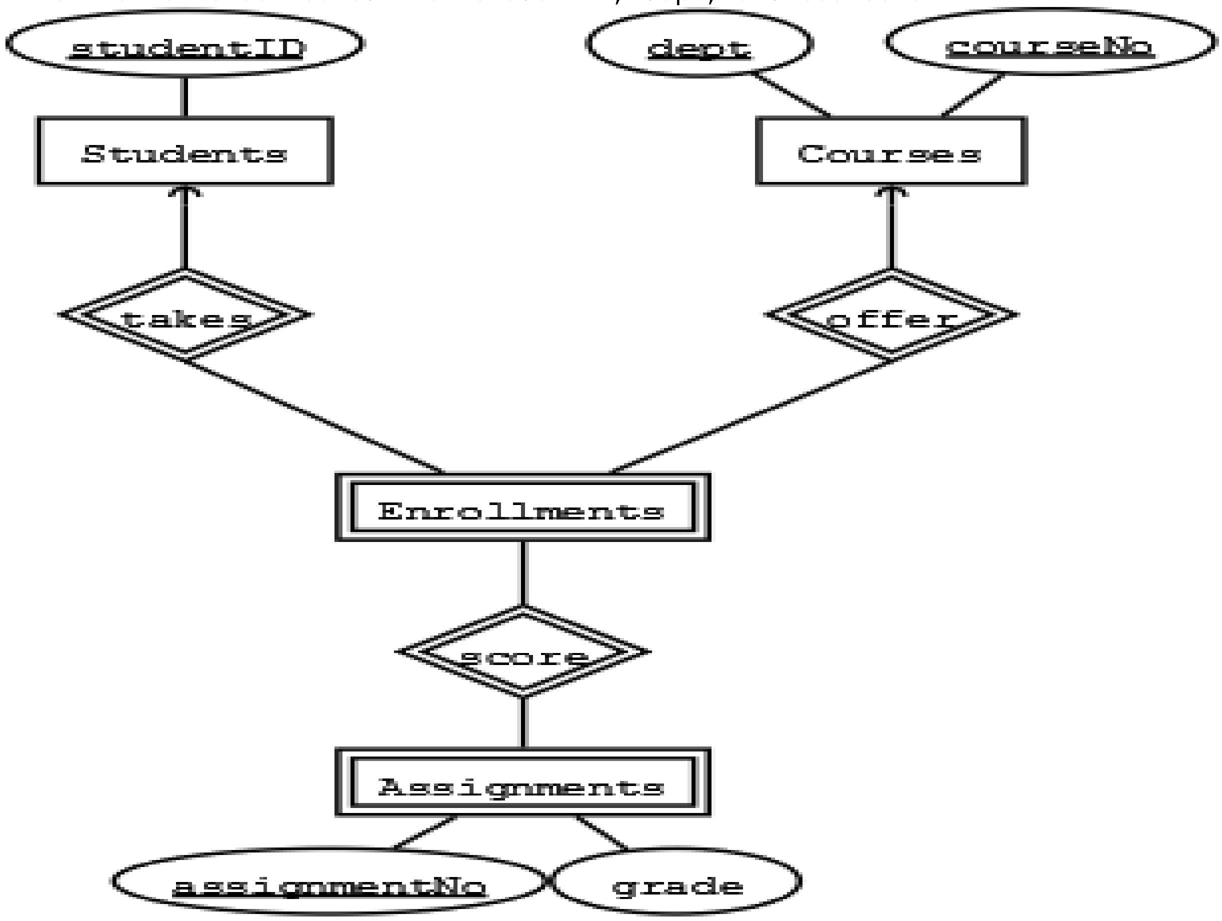
Otherwise: Combination of all Ki's where there does not exist an arrow going from R to Ei.

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.

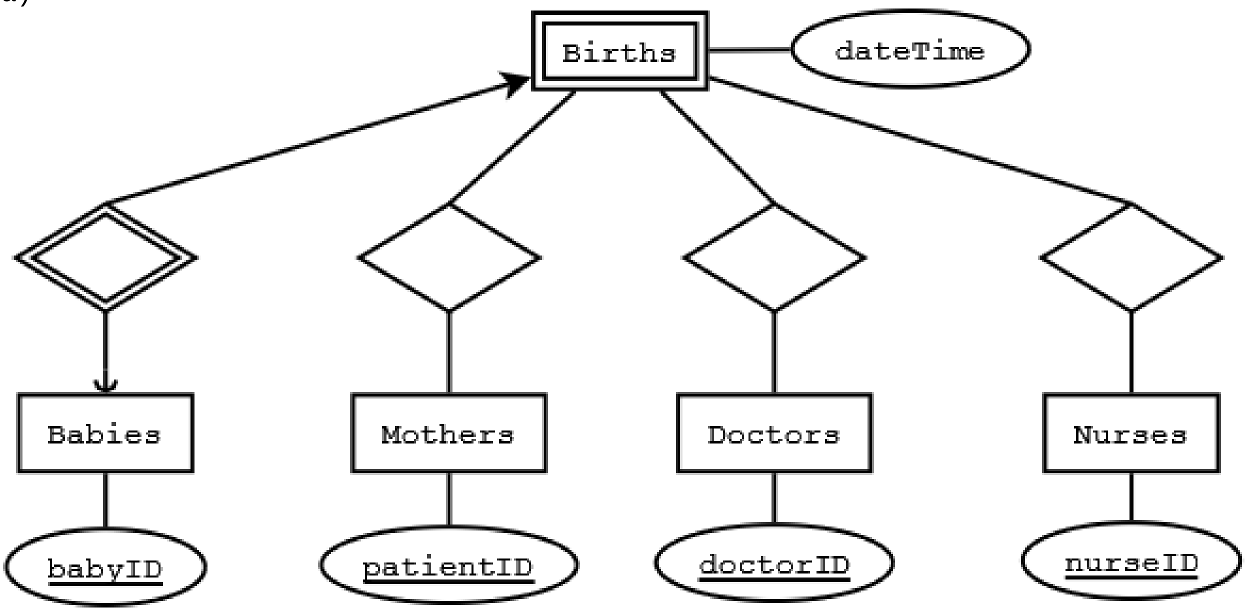


4.4.2

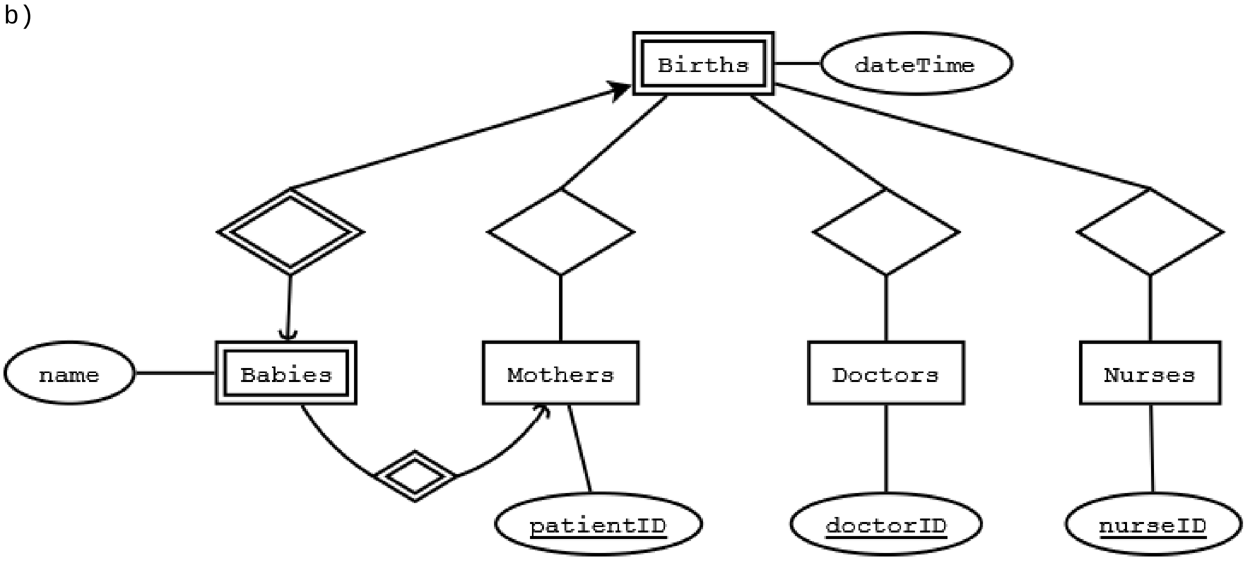
It is possible to make assignment number a weak key of Enrollments but this is not good design (redundancy since multiple assignments correspond to a course). A new entity set Assignment is created and it is also a weak entity set. Hence the key attributes of Assignment will come from the strong entity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



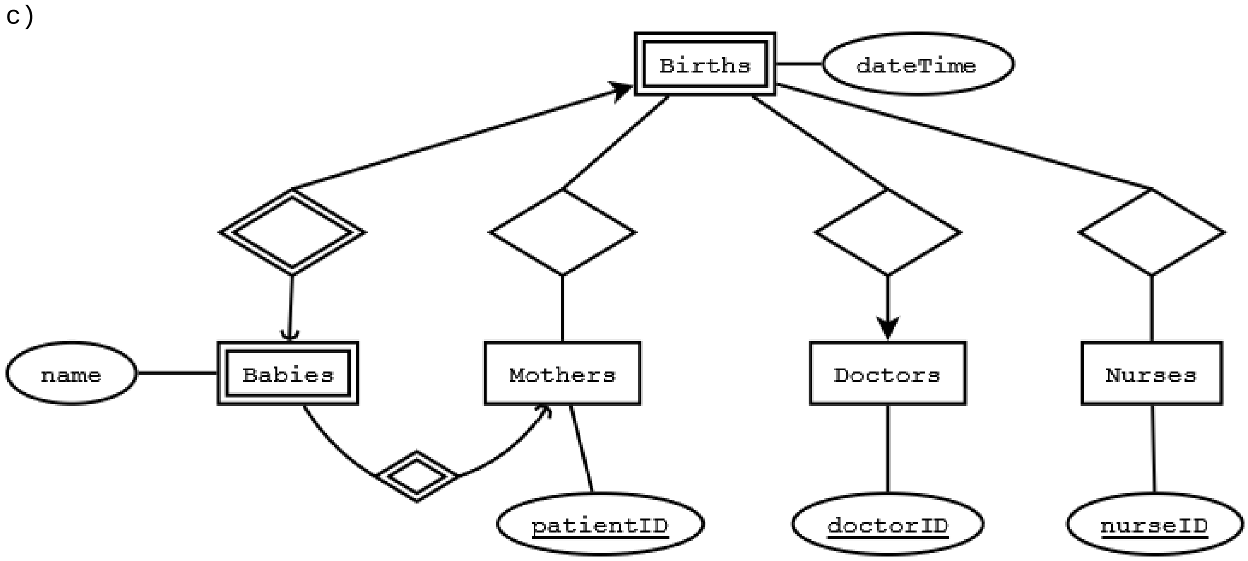
4.4.3
a)


Births
dateTime
Babies
Mothers
Doctors
Nurses
babyID
patientID
doctorID
nurseID

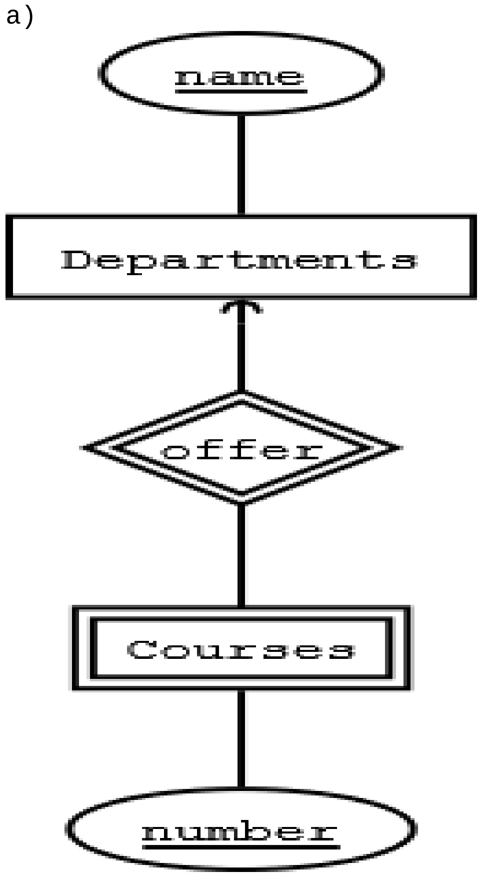
b)


Births
dateTime
name
Babies
Mothers
Doctors
Nurses
patientID
doctorID
nurseID

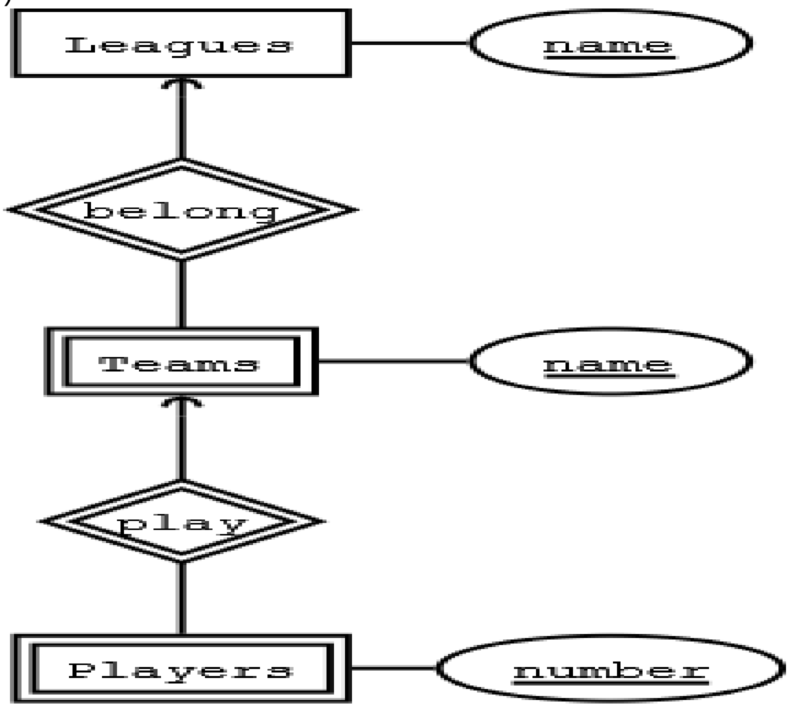
c)


Births
dateTime
name
Babies
Mothers
Doctors
Nurses
patientID
doctorID
nurseID

4.4.4
a)


name
Departments
offer
Courses
number

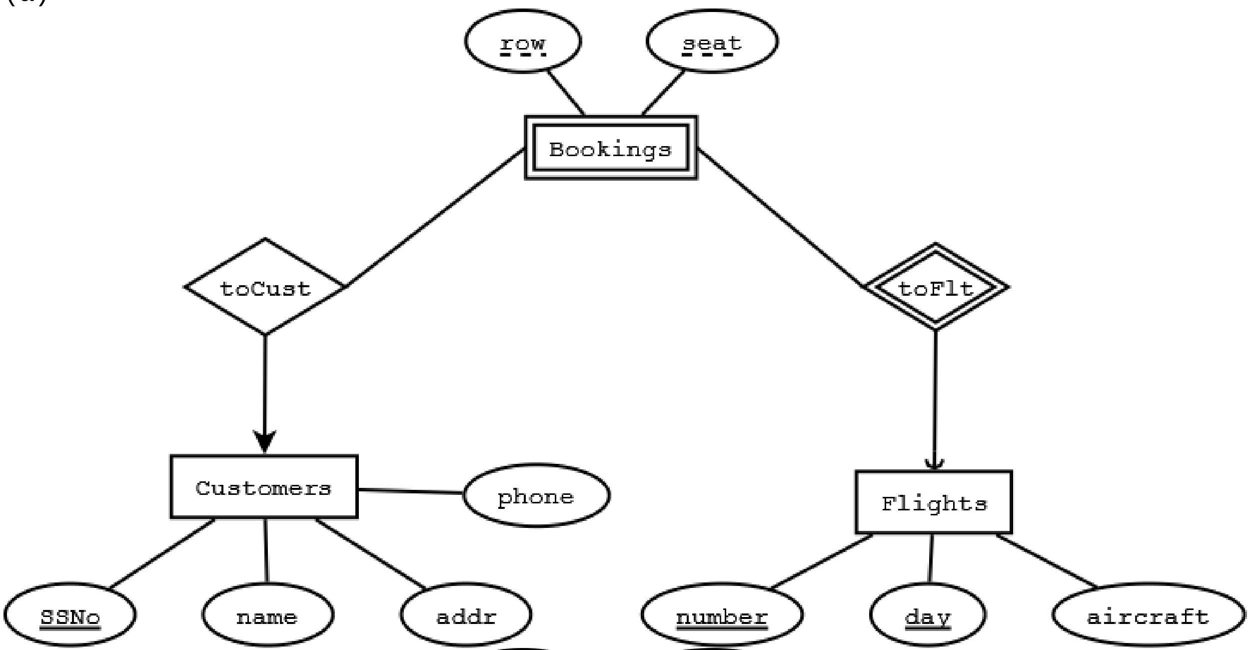
b)

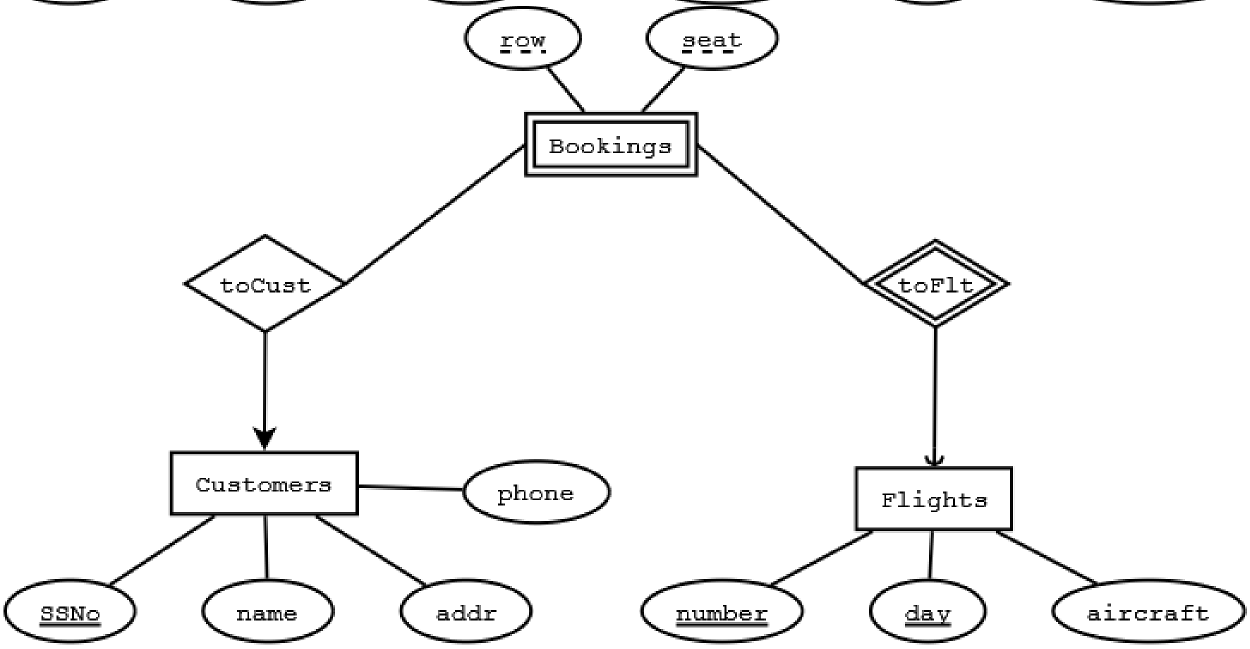


4.5.1
Customers(SSNo,name,addr,phone)
Flights(number,day,aircraft)
Bookings(custSSNo,flightNo,flightDay,row,seat)

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)

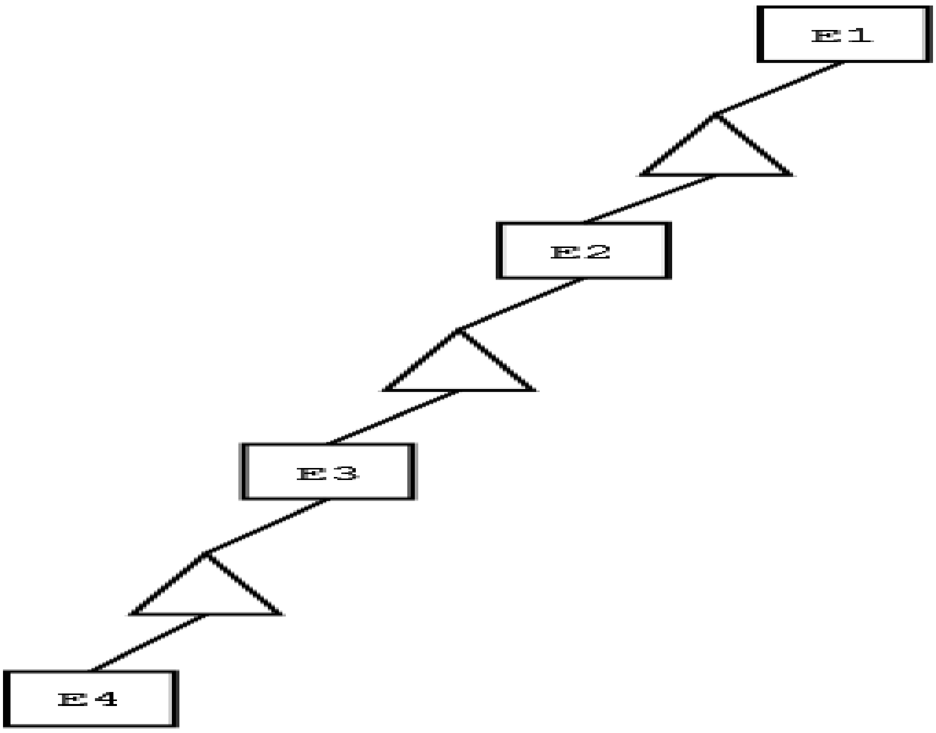
ChildOf(personName,childName)
FatherOf(childName,fatherName)
MotherOf(childName,motherName)

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.

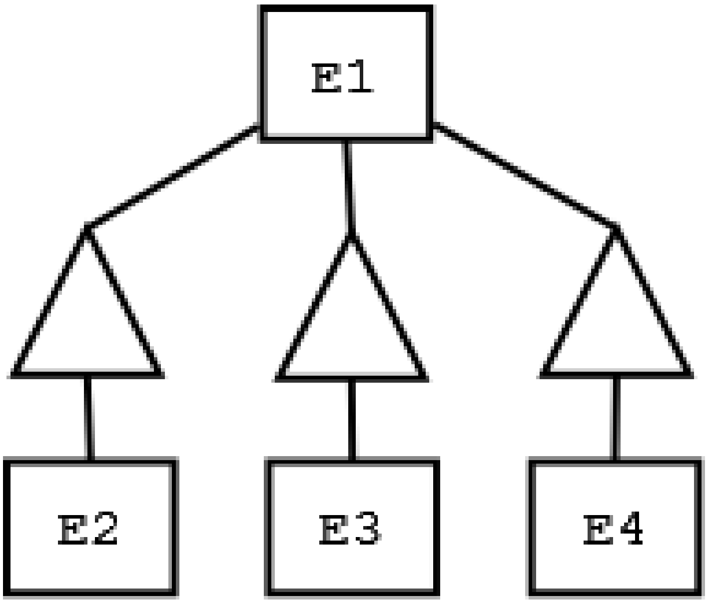
(c)
People(name,fatherName,motherName)
ChildOf(personName,childName)

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.

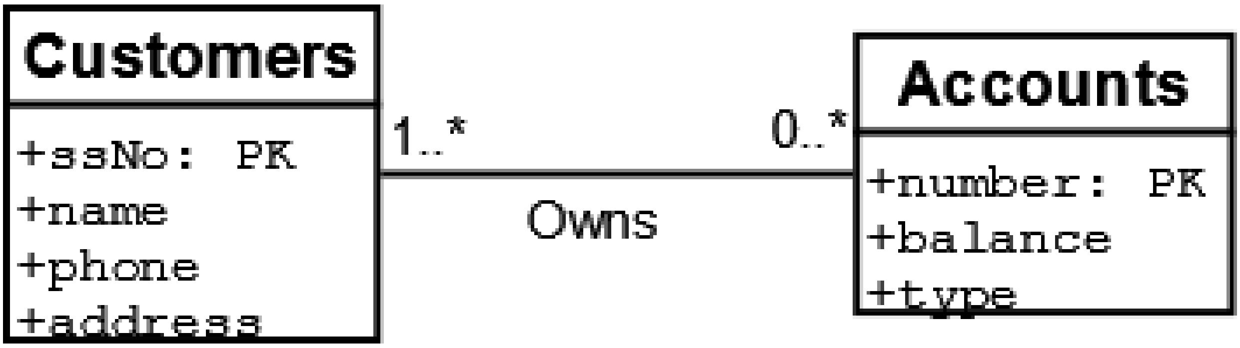


(c)
The nulls method always results in one relation and contains attributes from all e entities i.e. e*a attributes.

Summarizing for a,b, and c above;

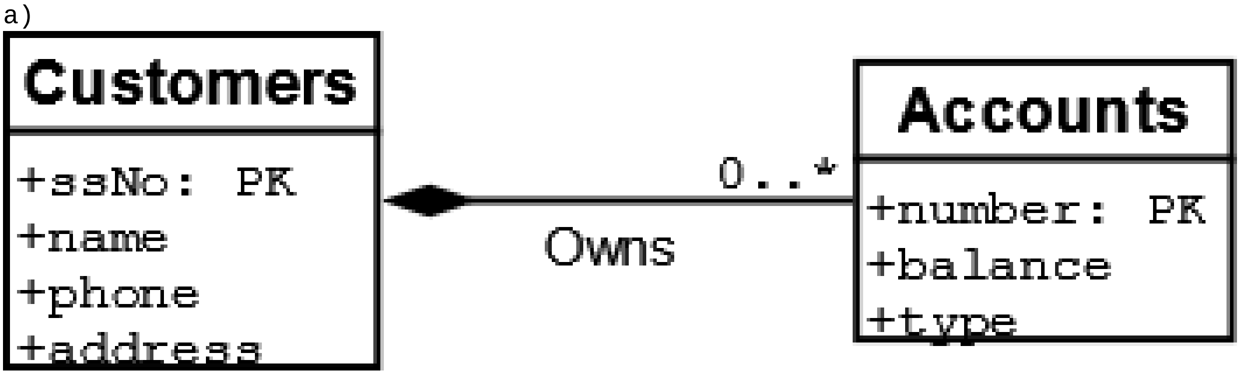
	#Components		#Relations	
	Min	Max	Min	Max
Method				
straight-E/R	a	a	e	e
object-oriented	a	e*a	e	2^(e-1)
nulls	e*a	e*a	1	1

4.7.1

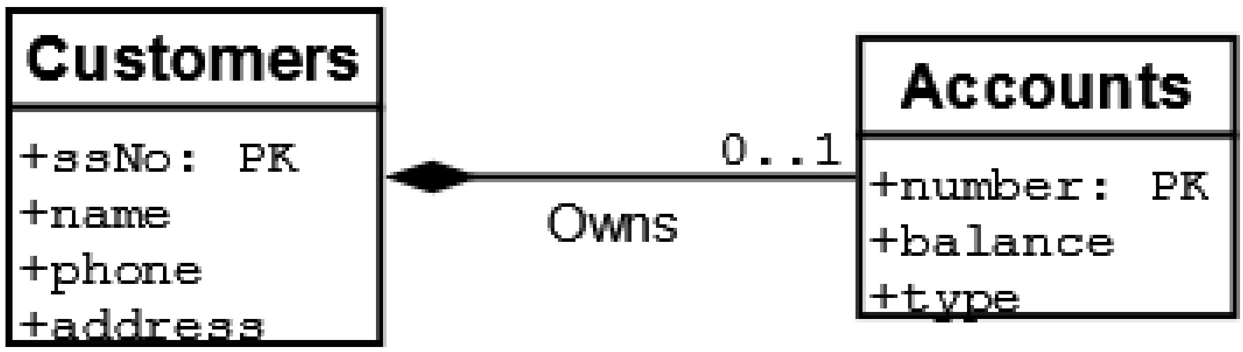

Customers
+ssNo: PK
+name
+phone
+address
1..*
Owns
0..*
Accounts
+number: PK
+balance
+type

4.7.2

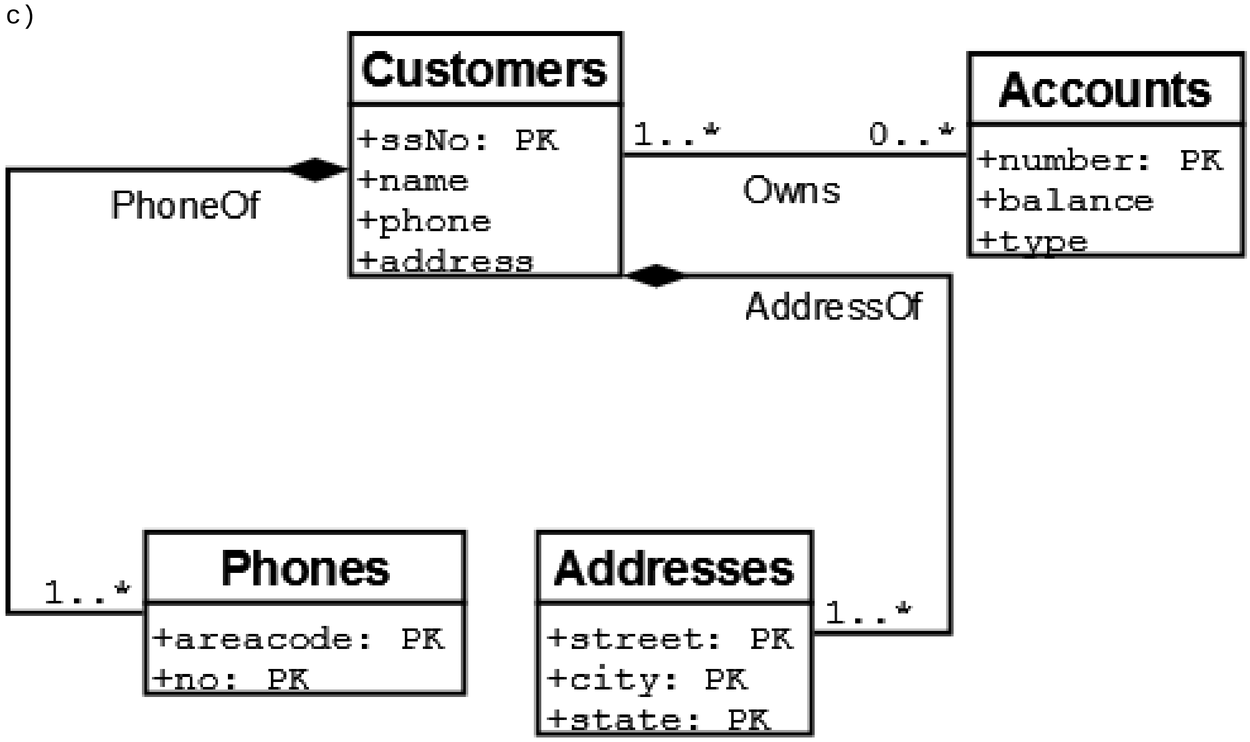
a)


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

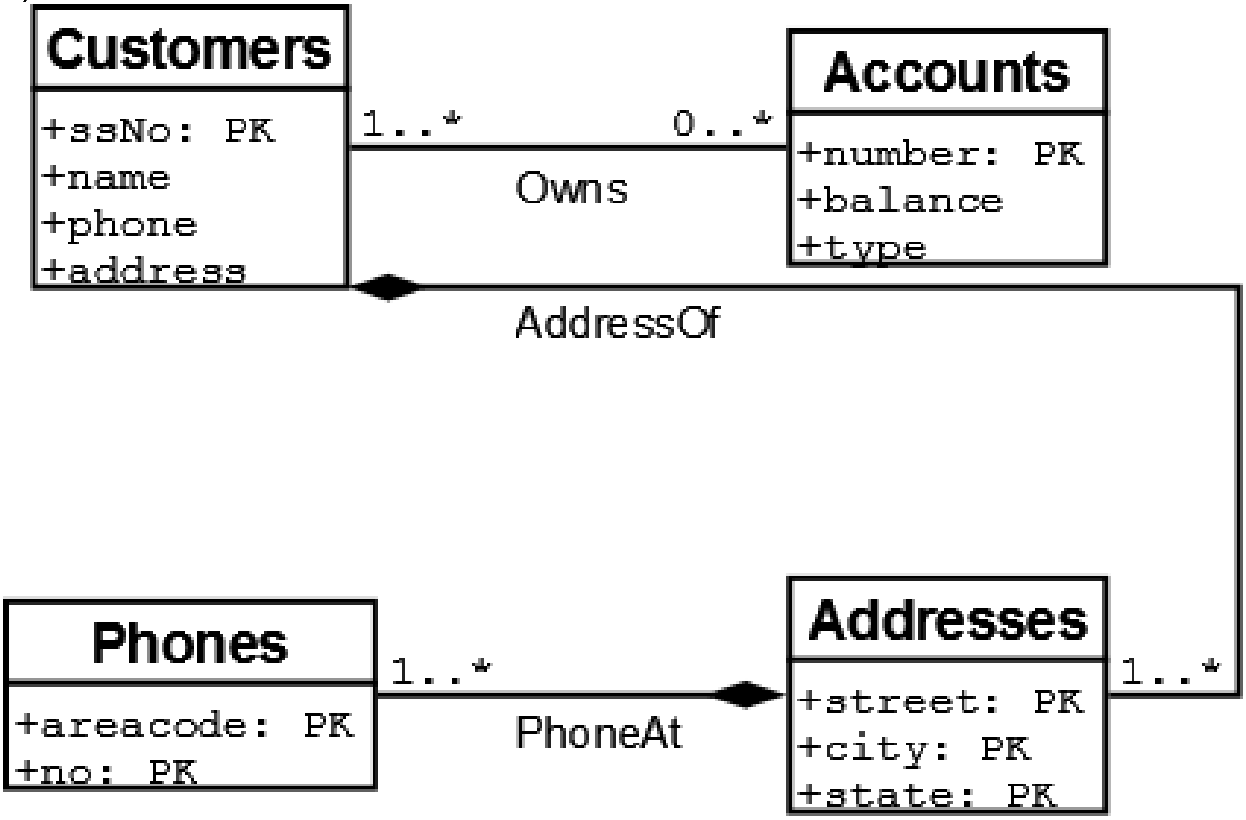
b)


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

c)

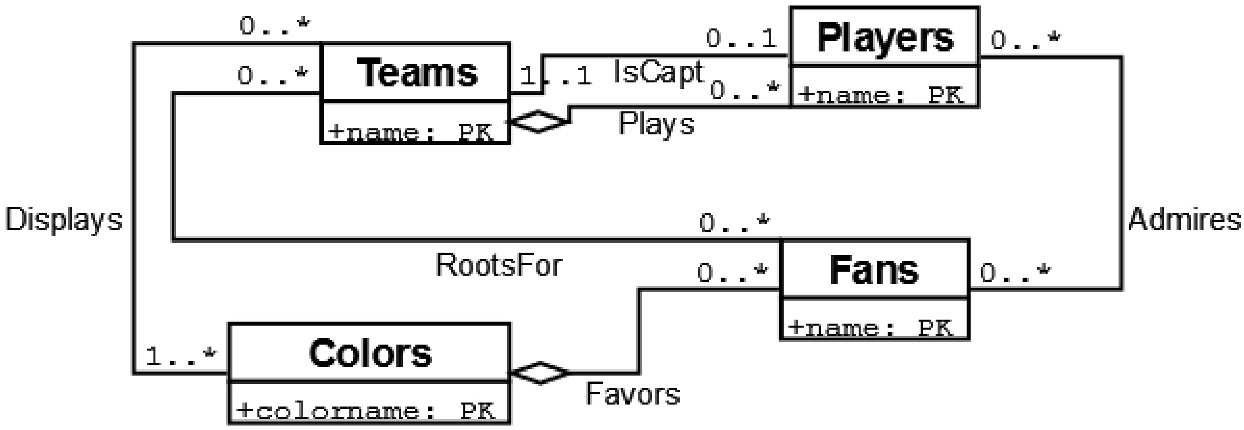

Customers
+ssNo: PK
+name
+phone
+address
1..*
0..*
Owns
Accounts
+number: PK
+balance
+type
PhoneOf
AddressOf
1..*
Phones
+areacode: PK
+no: PK
Addresses
+street: PK
+city: PK
+state: PK
1..*

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..*

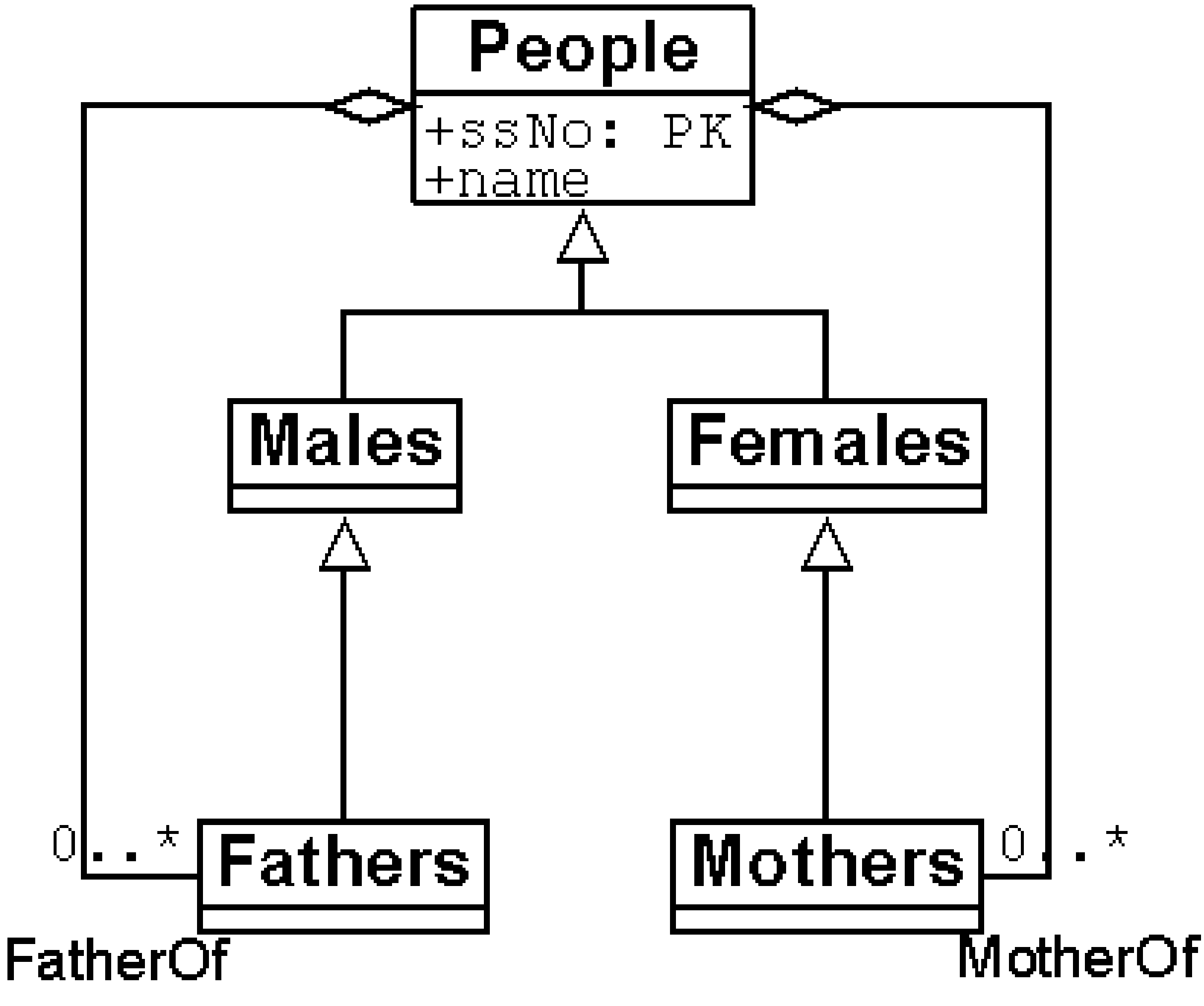
4.7.3



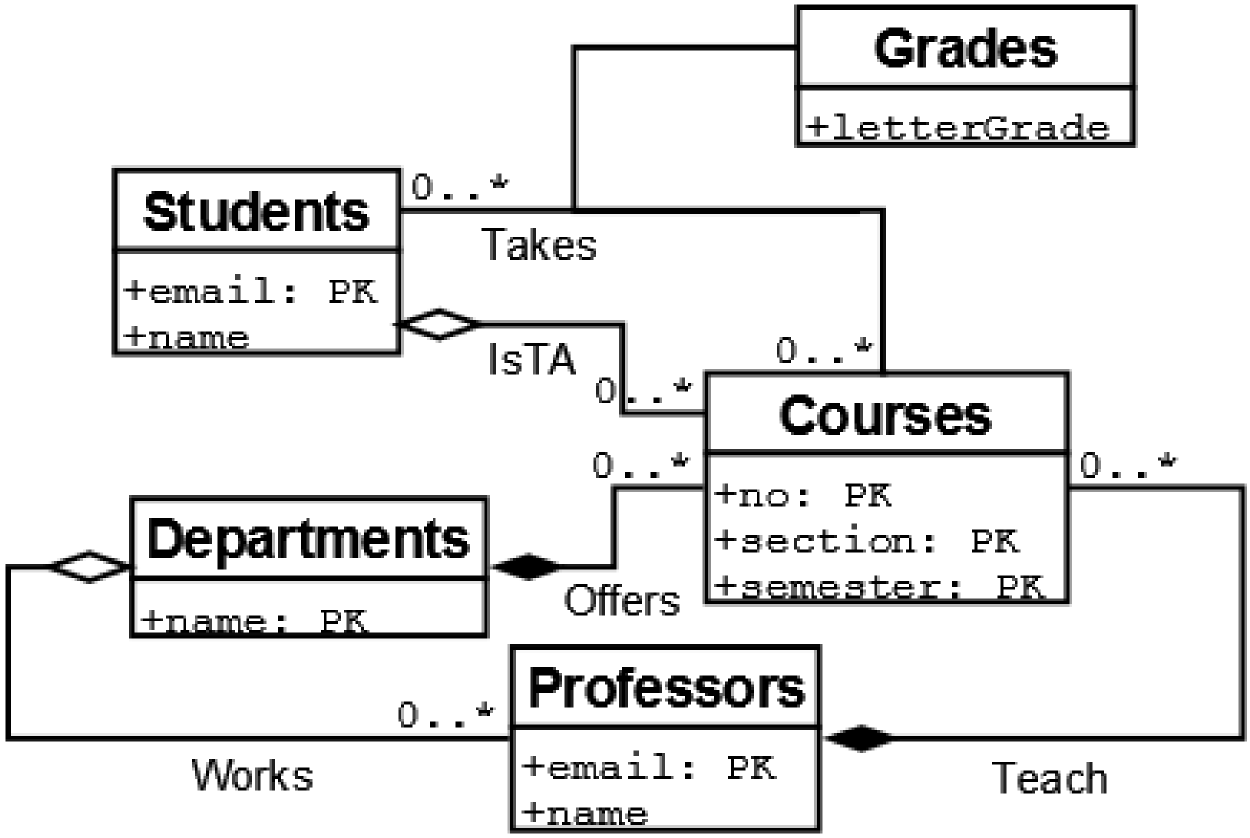
0..*
0..*
Teams
+name: PK
1..1
IsCapt
0..1
0..*
Players
+name: PK
0..*
Plays
Displays
0..*
RootsFor
0..*
Fans
+name: PK
0..*
Admires
1..*
Colors
+colorname: PK
Favors

4.7.4

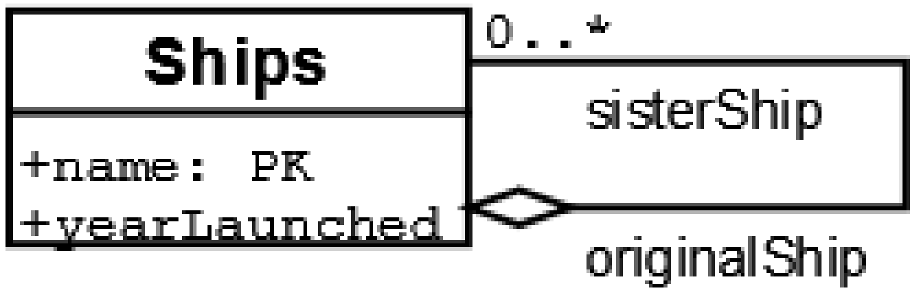
4.7.5
Males and Females subclasses are complete. Mothers and Fathers are partial. All subclasses are disjoint.



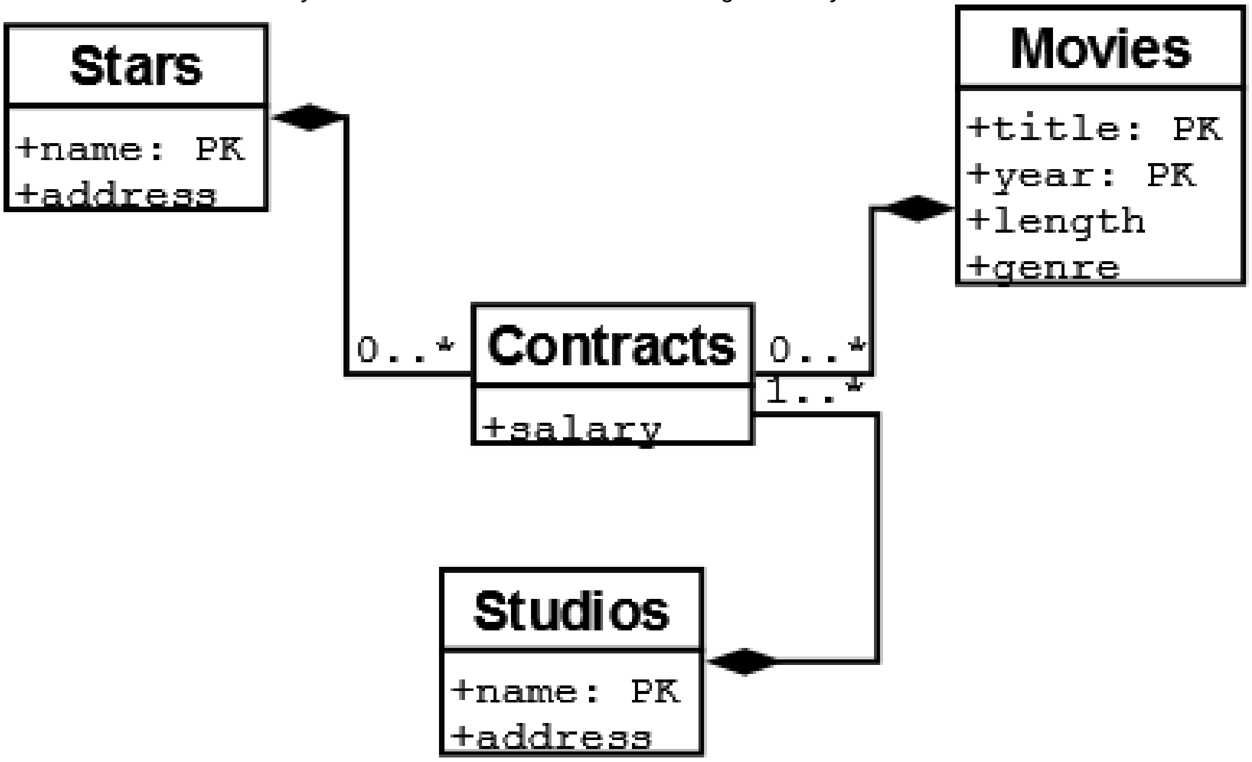
4.7.6


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

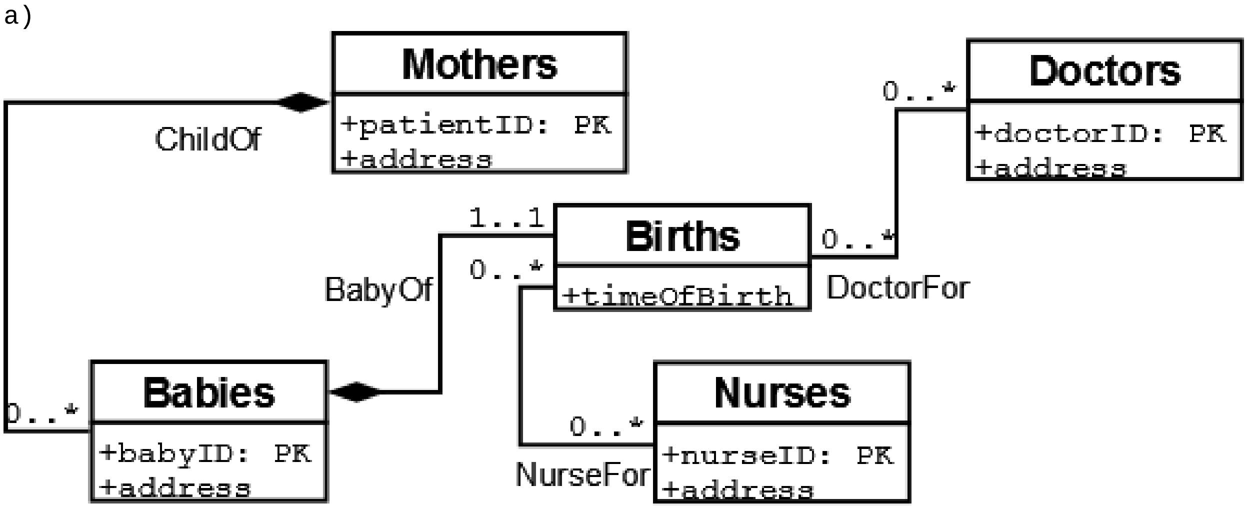
4.7.7


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

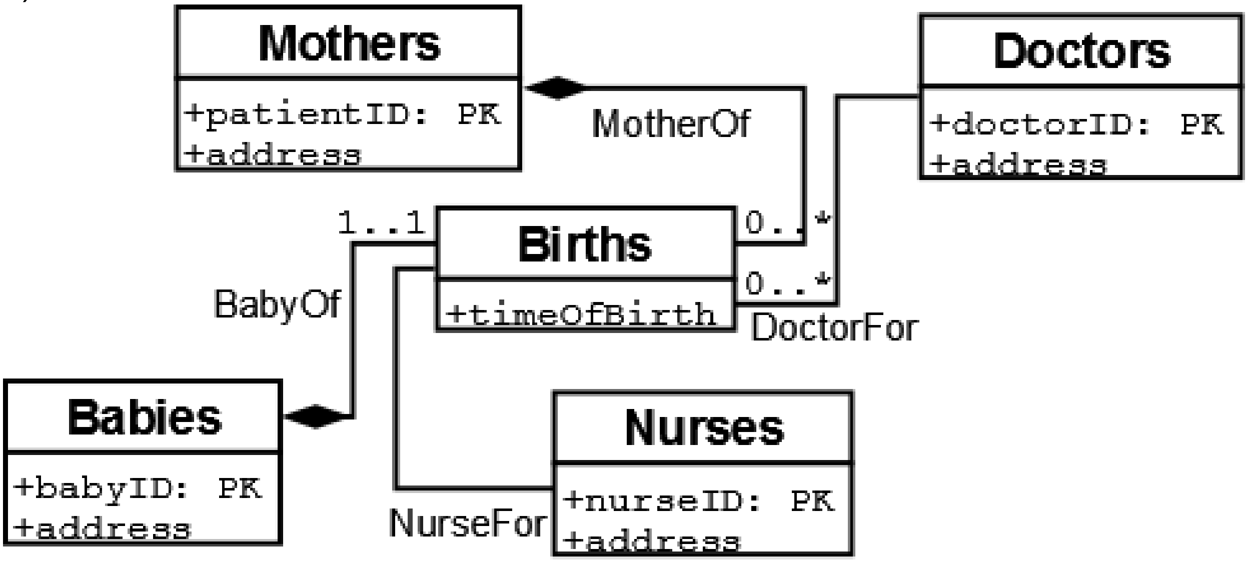
4.7.8
We convert the ternary relationship Contracts into three binary relationships between a new entity set Contracts and existing entity sets.



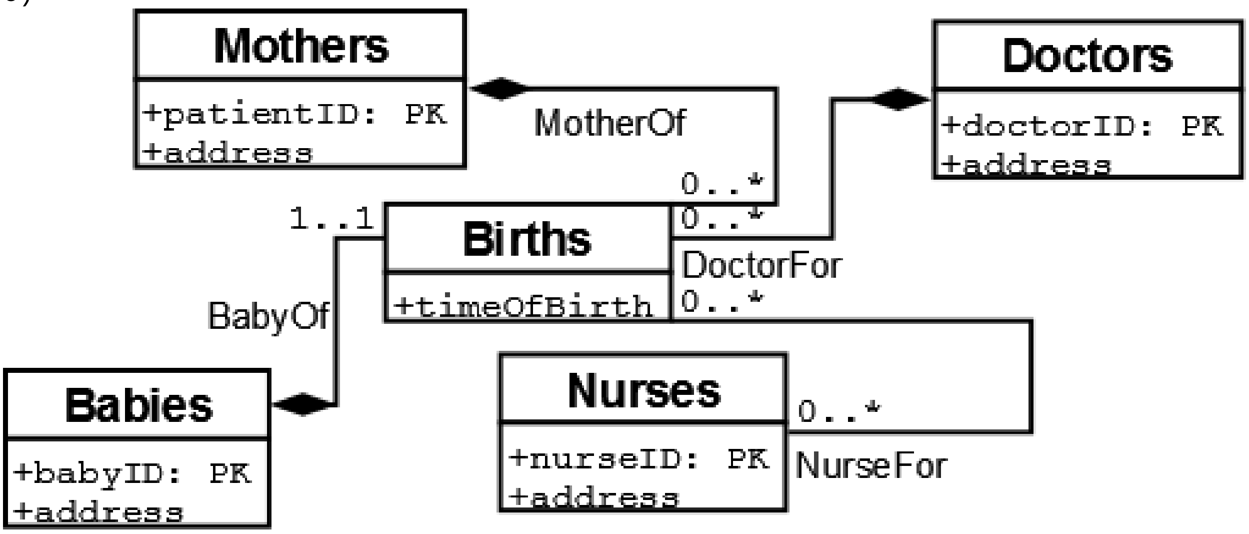
4.7.9
a)



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

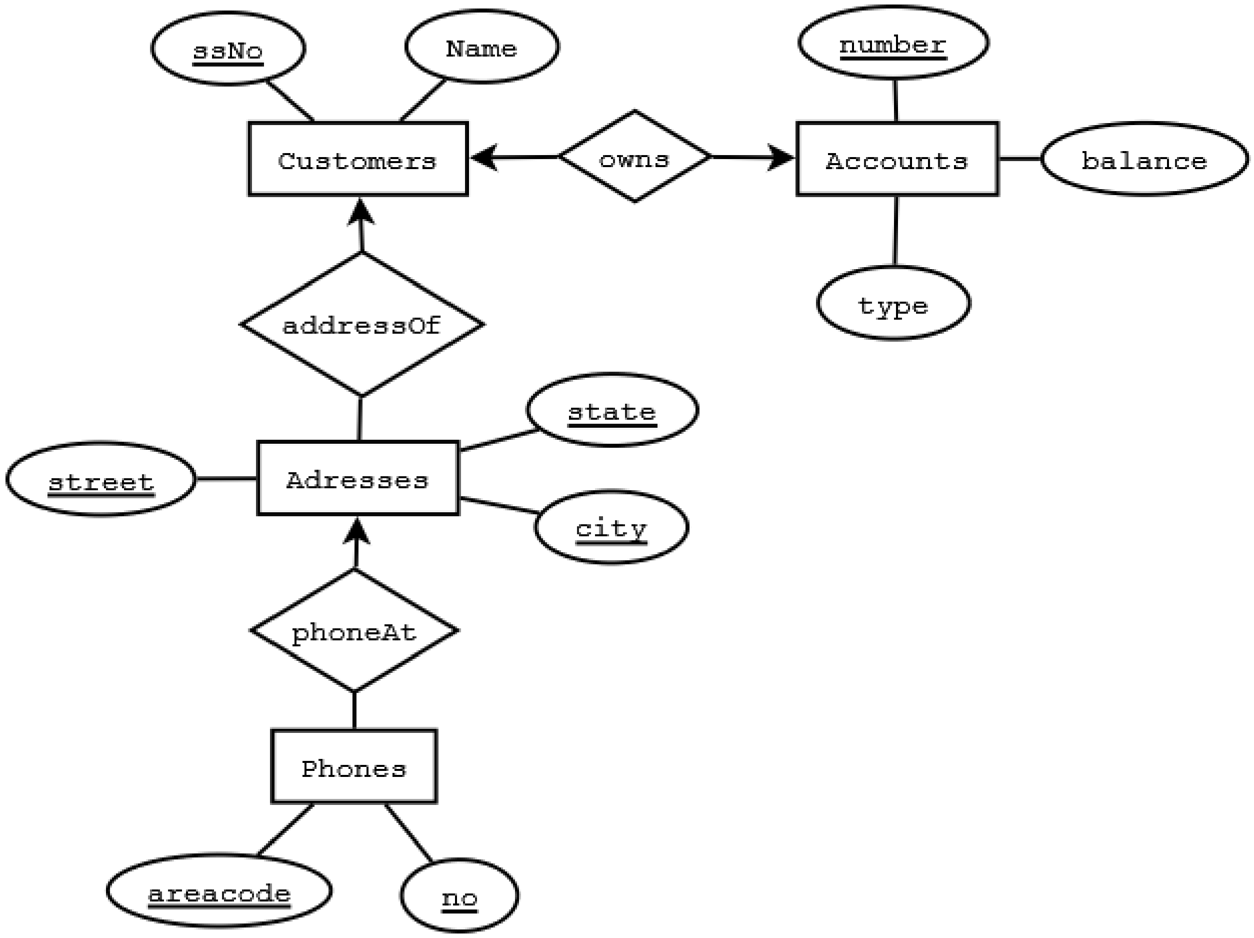
```
class Students (key email) {
  attribute string email;
  attribute string name;
  relationship Courses isTA
    inverse Courses::TA;
  relationship Courses Takes
    inverse Courses::TakenBy;
};

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)

Exercise 5.1.1

As a set:

speed
2.66
2.10
1.42
2.80
3.20
2.20
2.00
1.86
3.06

Average = 2.37

As a bag:

speed
2.66
2.10
1.42
2.80
3.20
3.20
2.20
2.20
2.00
2.80
1.86
2.80
3.06

Average = 2.48

Exercise 5.1.2

As a set:

hd
250
80
320
200
300
160

Average = 218

As a bag:

hd
250
250
80
250
250
320
200
250
250
300
160
160
80

Average = 215

Exercise 5.1.3a

As a set:

bore
15
16
14
18

As a bag:

bore
15
16
14
16
15
15
14
18

Exercise 5.1.3b

π_{bore}(Ships $\bowtie$ Classes)

Exercise 5.1.4a

For bags:

On the left-hand side:

Given bags R and S where a tuple t appears n and m times respectively, the union of bags R and S will have tuple t appear n + m times. The further union of bag T with the tuple t appearing o times will have tuple t appear n + m + o times in the final result.

On the right-hand side:

Given bags S and T where a tuple t appears m and o times respectively, the union of bags R and S will have tuple t appear m + o times. The further union of bag R with the tuple t appearing n times will have tuple t appear m + o + n times in the final result.

For sets:

This is a similar case when dealing with bags except the tuple t can only appear at most once in each set. The tuple t only appears in the result if all the sets have the tuple t. Otherwise, the tuple t will not appear in the result. Since we cannot have duplicates, the result only has at most one copy of the tuple t.

Exercise 5.1.4b

For bags:

On the left-hand side:

Given bags R and S where a tuple t appears n and m times respectively, the intersection of bags R and S will have tuple t appear min(n, m) times. The further intersection of bag T with the tuple t appearing o times will produce tuple t min(o, min(n, m)) times in the final result.

On the right-hand side:

Given bags S and T where a tuple t appears m and o times respectively, the intersection of bags R and S will have tuple t appear min(m, o) times. The further intersection of bag R with the tuple t appearing n times will produce tuple t min(n, min(m, o)) times in the final result.

The intersection of bags R,S and T will yield a result where tuple t appears min(n,m,o) times.

For sets:

This is a similar case when dealing with bags except the tuple t can only appear at most once in each set. The tuple t only appears in the result if all the sets have the tuple t. Otherwise, the tuple t will not appear in the result.

Exercise 5.1.4c

For bags:

On the left-hand side:

Given that tuple r in R, which appears m times, can successfully join with tuple s in S, which appears n times, we expect the result to contain mn copies. Also given that tuple t in T, which appears o times, can successfully join with the joined tuples of r and s, we expect the final result to have mno copies.

On the right-hand side:

Given that tuple s in S, which appears n times, can successfully join with tuple t in T, which appears o times, we expect the result to contain no copies. Also given that tuple r in R, which appears m times, can successfully join with the joined tuples of s and t, we expect the final result to have nom copies.

The order in which we perform the natural join does not matter for bags.

For sets:

This is a similar case when dealing with bags except the joined tuples can only appear at most once in each result. If there are tuples r,s,t in relations R,S,T that can successfully join, then the result will contain a tuple with the schema of their joined attributes.

Exercise 5.1.4d

For bags:

Suppose a tuple t occurs n and m times in bags R and S respectively. In the union of these two bags $R \cup S$, tuple t would appear $n + m$ times. Likewise, in the union of these two bags $S \cup R$, tuple t would appear $m + n$ times. Both sides of the relation yield the same result.

For sets:

A tuple t can only appear at most one time. Tuple t might appear each in sets R and S one or zero times. The combinations of number of occurrences for tuple t in R and S respectively are (0,0), (0,1), (1,0), and (1,1). Only when tuple t appears in both sets R and S will the union $R \cup S$ have the tuple t. The same reasoning holds when we take the union $S \cup R$.

Therefore the commutative law for union holds.

Exercise 5.1.4e

For bags:

Suppose a tuple t occurs n and m times in bags R and S respectively. In the intersection of these two bags $R \cap S$, tuple t would appear $\min(n,m)$ times. Likewise in the intersection of these two bags $S \cap R$, tuple t would appear $\min(m,n)$ times. Both sides of the relation yield the same result.

For sets:

A tuple t can only appear at most one time. Tuple t might appear each in sets R and S one or zero times. The combinations of number of occurrences for tuple t in R and S respectively are (0,0), (0,1), (1,0), and (1,1). Only when tuple t appears in at least one of the sets R and S will the intersection $R \cap S$ have the tuple t. The same reasoning holds when we take the intersection $S \cap R$.

Therefore the commutative law for intersection holds.

Exercise 5.1.4f

For bags:

Suppose a tuple t occurs n times in bag R and tuple u occurs m times in bag S. Suppose also that the two tuples t,u can successfully join. Then in the natural join of these two bags $R \bowtie S$, the joined tuple would appear nm times. Likewise in the natural join of these two bags $S \bowtie R$, the joined tuple would appear mn times. Both sides of the relation yield the same result.

For sets:

An arbitrary tuple t can only appear at most one time in any set. Tuples u,v might appear respectively in sets R and S one or zero times. The combinations of number of occurrences for tuples u,v in R and S respectively are (0,0), (0,1), (1,0), and (1,1). Only when tuple u exists in R and tuple v exists in S will the natural join R ⋈ S have the joined tuple. The same reasoning holds when we take the natural join S ⋈ R.

Therefore the commutative law for natural join holds.

Exercise 5.1.4g

For bags:

Suppose tuple t appears m times in R and n times in S. If we take the union of R and S first, we will get a relation where tuple t appears m + n times. Taking the projection of a list of attributes L will yield a resulting relation where the projected attributes from tuple t appear m + n times. If we take the projection of the attributes in list L first, then the projected attributes from tuple t would appear m times from R and n times from S. The union of these resulting relations would have the projected attributes of tuple t appear m + n times.

For sets:

An arbitrary tuple t can only appear at most one time in any set. Tuple t might appear in sets R and S one or zero times. The combinations of number of occurrences for tuple t in R and S respectively are (0,0), (0,1), (1,0), and (1,1). Only when tuple t exists in R or S (or both R and S) will the projected attributes of tuple t appear in the result.

Therefore the law holds.

Exercise 5.1.4h

For bags:

Suppose tuple t appears u times in R, v times in S and w times in T. On the left hand side, the intersection of S and T would produce a result where tuple t would appear min(v , w) times. With the addition of the union of R, the overall result would have u + min(v , w) copies of tuple t. On the right hand side, we would get a result of min(u + v, u + w) copies of tuple t. The expressions on both the left and right sides are equivalent.

For sets:

An arbitrary tuple t can only appear at most one time in any set. Tuple t might appear in sets R,S and T one or zero times. The combinations of number of occurrences for tuple t in R, S and T respectively are (0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0) and (1,1,1). Only when tuple t appears in R or in both S and T will the result have tuple t.

Therefore the distributive law of union over intersection holds.

Exercise 5.1.4i

Suppose that in relation R, u tuples satisfy condition C and v tuples satisfy condition D. Suppose also that w tuples satisfy both conditions C and D where $w \leq \min(v, w)$. Then the left hand side will return those w tuples. On the right hand side, $\sigma_C(R)$ produces u tuples and $\sigma_D(R)$ produces v tuples. However, we know the intersection will produce the same w tuples in the result.

When considering bags and sets, the only difference is bags allow duplicate tuples while sets only allow one copy of the tuple. The example above applies to both cases.

Therefore the law holds.

Exercise 5.1.5a

For sets, an arbitrary tuple t appears on the left hand side if it appears in both R,S and not in T. The same is true for the right hand side.

As an example for bags, suppose that tuple t appears one time each in both R,T and two times in S. The result of the left hand side would have zero copies of tuple t while the right hand side would have one copy of tuple t.

Therefore the law holds for sets but not for bags.

Exercise 5.1.5b

For sets, an arbitrary tuple t appears on the left hand side if it appears in R and either S or T. This is equivalent to saying tuple t only appears when it is in at least R and S or in R and T. The equivalence is exactly the right side’s expression.

As an example for bags, suppose that tuple t appears one time in R and two times each in S and T. Then the left hand side would have one copy of tuple t in the result while the right hand side would have two copies of tuple t.

Therefore the law holds for sets but not for bags.

Exercise 5.1.5c

For sets, an arbitrary tuple t appears on the left hand side if it satisfies condition C, condition D or both condition C and D. On the right hand side, $\sigma_C(R)$ selects those tuples that satisfy condition C while $\sigma_D(R)$ selects those tuples that satisfy condition D. However, the union operator will eliminate duplicate tuples, namely those tuples that satisfy both condition C and D. Thus we are ensured that both sides are equivalent.

As an example for bags, we only need to look at the union operator. If there are indeed tuples that satisfy both conditions C and D, then the right hand side will contain duplicate copies of those tuples. The left hand side, however, will only have one copy for each tuple of the original set of tuples.

Exercise 5.2.1a

A+B	A^2	B^2
1	0	1
5	4	9
1	0	1
6	4	16
7	9	16

Exercise 5.2.1b

B+1	C-1
1	0
3	3
3	4
4	3
1	1
4	3

Exercise 5.2.1c

A	B
0	1
0	1
2	3
2	4
3	4

Exercise 5.2.1d

B	C
0	1
0	2
2	4
2	5
3	4
3	4

Exercise 5.2.1e

A	B
0	1
2	3
2	4
3	4

Exercise 5.2.1f

B	C
0	1
2	4
2	5
3	4
0	2

Exercise 5.2.1g

A	SUM(B)
0	2
2	7
3	4

Exercise 5.2.1h

B	AVG(C)
0	1.5
2	4.5
3	4

Exercise 5.2.1i

A
0
2
3

Exercise 5.2.1j

A	MAX(C)
2	4

Exercise 5.2.1k

A	B	C
2	3	4
2	3	4
0	1	⊥
0	1	⊥
2	4	⊥
3	4	⊥

Exercise 5.2.1l

A	B	C
2	3	4
2	3	4
⊥	0	1
⊥	2	4
⊥	2	5
⊥	0	2

Exercise 5.2.1m

A	B	C
2	3	4
2	3	4
0	1	⊥
0	1	⊥
2	4	⊥
3	4	⊥
⊥	0	1
⊥	2	4
⊥	2	5
⊥	0	2

Exercise 5.2.1n

A	R.B	S.B	C
0	1	2	4
0	1	2	5
0	1	3	4
0	1	3	4
0	1	2	4
0	1	2	5
0	1	3	4
0	1	3	4
2	3	$\perp$	$\perp$
2	4	$\perp$	$\perp$
3	4	$\perp$	$\perp$
$\perp$	$\perp$	0	1
$\perp$	$\perp$	0	2

Exercise 5.2.2a

Applying the δ operator on a relation with no duplicates will yield the same relation. Thus idempotent.

Exercise 5.2.2b

The result of π_L is a relation over the list of attributes L. Performing the projection again will return the same relation because the relation only contains the list of attributes L. Thus π_L is idempotent.

Exercise 5.2.2c

The result of σ_C is a relation where condition C is satisfied by every tuple. Performing the selection again will return the same relation because the relation only contains tuples that satisfy the condition C. Thus σ_C is idempotent.

Exercise 5.2.2d

The result of γ_L is a relation whose schema consists of the grouping attributes and the aggregated attributes. If we perform the same grouping operation, there is no guarantee that the expression would make sense. The grouping attributes will still appear in the new result. However, the aggregated attributes may or may not appear correctly. If the aggregated attribute is given a different name than the original attribute, then performing γ_L would not make sense because it contains an aggregation for an attribute name that does not exist. In this case, the resulting relation would, according to the definition, only contain the grouping attributes. Thus, $_L$ is not idempotent.

Exercise 5.2.2e

The result of τ is a sorted list of tuples based on some attributes L. If L is not the entire schema of relation R, then there are attributes that are not sorted on. If in relation R there are two tuples that agree in all attributes L and disagree in some of the remaining attributes not in L, then it is arbitrary as to which order these two tuples appear in the result. Thus, performing the operation multiple times can yield a different relation where these two tuples are swapped. Thus, idempotent.

Exercise 5.2.3

If we only consider sets, then it is possible. We can take $_A$(R) and do a product with itself. From this product, we take the tuples where the two columns are equal to each other.

If we consider bags as well, then it is not possible. Take the case where we have the two tuples (1,0) and (1,0). We wish to produce a relation that contains tuples (1,1) and (1,1). If we use the classical operations of relational algebra, we can either get a result where there are no tuples or four copies of the tuple (1,1). It is not possible to get the desired relation because no operation can distinguish between the original tuples and the duplicated tuples. Thus it is not possible to get the relation with the two tuples (1,1) and (1,1).

Exercise 5.3.1

a) Answer(model) ← PC(model,speed,_,_,_) AND speed ≥ 3.00

b) Answer(maker) ← Laptop(model,_,_,hd,_,_) AND Product(maker,model,_) AND hd ≥ 100

c) Answer(model,price) ← PC(model,_,_,_,price) AND Product(maker,model,_) AND maker='B'
Answer(model,price) ← Laptop(model,_,_,_,_,price) AND Product(maker,model,_) AND maker='B'
Answer(model,price) ← Printer(model,_,_,price) AND Product(maker,model,_) AND maker='B'

d) Answer(model) ← Printer(model,color,type,_) AND color='true' AND type='laser'

e) PCMaker(maker) ← Product(maker,_,type) AND type='pc'
LaptopMaker(maker) ← Product(maker,_,type) AND type='laptop'
Answer(maker) ← LaptopMaker(maker) AND NOT PCMaker(maker)

f) Answer(hd) ← PC(model1,_,_,hd,_) AND PC(model2,_,_,hd,_) AND model1 <> model2

g) Answer(model1,model2) ← PC(model1,speed, ram,_,_) AND PC(model2,_speed,ram,_,_) AND model1 < model2

h) FastComputer(model) ← PC(model,speed,_,_,_) AND speed ≥ 2.80
FastComputer(model) ← Laptop(model,speed,_,_,_,_) AND speed ≥ 2.80
Answer(maker) ← Product(maker,model1,_) AND Product(maker,model2,_) AND FastComputer(model1) AND FastComputer(model2) AND model1 <> model2

i) Computers(model,speed) ← PC(model,speed,_,_,_)
Computers(model,speed) ← Laptop(model,speed,_,_,_,_)
SlowComputers(model) ← Computers(model,speed) AND Computers(model1,speed1) AND speed < speed1
FastestComputers(model) ← Computers(model,_) AND NOT SlowComputers(model)
Answer(maker) ← FastestComputers(model) AND Product(maker,model,_)

j) PCs(maker,speed) ← PC(model,speed,_,_,_) AND Product(maker,model,_)
Answer(maker) ← PCs(maker,speed) AND PCs(maker,speed1) AND PCs(maker,speed2) AND speed <> speed1 AND speed <> speed2 AND speed1 <> speed2

k) PCs(maker,model) ← Product(maker,model,type) AND type= ’pc’
Answer(maker) ← PCs(maker,model) AND PCs(maker,model1) AND PCs(maker,model2) AND PCs(maker,model3) AND model <> model1 AND model <> model2 AND model1 <> model2 AND (model3 = model OR model3 = model1 OR model3 = model2)

Exercise 5.3.2

a) Answer(class,country) ← Classes(class,_,country,_,bore,_) AND bore ≥ 16

b) Answer(name) ← Ships(name,_,launched) AND launched < 1921

c) Answer(ship) ← Outcomes(ship,battle,result) AND battle= ’Denmark Strait’ AND re = ‘sunk’

d) Answer(name) ← Classes(class,_,_,_,_,displacement) AND Ships(name,class,launched) AND displacement > 35000 AND launched > 1921

e) Answer(name,displacement,numGuns) ← Classes(class,_,_,numGuns,_,displacement) AND Ships(name,class,_) AND Outcomes (ship,battle,_) AND battle= ’Guadalcanal’ AND ship=name

f) Answer(name) ← Ships(name,_,_)
Answer(name) ← Outcomes(name,_,_) AND NOT Answer(name)

g) MoreThanOne(class) ← Ships(name,class,_) AND Ships(name1,class,_) AND name <> name1
Answer(class) ← Classes(class,_,_,_,_,_) AND NOT MoreThanOne(class)

h) Battleship(country) ← Classes(_,type,country,_,_,_) AND type= ’bb’
Battlecruiser(country) ← Classes(_,type,country,_,_,_) AND type= ’bc’
Answer(country) ← Battleship(country) AND Battlecruiser(country)

i) Results(ship,result,date) ← Battles(name,date) AND Outcomes(ship,battle,result) AND battle=name
Answer(ship) ← Results(ship,result,date) AND Results(ship,_,date1) AND result= ’damaged’ AND date < date1

Exercise 5.3.3

Answer(x,y) ← R(x,y) AND≠z z

Exercise 5.4.1a

Answer(a,b,c) ← R(a,b,c)
Answer(a,b,c) ← S(a,b,c)

Exercise 5.4.1b

Answer(a,b,c) ← R(a,b,c) AND S(a,b,c)

Exercise 5.4.1c

Answer(a,b,c) ← R(a,b,c) AND NOT S(a,b,c)

Exercise 5.4.1d

Union(a,b,c) ← R(a,b,c)
Union(a,b,c) ← S(a,b,c)
Answer(a,b,c) ← Union(a,b,c) AND NOT T(a,b,c)

Exercise 5.4.1e

J(a,b,c) ← R(a,b,c) AND NOT S(a,b,c)
K(a,b,c) ← R(,a,b,c) AND NOT T(a,b,c)
Answer(a,b,c) ← J(a,b,c) AND K(a,b,c)

Exercise 5.4.1f

Answer(a,b) $\leftarrow$ R(a,b,_)

Exercise 5.4.1g

J(a,b) $\leftarrow$ R(a,b,_)
K(a,b) $\leftarrow$ S(_,a,b)
Answer(a,b) $\leftarrow$ J(a,b) AND K(a,b)

Exercise 5.4.2a

Answer(x,y,z) $\leftarrow$ R(x,y,z) AND $x = y$

Exercise 5.4.2b

Answer(x,y,z) $\leftarrow$ R(x,y,z) AND $x < y$ AND $y < z$

Exercise 5.4.2c

Answer(x,y,z) $\leftarrow$ R(x,y,z) AND $x < y$
Answer(x,y,z) $\leftarrow$ R(x,y,z) AND $y < z$

Exercise 5.4.2d

Change: NOT($x < y$ OR $x > y$)
To: $x \geq y$ AND $x \leq y$

The above simplifies to $x = y$

Answer(x,y,z) $\leftarrow$ R(x,y,z) AND $x = y$

Exercise 5.4.2e

Change: NOT(($x < y$ OR $x > y$) AND $y < z$)
NOT($x < y$ OR $x > y$) OR $y \geq z$
($x \geq y$ AND $x \leq y$) OR $y \geq z$
To: $x = y$ OR $y \geq z$

Answer(x,y,z) $\leftarrow$ R(x,y,z) AND $x = y$
Answer(x,y,z) $\leftarrow$ R(x,y,z) AND $y \geq z$

Exercise 5.4.2f

Change: NOT(($x < y$ OR $x < z$) AND $y < z$)
NOT($x < y$ OR $x < z$) OR $y \geq z$
To: ($x \geq y$ AND $x \geq z$) OR $y \geq z$

Answer(x,y,z) $\leftarrow$ R(x,y,z) AND x $\geq$ y AND x $\geq$ z
Answer(x,y,z) $\leftarrow$ R(x,y,z) AND yz $\geq$

Exercise 5.4.3a

Answer(a,b,c,d) $\leftarrow$ R(a,b,c) AND S(b,c,d)

Exercise 5.4.3b

Answer(b,c,d,e) $\leftarrow$ S(b,c,d) AND T(d,e)

Exercise 5.4.3c

Answer(a,b,c,d,e) $\leftarrow$ R(a,b,c) AND S(b,c,d) AND T(d,e)

Exercise 5.4.4

a) Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND rx = sy

b) Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND rx < sy AND ry < sz

c) Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND rx < sy
Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND ry < sz

d) Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND rx = sy

e) Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND rx = sy
Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND ry $\geq$ sz

f) Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND rx $\geq$ sy AND rx $\geq$ s
Answer(rx,ry,rz,sx,sy,sz) $\leftarrow$ R(rx,ry,rz) AND S(sx,sy,sz) AND sz ry $\geq$

Exercise 5.4.5a

R1 := $\pi_{x,y}$(Q $\bowtie$ R)

Exercise 5.4.5b

R1 := $\rho_{R1(x,z)}$(Q)
R2 := $\rho_{R2(z,y)}$(Q)
R3 := $\pi_{x,y}$(R1 $\bowtie_{(R1.z = R2.z)}$ R2)

Exercise 5.4.5c

R1 := $\pi_{x,y}$(Q $\bowtie$ R)
R2 := $\sigma_{x<y}$(R1)

Solutions
Chapter 6

6.1.1
Attributes must be separated by commas. Thus here B is an alias of A.

6.1.2
a)
```
SELECT  address AS Studio_Address
FROM    Studio
WHERE   NAME = 'MGM';
```

b)
```
SELECT  birthdate AS Star_Birthdate
FROM    MovieStar
WHERE   name = 'Sandra Bullock';
```

c)
```
SELECT  starName
FROM    StarsIn
WHERE   movieYear = 1980
     OR movieTitle LIKE '%Love%';
```

However, above query will also return words that have the substring Love e.g. Lover. Below query will only return movies that have title containing the word Love.

```
SELECT  starName
FROM    StarsIn
WHERE   movieYear = 1980
     OR movieTitle LIKE 'Love %'
     OR movieTitle LIKE '% Love %'
     OR movieTitle LIKE '% Love'
     OR movieTitle = 'Love';
```

d)
```
SELECT  name AS Exec_Name
FROM    MovieExec
WHERE   netWorth >= 10000000;
```

e)
```
SELECT  name AS Star_Name
FROM    movieStar
WHERE   gender = 'M'
     OR address LIKE '% Malibu %';
```

6.1.3

a)

```
SELECT  model,
        speed,
        hd
FROM    PC
WHERE   price < 1000 ;

MODEL SPEED      HD
----- ---------- ------
1002        2.10    250
1003        1.42     80
1004        2.80    250
1005        3.20    250
1007        2.20    200
1008        2.20    250
1009        2.00    250
1010        2.80    300
1011        1.86    160
1012        2.80    160
1013        3.06     80

  11 record(s) selected.
```

b)

```
SELECT  model              ,
        speed AS gigahertz,
        hd    AS gigabytes
FROM    PC
WHERE   price < 1000 ;

MODEL GIGAHERTZ  GIGABYTES
----- ---------- ---------
1002        2.10       250
1003        1.42        80
1004        2.80       250
1005        3.20       250
1007        2.20       200
1008        2.20       250
1009        2.00       250
1010        2.80       300
1011        1.86       160
1012        2.80       160
1013        3.06        80

  11 record(s) selected.
```

c)

```
SELECT  maker
FROM    Product
WHERE   TYPE = 'printer' ;

MAKER
-----
D
D
E
E
E
H
H

  7 record(s) selected.
```

d)

```
SELECT  model,
        ram  ,
        screen
FROM    Laptop
WHERE   price > 1500 ;

MODEL RAM    SCREEN
----- ------ -------
2001    2048    20.1
2005    1024    17.0
2006    2048    15.4
2010    2048    15.4

  4 record(s) selected.
```

e)

```
SELECT  *
FROM    Printer
WHERE   color ;

MODEL CASE  TYPE     PRICE
----- ----- -------- ------
3001  TRUE  ink-jet      99
3003  TRUE  laser       999
3004  TRUE  ink-jet     120
3006  TRUE  ink-jet     100
3007  TRUE  laser       200

  5 record(s) selected.
```

Note: Implementation of Boolean type is optional in SQL standard (feature ID T031). PostgreSQL has implementation similar to above example. Other DBMS provide equivalent support. E.g. In DB2 the column type can be declare as SMALLINT with CONSTRAINT that the value can be 0 or 1. The result can be returned as Boolean type CHAR using CASE.

```
CREATE TABLE Printer
        (
                model CHAR(4) UNIQUE NOT NULL,
                color SMALLINT               ,
                type  VARCHAR(8)             ,
                price SMALLINT               ,
                CONSTRAINT Printer_ISCOLOR CHECK(color IN(0,1))
        );
SELECT  model,
        CASE color
                WHEN 1
                THEN 'TRUE'
                WHEN 0
                THEN 'FALSE'
                ELSE 'ERROR'
        END CASE            ,
                        type,
                        price
                FROM    Printer
                WHERE   color = 1;
```

f)

```
SELECT  model,
        hd
FROM    PC
WHERE   speed = 3.2
    AND price < 2000;

MODEL HD
----- ------
1005     250
1006     320

  2 record(s) selected.
```

6.1.4

a)

```
SELECT  class,
        country
FROM    Classes
WHERE   numGuns >= 10 ;

CLASS              COUNTRY
------------------ ------------
Tennessee          USA

  1 record(s) selected.
```

b)

```
SELECT  name AS shipName
FROM    Ships
WHERE   launched < 1918 ;

SHIPNAME
------------------
Haruna
Hiei
Kirishima
Kongo
Ramillies
Renown
Repulse
Resolution
Revenge
Royal Oak
Royal Sovereign

  11 record(s) selected.
```

c)

```
SELECT  ship AS shipName,
        battle
FROM    Outcomes
WHERE   result = 'sunk' ;

SHIPNAME           BATTLE
------------------ ------------------
Arizona            Pearl Harbor
Bismark            Denmark Strait
Fuso               Surigao Strait
Hood               Denmark Strait
Kirishima          Guadalcanal
Scharnhorst        North Cape
Yamashiro          Surigao Strait

  7 record(s) selected.
```

d)
```
SELECT  name AS shipName
FROM    Ships
WHERE   name = class ;

SHIPNAME
------------------
Iowa
Kongo
North Carolina
Renown
Revenge
Yamato

  6 record(s) selected.
```

e)
```
SELECT  name AS shipName
FROM    Ships
WHERE   name LIKE 'R%';

SHIPNAME
------------------
Ramillies
Renown
Repulse
Resolution
Revenge
Royal Oak
Royal Sovereign

  7 record(s) selected.
```

Note: As mentioned in exercise 2.4.3, there are some dangling pointers and to retrieve all ships a UNION of Ships and Outcomes is required.
Below query returns 8 rows including ship named Rodney.

```
SELECT  name AS shipName
FROM    Ships
WHERE   name LIKE 'R%'

UNION

SELECT  ship AS shipName
FROM    Outcomes
WHERE   ship LIKE 'R%';
```

f) Only using a filter like '% % %' will incorrectly match name such as ' a b ' since % can match any sequence of 0 or more characters.

```
SELECT  name AS shipName
FROM    Ships
WHERE   name LIKE '_% _% _%' ;

SHIPNAME
------------------

  0 record(s) selected.
```

Note: As in (e), UNION with results from Outcomes.

```
SELECT  name AS shipName
FROM    Ships
WHERE   name LIKE '_% _% _%'

UNION

SELECT  ship AS shipName
FROM    Outcomes
WHERE   ship LIKE '_% _% _%' ;
```

```
SHIPNAME
------------------
Duke of York
King George V
Prince of Wales

  3 record(s) selected.
```

6.1.5

a)

The resulting expression is false when neither of (a=10) or (b=20) is TRUE.

a = 10	b = 20	a = 10 OR b = 20
NULL	TRUE	TRUE
TRUE	NULL	TRUE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

b)

The resulting expression is only TRUE when both (a=10) and (b=20) are TRUE.

a = 10	b = 20	a = 10 AND b = 20
TRUE	TRUE	TRUE

c)
The expression is always TRUE unless a is NULL.

a < 10	a >= 10	a = 10 AND b = 20
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE

d)
The expression is TRUE when a=b except when the values are NULL.

a	b	a = b
NOT NULL	NOT NULL	TRUE when a=b; else FALSE

e)
Like in (d), the expression is TRUE when a<=b except when the values are NULL.

a	b	a <= b
NOT NULL	NOT NULL	TRUE when a<=b; else FALSE

6.1.6

```
SELECT  *
FROM    Movies
WHERE   LENGTH IS NOT NULL;
```

6.2.1
a)
```
SELECT  M.name AS starName
FROM    MovieStar M,
        StarsIn S
WHERE   M.name       = S.starName
    AND S.movieTitle = 'Titanic'
    AND M.gender     = 'M';
```

b)
```
SELECT  S.starName
FROM    Movies M ,
        StarsIn S,
        Studios T
WHERE   T.name       ='MGM'
    AND M.year       = 1995
    AND M.title      = S.movieTitle
    AND M.studioName = T.name;
```

c)
```
SELECT  X.name AS presidentName
FROM    MovieExec X,
        Studio T
WHERE   X.cert# = T.presC#
    AND T.name  = 'MGM';
```

d)
```
SELECT  M1.title
FROM    Movies M1,
        Movies M2
WHERE   M1.length > M2.length
    AND M2.title  ='Gone With the Wind' ;
```

e)
```
SELECT  X1.name AS execName
FROM    MovieExec X1,
        MovieExec X2
WHERE   X1.netWorth > X2.netWorth
    AND X2.name     = 'Merv Griffin' ;
```

6.2.2

a)
```
SELECT  R.maker AS manufacturer,
        L.speed AS gigahertz
FROM    Product R,
        Laptop L
WHERE   L.hd   >= 30
    AND R.model = L.model ;
```

```
MANUFACTURER GIGAHERTZ
------------ ----------
A                  2.00
A                  2.16
A                  2.00
B                  1.83
E                  2.00
E                  1.73
E                  1.80
F                  1.60
F                  1.60
G                  2.00

  10 record(s) selected.
```

b)
```
SELECT  R.model,
        P.price
FROM    Product R,
        PC P
WHERE   R.maker = 'B'
    AND R.model = P.model

UNION

SELECT  R.model,
        L.price
FROM    Product R,
        Laptop L
WHERE   R.maker = 'B'
    AND R.model = L.model

UNION

SELECT  R.model,
        T.price
FROM    Product R,
        Printer T
WHERE   R.maker = 'B'
    AND R.model = T.model ;

MODEL PRICE
----- ------
1004     649
1005     630
1006    1049
2007    1429

  4 record(s) selected.
```

c)
```
SELECT  R.maker
FROM    Product R,
        Laptop L
WHERE   R.model = L.model

EXCEPT

SELECT  R.maker
FROM    Product R,
        PC P
WHERE   R.model = P.model ;

MAKER
-----
F
G

  2 record(s) selected.
```

d)
```
SELECT DISTINCT P1.hd
FROM    PC P1,
        PC P2
WHERE   P1.hd    =P2.hd
    AND P1.model > P2.model ;
```

Alternate Answer:

```
SELECT DISTINCT P.hd
FROM    PC P
GROUP BY P.hd
HAVING COUNT(P.model) >= 2 ;
```

e)
```
SELECT  P1.model,
        P2.model
FROM    PC P1,
        PC P2
WHERE   P1.speed = P2.speed
    AND P1.ram   = P2.ram
    AND P1.model < P2.model ;

MODEL MODEL
----- -----
1004  1012

  1 record(s) selected.
```

f)
```
SELECT  M.maker
FROM
        (SELECT maker,
                R.model
        FROM    PC P,
                Product R
        WHERE   SPEED >= 3.0
            AND P.model=R.model

        UNION

        SELECT  maker,
                R.model
        FROM    Laptop L,
                Product R
        WHERE   speed >= 3.0
            AND L.model=R.model
        ) M
GROUP BY M.maker
HAVING COUNT(M.model) >= 2 ;

MAKER
-----
B

  1 record(s) selected.
```

6.2.3

a)

```
SELECT  S.name
FROM    Ships S,
        Classes C
WHERE   S.class        = C.class
    AND C.displacement > 35000;

NAME
------------------
Iowa
Missouri
Musashi
New Jersey
North Carolina
Washington
Wisconsin
Yamato

  8 record(s) selected.
```

b)

```
SELECT  S.name        ,
        C.displacement,
        C.numGuns
FROM    Ships S   ,
        Outcomes O,
        Classes C
WHERE   S.name   = O.ship
    AND S.class  = C.class
    AND O.battle = 'Guadalcanal' ;

NAME               DISPLACEMENT NUMGUNS
------------------ ------------ -------
Kirishima                 32000       8
Washington                37000       9

  2 record(s) selected.
```

Note:South Dakota was also engaged in battle of Guadalcanal but not chosen since it is not in Ships table(Hence, no information regarding it's Class is available).

c)

```
SELECT  name shipName
FROM    Ships

UNION

SELECT  ship shipName
FROM    Outcomes ;

SHIPNAME
------------------
Arizona
Bismark
California
Duke of York
Fuso
Haruna
Hiei
Hood
Iowa
King George V
Kirishima
Kongo
Missouri
Musashi
New Jersey
North Carolina
Prince of Wales
Ramillies
Renown
Repulse
Resolution
Revenge
Rodney
Royal Oak
Royal Sovereign
Scharnhorst
South Dakota
Tennesee
Tennessee
Washington
West Virginia
Wisconsin
Yamashiro
Yamato

  34 record(s) selected.
```

d)
```
SELECT  C1.country
FROM    Classes C1,
        Classes C2
WHERE   C1.country = C2.country
    AND C1.type    = 'bb'
    AND C2.type    = 'bc' ;

COUNTRY
------------
Gt. Britain
Japan

  2 record(s) selected.
```

e)
```
SELECT  O1.ship
FROM    Outcomes O1,
        Battles B1
WHERE   O1.battle = B1.name
    AND O1.result = 'damaged'
    AND EXISTS
        (SELECT B2.date
        FROM    Outcomes O2,
                Battles B2
        WHERE   O2.battle=B2.name
            AND O1.ship  = O2.ship
            AND B1.date  < B2.date
        ) ;

SHIP
------------------

  0 record(s) selected.
```

f)
```
SELECT  O.battle
FROM    Outcomes O,
        Ships S   ,
        Classes C
WHERE   O.ship  = S.name
    AND S.class = C.class
GROUP BY C.country,
        O.battle
HAVING COUNT(O.ship) > 3;
SELECT  O.battle
FROM    Ships S  ,
        Classes C,
        Outcomes O
WHERE   C.Class = S.class
    AND O.ship  = S.name
GROUP BY C.country,
        O.battle
HAVING COUNT(O.ship) >= 3;
```

6.2.4

Since tuple variables are not guaranteed to be unique, every relation Ri should be renamed using an alias. Every tuple variable should be qualified with the alias. Tuple variables for repeating relations will also be distinctly identified this way.

Thus the query will be like

```
SELECT A1.COLL1,A1.COLL2,A2.COLL1,…
FROM R1  A1,R2  A2,…,Rn  An
WHERE A1.COLL1=A2.COLC2,…
```

6.2.5

Again, create a tuple variable for every Ri, i=1,2,...,n

That is, the FROM clause is

```
     FROM R1  A1, R2  A2,...,Rn  An.
```

Now, build the WHERE clause from C by replacing every reference to some attribute COL1 of Ri by Ai.COL1. In addition apply Natural Join i.e. add condition to check equality of common attribute names between Ri and Ri+1 for all i from 0 to n-1. Also, build the SELECT clause from list of attributes L by replacing every attribute COLj of Ri by Ai.COLj.

6.3.1

a)

```
SELECT DISTINCT maker
FROM    Product
WHERE   model IN
        (SELECT model
        FROM    PC
        WHERE   speed >= 3.0
        );
SELECT DISTINCT R.maker
FROM    Product R
WHERE   EXISTS
        (SELECT P.model
        FROM    PC P
        WHERE   P.speed >= 3.0
            AND P.model  =R.model
        );
```

b)

```
SELECT  P1.model
FROM    Printer P1
WHERE   P1.price >= ALL
        (SELECT P2.price
        FROM    Printer P2
        ) ;
SELECT  P1.model
FROM    Printer P1
WHERE   P1.price IN
        (SELECT MAX(P2.price)
        FROM    Printer P2
        ) ;
```

c)

```
SELECT  L.model
FROM    Laptop L
WHERE   L.speed < ANY
        (SELECT P.speed
        FROM    PC P
        ) ;
SELECT  L.model
FROM    Laptop L
WHERE   EXISTS
        (SELECT P.speed
        FROM    PC P
        WHERE   P.speed >= L.speed
        ) ;
```

d)

```
SELECT  model
FROM
        (SELECT model,
                price
        FROM    PC

        UNION

        SELECT  model,
                price
        FROM    Laptop

        UNION

        SELECT  model,
                price
        FROM    Printer
        ) M1
WHERE   M1.price >= ALL
        (SELECT price
        FROM    PC

        UNION

        SELECT  price
        FROM    Laptop

        UNION

        SELECT  price
        FROM    Printer
        ) ;
```

(d) – contd --

```
SELECT  model
FROM
        (SELECT model,
                price
        FROM    PC

        UNION

        SELECT  model,
                price
        FROM    Laptop

        UNION

        SELECT  model,
                price
        FROM    Printer
        ) M1
WHERE   M1.price IN
        (SELECT MAX(price)
        FROM
                (SELECT price
                FROM    PC

                UNION

                SELECT  price
                FROM    Laptop

                UNION

                SELECT  price
                FROM    Printer
                ) M2
        ) ;
```

e)

```
SELECT  R.maker
FROM    Product R,
        Printer T
WHERE   R.model  =T.model
    AND T.price <= ALL
        (SELECT MIN(price)
        FROM    Printer
        );
SELECT  R.maker
FROM    Product R,
        Printer T1
WHERE   R.model   =T1.model
    AND T1.price IN
        (SELECT MIN(T2.price)
        FROM    Printer T2
        );
```

f)

```
SELECT  R1.maker
FROM    Product R1,
        PC P1
WHERE   R1.model=P1.model
    AND P1.ram IN
        (SELECT MIN(ram)
        FROM    PC
        )
    AND P1.speed >= ALL
        (SELECT P1.speed
        FROM    Product R1,
                PC P1
        WHERE   R1.model=P1.model
            AND P1.ram IN
                (SELECT MIN(ram)
                FROM    PC
                )
        );
SELECT  R1.maker
FROM    Product R1,
        PC P1
WHERE   R1.model=P1.model
    AND P1.ram  =
        (SELECT MIN(ram)
        FROM    PC
        )
    AND P1.speed IN
        (SELECT MAX(P1.speed)
        FROM    Product R1,
                PC P1
        WHERE   R1.model=P1.model
            AND P1.ram IN
                (SELECT MIN(ram)
                FROM    PC
                )
        );
```

6.3.2
a)

```
SELECT  C.country
FROM    Classes C
WHERE   numGuns IN
        (SELECT MAX(numGuns)
        FROM    Classes
        );
SELECT  C.country
FROM    Classes C
WHERE   numGuns >= ALL
        (SELECT numGuns
        FROM    Classes
        );
```

b)

```
SELECT DISTINCT C.class
FROM    Classes C,
        Ships S
WHERE   C.class = S.class
    AND EXISTS
        (SELECT ship
        FROM    Outcomes O
        WHERE   O.result='sunk'
            AND O.ship  = S.name
        ) ;
SELECT DISTINCT C.class
FROM    Classes C,
        Ships S
WHERE   C.class = S.class
    AND S.name IN
        (SELECT ship
        FROM    Outcomes O
        WHERE   O.result='sunk'
        ) ;
```

c)

```
SELECT  S.name
FROM    Ships S
WHERE   S.class IN
        (SELECT class
        FROM    Classes C
        WHERE   bore=16
        ) ;
SELECT  S.name
FROM    Ships S
WHERE   EXISTS
        (SELECT class
        FROM    Classes C
        WHERE   bore    =16
            AND C.class = S.class
        );
```

d)

```
SELECT  O.battle
FROM    Outcomes O
WHERE   O.ship IN
        (SELECT name
        FROM    Ships S
        WHERE   S.Class ='Kongo'
        );
SELECT  O.battle
FROM    Outcomes O
WHERE   EXISTS
        (SELECT name
        FROM    Ships S
        WHERE   S.Class ='Kongo'
            AND S.name  = O.ship
        );
```

e)

```
SELECT  S.name
FROM    Ships S,
        Classes C
WHERE   S.Class  = C.Class
    AND numGuns >= ALL
        (SELECT numGuns
        FROM    Ships S2,
                Classes C2
        WHERE   S2.Class = C2.Class
            AND C2.bore  = C.bore
        ) ;
SELECT  S.name
FROM    Ships S,
        Classes C
WHERE   S.Class  = C.Class
    AND numGuns IN
        (SELECT MAX(numGuns)
        FROM    Ships S2,
                Classes C2
        WHERE   S2.Class = C2.Class
            AND C2.bore  = C.bore
        ) ;
```

Better answer;

```
SELECT  S.name
FROM    Ships S,
        Classes C
WHERE   S.Class  = C.Class
    AND numGuns >= ALL
        (SELECT numGuns
        FROM    Classes C2
        WHERE   C2.bore = C.bore
        ) ;
SELECT  S.name
FROM    Ships S,
        Classes C
WHERE   S.Class  = C.Class
    AND numGuns IN
        (SELECT MAX(numGuns)
        FROM    Classes C2
        WHERE   C2.bore = C.bore
        ) ;
```

6.3.3

```
SELECT  title
FROM    Movies
GROUP BY title
HAVING COUNT(title) > 1 ;
```

6.3.4

```
SELECT  S.name
FROM    Ships S,
        Classes C
WHERE   S.Class = C.Class ;
```

Assumption: In R1 join R2, the rows of R2 are unique on the joining columns.

```
SELECT  COLL12,
        COLL13,
        COLL14
FROM    R1
WHERE   COLL12 IN
        (SELECT COL22
        FROM    R2
        )
    AND COLL13 IN
        (SELECT COL33
        FROM    R3
        )
    AND COLL14 IN
        (SELECT COL44
        FROM    R4
        ) ...
```

6.3.5

(a)

```
SELECT  S.name,
        S.address
FROM    MovieStar S,
        MovieExec E
WHERE   S.gender   ='F'
    AND E.netWorth > 10000000
    AND S.name     = E.name
    AND S.address  = E.address ;
```

Note: As mentioned previously in the book, the names of stars are unique. However no such restriction exists for executives. Thus, both name and address are required as join columns.

Alternate solution:

```
SELECT  name,
        address
FROM    MovieStar
WHERE   gender           = 'F'
    AND (name, address) IN
        (SELECT name,
                address
        FROM    MovieExec
        WHERE   netWorth > 10000000
        ) ;
```

(b)

```
SELECT  name,
        address
FROM    MovieStar
WHERE (name,address) NOT IN
        (SELECT name address
        FROM    MovieExec
        ) ;
```

6.3.6

By replacing the column in subquery with a constant and using IN subquery for the constant, statement equivalent to EXISTS can be found.

i.e. replace "WHERE EXISTS (SELECT C1 FROM R1..)" by "WHERE 1 IN (SELECT 1 FROM R1...)"

Example:

```
SELECT DISTINCT R.maker
FROM    Product R
WHERE   EXISTS
        (SELECT P.model
        FROM    PC P
        WHERE   P.speed >= 3.0
            AND P.model  =R.model
        ) ;
```

Above statement can be transformed to below statement.

```
SELECT DISTINCT R.maker
FROM    Product R
WHERE   1 IN
        (SELECT 1
        FROM    PC P
        WHERE   P.speed >= 3.0
            AND P.model  =R.model
        ) ;
```

6.3.7

(a)

n*m tuples are returned where there are n studios and m executives. Each studio will appear m times; once for every exec.

(b)

There are no common attributes between StarsIn and MovieStar; hence no tuples are returned.

(c)

There will be at least one tuple corresponding to each star in MovieStar. The unemployed stars will appear once with null values for StarsIn. All employed stars will appear as many times as the number of movies they are working in. In other words, for each tuple in StarsIn(starName), the correspoding tuple from MovieStar(name)) is joined and returned. For tuples in MovieStar that do not have a corresponding entry in StarsIn, the MovieStar tuple is returned with null values for StarsIn columns.

6.3.8

Since model numbers are unique, a full natural outer join of PC, Laptop and Printer will return one row for each model. We want all information about PCs, Laptops and Printers even if the model does not appear in Product but vice versa is not true. Thus a left natural outer join between Product and result above is required. The type attribute from Product must be renamed since Printer has a type attribute as well and the two attributes are different.

```
(SELECT maker,
        model,
        type AS productType
FROM    Product
) RIGHT NATURAL OUTER JOIN ((PC FULL NATURAL OUTER JOIN Laptop) FULL NATURAL
OUTER JOIN Printer);
```

Alternately, the Product relation can be joined individually with each of PC,Laptop and Printer and the three results can be Unioned together. For attributes that do not exist in one relation, a constant such as 'NA' or 0.0 can be used. Below is an example of this approach using PC and Laptop.

```
SELECT  R.MAKER      ,
        R.MODEL      ,
        R.TYPE       ,
        P.SPEED      ,
        P.RAM        ,
        P.HD         ,
        0.0 AS SCREEN,
        P.PRICE
FROM    PRODUCT R,
        PC P
WHERE   R.MODEL = P.MODEL

UNION

SELECT  R.MAKER ,
        R.MODEL ,
        R.TYPE  ,
        L.SPEED ,
        L.RAM   ,
        L.HD    ,
        L.SCREEN,
        L.PRICE
FROM    PRODUCT R,
        LAPTOP L
WHERE   R.MODEL = L.MODEL;
```

6.3.9

```
SELECT  *
FROM    Classes RIGHT NATURAL
        OUTER JOIN Ships ;
```

6.3.10

```
SELECT  *
FROM    Classes RIGHT NATURAL
        OUTER JOIN Ships

UNION
        (SELECT C2.class       ,
                C2.type        ,
                C2.country     ,
                C2.numguns     ,
                C2.bore        ,
                C2.displacement,
                C2.class NAME  ,
                0
        FROM    Classes C2,
                Ships S2
        WHERE   C2.Class NOT IN
                (SELECT Class
                FROM    Ships
                )
        ) ;
```

6.3.11

(a)

```
SELECT  *
FROM    R,
        S ;
```

(b)

Let Attr consist of
AttrR = attributes unique to R
AttrS = attributes unique to S
AttrU = attributes common to R and S
Thus in Attr, attributes common to R and S are not repeated.

```
SELECT  Attr
FROM    R,
        S
WHERE   R.AttrU1 = S.AttrU1
    AND R.AttrU2 = S.AttrU2 ...
    AND R.AttrUi = S.AttrUi ;
```

(c)

```
SELECT  *
FROM    R,
        S
WHERE   C ;
```

6.4.1
(a)
DISTINCT keyword is not required here since each model only occurs once in PC relation.

```
SELECT  model
FROM    PC
WHERE   speed >= 3.0 ;
```

(b)

```
SELECT DISTINCT R.maker
FROM    Product R,
        Laptop L
WHERE   R.model = L.model
    AND L.hd    > 100 ;
```

(c)

```
SELECT  R.model,
        P.price
FROM    Product R,
        PC P
WHERE   R.model = P.model
    AND R.maker = 'B'

UNION

SELECT  R.model,
        L.price
FROM    Product R,
        Laptop L
WHERE   R.model = L.model
    AND R.maker = 'B'

UNION

SELECT  R.model,
        T.price
FROM    Product R,
        Printer T
WHERE   R.model = T.model
    AND R.maker = 'B' ;
```

(d)
```
SELECT  model
FROM    Printer
WHERE   color=TRUE
    AND type ='laser' ;
```

(e)
```
SELECT DISTINCT R.maker
FROM    Product R,
        Laptop L
WHERE   R.model      = L.model
    AND R.maker NOT IN
        (SELECT R1.maker
        FROM    Product R1,
                PC P
        WHERE   R1.model = P.model
        ) ;
```

better:
```
SELECT DISTINCT R.maker
FROM    Product R
WHERE   R.type       = 'laptop'
    AND R.maker NOT IN
        (SELECT R.maker
        FROM    Product R
        WHERE   R.type = 'pc'
        ) ;
```

(f)
With GROUP BY hd, DISTINCT keyword is not required.

```
SELECT  hd
FROM    PC
GROUP BY hd
HAVING COUNT(hd) > 1 ;
```

(g)
```
SELECT  P1.model,
        P2.model
FROM    PC P1,
        PC P2
WHERE   P1.speed = P2.speed
    AND P1.ram   = P2.ram
    AND P1.model < P2.model ;
```

(h)

```
SELECT  R.maker
FROM    Product R
WHERE   R.model IN
        (SELECT P.model
        FROM    PC P
        WHERE   P.speed >= 2.8
        )
     OR R.model IN
        (SELECT L.model
        FROM    Laptop L
        WHERE   L.speed >= 2.8
        )
GROUP BY R.maker
HAVING COUNT(R.model) > 1 ;
```

(i)
After finding the maximum speed, an IN subquery can provide the manufacturer name.

```
SELECT  MAX(M.speed)
FROM
        (SELECT speed
        FROM    PC

        UNION

        SELECT  speed
        FROM    Laptop
        ) M ;

SELECT  R.maker
FROM    Product R,
        PC P
WHERE   R.model  = P.model
    AND P.speed IN
        (SELECT MAX(M.speed)
        FROM
                (SELECT speed
                FROM    PC

                UNION

                SELECT  speed
                FROM    Laptop
                ) M
        )

UNION

SELECT  R2.maker
FROM    Product R2,
        Laptop L
WHERE   R2.model = L.model
    AND L.speed IN
        (SELECT MAX(N.speed)
```

```
        FROM
                (SELECT speed
                FROM    PC

                UNION

                SELECT  speed
                FROM    Laptop
                ) N
        ) ;
```

Alternately,

```
SELECT  COALESCE(MAX(P2.speed),MAX(L2.speed),0) SPEED
FROM    PC P2
        FULL OUTER JOIN Laptop L2
        ON      P2.speed = L2.speed ;
SELECT  R.maker
FROM    Product R,
        PC P
WHERE   R.model  = P.model
    AND P.speed IN
        (SELECT COALESCE(MAX(P2.speed),MAX(L2.speed),0) SPEED
        FROM    PC P2
                FULL OUTER JOIN Laptop L2
                ON      P2.speed = L2.speed
        )

UNION

SELECT  R2.maker
FROM    Product R2,
        Laptop L
WHERE   R2.model = L.model
    AND L.speed IN
        (SELECT COALESCE(MAX(P2.speed),MAX(L2.speed),0) SPEED
        FROM    PC P2
                FULL OUTER JOIN Laptop L2
                ON      P2.speed = L2.speed
        )
```

(j)

```
SELECT  R.maker
FROM    Product R,
        PC P
WHERE   R.model = P.model
GROUP BY R.maker
HAVING COUNT(DISTINCT speed) >= 3 ;
```

(k)

```
SELECT  R.maker
FROM    Product R,
        PC P
WHERE   R.model = P.model
GROUP BY R.maker
HAVING COUNT(R.model) = 3 ;
```

better;

```
SELECT  R.maker
FROM    Product R
WHERE   R.type='pc'
GROUP BY R.maker
HAVING COUNT(R.model) = 3 ;
```

6.4.2

(a)

We can assume that class is unique in Classes and DISTINCT keyword is not required.

```
SELECT  class,
        country
FROM    Classes
WHERE   bore >= 16 ;
```

(b)

Ship names are not unique (In absence of hull codes, year of launch can help distinguish ships).

```
SELECT DISTINCT name AS Ship_Name
FROM    Ships
WHERE   launched < 1921 ;
```

(c)

```
SELECT DISTINCT ship AS Ship_Name
FROM    Outcomes
WHERE   battle = 'Denmark Strait'
    AND result = 'sunk' ;
```

(d)

```
SELECT DISTINCT S.name AS Ship_Name
FROM    Ships S,
        Classes C
WHERE   S.class        = C.class
    AND C.displacement > 35000 ;
```

(e)

```
SELECT DISTINCT O.ship AS Ship_Name,
       C.displacement            ,
       C.numGuns
FROM   Classes C ,
       Outcomes O,
       Ships S
WHERE  C.class  = S.class
   AND S.name   = O.ship
   AND O.battle = 'Guadalcanal' ;

SHIP_NAME          DISPLACEMENT NUMGUNS
------------------ ------------ -------
Kirishima                 32000       8
Washington                37000       9

  2 record(s) selected.
```

Note: South Dakota was also in Guadalcanal but its class information is not available. Below query will return name of all ships that were in Guadalcanal even if no other information is available (shown as NULL). The above query is modified from INNER joins to LEFT OUTER joins.

```
SELECT DISTINCT O.ship AS Ship_Name,
       C.displacement            ,
       C.numGuns
FROM   Outcomes O
       LEFT JOIN Ships S
       ON      S.name = O.ship
       LEFT JOIN Classes C
       ON      C.class = S.class
WHERE  O.battle        = 'Guadalcanal' ;


SHIP_NAME          DISPLACEMENT NUMGUNS
------------------ ------------ -------
Kirishima                 32000       8
South Dakota                  -       -
Washington                37000       9

  3 record(s) selected.
```

(f)

The Set opearator UNION guarantees unique results.

```
SELECT  ship AS Ship_Name
FROM    Outcomes

UNION

SELECT  name AS Ship_Name
FROM    Ships ;
```

(g)
```
SELECT  C.class
FROM    Classes C,
        Ships S
WHERE   C.class = S.class
GROUP BY C.class
HAVING COUNT(S.name) = 1 ;
```

better:
```
SELECT  S.class
FROM    Ships S
GROUP BY S.class
HAVING COUNT(S.name) = 1 ;
```

(h)
The Set opearator INTERSECT guarantees unique results.
```
SELECT  C.country
FROM    Classes C
WHERE   C.type='bb'

INTERSECT

SELECT  C2.country
FROM    Classes C2
WHERE   C2.type='bc' ;
```

However, above query does not account for classes without any ships belonging to them.

```
SELECT  C.country
FROM    Classes C,
        Ships S
WHERE   C.class = S.class
    AND C.type  ='bb'

INTERSECT

SELECT  C2.country
FROM    Classes C2,
        Ships S2
WHERE   C2.class = S2.class
    AND C2.type  ='bc' ;
```

(i)

```
SELECT  O2.ship AS Ship_Name
FROM    Outcomes O2,
        Battles B2
WHERE   O2.battle = B2.name
    AND B2.date   > ANY
        (SELECT B.date
        FROM    Outcomes O,
                Battles B
        WHERE   O.battle = B.name
            AND O.result ='damaged'
            AND O.ship   = O2.ship
        );
```

6.4.3
a)

```
SELECT DISTINCT R.maker
FROM    Product R,
        PC P
WHERE   R.model  = P.model
    AND P.speed >= 3.0;
```

b)
Models are unique.

```
SELECT  P1.model
FROM    Printer P1
        LEFT OUTER JOIN Printer P2
        ON (P1.price < P2.price)
WHERE   P2.model    IS NULL ;
```

c)

```
SELECT DISTINCT L.model
FROM    Laptop L,
        PC P
WHERE   L.speed < P.speed ;
```

d)
Due to set operator UNION, unique results are returned.
It is difficult to completely avoid a subquery here. One option is to use Views.

```
CREATE VIEW AllProduct AS
SELECT  model,
        price
FROM    PC

UNION

SELECT  model,
        price
FROM    Laptop

UNION

SELECT  model,
        price
FROM    Printer ;
SELECT  A1.model
FROM    AllProduct A1
        LEFT OUTER JOIN AllProduct A2
        ON (A1.price < A2.price)
WHERE   A2.model    IS NULL ;
```

But if we replace the View, the query contains a FROM subquery.

```
SELECT  A1.model
FROM
        (SELECT model,
                price
        FROM    PC

        UNION

        SELECT  model,
                price
        FROM    Laptop

        UNION

        SELECT  model,
                price
        FROM    Printer
        ) A1
        LEFT OUTER JOIN
                (SELECT model,
                        price
                FROM    PC

                UNION

                SELECT  model,
                        price
                FROM    Laptop
```

```
                UNION

                SELECT  model,
                        price
                FROM    Printer
                ) A2
        ON (A1.price < A2.price)
WHERE   A2.model    IS NULL ;
```

e)

```
SELECT DISTINCT R.maker
FROM    Product R,
        Printer T
WHERE   R.model  =T.model
    AND T.price <= ALL
        (SELECT MIN(price)
        FROM    Printer
        );
```

f)

```
SELECT DISTINCT R1.maker
FROM    Product R1,
        PC P1
WHERE   R1.model=P1.model
    AND P1.ram IN
        (SELECT MIN(ram)
        FROM    PC
        )
    AND P1.speed >= ALL
        (SELECT P1.speed
        FROM    Product R1,
                PC P1
        WHERE   R1.model=P1.model
            AND P1.ram IN
                (SELECT MIN(ram)
                FROM    PC
                )
        );
```

6.4.4

a)

```
SELECT DISTINCT C1.country
FROM    Classes C1
        LEFT OUTER JOIN Classes C2
        ON (C1.numGuns < C2.numGuns)
WHERE   C2.country    IS NULL ;
```

b)

```
SELECT DISTINCT C.class
FROM    Classes C,
        Ships S  ,
        Outcomes O
WHERE   C.class = S.class
    AND S.name  = O.ship
    AND O.result='sunk' ;
```

c)

```
SELECT  S.name
FROM    Ships S,
        Classes C
WHERE   C.class = S.class
    AND C.bore  =16 ;
```

d)

```
SELECT  O.battle
FROM    Outcomes O,
        Ships S
WHERE   S.Class ='Kongo'
    AND S.name  = O.ship ;
```

e)

```
SELECT  S.name
FROM    Classes C1
        LEFT OUTER JOIN Classes C2
        ON (C1.bore        = C2.bore
            AND C1.numGuns < C2.numGuns)
        INNER JOIN Ships S
        ON      C1.class = S.class
WHERE   C2.class        IS NULL ;
```

6.4.5
Yes, duplicates are possible. If a person produced more than one movie of Harrison Ford's, the temporary relation Prod will contain duplicates. The join of Prod and MovieExec will also repeat the name.

6.4.6
(a)

```
SELECT  AVG(speed) AS Avg_Speed
FROM    PC ;
```

```
AVG_SPEED
---------------------------------
      2.4846153846153846153846153

  1 record(s) selected.
```

(b)

```
SELECT  AVG(speed) AS Avg_Speed
FROM    Laptop
WHERE   price > 1000 ;

AVG_SPEED
---------------------------------
      1.99833333333333333333333333

  1 record(s) selected.
```

(c)

```
SELECT  AVG(P.price) AS Avg_Price
FROM    Product R,
        PC P
WHERE   R.model=P.model
    AND R.maker='A' ;

AVG_PRICE
-----------
       1195

  1 record(s) selected.
```

(d)

```
SELECT  AVG(M.price) AS Avg_Price
FROM
        (SELECT P.price
        FROM    Product R,
                PC P
        WHERE   R.model = P.model
            AND R.maker = 'D'

        UNION ALL

        SELECT  L.price
        FROM    Product R,
                Laptop L
        WHERE   R.model = L.model
            AND R.maker = 'D'
        ) M ;

AVG_PRICE
-----------
        730

  1 record(s) selected.
```

(e)

```
SELECT  SPEED,
        AVG(price) AS AVG_PRICE
FROM    PC
GROUP BY speed ;


SPEED      AVG_PRICE
---------- -----------
      1.42         478
      1.86         959
      2.00         650
      2.10         995
      2.20         640
      2.66        2114
      2.80         689
      3.06         529
      3.20         839

  9 record(s) selected.
```

(f)

```
SELECT  R.maker,
        AVG(L.screen) AS Avg_Screen_Size
FROM    Product R,
        Laptop L
WHERE   R.model = L.model
GROUP BY R.maker ;




MAKER AVG_SCREEN_SIZE
----- ---------------------------------
A        15.2333333333333333333333333
B        13.3000000000000000000000000
E        17.5000000000000000000000000
F        14.7500000000000000000000000
G        15.4000000000000000000000000

  5 record(s) selected.
```

(g)
```
SELECT  R.maker
FROM    Product R,
        PC P
WHERE   R.model = P.model
GROUP BY R.maker
HAVING COUNT(R.model) >=3 ;
```

better:
```
SELECT  maker
FROM    Product
WHERE   type='pc'
GROUP BY maker
HAVING COUNT(model) >=3 ;
```

```
MAKER
-----
A
B
D
E

  4 record(s) selected.
```

(h)
```
SELECT  R.maker,
        MAX(P.price) AS Max_Price
FROM    Product R,
        PC P
WHERE   R.model = P.model
GROUP BY R.maker ;
```

```
MAKER MAX_PRICE
----- ---------
A          2114
B          1049
C           510
D           770
E           959

  5 record(s) selected.
```

(i)
```
SELECT  speed,
        AVG(price) AS Avg_Price
FROM    PC
WHERE   speed > 2.0
GROUP BY speed ;

SPEED      AVG_PRICE
---------- -----------
      2.10         995
      2.20         640
      2.66        2114
      2.80         689
      3.06         529
      3.20         839

  6 record(s) selected.
```

(j)
```
SELECT  AVG(P.hd) AS Avg_HD_Size
FROM    Product R,
        PC P
WHERE   R.model  = P.model
    AND R.maker IN
        (SELECT maker
        FROM    Product
        WHERE   type = 'printer'
        ) ;

AVG_HD_SIZE
-----------
        200

  1 record(s) selected.
```

6.4.7
(a)
```
SELECT  COUNT(C.type) AS NO_Classes
FROM    Classes
WHERE   type ='bb' ;

NO_CLASSES
-----------
          6

  1 record(s) selected.
```

(b)
```
SELECT  AVG(C.numGuns) AS Avg_Guns
FROM    Classes
WHERE   type ='bb' ;

AVG_GUNS
-----------
          9

  1 record(s) selected.
```

(c)
We weight by the number of ships and the answer could be different.

```
SELECT  AVG(C.numGuns) AS Avg_Guns
FROM    Classes C
        INNER JOIN Ships S
        ON (C.class = S.class)
WHERE   C.type      ='bb';


AVG_GUNS
-----------
          9

  1 record(s) selected.
```

(d)
Even though the book mentions that the first ship has the same name as class, we can also calculate answer differently.

```
SELECT  C.class,
        MIN(S.launched) AS First_Launched
FROM    Classes C,
        Ships S
WHERE   C.class = S.class
GROUP BY C.class ;

CLASS              FIRST_LAUNCHED
------------------ --------------
Iowa                         1943
Kongo                        1913
North Carolina               1941
Renown                       1916
Revenge                      1916
Tennessee                    1920
Yamato                       1941

  7 record(s) selected.
```

(e)

```
SELECT  C.class,
        COUNT(O.ship) AS No_Sunk
FROM    Classes C ,
        Outcomes O,
        Ships S
WHERE   C.class  = S.class
    AND S.name   = O.ship
    AND O.result = 'sunk'
GROUP BY C.Class ;

CLASS              NO_SUNK
------------------ -----------
Kongo                        1

  1 record(s) selected.
```

(f)

```
SELECT  M.class,
        COUNT(O.ship) AS No_Sunk
FROM    Outcomes O,
        Ships S   ,
        (SELECT C.class
        FROM    Classes C,
                Ships S
        WHERE   C.class = S.class
        GROUP BY C.class
        HAVING COUNT(S.name) >= 3
        ) M
WHERE   O.result = 'sunk'
    AND O.ship   = S.name
    AND S.class  = M.class
GROUP BY M.class ;

CLASS              NO_SUNK
------------------ -----------
Kongo                        1

  1 record(s) selected.
```

(g)

```
SELECT  C.country,
        AVG(C.bore*C.bore*C.bore*0.5) Avg_Shell_Wt
FROM    Classes C,
        Ships S
WHERE   C.class = S.class
GROUP BY C.country ;

COUNTRY      AVG_SHELL_WT
------------ ---------------------------------
Gt. Britain           1687.5000000000000000000
Japan                 1886.6666666666666666666
USA                   1879.0000000000000000000

  3 record(s) selected.
```

6.4.8

```
SELECT  starName,
        MIN(YEAR) AS minYear
FROM    StarsIn
GROUP BY starName
HAVING COUNT(title) >= 3 ;
```

6.4.9

Yes, it is possible. We can include in gamma operator the aggregation for HAVING condition (including renaming it). Then the sigma operator can be used to apply the HAVING condition using the renamed attribute. The pi operator can be used to filter out the renamed attribute from query result.

6.5.1

(a)

```
INSERT
INTO    Product VALUES
        (
                'C'   ,
                '1100',
                'pc'
        ) ;
INSERT
INTO    PC VALUES
        (
                '1100',
                3.2   ,
                1024,180,2499
        ) ;
```

(b)

```
INSERT
INTO    Product
SELECT  make      ,
        model+1100,
        'laptop'
FROM    Product
WHERE   type = 'pc' ;
INSERT
INTO    Laptop
SELECT  model+1100,
        speed     ,
        ram       ,
        hd        ,
        17        ,
        price+500
FROM    PC ;
```

Or if model is character data type

```
INSERT
INTO    Product
SELECT  make                 ,
        CHAR(INT(model)+1100),
        'laptop'
FROM    Product
WHERE   type = 'pc' ;
INSERT
INTO    Laptop
SELECT  CHAR(INT(model)+1100),
        speed                ,
        ram                  ,
        hd                   ,
        17                   ,
        price+500
FROM    PC ;
```

(c)

```
DELETE
FROM    PC
WHERE   hd < 100 ;
```

(d)

```
DELETE
FROM    Laptop L
WHERE   L.model IN
        (SELECT R2.model
        FROM    Product R2
        WHERE   R2.maker IN
                (SELECT DISTINCT R.maker
                FROM    Product R
                WHERE   R.maker NOT IN
                        (SELECT R2.maker
                        FROM    Product R2
                        WHERE   R2.type = 'printer'
                        )
                )
        ) ;
DELETE
FROM    PRODUCT R3
WHERE   R3.model IN
        (SELECT R2.model
        FROM    Product R2
        WHERE   R2.maker IN
                (SELECT DISTINCT R.maker
                FROM    Product R
                WHERE   R.maker NOT IN
                        (SELECT R2.maker
                        FROM    Product R2
                        WHERE   R2.type = 'printer'
                        )
                )
        )
    AND R3.type = 'laptop';
```

(e)

```
UPDATE Product
SET     maker = 'A'
WHERE   maker = 'B' ;
```

(f)

```
UPDATE PC
SET     ram = ram*2,
        hd  =hd  +60 ;
```

(g)

```
UPDATE Laptop L
SET     L.screen = L.screen+1,
        L.price  =L.price  -100
WHERE   L.model           IN
        (SELECT R.model
        FROM    Product R
        WHERE   R.maker = 'B'
        ) ;
```

6.5.2

(a)

```
INSERT
INTO    Classes VALUES
        (
                'Nelson'     ,
                'bb'         ,
                'Gt. Britain',
                9,16,34000
        ) ;
INSERT
INTO    Ships VALUES
        (
                'Nelson',
                'Nelson',
                1927
        );
INSERT
INTO    Ships VALUES
        (
                'Rodney',
                'Nelson',
                1927
        );
```

(b)

```
INSERT
INTO    Classes VALUES
        (
                'Vittorio Veneto',
                'bb'             ,
                'Italy'          ,
                9,15,41000
        ) ;
INSERT
INTO    Ships VALUES
        (
                'Vittorio Veneto',
                'Vittorio Veneto',
                1940
        );
INSERT
INTO    Ships VALUES
        (
                'Italia'         ,
                'Vittorio Veneto',
                1940
        );
INSERT
INTO    Ships VALUES
        (
                'Roma'           ,
                'Vittorio Veneto',
                1940
        );
```

(c)

```
DELETE
FROM    Ships S
WHERE   S.name IN
        (SELECT ship
        FROM    Outcomes
        WHERE   result='sunk'
        ) ;
```

(d)

```
UPDATE Classes
SET     bore        =2.5         *bore,
        displacement=displacement/1.1 ;
```

(e)

```
DELETE
FROM    Classes C
WHERE   C.class IN
        (SELECT C2.class
        FROM    Classes C2,
                Ships S
        WHERE   C2.class = S.Class
        GROUP BY C2.class
        HAVING COUNT(C2.class) < 3
        ) ;
```

6.6.1
(a)

```
EXEC SQL BEGIN DECLARE SECTION;
    int modelNo;
    int pcPrice;
    int pcRAM;
    float pcSpeed;
EXEC SQL END DECLARE SECTION;

void lookupPC(int iSpeed,int fRAM) {
     EXEC SQL SET TRANSACTION READ ONLY ISOLATION READ COMMITTED;
     EXEC SQL DECLARE pcCursor CURSOR FOR
              SELECT model,price
              FROM PC
              WHERE speed=:pcSpeed
                AND ram=:pcRAM;

     pcSpeed = iSpeed;
     pcRAM = fRAM;

     EXEC SQL OPEN pcCursor;

     EXEC SQL FETCH pcCursor
              INTO :modelNo, :pcPrice;
     while (SQLCODE == 0)
     {
        printf("Model No: %d   Price: %d", modelNo, pcPrice );
        EXEC SQL FETCH pcCursor
                 INTO :modelNo, :pcPrice;

     }
     EXEC SQL CLOSE pcCursor;
     EXEC SQL COMMIT;
}
```

This is a READ ONLY transaction and READ COMMITTED provides the optimum ISOLATION LEVEL for concurrency while not allowing dirty reads.

(b)

```
EXEC SQL BEGIN DECLARE SECTION;
    int modelNo;
EXEC SQL END DECLARE SECTION;

void deleteModel(int iModel) {
     EXEC SQL SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
     modelNo  = iModel;

     EXEC SQL DELETE FROM Product
              WHERE model = :modelNo;

     EXEC SQL DELETE FROM PC
              WHERE model = :modelNo;

     EXEC SQL COMMIT;
}
```

The ISOLATION LEVEL is set to SERIALIZABLE but it could be anything since there is no risk of dirty read (no select statement).

(c)

```
EXEC SQL BEGIN DECLARE SECTION;
    int modelNo;
EXEC SQL END DECLARE SECTION;

void updatePCPrice(int iModel) {
     EXEC SQL SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
     modelNo  = iModel;

     EXEC SQL UPDATE PC
              SET price = price - 100
              WHERE model = :modelNo;


     EXEC SQL COMMIT;
}
```

For reason same as in (b) above, the isolation level is set to SERIALIZABLE.

(d)

```
EXEC SQL BEGIN DECLARE SECTION;
    char maker[1];
    int exists = 0;
    int modelNo;
    int pcPrice;
    int pcRAM;
    int pcHDD;
    float pcSpeed;
EXEC SQL END DECLARE SECTION;

void insertPC(char cMaker[1],int iModel,int iSpeed,float fRAM,int iHDD,
              int iPrice) {
     EXEC SQL SET TRANSACTION ISOLATION READ COMMITTED;
     EXEC SQL DECLARE newCursor CURSOR FOR
              SELECT  1
              FROM    Product R
              WHERE   R.model=:modelNo;

     maker = cMaker;
     modelNo  = iModel;
     pcSpeed = iSpeed;
     pcRAM = fRAM;
     pcHDD  = iHDD;
     pcPrice   = iPrice;

     EXEC SQL OPEN newCursor;

     EXEC SQL FETCH newCursor
              INTO :exists;
     if (exists == 1)
     {
        printf("ERROR:Model No: %d   already exists in database", modelNo);

     }
     else /*  Add model into database  */
     {
        EXEC SQL INSERT INTO Product
                 VALUES(:maker,:modelNo,'pc') ;
        EXEC SQL INSERT INTO PC
                 VALUES(:modelNo,:pcSpeed,:pcRAM,:pcHDD,:pcPrice) ;

     }
     EXEC SQL CLOSE newCursor;
     EXEC SQL COMMIT;
}
```

6.6.2

(a) It is a READ ONLY transaction. Thus there is no write or update atomicity problem. However, a system crash can cause truncated result and application may need to rerun on system restart.

(b) If the system crash occurs after the model was deleted from Product but before deletion from PC, an atomicity problem occurs. Databases keep a log of activities and use the log with some kind of recovery strategy to bring the database to a consistent state on system restart.

(c) There is no atomicity problem here since there is only one sql statement and each sql statement is atomic by nature. However, the application may need to call updatePCPrice again if the system crashed before update completed.

(d) Similar to (b). If system crashed between inserts, atomicity problem occurs and database is left in inconsistent state.

6.6.3

(a)

T is the READ ONLY transaction from 6.6.1 (a). Another READ ONLY transaction can run concurrently without any difference (i.e. As if all transactions ran in SERIALIZABLE isolation).

If deleteModel from 6.6.1 (b) was running concurrently with T, T may not return a PC model which had been deleted from Product and then deleteModel rolled back. With SERIALIZABLE isolation, T would return the PC model unless the delete transaction committed.

If updatePCPrice from 6.6.1 (c) was running concurrently with T, the reduced PC price(dirty read) could be returned by T even if updatePCPrice later rolled back. Similarly, T could return the inserted PC model by insertPC (phantom read) even if insertPC later rolled back.

(b)

T is the deleteModel from 6.6.1 (b). If running insertPC concurrently with T, insertPC checked that the model does not exist since T just deleted the model, but then T rolled back. Thus insertPC attempts to insert a model that already exists.

(c)

T is updatePCPrice from 6.6.1 (c). When running concurrently with another updatePCPrice for same model, T could read the updated price (dirty data) and decrement model price by $100. But then first updatePCPrice rolled back. However, the pc price for the model was reduced by $200 though only one updatePCPrice completed.

(d)

T is insertPC from 6.6.1 (d).

When running concurrently with another insertPC, both could check that there is no product with the model, and then try to insert the model.

6.6.4

Serializable: T will never see changes to the database and keep printing the same list of PCs. This does not serve any useful purpose. Application may need to periodically stop T and then restart it to see data committed in the meantime.

Repeatable Read: T will continue to see the list of PCs it saw once. However, T will also see any new PCs that are inserted in the database. Locking issues can occur if another transaction such as 6.6.1 (b) or (c) tries to update/delete the rows read by T. 6.6.1 (d) inserts a new row and thus can run concurrently with T.

Read Committed: Perhaps the best option. T can see new or updated rows after other transactions such as 6.6.1 (c) or (d) commit. However, if T reads the same table twice, the results are not consistent because some rows may have been updated (6.6.1 (c) or deleted(6.6.1 (b)) by other transaction. Moreover, if T reads a row and based on the result then tries to read/update/delete the row; the state of row may have changed in the meantime.

Read Uncommitted: T will not cause any locking (high concurrency) but uncommitted PC data might be printed out due to insert/update by other transaction e.g. 6.6.1 (c) or (d). However, the other transaction might rollback resulting in wrong reports.

Solutions
Chapter 7

7.1.1

a)

```
CREATE TABLE Movies (
title           CHAR(100),
year            INT,
length          INT,
genre           CHAR(10),
studioName      CHAR(30),
producerC#      INT,
PRIMARY KEY (title, year),
FOREIGN KEY (producerC#)  REFERENCES MovieExec(cert#)
);
```

or

```
CREATE TABLE Movies (
title           CHAR(100),
year            INT,
length          INT,
genre           CHAR(10),
studioName      CHAR(30),
producerC#      INT         REFERENCES MovieExec(cert#),
PRIMARY KEY (title, year)
);
```

b)

```
CREATE TABLE Movies (
title           CHAR(100),
year            INT,
length          INT,
genre           CHAR(10),
studioName      CHAR(30),
producerC#      INT         REFERENCES MovieExec(cert#)
ON DELETE SET NULL
ON UPDATE SET NULL,
PRIMARY KEY (title, year)
);
```

c)

```
CREATE TABLE Movies (
title           CHAR(100),
year            INT,
length          INT,
```

```
genre                CHAR(10),
studioName           CHAR(30),
producerC#           INT          REFERENCES MovieExec(cert#)
ON DELETE CASCADE
ON UPDATE CASCADE,
PRIMARY KEY (title, year)
);
```

d)

```
CREATE TABLE StarsIn (
movieTitle           CHAR(100)   REFERENCES  Movie(title),
movieYear            INT,
starName             CHAR(30),
PRIMARY KEY (movieTItle, movieYear, starName)
);
```

e)

```
CREATE TABLE StarsIn (
movieTitle           CHAR(100)   REFERENCES  Movie(title)
          ON DELETE CASCADE,
movieYear            INT,
starName             CHAR(30),
PRIMARY KEY (movieTItle, movieYear, starName)
);
```

7.1.2

To declare such a foreign-key constraint between the relations Movie and StarsIn, values of the referencing attributes in Movie should appear in MovieStar as unique values. However, based on primary key declaration in relation StarIn, the uniqueness of movies is guaranteed with movieTitle, movieYear, and starName attributes. Even with title and year as referencing attributes there is no way of referencing unique movie from StarsIn without starName information. Therefore, such a constraint can not be expressed using a foreign-key constraint.

7.1.3

```
ALTER TABLE Product
          ADD PRIMARY KEY (model);

ALTER TABLE PC
          ADD FOREIGN KEY (model)  REFERENCES  Product (model);


ALTER TABLE Laptop
          ADD FOREIGN KEY (model)  REFERENCES  Product(model);


ALTER TABLE Printer
          ADD FOREIGN KEY (model)  REFERENCES  Product (model);
```

7.1.4

```
ALTER TABLE Classes
        ADD PRIMARY KEY (class);

ALTER TABLE Ships
        ADD PRIMARY KEY (name);

ALTER TABLE Ships
        ADD FOREIGN KEY (class)  REFERENCES Classes (calss);

ALTER TABLE Battles
        ADD PRIMARY KEY (name);

ALTER TABLE Outcomes
        ADD FOREIGN KEY (ship) REFERENCES Ships (name);

ALTER TABLE Outcomes
        ADD FOREIGN KEY (battle) REFERENCES Battles (name);
```

7.1.5

a)

```
ALTER TABLE Ships
        ADD FOREIGN KEY (class) REFERENCES Classes (class)
                                ON DELETE SET NULL
                                ON UPDATE SET NULL;
```

In addition to the above declaration, class must be declared the primary key for Classes.

b)

```
ALTER TABLE Outcome
        ADD FOREIGN KEY (battle) REFERENCES Battles (name)
                                 ON DELETE SET NULL
                                 ON UPDATE SET NULL;
```

c)

```
ALTER TABLE Outcomes
        ADD FOREIGN KEY (ship) REFERENCES Ships (name)
                               ON DELETE SET NULL
                               ON UPDATE SET NULL;
```

7.2.1

a)

```
year            INT       CHECK (year >= 1915)
```

b)

```
length          INT       CHECK (length >= 60 AND length <= 250)
```

c)

```
studioName      CHAR(30)
  CHECK (studioName IN ( , Disney ?, Fox ?, , MGM?, , Paramount ?) )
```

7.2.2

a)

```
CREATE TABLE Laptop (
   …
  speed           DECIMAL(4,2)      CHECK (speed >= 2.0)
   …
);
```

b)

```
CREATE TABLE Printer (
   …
  type            VARCHAR(10)
    CHECK (type IN ( ,laser? , , ink-jet ?, , bubble-jet ?))
   …
);
```

c)

```
CREATE TABLE Product (
   …
  type            VARCHAR(10)
    CHECK (type IN( ,pc? , , laptop ?, , printer ?))
   …
);
```

d)

```
CREATE TABLE Product (
   …
  model           CHAR(4)
    CHECK (model IN (SELECT model FROM PC
                                          UNION ALL
                     SELECT model FROM laptop
                                          UNION ALL
                     SELECT model FROM printer))
   …
);
```

* note this doesn ?t check the attribute constraint violation caused by deletions from PC, laptop, or printer

7.2.3

a)

```
CREATE TABLE StarsIn (
    …
  starName    CHAR(30)
      CHECK (starName IN (SELECT name FROM MovieStar
                                  WHERE YEAR(birthdate) > movieYear))
    …
);
```

b)

```
CREATE TABLE Studio (
    …
  address     CHAR(255)                CHECK (address IS UNIQUE)
    …
);
```

c)

```
CREATE TABLE MovieStar (
    …
  name     CHAR(30)          CHECK (name NOT IN (SELECT name FROM MovieExec))
    …
);
```

d)

```
CREATE TABLE Studio (
    …
    Name    CHAR(30)         CHECK (name IN (SELECT studioName FROM Movies))
    …
);
```

e)

```
CREATE TABLE Movies (
    …
    CHECK (producerC# NOT IN (SELECT presC# FROM Studio) OR
              studioName IN (SELECT name FROM Studio
                                  WHERE presC# = producerC#))
    …
);
```

7.2.4

a)

```
CHECK (speed >= 2.0 OR price <= 600)
```

b)

```
CHECK (screen >= 15 OR hd >= 40 OR price <= 1000)
```

7.2.5

a)

```
CHECK (class NOT IN (SELECT class FROM Classes
WHERE bore > 16))
```

b)

```
CHECK (class NOT IN (SELECT class  FROM Classes
                         WHERE numGuns > 9 AND bore > 14))
```

c)

```
CHECK (ship IN (SELECT s.name FROM Ships s, Battles b, Outcomes o
                                WHERE s.name = o.ship AND
                b.name = o.battle AND
                s.launched > YEAR(b.date)))
```

7.2.6

The constraint in Example 7.6 does not allow NULL value for gender while the constraint in Example 7.8 allows NULL.

7.3.1

a)
```
ALTER TABLE Movie ADD CONSTRAINT        myKey
        PRIMARY KEY (title, year);
```

b)
```
ALTER TABLE Movie ADD CONSTRAINT        producerCheck
        FOREIGN KEY (producerC#) REFERENCES MovieExec (cert#);
```

c)
```
ALTER TABLE Movie ADD CONSTRAINT        lengthCheck
        CHECK (length >= 60 AND length <= 250);
```

d)
```
ALTER TABLE MovieStar ADD CONSTRAINT noDupInExec
    CHECK (name NOT IN (SELECT name FROM MovieExec));

ALTER TABLE MovieExec ADD CONSTRAINT noDupInStar
    CHECK (name NOT IN (SELECT name FROM MovieStar));
```

e)
```
ALTER TABLE Studio ADD CONSTRAINT noDupAddr
        CHECK (address is UNIQUE);
```

7.3.2

a)
```
ALTER TABLE Classes ADD CONSTRAINT        myKey
        PRIMARY KEY (class, country);
```
b)
```
ALTER TABLE Outcomes ADD CONSTRAINT        battleCheck
        FOREIGN KEY (battle) REFERENCES Battles (name);
```

c)
```
ALTER TABLE Outcomes ADD CONSTRAINT        shipCheck
        FOREIGN KEY (ship) REFERENCES Ships (name);
```

d)
```
ALTER TABLE Ships ADD CONSTRAINT classGun        Check
        CHECK (class NOT IN (SELECT class FROM Classes
                WHERE numGuns > 14));
```

e)
```
ALTER TABLE Ships ADD CONSTRAINT shipDateCheck
   CHECK (ship IN (SELECT s.name FROM Ships s, Battles b, Outcomes o
                        WHERE s.name = o.ship AND
              b.name = o.battle AND
              s.launched >= YEAR(b.date)))
```

7.4.1

a)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
      (
          (SELECT maker FROM Product NATURAL JOIN PC)
       INTERSECT
          (SELECT maker FROM Product NATURAL JOIN Laptop)
      )
   );
```

b)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
      (SELECT maker
       FROM Product NATURAL JOIN PC
       WHERE speed > ALL
          (SELECT L2.speed
           FROM Product P2, Laptop L2
           WHERE P2.maker = maker AND
                P2.model = L2.model
          )
     )
   );
```

c)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
      (SELECT model
       FROM Laptop
       WHERE price <= ALL
                            (SELECT price FROM PC
                             WHERE PC.ram < Laptop.ram
                            )
        )
     )
   );
```

d)
```
CREATE ASSERTION CHECK
   (EXISTS
      (SELECT p2.model FROM Product p1, PC p2
                     WHERE p1.type = ,pc? AND
                            P1.model = p2.model)
     UNION ALL
      (SELECT l.model
        FROM Product p, Laptop l
                     WHERE p.type = ,laptop? AND
                            p.model = l.model)
     UNION ALL
       (SELECT p2.model
         FROM Product p1, Printer p2
                     WHERE p1.type = ,printer? AND
                            P1.model = p2.model)
    );
```

7.4.2

a)
```
CREATE ASSERTION CHECK
   ( 2 >= ALL
              (SELECT COUNT(*) FROM Ships GROUP BY class)
    );
```

b)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
              (SELECT country FROM Classes
                     WHERE type = ‚bb?
               )
              INTERSECT
              (SELECT country FROM Classes
                     WHERE type = ‚bc?
               )
    );
```

c)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
              (SELECT o.battle FROM Outcomes o, Ships s, Classes c
                WHERE o.ship = s.name AND s.class = c.class AND c.numGuns > 9
               )
     INTERSECT
              (SELECT o.battle FROM Outcomes o, Ships s, Classes c
                    WHERE o.result = ‚sunk? AND o.ship = s.name AND
             s.class = c.class AND c.numGuns < 9
               )
    );
```

d)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
              (SELECT s1.name FROM Ships s1
             WHERE s1.launched < (SELECT s2.launched FROM Ships s2
                                                              WHERE s2.name = s1.class
                            )
      )
   );
```

e)
```
CREATE ASSERTION CHECK
   (ALL (SELECT class FROM Classes c)
     IN (SELECT class FROM Ships GROUP BY class)
    );
```

7.4.3

1)
```
presC# INT CHECK
              (presC# IN (SELECT cert# FROM MovieExec
                              WHERE netWorth >= 10000000
                            )
```

```
                )
2)
presC# INT Check
                (presC# NOT IN (SELECT cert# FROM MovieExec
                                        WHERE netWorth < 10000000
                                      )
                )
```

7.5.1

```
CREATE TRIGGER AvgNetWorthTrigger
AFTER INSERT ON MovieExec
REFERENCING
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
DELETE FROM MovieExec
        WHERE (name, address, cert#, netWorth) IN NewStuff;
```

```
CREATE TRIGGER AvgNetWorthTrigger
AFTER DELETE ON MovieExec
REFERENCING
        OLD TABLE AS OldStuff
FOR EACH STATEMENT
WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
INSERT INTO MovieExec
        (SELECT * FROM OldStuff);
```

7.5.2

a)

```
CREATE TRIGGER LowPricePCTrigger
AFTER UPDATE OF price ON PC
REFERENCING
        OLD ROW AS OldRow,
        OLD TABLE AS OldStuff,
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NewRow.price < ALL
        (SELECT PC.price FROM PC
         WHERE PC.speed = NewRow.speed))
BEGIN
        DELETE FROM PC
        WHERE (model, speed, ram, hd, price) IN NewStuff;
        INSERT INTO PC
                (SELECT * FROM OldStuff);
END;
```

b)

```
CREATE TRIGGER NewPrinterTrigger
AFTER INSERT ON Printer
REFERENCING
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NOT EXISTS (SELECT * FROM Product
                        WHERE Product.model = NewRow.model))
DELETE FROM Printer
        WHERE (model, color, type, price) IN NewStuff;
```

c)

```
CREATE TRIGGER AvgPriceTrigger
```

```
AFTER UPDATE OF price ON Laptop
REFERENCING
        OLD TABLE AS OldStuff,
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (1500 > (SELECT AVG(price) FROM Laptop))
BEGIN
        DELETE FROM Laptop
        WHERE (model, speed, ram, hd, screen, price) IN NewStuff;
        INSERT INTO Laptop
                (SELECT * FROM OldStuff);
END;
```

d)

```
CREATE TRIGGER HardDiskTrigger
AFTER UPDATE OF hd, ram ON PC
REFERENCING
        OLD ROW AS OldRow,
        OLD TABLE AS OldStuff,
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NewRow.hd < NewRow.ram * 100)
BEGIN
        DELETE FROM PC
        WHERE (model, speed, ram, hd, price) IN NewStuff;
        INSERT INTO PC
                (SELECT * FROM OldStuff);

END;
```

e)

```
CREATE TRIGGER DupModelTrigger
BEFORE INSERT ON PC, Laptop, Printer
REFERENCING
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (EXISTS (SELECT * FROM NewStuff NATUAL JOIN PC)
          UNION ALL
          (SELECT * FROM NewStuff NATUAL JOIN Laptop)
          UNION ALL
          (SELECT * FROM NewStuff NATUAL JOIN Printer))
BEGIN
        SIGNAL SQLSTATE ,10001?
           (,Duplicate Model     –     Insert Failed?);
END;
```

7.5.3

a)

```
CREATE TRIGGER NewClassTrigger
AFTER INSERT ON Classes
REFERENCING
        NEW ROW AS NewRow
FOR EACH ROW
```

```
BEGIN
        INSERT INTO Ships (name, class, lunched)
                VALUES (NewRow.class, NewRow.class, NULL);
END;
```

b)

```
CREATE TRIGGER ClassDisTrigger
BEFORE INSERT ON Classes
REFERENCING
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NewRow.displacement > 35000)
UPDATE NewStuff SET displacement = 35000;
```

c)

```
CREATE TRIGGER newOutcomesTrigger
AFTER INSERT ON Outcomes
REFERENCING
        NEW ROW AS NewRow
FOR EACH ROW
WHEN (NewRow.ship NOT EXISTS (SELECT name FROM Ships))
INSERT INTO Ships (name, class, lunched)
        VALUES (NewRow.ship, NULL, NULL);


CREATE TRIGGER newOutcomesTrigger2
AFTER INSERT ON Outcomes
REFERENCING
        NEW ROW AS NewRow
FOR EACH ROW
WHEN (NewRow.battle NOT EXISTS (SELECT name FROM Battles))
INSERT INTO Battles (name, date)
        VALUES (NewRow.battle, NULL);
```

d)

```
CREATE TRIGGER changeShipTrigger
AFTER INSERT ON Ships
REFERENCING
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN ( 20 < ALL
        (SELECT COUNT(name) From Ships NATURAL JOIN Classes
                GROUP BY country))
DELETE FROM Ships
WHERE (name, class, launched) IN NewStuff;

CREATE TRIGGER changeShipTrigger2
AFTER UPDATE ON Ships
REFERENCING
        OLD TABLE AS OldStuff,
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN ( 20 < ALL
```

```
                (SELECT COUNT(name) From Ships NATURAL JOIN Classes
                                GROUP BY country))
BEGIN
                DELETE FROM Ships
                WHERE (name, class, launched) IN NewStuff;
                INSERT INTO Ships
                                (SELECT * FROM OldStuff);
END;
```

e)

```
CREATE TRIGGER sunkShipTrigger
AFTER INSERT ON Outcomes
REFERENCING
                NEW ROW AS NewRow
                NEW TABLE AS NewStuff
FOR EACH ROW
WHEN ( (SELECT date FROM Battles WHERE name = NewRow.battle)
      < ALL
                (SELECT date FROM Battles
                    WHERE name IN (SELECT battle FROM Outcomes
                            WHERE ship = NewRow.ship AND
                                                                                result =      " sunk "
                       )
       )
     )

DELETE FROM Outcomes
WHERE (ship, battle, result) IN NewStuff;

CREATE TRIGGER sunkShipTrigger2
AFTER UPDATE ON Outcomes
REFERENCING
                NEW ROW AS NewRow,
                NEW TABLE AS NewStuff
FOR EACH ROW
FOR EACH ROW
WHEN ( (SELECT date FROM Battles WHERE name = NewRow.battle)
      < ALL
                (SELECT date FROM Battles
                    WHERE name IN (SELECT battle FROM Outcomes
                            WHERE ship = NewRow.ship AND
                                                                                result =      " sunk "
                       )
       )
     )
BEGIN
                DELETE FROM Outcomes
                WHERE (ship, battle, result) IN NewStuff;
                INSERT INTO Outcomes
                                (SELECT * FROM OldStuff);
END;
```

7.5.4

```
CREATE TRIGGER changeStarsInTrigger
AFTER INSERT ON StarsIn
```

```
REFERENCING
                NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.starName NOT EXISTS
                                (SELECT name FROM MovieStar))
INSERT INTO MovieStar(name)
            VALUES(NewRow.starName);


CREATE TRIGGER changeStarsInTrigger2
AFTER UPDATE ON StarsIn
REFERENCING
                NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.starName NOT EXISTS
                                (SELECT name FROM MovieStar))
INSERT INTO MovieStar(name)
            VALUES(NewRow.starName);
```

b)

```
CREATE TRIGGER changeMovieExecTrigger
AFTER INSERT ON MovieExec
REFERENCING
                NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.cert# NOT EXISTS
                                (SELECT presC# FROM Studio)
            UNION ALL
              SELECT producerC# FROM Movies)
    )
INSERT INTO Movies(procucerC#)
            VALUES(NewRow.cert#);
```

* insert into the relation Movies rather than Studio since there?s no associated info with Studio.

```
CREATE TRIGGER changeMovieExecTrigger2
AFTER UPDATE ON MovieExec
REFERENCING
                NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.cert# NOT EXISTS
                                (SELECT presC# FROM Studio)
            UNION ALL
              SELECT producerC# FROM Movies)
    )
INSERT INTO Movies(procucerC#)
            VALUES(NewRow.cert#);
```

c)

```
CREATE TRIGGER changeMovieTrigger
AFTER DELETE ON MovieStar
REFERENCING
                OLD TABLE AS OldStuff,
FOR EACH STATEMENT
WHEN ( 1 > ALL (SELECT COUNT(*) FROM StarIn s, MovieStar m
```

```
                    WHERE s.starName = m.name
                          GROUP BY s.movieTitle, m.gendar)
  )
INSERT INTO MovieStar
      (SELECT * FROM OldStuff);
```

* only considering DELETE from MovieStar since the assumption was the desired condistion was satisfied before any change.
** not considering INSERT into StarsIn since no gender info can be extracted from a new row for StarsIn.

d)

```
CREATE TRIGGER numMoviesTrigger
AFTER   INSERT   ON Movies
REFERENCING
      NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (100 < ALL
      (SELECT COUNT(*) FROM Movies
            GROUP BY studioName, year))
DELETE FROM Movies
WHERE (title, year, length, genre, StudioName, procedureC#)IN NewStuff;


CREATE TRIGGER numMoviesTrigger2
AFTER   UPDATE ON Movies
REFERENCING
      OLD TABLE AS OldStuff
      NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (100 < ALL
      (SELECT COUNT(*) FROM Movies
            GROUP BY studioName, year))
BEGIN
      DELETE FROM Movies
      WHERE (title, year, length, genre, StudioName, procedureC#)
      IN NewStuff;
      INSERT INTO Movies
        (SELECT * FROM OldStuff);
END;
```

e)

```
CREATE TRIGGER avgMovieLenTrigger
AFTER   INSERT   ON Movies
REFERENCING
      NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (120 < ALL
      (SELECT AVG(length) FROM Movies
            GROUP BY year))
DELETE FROM Movies
WHERE (title, year, length, genre, StudioName, procedureC#)IN NewStuff;


CREATE TRIGGER avgMovieLenTrigger2
AFTER   UPDATE ON Movies
```

```
REFERENCING
        OLD TABLE AS OldStuff
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (120 < ALL
        (SELECT AVG(length) FROM Movies
                GROUP BY year))
BEGIN
        DELETE FROM Movies
        WHERE (title, year, length, genre, StudioName, procedureC#)
        IN NewStuff;
        INSERT INTO Movies
           (SELECT * FROM OldStuff);
END;
```

Section 1

Exercise 8.1.1

a)

```
CREATE VIEW RichExec AS
	SELECT * FROM MovieExec WHERE netWorth >= 10000000;
```

b)

```
CREATE VIEW StudioPres (name, address, cert#) AS
	SELECT MovieExec.name, MovieExec.address, MovieExec.cert# FROM MovieExec,
	Studio WHERE MovieExec.cert# = Studio.presC#;
```

c)

```
CREATE VIEW ExecutiveStar (name, address, gender, birthdate, cert#, netWorth) AS
	SELECT star.name, star.address, star.gender, star.birthdate, exec.cert#, exec.netWorth
	FROM MovieStar star, MovieExec exec WHERE star.name = exec.name AND
	star.address = exec.address;
```

Exercise 8.1.2

a)

```
SELECT name from ExecutiveStar WHERE gender = ‘f’;
```

b)

```
SELECT RichExec.name from RichExec, StudioPres where RichExec.name = StudioPres.name;
```

c)

```
SELECT ExecutiveStar.name from ExecutiveStar, StudioPres
	WHERE ExecutiveStar.netWorth  >= 50000000 AND
	StudioPres.cert# = RichExec.cert#;
```

Section 2

Exercise 8.2.1

The views RichExec and StudioPres are updatable; however, the StudioPres view needs to be created with a subquery.

```
CREATE VIEW StudioPres (name, address, cert#) AS
	SELECT MovieExec.name, MovieExec.address, MovieExec.cert# FROM MovieExec
	WHERE MovieExec.cert# IN (SELECT presCt# from Studio);
```

Exercise 8.2.2

a) Yes, the view is updatable.

b)

```
CREATE TRIGGER DisneyComedyInsert
INSTEAD OF INSERT ON DisneyComedies
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
INSERT INTO Movies(title, year, length, studioName, genre)
VALUES(NewRow.title, NewRow.year, NewYear.length, ‘Disney’, ‘comedy’);
```

c)

```
CREATE TRIGGER DisneyComedyUpdate
INSTEAD OF UPDATE ON DisneyComedies
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
UPDATE Movies SET length NewRow.length
WHERE title = NewRow.title AND year = NEWROW.year AND
studionName = ‘Disney’ AND genre = ‘comedy’;
```

Exercise 8.2.3

a) No, the view is not updatable since it is constructed from two different relations.

b)

```
CREATE TRIGGER NewPCInsert
INSTEAD OF INSERT ON NewPC
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
(INSERT INTO Product VALUES(NewRow.maker, NewRow.model, ‘pc’))
(INSERT INTO PC VALUES(NewRow.model, NewRow.speed, NewRow.ram, NewRow.hd,
NewRow.price));
```

c)

```
CREATE TRIGGER NewPCUpdate
INSTEAD OF UPDATE ON NewPC
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
UPDATE PC SET price = NewPC.price where model = NewPC.model;
```

d)

```
CREATE TRIGGER NewPCDelete
INSTEAD OF DELETE ON NeePC
REFERENCING OLD ROW AS OldRow
FOR EACH ROW
(DELETE FROM Product WHERE model = OldRow.model)
(DELETE FROM PC where model = OldRow.model);
```

Section 3

Exercise 8.3.1
a)
CREATE INDEX NameIndex on Studio(name);

b)
CREATE INDEX AddressIndex on MovieExec(address);

c)
CREATE INDEX GenreIndex on Movies(genre, length);

Section 4

Exercise 8.4.1

Action	No Index	Star Index	Movie Index	Both Indexes
Q1	100	4	100	4
Q2	100	100	4	4
I	2	4	4	6
Average	$2 + 98p_1 + 98p_2$	$4 + 96\,p_2$	$4 + 96\,p_1$	$6 – 2\,p_1 – 2\,p_2$

Exercise 8.4.2

Q1 = SELECT * FROM Ships WHERE name = n;
Q2 = SELECT * FROM Ships WHERE class = c;
Q3 = SELECT * FROM Ships WHERE launched = y;
I = Inserts

Indexes Actions	None	Name	Class	Launched	Name & Class	Name & Launched	Class & Launched	Three Indexes
Q1	50	2	50	50	2	2	50	2
Q2	1	1	2	1	2	1	2	2
Q3	50	50	50	26	50	26	26	26
I	2	4	4	4	6	6	6	8
Average	$2 + 48p_1 - p_2 + 48p_3$	$4 + 46\,p_3 - 2\,p_1 - 3\,p_2$	$4 + 46p_1 - 2p_2 + 46p_3$	$4 + 46p_1 - 3p_2 + 22p_3$	$6 - 4p_1 - 4p_2 + 44p_3$	$6 - 4p_1 - 5p_2 + 20p_3$	$6 - 44p_1 - 4p_2 + 20p_3$	$8 - 6p_1 - 6p_2 + 18p_3$

The best choice of indexes (name and launched) has an average cost of $6 - 4p_1 - 5p_2 + 20p_3$ per operation.

Section 5

Exercise 8.5.1

Updates to movies that involves title or year

UPDATE MovieProd SET title = ‘newTitle’ where title=’oldTitle’ AND year = oldYear;

UPDATE MovieProd SET year = newYear where title=’oldYitle’ AND year = oldYear;

Update to MovieExec involving cert#

```
DELETE FROM MovieProd
WHERE (title, year) IN (
        SELECT title, year
        FROM Movies, MovieExec
        WHERE cert# = oldCert# AND cert# = producerC#
        );
```

```
INSERT INTO MovieProd
        SELECT title, year, name
        FROM Movies, MovieExec
        WHERE cert# = newCert# AND cert# = producerC#;
```

Exercise 8.5.2

Insertions, deletions, and updates to the base tables Product and PC would require a modification of the materialized view.

Insertions into Product with type equal to ‘pc’:

```
        INSERT INTO NewPC
                SELECT maker, model, speed, ram, hd, price FROM Product, PC WHERE
                Product.model = newModel and Product.model = PC.model;
```

Insertions into PC:

```
        INSERT INTO NewPC
                SELECT maker, ‘newModel’, ‘newSpeed’, ‘newRam’, ‘newHd’, ‘newPrice’
                FROM Product WHERE model = ‘newModel’;
```

Deletions from Product with type equal to ‘pc’:

```
        DELETE FROM NewPC WHERE maker = ‘deletedMaker’ AND model=’deletedModel’;
```

Deletions from PC:

DELETE FROM NewPC WHERE model = ‘deletedModel’;

Updates to PC:

Update NewPC SET speed=PC.speed, ram=PC.ram, hd=PC.hd, price=PC.price FROM PC where model=pc.model;

Update to the attribute ‘model’ needs to be treated as a delete and an insert.

Updates to Product:

Any changes to a Product tuple whose type is ‘pc’ need to be treated as a delete or an insert, or both.

Exercise 8.5.3

Modifications to the base tables that would require a modification to the materialized view: inserts and deletes from Ships, deletes from class, updates to a Class’ displacement.

Deletions from Ship:

```
UPDATE ShipStats SET
        displacement=((displacement * count) –
                (SELECT displacement
                FROM Classses
                WHERE class  = ‘DeletedShipClass’)
                ) / (count – 1),
        count = count – 1
WHERE
        country = (SELECT country FROM Classes WHERE class=’DeletedShipClass’);
```

Insertions into Ship:

```
Update ShipStat SET
        displacement=((displacement*count) +
                (SELECT displacement FROM Classes
                WHERE class=’InsertedShipClass’)
                ) / (count + 1),
        count = count + 1
WHERE
        country = (SELECT country FROM Classes WHERE classes=’InsertedShipClass);
```

Deletes from Classes:

```
NumRowsDeleted  = SELECT count(*) FROM ships WHERE class = ‘DeletedClass’;
```

```
UPDATE ShipStats SET
        displacement = (displacement * count)  - (DeletedClassDisplacement *
```

```
                    NumRowsDeleted)) /  (count – NumRowsDeleted),
          count = count – NumRowsDeleted
WHERE country = 'DeletedClassCountry';
```

Update to a Class' displacement:

```
N = SELECT count(*) FROM Ships where class = 'UpdatedClass';

UPDATE ShipsStat SET
          displacement = ((displacement * count) + ((oldDisplacement – newDisplacement) *
          N))/count
WHERE
          country = 'UpdatedClassCountry';
```

Exercise 8.5.4

Queries that can be rewritten with the materialized view:

Names of stars of movies produced by a certain producer

```
SELECT starName
FROM StarsIn, Movies, MovieExec
WHERE movieTitle = title AND movieYear = year AND producerC# = cert# AND
          name = 'Max Bialystock';
```

Movies produced by a certain producer

```
SELECT title, year
FROM Movies, MovieExec
Where producerC# = cert# AND name = 'George Lucas';
```

Names of producers that a certain star has worked with

```
SELECT name
FROM Movies, MovieExec, StarsIn
Where producerC#=cert# AND title=movieTitle AND year=movieYear AND
          starName='Carrie Fisher';
```

The number of movies produced by given producer

```
SELECT count(*)
FROM Movies, MovieExec
WHERE producerC#=cert# AND name = 'George Lucas';
```

Names of producers who also starred in their own movies

```
SELECT name
FROM Movies, StarsIn, MovieExec
WHERE producerC#=cert# AND movieTitle = title AND movieYear = year AND
        MovieExec.name = starName;
```

The number of stars that have starred in movies produced by a certain producer

```
SELECT count(DISTINCT starName)
FROM Movies, StarsIn, MovieExec
WHERE producerC#=cert# AND movieTitle = title AND movieYear = year AND
        name ‘George Lucas’;
```

The number of movies produced by each producer

```
SELECT name, count(*)
FROM Movies, MovieExec
WHERE producerC#=cert# GROUP BY name
```

9.3.1

a)

In the following, we use macro NOT_FOUND as defined in the section.

```
void closestMatchPC() {

    EXEC SQL BEGIN DECLARE SECTION;

        char manf, SQLSTATE[6];
        int targetPrice, /* holds price given by user */
        float tempSpeed, speedOfClosest;
        char tempModel[4], modelOfClosest[4];
        int tempPrice, priceOfClosest;
        /* for tuple just read from PC & closest price found so far */

    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE pcCursor CURSOR FOR
        SELECT model, price, speed FROM PC;

    EXEC SQL OPEN pcCursor;

    /* ask user for target price and read the answer into variable
       targetPrice */

    /* Initially, the first PC is the closest to the target price.
       If PC is empty, we cannot answer the question, and so abort. */

    EXEC SQL FETCH FROM pcCursor INTO :modelOfClosest, :priceOfClosest,
                                      :speedOfClosest;
    if(NOT_FOUND) /* print message and exit */ ;

    while(1) {
        EXEC SQL FETCH pcCursor INTO :tempModel, :tempPrice,
                                     :tempSpeed;
        if(NOT_FOUND) break;
        if(/*tempPrice closer to targetPrice than is priceOfClosest */)
        {
            modelOfClosest = tempModel;
            priceOfClosest = tempPrice;
            speedOfClosest = tempSpeed;
        }
    }
    /* Now, modelOfClosest is the model whose price is closest to
       target. We must get its manufacturer with a single-row select */

   EXEC SQL SELECT maker
            INTO :manf
            FROM Product
            WHERE model = :modelOfClosest;

    printf("manf=%s, model=%d, speed=%d\n",
             manf, modelOfClosest, speedOfClosest);

    EXEC SQL CLOSE CURSOR pcCursor;
```

```
}

b)
void acceptableLaptop() {

    EXEC SQL BEGIN DECLARE SECTION;

        int minRam, minHd, minScreen; /* given by user */
        float minSpeed;
        char model[4], maker,
        float speed;
        int ram, hd, screen, price;

    EXEC SQL END DECLARE SECTION;

    EXEC SQL PREPARE query1 FROM
        ‘SELECT model, speed, ram, hd, screen, price, maker
             FROM Laptop l, Product p
               WHERE speed >= ? AND
                     ram >= ? AND
                     hd >= ? AND
                     screen >= ? AND
                     l.model = p.model’

    EXEC SQL DECLARE cursor1 CURSOR FOR query1;

    /* ask user for minimum speed, ram, hd size, and screen size */

    EXEC SQL OPEN cursor1 USING :minSpeed, :minRam, :minHd, :minScreen;

    while(!NOT_FOUND) {

        EXEC SQL FETCH cursor1 INTO
                   :model, :speed, :ram, :hd, :screen, :price, :maker;

        if(FOUND)
        {
            printf("maker:%s, model:%d, \n
                    speed:%.2f, ram:%d, hd:%d, screen:%d, price:%d\n",
                    maker, model, speed, ram, hd, screen, price);

        }
    }

    EXEC SQL CLOSE CURSOR cursor1;
}

c)

void productsByMaker() {

    EXEC SQL BEGIN DECLARE SECTION;

        char maker, model[4], type[10], color[6];
        float speed;
        int ram, hd, screen, price;
```

```
EXEC SQL END DECLARE SECTION;

EXEC SQL PREPARE query1 FROM
    'SELECT * FROM PC
           WHERE model IN (SELECT model FROM Product
                                WHERE maker = ? AND
                                      type = 'pc');
EXEC SQL PREPARE query2 FROM
    'SELECT * FROM Laptop
           WHERE model IN (SELECT model FROM Product
                                WHERE maker = ? AND
                                      type = 'laptop');
EXEC SQL PREPARE query3 FROM
    'SELECT * FROM Printer
           WHERE model IN (SELECT model FROM Product
                                WHERE maker = ? AND
                                      type = 'printer');

EXEC SQL DECLARE cursor1 CURSOR FOR query1;
EXEC SQL DECLARE cursor2 CURSOR FOR query2;
EXEC SQL DECLARE cursor3 CURSOR FOR query3;

/* ask user for manufacturer */
Printf("maker:%s\n", maker);

/* get PCs made by the manufacturer */
EXEC SQL OPEN cursor1 USING :maker;

Printf("product type: PC\n");

while(!NOT_FOUND) {

    EXEC SQL FETCH cursor1 INTO
               :model, :speed, :ram, :hd, :price;

    if(FOUND)
    {
        printf("model:%d,speed:%.2f, ram:%d, hd:%d, price:%d\n",
                 model, speed, ram, hd, price);

    }
}

/* get Laptops made by the manufacturer */
EXEC SQL OPEN cursor2 USING :maker;

Printf("product type: Laptop\n");

while(!NOT_FOUND) {

    EXEC SQL FETCH cursor2 INTO
               :model, :speed, :ram, :hd, :screen, :price;

    if(FOUND)
    {
        printf("model:%d, speed:%.2f, ram:%d, hd:%d, screen:%d,
                price:%d\n", model, speed, ram, hd, screen, price);
```

```
        }
    }

    /* get Printers made by the manufacturer */
    EXEC SQL OPEN cursor3 USING :maker;

    Printf("product type: Printer\n");

    while(!NOT_FOUND) {

        EXEC SQL FETCH cursor3 INTO
                   :model, :color, :type, :price;

        if(FOUND)
        {
            printf("model:%d, color:%s, type:%s, price:%d\n",
                                        model, color, type, price);

        }
    }

    EXEC SQL CLOSE CURSOR cursor1;
    EXEC SQL CLOSE CURSOR cursor2;
    EXEC SQL CLOSE CURSOR cursor3;
}
```

d)

```
void withinBudget() {

    EXEC SQL BEGIN DECLARE SECTION;

       int total_budget, rest_budget, pc_price, printer_price;
       char pc_model[4], printer_model[4], color[6];
       float min_speed;

    EXEC SQL END DECLARE SECTION;

    EXEC SQL PREPARE query1 FROM
        'SELECT model, price FROM PC
               WHERE speed >= ? AND price <= ?
                       ORDER BY price';

    EXEC SQL PREPARE query2 FROM
        'SELECT model, price FROM Printer
               WHERE price <= ? AND color = ?
                       ORDER BY price';

    EXEC SQL DECLARE cursor1 CURSOR FOR query1;
    EXEC SQL DECLARE cursor2 CURSOR FOR query2;

    /* ask user for budget & the minimum speed of pc */

    /* get the cheapest PC of the minimum speed */
    EXEC SQL OPEN cursor1 USING :min_speed, :total_budget;
```

```
    EXEC SQL FETCH cursor1 INTO :pc_model, :pc_price;

    if (NOT_FOUND)
        Printf("no pc found within the budget\n");
    else
    {
        Printf("pc model: %s\n", pc_model);
    }

    /* get Printer within the budget */
    rest_budget = total_budget – pc_price;
    color = "true";

    EXEC SQL OPEN cursor2 USING :rest_budget, :color;

    EXEC SQL FETCH cursor2 INTO :printer_model;

    if(NOT_FOUND) {
        EXEC SQL CLOSE CURSOR cursor2;
        color = "false";
        EXEC SQL OPEN cursor2 USING :rest_budget, :color;

        if(NOT_FOUND)
           printf("no printer found within the budget\n");
        else {
           EXEC SQL FETCH cursor2 INTO :printer_model;
           printf("printer model: %s\n", printer_model);
        }
    }
    else {
        printf("printer model: %s\n", printer_model);
    }

    EXEC SQL CLOSE CURSOR cursor1;
    EXEC SQL CLOSE CURSOR cursor2;
}
```

e)

```
void newPCproduct() {

    EXEC SQL BEGIN DECLARE SECTION;

        char pmaker, pmodel[4], ptype[6];
        float pspeed;
        int pram, phd, pscreen, pprice;
        int pcount;

    EXEC SQL END DECLARE SECTION;

    EXEC SQL PREPARE stmt1 FROM
        'SELECT COUNT(*) INTO :count
               FROM PC
               WHERE MODEL = ?;

    EXEC SQL PREPARE stmt2 FROM
        'INSERT INTO Product VALUES(?, ?, ?)';
```

```
    EXEC SQL PREPARE stmt3 FROM
        'INSERT INTO PC VALUES(?, ?, ?, ?, ?)';

    /* ask user for manufacturer, model, speed, RAM, hard-disk,
       & price of a new PC                                          */

    EXEC SQL EXECUTE stmt1 USING :pmodel;

    IF (count > 0)
      Printf("Warnning: The PC model already exists\n");
    ELSE
    {
       EXEC SQL EXECUTE stmt2 USING :pmaker, :pmodel, :ptype;

       EXEC SQL EXECUTE stmt3 UINGNG :pmodel, :pspeed, :pram,
                                      :phd, :pprice
    }

}
```

9.3.2

a)

```
void largestFirepower() {

    EXEC SQL BEGIN DECLARE SECTION;

        char cclass[20], maxFirepowerClass[20];
        int cnumGuns, cbore;
        float firepower, maxFirepower;

    EXEC SQL END DECLARE SECTION;

    EXEC SQL DECLARE cursor1 CURSOR FOR
        SELECT class, numGuns, bore FROM Classes;

    EXEC SQL OPEN cursor1;

    EXEC SQL FETCH FROM cursor1 INTO :cclass, :cnumGuns, :cbore;

    if(NOT_FOUND) /* print message and exit */ ;

    maxFirepower = cnumGuns * (power (cbore, 3));
    strcpy(maxFirepowerClass, cclass);

    while(1) {
        EXEC SQL FETCH cursor1 INTO :cclass, :cnumGuns, :cbore;

        if(NOT_FOUND) break;

        firepower = cnumGuns * (power (cbore, 3));

        if( firepower > maxFirepower )
        {
            maxFirepower = firepower;
```

```
            strcpy(maxFirepowerClass, cclass);
        }
    }

    printf("Class of maximum firepower :%s\n", maxFirepowerClass);

    EXEC SQL CLOSE CURSOR cursor1;
}
```

b)

```
void getCountry() {

    EXEC SQL BEGIN DECLARE SECTION;

        char ibattle[20], iresult[10], ocountry[20];
        char stmt1[200], stmt2[200];

    EXEC SQL END DECLARE SECTION;

    strcpy(stmt1, "SELECT COUNTRY FROM Classes C
                WHERE C.class IN (
                   SELECT S.class FROM Ships S
                      WHERE S.name IN (
                         SELECT ship FROM Outcomes
                            WHERE battle = ?))" );

    Strcpy(stm2, "SELECT country FROM Classes
                WHERE class = ( SELECT MAX(COUNT(class))
                                  FROM Ships s, Outcomes o
                                     WHERE o.name = s.ship AND
                                         s.result = '?')" );

    EXEC SQL PREPARE query1 FROM stmt1;
    EXEC SQL PREPARE query2 FROM stmt2;

    EXEC SQL DECLARE cursor1 CURSOR FOR query1;
    EXEC SQL DECLARE cursor2 CURSOR FOR query2;

    /* ask user for battle */

    /* get countries of the ships involved in the battle */
    EXEC SQL OPEN cursor1 USING :ibattle;

    while(!NOT_FOUND) {

        EXEC SQL FETCH cursor1 INTO :ocountry;

        if(FOUND)
            printf("contry:%s\n", ocoutry);
    }

    EXEC SQL CLOSE CURSOR cursor1;

    /* get the country with the most ships sunk */
    strcpy(iresult, "sunk");
```

```
    EXEC SQL OPEN cursor2 USING :iresult;

    /* loop for the case there’s the same max# of ships sunk */

    While(!NOT_FOUND) {

        EXEC SQL FETCH cursor2 INTO :ocountry;

        If(FOUND)
            Printf(“country with the most ships sunk: %s, ocountry);

    }

    /* get the country with the most ships damaged */
    strcpy(iresult, “damaged”);

    EXEC SQL OPEN cursor2 USING :iresult;

    /* loop for the case there’s the same max# of ships damaged */

    While(!NOT_FOUND) {

        EXEC SQL FETCH cursor2 INTO :ocountry;

        If(FOUND)
            Printf(“country with the most ships damaged: %s, ocountry);

    }

}
```

c)

```
void addShips() {

    EXEC SQL BEGIN DECLARE SECTION;

        char iclass[20], itype[3], icontry[20], iship[20];
        int inumGuns, ibore, idisplacement, ilaunched;
        char stmt1[100], stmt2[100];

    EXEC SQL END DECLARE SECTION;

    strcpy(stmt1, “INSERT INTO Classes VALUES (?, ?, ?, ?, ?, ?)”);

    strcpy(stmt2, “INSERT INTO Ships VALUES (?, ?, ?)”);

    /* ask user for a class and other info for Classes table */

    EXEC SQL EXECUTE IMMEDATE :stmt1
                    USING :iclass, :itype, :icontry,
                          :inumGuns, :ibore, :idisplacement;

    /* ask user for a ship and launched */

    WHILE(there_is_input)
    {
```

```
        EXEC SQL EXECUTE IMMEDATE :stmt2
                            USING :iship, :iclass, ilaunched;

        /* ask user for a ship and launched */
    }

}
```

d)

```
void findError() {

    EXEC SQL BEGIN DECLARE SECTION;

        char bname[20], bdate[8], newbdate[8];
        char sname[20], lyear[4], newlyear[4];
        char stmt1[100], stmt2[100];

    EXEC SQL END DECLARE SECTION;

    strcpy(stmt1, "UPDATE Battles SET date = ? WHERE name = ?");
    strcpy(stmt2, "UPDATE Ships SET launched = ? WHERE name = ?");

    EXEC SQL DECLARE C1 CURSOR FOR
        Select b.name, b.date, s.name, s.launched
            FROM Battles b, Outcomes o, Ships s
                WHERE b.name = o.battle AND
                      o.ship = s.name AND
              YEAR(b.date) < s.launched;

    EXEC SQL OPEN C1;

    while(!NOT_FOUND) {

        EXEC SQL FETCH C1 INTO :bname, :bdate, :sname, :lyear;

        /* prompt user and ask if a change is needed */

        if(change_battle)
        {
           /* get a new battle date to newbdate */
           EXEC SQL EXECUTE IMMEDATE :stmt1
                            USING :bname, :newbdate;
        }

        if(change_ship)
        {
           /* get a new launched year to newlyear */
           EXEC SQL EXECUTE IMMEDATE :stmt2
                            USING :sname, :newlyear;
        }
     }
}
```

9.4.1

a)

```
CREATE FUNCTION PresNetWorth(studioName CHAR[15]) RETURNS INTEGER

DECLARE presNetWorth INT;

BEGIN
    SELECT netWorth
    INTO presNetWorth
    FROM Studio, MovieExec
    WHERE Studio.name = studioName AND presC# = cert#;
    RETURN(presNetWorth);
END;
```

b)

```
CREATE FUNCTION status(person CHAR(30), addr CHAR(255)) RETURNS INTEGER

DECLARE isStar INT;
DECLARE isExec INT;

BEGIN
    SELECT COUNT(*)
    INTO isStar
    FROM MovieStar
    WHERE MovieStar.name = person AND MovieStar.address = addr;
    SELECT COUNT(*)
    INTO isExec
    FROM MovieExec
    WHERE MovieExec.name = person AND MovieExec.address = addr;
    IF isStar + isExec = 0 THEN RETURN(4)
    ELSE RETURN(isStar + 2*isExec)
    END IF;
END;
```

c)

```
CREATE PROCEDURE twoLongest(
    IN studio CHAR(15),
    OUT longest VARCHAR(255),
    OUT second VARCHAR(255)
)

DECLARE t VARCHAR(255);
DECLARE i INT;
DECLARE Not_Found CONDITION FOR SQLSTATE = '02000';
DECLARE MovieCursor CURSOR FOR
    SELECT title FROM Movies WHERE studioName = studio
       ORDER BY length DESC;

BEGIN
    SET longest = NULL;
    SET second = NULL;
    OPEN MovieCursor;
    SET i = 0;
```

```
    mainLoop: WHILE (i < 2) DO
        FETCH MovieCursor INTO t;
        IF Not_Found THEN LEAVE mainLoop END IF;
        SET i = i + 1;
    END WHILE;
    CLOSE MovieCursor;
END;
```

d)

```
CREATE PROCEDURE earliest120mMovie(
    IN star CHAR(30),
    OUT earliestYear INT
)

DECLARE Not_Found CONDITION FOR SQLSTATE = '02000';
DECLARE MovieCursor CURSOR FOR
    SELECT MIN(year) FROM Movies
         WHERE length > 120 AND
               title IN (SELECT movieTitle FROM StarsIn
                                   WHERE starName = star);
BEGIN
   SET earliestYear = 0;
   OPEN MovieCursor;
   FETCH MovieCursor INTO earliestYear;
   CLOSE MovieCursor;
END;
```

e)

```
CREATE PROCEDURE uniqueStar(
    IN addr CHAR(255),
    OUT star CHAR(30)
)

BEGIN
   SET star = NULL;
   IF 1 = (SELECT COUNT(*) FROM MovieStar WHERE address = addr)
   THEN
      SELECT name INTO star FROM MovieStar WHERE address = addr;
END;
```

f)

```
CREATE PROCEDURE removeStar(
    IN star CHAR(30)
)

BEGIN
   DELETE FROM Movies WHERE title IN
                (SELECT movieTitle FROM StarsIn WHERE starName = star);
   DELETE FROM StarsIn WHERE starName = star;
   DELETE FROM MovieStar WHERE name = star;
END;
```

9.4.2

a)

```
CREATE FUNCTION closestMatchPC(targetPrice INT) RETURNS CHAR

DECLARE closestModel CHAR(4);
DECLARE diffSq INT;
DECLARE currSq INT;
DECLARE m CHAR(4);
DECLARE p INT;
DECLARE Not_Found CONDITION FOR SQLSTATE '02000';
DECLARE PCCursor CURSOR FOR
    SELECT model, price FROM PC;

BEGIN
    SET closestModel = NULL;
    SET diffSq = -1;
    OPEN PCCursor;
    mainLoop: LOOP
        FETCH PCCursor INTO m, p;
        IF Not_Found THEN LEAVE mainLoop END IF;
        SET currSq = (p - targetPrice)*(p - targetPrice);
        IF diffSq = -1 OR diffSq > currSq
            THEN BEGIN
                SET closestModel = m;
                SET diffSq = currSq;
        END IF;
    END LOOP;
    CLOSE PDCursor;
    RETURN(closestModel);
END;
```

b)

```
CREATE FUNCTION getPrice(imaker CHAR(1), imodel CHAR(4))
                    RETURNS INTEGER

DECLARE ptype VARCHAR(10);
DECLARE pprice INT;
DECLARE Not_Found CONDITION FOR SQLSTATE '02000';

BEGIN
    SELECT type INTO ptype FROM Product
                        WHERE maker = imaker AND model = imodel;
    IF ptype = ‘pc’ THEN
       SELECT price INTO pprice FROM PC
                           WHERE model = imodel;
    ELSE IF ptype = ‘laptop’ THEN
       SELECT price INTO pprice FROM Laptop
                           WHERE model = imodel;
    ELSE IF ptype = ‘printer’ THEN
       SELECT price INTO pprice FROM Printer
                           WHERE model = imodel;
    ELSE
       pprice = NULL;
    END IF;
```

```
    RETURN (pprice);
END;
```

c)

```
CREATE PROCEDURE addPC(
    IN imodel INT,
    IN ispeed DECIMAL(3,2),
    IN iram   INT,
    IN ihd    INT,
    IN iprice INT
)

DECLARE Already_Exist CONDITION FOR SQLSTATE '02300';

BEGIN


   INSERT INTO PC VALUES(imodel, ispeed, iram, ihd, iprice);
   WHILE (Already_Exist) DO
     SET imodel = imodel + 1;
     INSERT INTO PC VALUES(imodel, ispeed, iram, ihd, iprice);
   END WHILE;
END;
```

d)

```
CREATE PROCEDURE getNumOfHigherPrice(
    IN  iprice INT,
    OUT NumOfPCs INT,
    OUT NumOfLaptops INT,
    OUT NumOfPrinters INT
)

BEGIN
   SET NumOfPCs = 0;
   SET NumOfLaptops = 0;
   SET NumOfPrinters = 0;

   SELECT COUNT(*) INTO NumOfPCs FROM PC
                WHERE price > iprice;

   SELECT COUNT(*) INTO NumOfLaptops FROM Laptop
                WHERE price > iprice;

   SELECT COUNT(*) INTO NumOfPrinters FROM Printer
                WHERE price > iprice;
END;
```

9.4.3

a)

```
CREATE FUNCTION getFirepower(iclass VARCHAR(10)) RETURNS INTEGER

DECLARE firepower INT;
DECLARE nguns INT;
DECLARE nbore INT;

BEGIN
  SELECT numGuns, bore INTO nguns, nbore FROM Classes
                     WHERE class = iclass;
  SET firepower = nguns * (nbore * nbore * nbore);
  RETURN(firepower);
END;
```

b)

```
CREATE PROCEDURE twoCountriesInBattle(
    IN ibattle VARCHAR(20),
    OUT firstCountry VARCHAR(20),
    OUT secondCountry VARCHAR(20)
)

DECLARE i INT;
DECLARE ocountry VARCHAR(20);
DECLARE classCursor CURSOR FOR
    SELECT country FROM Classes
              WHERE class IN(SELECT class FROM Ships
                                 WHERE name IN(
                                              SELECT ship FROM Outcomes
                                                WHERE battle = ibattle
                                              )
                             );

BEGIN
   SET firstCountry = NULL;
   SET secondCountry = NULL;
   SET i = 0;
   IF 2 = (SELECT COUNT(*) count FROM Classes
              WHERE class IN(SELECT class FROM Ships
                                 WHERE name IN(
                                              SELECT ship FROM Outcomes
                                                WHERE battle = ibattle
                                              )
                             )
           )
   THEN
      OPEN classCursor;
      WHILE (i < 2) DO
         FETCH classCursor INTO ocountry;
         IF (i = 0) THEN
            SET firstCountry = ocountry;
         ELSE
            SET secoundCountry = ocountry;
         END IF;
```

```
      END WHILE;
   END IF;
   CLOSE calssCursor;
END;
```

c)

```
CREATE PROCEDURE addClass(
    IN iship VARCHAR(20),
    IN iclass VARCHAR(20),
    IN itype CHAR(2),
    IN icountry VARCHAR(20),
    IN inumGuns INT,
    IN ibore INT,
    IN idisplacement INT
)

BEGIN

   INSERT INTO Classes VALUES(iclass, itype, icountry,
                                    inumGuns, ibore, idisplacement);
   INSERT INTO Ships VALUES(iship, iclass, NULL);
END;
```

d)

```
CREATE PROCEDURE checkLaunched(
    IN ship VARCHAR(20)
)
DECLARE bname VARCHAR(20);

BEGIN
   IF EXIST (SELECT b.name INTO bname
                 FROM Battles b, Outcomes o, Ships s
                         WHERE b.name = o.battle AND
                               o.ship = s.name AND
                               YEAR(b.date) < s.launched)
   THEN
     UPDATE Ships SET launced = 0 WHERE name = iship;
     UPDATE Battles SET date = 0 WHERE name = bname;
   END IF;
END
```

9.4.4

$$
\begin{aligned}
\Big[\sum_{i=1}^{n} (x_i - \bar{x})^2 \Big] \;/\; \text{n} &= 1/\text{n} \Big[\sum_{i=1}^{n} \left(x_i^2 - 2\bar{x}x_i + \bar{x}^2 \right) \Big] \\
&= 1/\text{n} \Big[\sum x_i^2 - \sum 2\bar{x}x_i + \sum \bar{x}^2 \Big] \\
&= 1/\text{n} \Big[\sum x_i^2 - 2\bar{x} \sum x_i + \bar{x}^2 \sum 1 \Big] \\
&= 1/\text{n} \Big[\sum x_i^2 - 2\bar{x}(n\bar{x}) + \bar{x}^2(n) \Big] \\
&= 1/\text{n} \Big[\sum x_i^2 - n\bar{x}^2 \Big] \quad \text{, since} \quad \bar{x} = \sum_{i=1}^{n} (x_i) \;/\; \text{n} \\
&= \Big[\sum_{i=1}^{n} (x_i)^2 \Big] / \text{n} - \Big[\Big(\sum_{i=1}^{n} x_i \Big) / \text{n} \Big]^2
\end{aligned}
$$

9.5.1

a)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR manf, tempModel[4];
SQLFLOAT tempSpeed;
SQLINTEGER tempPrice;
SQLINTEGER colInfo;

Int targetPrice;
char modelOfClosest[4];
float speedOfClosest;
int priceOfClosest;

/* ask user for target price and read the answer into variable
   targetPrice   */

errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf(”Error for SQL_HANDLE_ENV.\n”);
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf(”Error for SQL_ HANDLE_DBC.\n”);
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf(”Error for SQL_ HANDLE_STMT.\n”);
   exit(1);
}

SQLExecDirect(execStat, “SELECT model, price, speed FROM PC”, SQL_NTS);

SQLBindCol(execStat, 1, SQL_CHAR, tempModel,
                                sizeof(tempModel), &colInfo);
SQLBindCol(execStat, 2, SQL_INTEGER, tempPrice,
                                sizeof(tempPrice), &colInfo);
SQLBindCol(execStat, 3, SQL_FLOAT, tempSpeed,
                                sizeof(tempSpeed), &colInfo);

priceOfClosest = NULL;
while(SQLFetch(execStat) != SQL_NO_DATA) {

   if( /* the 1st fetch or tempPrice closer to targetPrice */
        modelOfClosest = tempModel;
```

```
        priceOfClosest = tempPrice;
        speedOfClosest = tempSpeed;
   }

}

/* Now, modelOfClosest is the model whose price is closest to
       target.  We must get its manufacturer with a single-row select
*/

if (priceOfClosest == NULL ) /* no data fetched */
  /* print error message and exit */

SQLPrepare (execStat,
                "SELECT maker FROM Product WHERE model = ?", SQL_NTS);
SQLBindParameter(execStat, 1,…,modelOfClosest, …);
SQLExecute(execStat);
SQLBindCol(execStat, 1, SQLCHAR, &manf, sizeof(manf), &colInfo);

/* print manf */
```

b)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR model[4], maker;
SQLFLOAT minSpeed;
SQLINTEGER minRam, minHd, minScreen;
SQLFLOAT speed;
SQLINTEGER ram, hd, screen;
SQLINTEGER colInfo;

/* ask user for minimum speed, ram, hd size, and screen size */

errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf("Error for SQL_HANDLE_ENV.\n");
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf("Error for SQL_ HANDLE_DBC.\n");
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf("Error for SQL_ HANDLE_STMT.\n");
   exit(1);
}
```

```
SQLPrepare(execStat,
             "SELECT model, speed, ram, hd, screen, price, maker " ||
               "FROM Laptop l, Product p " ||
                  "WHERE speed >= ? AND " ||
                        "ram >= ? AND " ||
                        "hd >= ? AND " ||
                        "screen >= ? AND " ||
                        "l.model = p.model",
             SQL_NTS);

SQLBindParameter(execStat, 1, SQL_FLOAT, …, minSpeed, …);
SQLBindParameter (execStat, 2, SQL_INTEGER, …, minRam, …);
SQLBindParameter (execStat, 3, SQL_ INTEGER, …, minHd, …);
SQLBindParameter (execStat, 4, SQL_ INTEGER, …, minScreen, …);

SQLExecute(execStat);

SQLBindCol(execStat, 1, SQL_CHAR, model, sizeof(model), &colInfo);
SQLBindCol(execStat, 2, SQL_FLOAT, speed,
                                      sizeof(speed), &colInfo);
SQLBindCol(execStat, 3, SQL_INTEGER, ram,
                                      sizeof(ram), &colInfo);
SQLBindCol(execStat, 4, SQL_INTEGER, hd,
                                      sizeof(hd), &colInfo);
SQLBindCol(execStat, 5, SQL_INTEGER, screen,
                                      sizeof(screen), &colInfo);
SQLBindCol(execStat, 6, SQL_INTEGER, price,
                                      sizeof(price), &colInfo);
SQLBindCol(execStat, 7, SQL_CHAR, maker,
                                      sizeof(maker), &colInfo);

while(SQLFetch(execStat) != SQL_NO_DATA) {

   if( FOUND )
     /* print fetched info */
}
```

c)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR maker, model[4], type[10], color[6];
SQLFLOAT speed;
SQLINTEGER ram, hd, screen, price;
SQLINTEGER colInfo;

/* ask user for minimum speed, ram, hd size, and screen size */

errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf("Error for SQL_HANDLE_ENV.\n");
   exit(1);
}
```

```
errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf(”Error for SQL_ HANDLE_DBC.\n”);
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf(”Error for SQL_ HANDLE_STMT.\n”);
   exit(1);
}

/* get PCs made by the manufacturer */

SQLPrepare(execStat,
             “SELECT * FROM PC WHERE model IN (” ||
                              “SELECT model FROM Product ” ||
                                  “WHERE maker = ? AND ” ||
                                        “type = ‘pc’”,

             SQL_NTS);

SQLBindParameter(execStat, 1, SQL_CHAR, …, maker, …);

SQLExecute(execStat);

SQLBindCol(execStat, 1, SQL_CHAR, model, sizeof(model), &colInfo);
SQLBindCol(execStat, 2, SQL_FLOAT, speed,
                                       sizeof(speed), &colInfo);
SQLBindCol(execStat, 3, SQL_INTEGER, ram,
                                       sizeof(ram), &colInfo);
SQLBindCol(execStat, 4, SQL_INTEGER, hd,
                                       sizeof(hd), &colInfo);
SQLBindCol(execStat, 5, SQL_INTEGER, price,
                                       sizeof(price), &colInfo);

while(SQLFetch(execStat) != SQL_NO_DATA) {

   if( FOUND )
     /* print fetched info */
}
/* get Laptops made by the manufacturer */

SQLPrepare(execStat,
             “SELECT * FROM Laptop WHERE model IN (” ||
                              “SELECT model FROM Product ” ||
                                  “WHERE maker = ? AND ” ||
                                        “type = ‘laptop’”,

             SQL_NTS);

SQLBindParameter(execStat, 1, SQL_CHAR, …, maker, …);

SQLExecute(execStat);
```

```
SQLBindCol(execStat, 1, SQL_CHAR, model, sizeof(model), &colInfo);
SQLBindCol(execStat, 2, SQL_FLOAT, speed,
                               sizeof(speed), &colInfo);
SQLBindCol(execStat, 3, SQL_INTEGER, ram,
                               sizeof(ram), &colInfo);
SQLBindCol(execStat, 4, SQL_INTEGER, hd,
                               sizeof(hd), &colInfo);
SQLBindCol(execStat, 5, SQL_INTEGER, screen,
                               sizeof(screen), &colInfo);
SQLBindCol(execStat, 6, SQL_INTEGER, price,
                               sizeof(price), &colInfo);

while(SQLFetch(execStat) != SQL_NO_DATA) {

   if( FOUND )
     /* print fetched info */
}

/* get Printers made by the manufacturer */

SQLPrepare(execStat,
             "SELECT * FROM Printer WHERE model IN (" ||
                             "SELECT model FROM Product " ||
                                 "WHERE maker = ? AND " ||
                                       "type = 'printer'",

             SQL_NTS);

SQLBindParameter(execStat, 1, SQL_CHAR, …, maker, …);

SQLExecute(execStat);

SQLBindCol(execStat, 1, SQL_CHAR, model, sizeof(model), &colInfo);
SQLBindCol(execStat, 2, SQL_CHAR, color, sizeof(color), $colInfo);
SQLBindCol(execStat, 3, SQL_CHAR, type, sizeof(type), $colInfo);
SQLBindCol(execStat, 4, SQL_INTEGER, price, sizeof(price), &colInfo);
```

d)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLINTEGER total_budget, rest_budget, pc_price, printer_price;
SQLCHAR pc_model[4], printer_model[4], color[6];
SQLFLOAT min_speed;


errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf("Error for SQL_HANDLE_ENV.\n");
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);
```

```
if(errCode2) {
   printf("Error for SQL_ HANDLE_DBC.\n");
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf("Error for SQL_ HANDLE_STMT.\n");
   exit(1);
}

SQLPrepare(execStat,
             "SELECT model, price FROM PC
               WHERE speed >= ? AND price <= ?
                       ORDER BY price",
             SQL_NTS);

/* ask user for budget & the minimum speed of pc */

/* get the cheapest PC of the minimum speed */

SQLBindParameter(execStat, 1, SQL_FLOAT, …, min_speed, …);
SQLBindParameter(execStat, 2, SQL_INTEGER, …, total_budget, …);

SQLExecute(execStat);

SQLBindCol(execStat, 1, SQL_CHAR, pc_model, sizeof(pc_model),&colInfo);
SQLBindCol(execStat, 2, SQL_INGETER, pc_price,
                                        sizeof(pc_price),&colInfo);

SQLFetch(execStat);

if (NOT_FOUND) {
     printf("no pc found within the budget\n");
}
else {
     printf("pc model: %s\n", pc_model);
}

/* get Printer within the budget */
rest_budget = total_budget - pc_price;
color = "true";

SQLPrepare(execStat, "SELECT model, price FROM Printer
                            WHERE price <= ? AND color = ?
                                            ORDER BY price", SQL_NTS);

SQLBindParameter(execStat, 1, SQL_INTEGER, …, rest_budget, …);
SQLBindParameter(execStat, 2, SQL_CHAR, …, color, …);

SQLExecute(execStat);

SQLBindCol(execStat, 1, SQL_CHAR, print_model,
                                         sizeof(print_model),&colInfo);
SQLBindCol(execStat, 2, SQL_INGETER, print_price,
                                         sizeof(print_price),&colInfo);
```

```
SQLFetch(execStat);

if(NOT_FOUND) {
    color = "false";

    SQLBindParameter(execStat, 1, SQL_INTEGER, …, rest_budget, …);
    SQLBindParameter(execStat, 2, SQL_CHAR, …, color, …);

    SQLExecute(execStat);

    SQLBindCol(execStat, 1, SQL_CHAR, print_model,
                                     sizeof(print_model),&colInfo);
    SQLBindCol(execStat, 2, SQL_INGETER, print_price,
                                     sizeof(print_price),&colInfo);

    SQLFetch(execStat);

    if(NOT_FOUND)
        printf("no printer found within the budget\n");
    else
        printf("printer model: %s\n", printer_model);

}
else
   printf("printer model: %s\n", printer_model);

}
```

e)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR pmodel[4], pmaker, ptype[6];
SQLFLOAT pspeed;
SQLINTEGER pram, phd, pscreen, pprice, count;
SQLINTEGER colInfo;

/* ask user for minimum speed, ram, hd size, and screen size */

errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf("Error for SQL_HANDLE_ENV.\n");
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf("Error for SQL_ HANDLE_DBC.\n");
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);
```

```
if(errCode3) {
   printf(”Error for SQL_ HANDLE_STMT.\n”);
   exit(1);
}

/* ask user for manufacturer, model, speed, RAM, hard-disk, */
/* & price of a new PC */

SQLPrepare(execStat,
             “SELECT COUNT(*) FROM PC WHERE model = ?”,
             SQL_NTS);

SQLBindParameter(execStat, 1, SQL_CHAR, …, pmodel, …);

SQLExecute(execStat);

SQLBindCol(execStat, 1, SQL_INTEGER, count, sizeof(count), &colInfo);

SQLFetch(execStat);

if (count  > 0) {
      Printf(“Warnning: The PC model already exists\n”);
}
else {

   SQLPrepare(execStat, “INSERT INTO Product VALUES(?, ?, ?)”,
                                                        SQL_NTS);
   SQLBindParameter(execStat, 1, SQL_CHAR, …, pmaker, …);
   SQLBindParameter(execStat, 2, SQL_CHAR, …, pmodel, …);
   SQLBindParameter(execStat, 3, SQL_CHAR, …, ptype, …);
   SQLExecute(execStat);


   SQLPrepare(execStat, “INSERT INTO PC VALUES(?, ?, ?, ?, ?)”,
                                                        SQL_NTS);
   SQLBindParameter(execStat, 1, SQL_CHAR, …, pmodel, …);
   SQLBindParameter(execStat, 2, SQL_FLOAT, …, pspeed, …);
   SQLBindParameter(execStat, 3, SQL_INTEGER, …, pram, …);
   SQLBindParameter(execStat, 4, SQL_INTEGER, …, phd, …);
   SQLBindParameter(execStat, 5, SQL_INTEGER, …, pprice, …);
   SQLExecute(execStat);

}
```

9.5.2

a)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR cclass[20], maxFirepowerClass[20];
SQLFLOAT firepower, maxFirepower;
SQLINTEGER cnumGuns, cbore;
```

```
SQLINTEGER colInfo;


errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf(”Error for SQL_HANDLE_ENV.\n”);
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf(”Error for SQL_ HANDLE_DBC.\n”);
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf(”Error for SQL_ HANDLE_STMT.\n”);
   exit(1);
}


SQLExecDirect(execStat,
             “SELECT class, numGuns, bore FROM Classes”, SQL_NTS);

SQLBindCol(execStat, 1, SQL_CHAR, cclass, sizeof(cclass), &colInfo);
SQLBindCol(execStat, 2, SQL_INTEGER, cnumGuns, sizeof(cnumGuns),
                                                           &colInfo);
SQLBindCol(execStat, 3, SQL_INTEGER, cbore, sizeof(cbore), &colInfo);

SQLFetch(execStat);

if(NOT_FOUND) /* print message and exit */ ;

maxFirepower = cnumGuns * (power (cbore, 3));
strcpy(maxFirepowerClass, cclass);

while(1) {

    SQLFetch(execStat);

    if(NOT_FOUND) break;

    firepower = cnumGuns * (power (cbore, 3));

    if( firepower > maxFirepower )
    {
        maxFirepower = firepower;
        strcpy(maxFirepowerClass, cclass);
    }
}

printf("Class of maximum firepower :%s\n", maxFirepowerClass);
```

b)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR ibattle[20], ocountry[20];
SQLINTEGER colInfo;

errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf("Error for SQL_HANDLE_ENV.\n");
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf("Error for SQL_ HANDLE_DBC.\n");
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf("Error for SQL_ HANDLE_STMT.\n");
   exit(1);
}

SQLPrepare(execStat,
             "SELECT COUNTRY FROM Classes C WHERE C.class IN (
                     SELECT S.class FROM Ships S WHERE S.name IN (
                          SELECT ship FROM Outcomes WHERE battle = ?))",

             SQL_NTS);

SQLBindParameter(execStat, 1, SQL_CHAR, …, ibattle, …);

SQLExecute(execStat);

SQLBindCol(execStat, 1, SQL_CHAR, ocountry, sizeof(ocountry),
&colInfo);

while(SQLFetch(execStat) != SQL_NO_DATA) {

    printf("contry:%s\n", ocoutry);

}
```

c)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
```

```
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR iclass[20], itype[3], icountry[20], iship[20];
SQLINTEGER inumGuns, ibore, idisplacement, ilaunched;
SQLINTEGER colInfo;


errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf(”Error for SQL_HANDLE_ENV.\n”);
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf(”Error for SQL_ HANDLE_DBC.\n”);
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf(”Error for SQL_ HANDLE_STMT.\n”);
   exit(1);
}

/* ask user for a class and other info for Classes table */

SQLPrepare(execStat,
             “INSERT INTO Classes VALUES (?, ?, ?, ?, ?, ?)”, SQL_NTS);

SQLBindParameter(execStat, 1, SQL_CHAR, …, iclass, …);
SQLBindParameter(execStat, 2, SQL_CHAR, …, itype, …);
SQLBindParameter(execStat, 3, SQL_CHAR, …, icountry, …);
SQLBindParameter(execStat, 4, SQL_INTEGER, …, inumGuns, …);
SQLBindParameter(execStat, 5, SQL_INTEGER, …, ibore, …);
SQLBindParameter(execStat, 6, SQL_INTEGER, …, idisplacement,…);

SQLExecute(execStat);

/* ask user for a ship and launched */

SQLPrepare(execStat, “INSERT INTO Ships VALUES (?, ?, ?)”, SQL_NTS);

WHILE(there_is_input)
{
   SQLBindParameter(execStat, 1, SQL_CHAR, …, iship, …);
   SQLBindParameter(execStat, 2, SQL_CHAR, …, iclass, …);
   SQLBindParameter(execStat, 3, SQL_INTEGER, …, ilaunched, …);

   SQLExecute(execStat);

   /* ask user for a ship and launched */
}
```

d)

```
#include sqlcli.h
SQLHENV myEnv;
SQLHDBC mycon;
SQLHSTMT execStat;
SQLRETURN errCode1, errCode2, errCode3;
SQLCHAR bname[20], bdate[8], newbdate[8];
SQLCHAR sname[20], lyear[4], newlyear[4];
SQLINTEGER colInfo;


errCode1 = SQLAllocHandle(SQL_HANDLE_ENV, SQL_NULL_HANDLE, &myEnv);
if(errCode1) {
   printf(”Error for SQL_HANDLE_ENV.\n”);
   exit(1);
}

errCode2 = SQLAllocHandle(SQL_HANDLE_DBC, myEnv, &myCon);

if(errCode2) {
   printf(”Error for SQL_ HANDLE_DBC.\n”);
   exit(1);
}

errCode3 = SQLAllocHandle(SQL_HANDLE_STMT, myCon, &execStat);

if(errCode3) {
   printf(”Error for SQL_ HANDLE_STMT.\n”);
   exit(1);
}


SQLExecDirect(execStat,
             “Select b.name, b.date, s.name, s.launched ” ||
                  “FROM Battles b, Outcomes o, Ships s ” ||
                     “WHERE b.name = o.battle AND ” ||
                           “o.ship = s.name AND ” ||
                           “YEAR(b.date) < s.launched ”,
             SQL_NTS);

SQLBindCol(execStat, 1, SQL_CHAR, bname, sizeof(bname), &colInfo);
SQLBindCol(execStat, 2, SQL_CHAR, bdate, sizeof(bdate), &colInfo);
SQLBindCol(execStat, 3, SQL_CHAR, sname, sizeof(sname), &colInfo);
SQLBindCol(execStat, 4, SQL_CHAR, lyear, sizeof(lyear), &colInfo);

while(SQLFetch(execStat) != SQL_NO_DATA) {

   /* prompt user and ask if a change is needed */

   if(change_battle)
   {
     /* get a new battle date to newbdate */
     SQLPrepare(execStat, “UPDATE Battles SET date = ? WHERE name = ?”,
                    SQL_NTS);
     SQLBindParameter(execStat, 1, …, newdate, …);
     SQLBindParameter(execStat, 2, …, bname, …);

     SQLExecute(execStat);
```

```
    }

    if(change_ship)
    {
      /* get a new launched year to newlyear */
      SQLPrepare(execStat, “UPDATE Ships SET launched = ? WHERE name= ?”,
                       SQL_NTS);
      SQLBindParameter(execStat, 1, …, newlyear, …);
      SQLBindParameter(execStat, 2, …, sname, …);

      SQLExecute(execStat);

    }
}
```

9.6.1

a)

```
import java.sql.*;

char manf, tempModel[4];
float tempSpeed;
int tempPrice;

Int targetPrice;
char modelOfClosest[4];
float speedOfClosest;
int priceOfClosest;


Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);


/* ask user for target price and read the answer into variable
       targetPrice */

PreparedStatement execStat = myCon.prepareStatement(
                  "SELECT model, price, speed FROM PC");
ResultSet pcs = execStat.executeQuery();

While(pcs.next()) {

    tempModel = pcs.getString(1);
    tempPrice = pcs.getInt(2);
    tempSpeed = pcs.getFloat(3);

    if( /* the 1st fetch or tempPrice closer to targetPrice */ ) {
        modelOfClosest = tempModel;
        priceOfClosest = tempPrice;
        speedOfClosest = tempSpeed;
    }

}

/* Now, modelOfClosest is the model whose price is closest to
       target.  We must get its manufacturer with a single-row select
*/

if (priceOfClosest == NULL ) /* no data fetched */
  /* print error message and exit */

PreparedStatement execStat2 = myCon.prepareStatement(
                  "SELECT maker FROM Product WHERE model = ?");
execStat2.setString(1, modelOfClosest);

ResultSet makers = execStat2.executeQuery();

/* print manf */
```

b)

```
import java.sql.*;

char model[4], maker;
float minSpeed;
int minRam, minHd, minScreen;
float speed;
int ram, hd, screen;


Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);

/* ask user for minimum speed, ram, hd size, and screen size */

PreparedStatement execStat = myCon.prepareStatement(
             "SELECT model, speed, ram, hd, screen, price, maker " +
               "FROM Laptop l, Product p " +
                  "WHERE speed >= ? AND " +
                        "ram >= ? AND " +
                        "hd >= ? AND " +
                        "screen >= ? AND " +
                        "l.model = p.model");

execStat.setFloat(1, minSpeed);
execStat.setInt(2, minRam);
execStat.setInt(3, minHd);
execStat.setInt(4, minScreen);

ResultSet products = execStat.executeQuery();

While(products.next()) {

    model = getString(1);
    speed = getFloat(2);
    ram = getInt(3);
    hd = getInt(4);
    screen = getInt(5);
    price = getInt(6);
    maker = getString(7);

    /* print fetched info */

}
```

c)

```
import java.sql.*;

char maker, model[4], type[10], color[6];
float speed;
int ram, hd, screen, price;
```

```
Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);

/* ask user for manufacturer */


/* get PCs made by the manufacturer */

PreparedStatement execStat = myCon.prepareStatement(
             "SELECT * FROM PC WHERE model IN (" +
                             "SELECT model FROM Product " +
                                 "WHERE maker = ? AND " +
                                       "type = 'pc'");
execStat.setString(1, maker);

ResultSet pcs = execStat.executeQuery();

while (pcs.next()) {

   model = pcs.getString(1);
   speed = pcs.getFloat(2);
   ram = pcs.getInt(3);
   hd = pcs.getInt(4);
   price = pcs.getInt(5);

   /* print fetched info */

}

/* get Laptops made by the manufacturer */

PreparedStatement execStat2 = myCon.prepareStatement(
             "SELECT * FROM Laptop WHERE model IN (" +
                             "SELECT model FROM Product " +
                                 "WHERE maker = ? AND " +
                                       "type = 'laptop'");

execStat.setString(1, maker);

ResultSet laptops = execStat2.executeQuery();

while (laptops.next()) {

   model = laptops.getString(1);
   speed = laptops.getFloat(2);
   ram = laptops.getInt(3);
   hd = laptops.getInt(4);
   screen = laptops.getInt(5);
   price = laptops.getInt(6);

   /* print fetched info */
}
```

```
/* get Printers made by the manufacturer */

PreparedStatement execStat3 =
             "SELECT * FROM Printer WHERE model IN (" +
                             "SELECT model FROM Product " +
                                 "WHERE maker = ? AND " +
                                       "type = 'printer'");


execStat3.setString(1, maker);

ResultSet printers = execStat3.executeQuery();

while (printers.next()) {

   model = printers.getString(1);
   color = printers.getString(2);
   type = printers.getString(3);
   price = printers.getInt(4);

   /* print fetched info */

}
```

d)

```
import java.sql.*;

int total_budget, rest_budget, pc_price, printer_price;
char pc_model[4], printer_model[4], color[6];
float min_speed;


Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);


/* ask user for budget & the minimum speed of pc */

/* get the cheapest PC of the minimum speed */
PreparedStatement execStat = myCon.prepareStatement(
                                 "SELECT model, price FROM PC
                                     WHERE speed >= ? AND price <= ?
                                                    ORDER BY price");
execStat.setFloat(1, min_speed);
execStat.setInt(2, total_budget);

ResultSet rs = execStat.executeQuery();

pc_model = rs.getString(1);
pc_price = rs.getInt(2);

if (!rs.next()) {
     printf("no pc found within the budget\n");
}
```

```
else {
     printf("pc model: %s\n", pc_model);
}

/* get Printer within the budget */
rest_budget = total_budget – pc_price;
color = "true";

PreparedStatement execStat = myCon.prepareStatement(
                                 "SELECT model, price FROM Printer
                                       WHERE price <= ? AND color = ?
                                                 ORDER BY price");
execStat.setInt(1, rest_budget);
execStat.setString(2, color);

ResultSet rs = execStat.executeQuery();

print_model = rs.getString(1);
print_price = rs.getInt(2);

if (!rs.next()) {

     color = "false";

     execStat.setInt(1, rest_budget);
     execStat.setString(2, color);

     ResultSet rs = execStat.executeQuery();

     print_model = rs.getString(1);
     print_price = rs.getInt(2);

     if (!rs.next())
        printf("no printer found within the budget\n");
     else
        printf("printer model: %s\n", printer_model);

}
else {
     printf("printer model: %s\n", printer_model);
}
```

e)

```
import java.sql.*;

char pmodel[4], pmaker, ptype[6];
float pspeed;
int pram, phd, pscreen, pprice, count;

char maker, model[4], type[10], color[6];
float speed;
int ram, hd, screen, price;


Class.forName("<drive name>");
```

```
Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);


/* ask user for manufacturer, model, speed, RAM, hard-disk, */
/* & price of a new PC */

PreparedStatement execStat = myCon.prepareStatement(
             "SELECT COUNT(*) FROM PC WHERE model = ?");
execStat.setString(1, pmodel);

ResultSet rs = execStat.executeQuery();

Count = rs.getInt(1);

if (count  > 0) {
      Printf("Warnning: The PC model already exists\n");
}
else {

   PreparedStatement execStat2 = myCon.prepareStatement(
                      "INSERT INTO Product VALUES(?, ?, ?)");

   execStat2.setString(1, pmaker);
   execStat2.setString(2, pmodel);
   execStat2.setString(3, ptype);
   execStat2.executeUpdate();


   PreparedStatement execStat3 = myCon.prepareStatement(
                   "INSERT INTO PC VALUES(?, ?, ?, ?, ?)");

   execStat3.setString(1, pmodel);
   execStat3.setFloat(2, pspeed);
   execStat3.setInt(3, pram);
   execStat3.setInt(4, phd);
   execStat3.setInt(5, pprice);
   execStat3.executeUpdate();

}
```

9.6.2

a)

```
import java.sql.*;

char cclass[20], maxFirepowerClass[20];
float firepower, maxFirepower;
int cnumGuns, cbore;

Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);
```

```
PreparedStatement execStat = myCon.repareStatement(
             "SELECT class, numGuns, bore FROM Classes");
ResultSet classrs = execStat.executeQuery();

if(!classrs.next())
  /* print message and exit */ ;

cclass = classrs.getString(1);
cnumGuns = classrs.getString(2);
cbore = classrs.getString(3);

maxFirepower = cnumGuns * (power (cbore, 3));
maxFirepowerClass = cclass;

while(classrs.next()) {

    cclass = classrs.getString(1);
    cnumGuns = classrs.getString(2);
    cbore = classrs.getString(3);

    firepower = cnumGuns * (power (cbore, 3));

    if( firepower > maxFirepower )
    {
        maxFirepower = firepower;
        maxFirepowerClass = cclass;
    }
}

/* print maxFirepowerClass */
```

b)

```
import java.sql.*;

char ibattle[20], ocountry[20];
int colInfo;

Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);


PreparedStatement execStat = myCon.prepareStatement(
             "SELECT COUNTRY FROM Classes C WHERE C.class IN (
                   SELECT S.class FROM Ships S WHERE S.name IN (
                        SELECT ship FROM Outcomes WHERE battle = ?))");

execStat.setString(1, ibattle);
ResultSet classrs = execStat.executeQuery();

while(classrs.next()) {

   ocountry = classrs.getString(1);
```

```
   /* print ocountry */
}
```

c)

```
import java.sql.*;

char iclass[20], itype[3], icountry[20], iship[20];
int inumGuns, ibore, idisplacement, ilaunched;

Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);


/* ask user for a class and other info for Classes table */

PreparedStatement execStat = myCon.prepareStatement(
             "INSERT INTO Classes VALUES (?, ?, ?, ?, ?, ?)");

execStat.setString(1, iclass);
execStat.setString(2, itype);
execStat.setString(3, icountry);
execStat.setInt(4, inumGuns);
execStat.setInt(5, ibore);
execStat.setInt(6, idisplacement);

execStat.executeUpdate();

/* ask user for a ship and launched */

PreparedStatement execStat2 = myCon.prepareStatement(
              "INSERT INTO Ships VALUES (?, ?, ?)");

while(there_is_input)
{
   execStat2.setSting(1, iship);
   execStat2.setSting(2, iclass);
   execStat2.setSting(3, ilaunched);

   execStat2.executeUpdate();

   /* ask user for a ship and launched */
}
```

d)

```
import java.sql.*;

char bname[20], bdate[8], newbdate[8];
char sname[20], lyear[4], newlyear[4];

Class.forName("<drive name>");

Connection myCon =
          DriverManager.getConnection(<URL>, <username>, <password>);
```

```
PreparedStatement execStat = myCon.prepareStatement(
             "Select b.name, b.date, s.name, s.launched " +
                  "FROM Battles b, Outcomes o, Ships s " +
                     "WHERE b.name = o.battle AND " +
                           "o.ship = s.name AND " +
                           "YEAR(b.date) < s.launched ");

ResultSet rs = execStat.executeQuery();

while(rs.next()) {


   bname = rs.getString(1);
   bdate = rs.getString(2);
   sname = rs.getString(3);
   lyear = rs.getString(4);

   /* prompt user and ask if a change is needed */

   if(change_battle)
   {
     /* get a new battle date to newbdate */
     PreparedStatement execStat2 = myCon.prepareStatement(
                         "UPDATE Battles SET date = ? WHERE name = ?");
     execStat2.setString(1, newdate);
     execStat2.setString(2, bname);
     execStat2.executeUpdate();
   }

   if(change_ship)
   {
     /* get a new launched year to newlyear */
     PreparedStatement execStat3 = myCon.prepareStatement(
                        "UPDATE Ships SET launched = ? WHERE name= ?");

     execStat3.setString(1, newlyear);
     execStat3.setString(2, sname);
     execStat3.executeUpdate();

   }
}
```

9.7.1

a)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);

/* ask user for target price and read the answer into variable
       targetPrice */

$pcs = $myCon->query("SELECT model, price, speed FROM PC");

while($tuple = $pcs->fetchRow()) {

    $tempModel = $tuple[0];
    $tempPrice = $tuple[1];
    $tempSpeed = $tuple[2];

    if( /* the 1st fetch or tempPrice closer to targetPrice */ ) {
        $modelOfClosest = $tempModel;
        $priceOfClosest = $tempPrice;
        $speedOfClosest = $tempSpeed;
    }

}

/* Now, modelOfClosest is the model whose price is closest to
       target.  We must get its manufacturer with a single-row select
*/

if (priceOfClosest == NULL ) /* no data fetched */
  /* print error message and exit */

$prepQuery = #myCon->prepare(
                  "SELECT maker FROM Product WHERE model = ?");
$makers = $myCon->execute($prepQuery, $priceOfCloset);

/* print manf */
```

b)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);

/* ask user for minimum speed, ram, hd size, and screen size & get into
args array */

$prepQuery = $myCon->prepare(
             "SELECT model, speed, ram, hd, screen, price, maker " .
               "FROM Laptop l, Product p " .
                  "WHERE speed >= ? AND " .
                        "ram >= ? AND " .
                        "hd >= ? AND " .
                        "screen >= ? AND " .
                        "l.model = p.model");
```

```
$products = $myCon->execute($prepQuery, $args);

while($tuple = $products->fetchRow()) {

    $model = $tuple[0];
    $speed = $tuple[1];
    $ram = $tuple[2];
    $hd = $tuple[3];
    $screen = $tuple[4];
    $price = $tuple[5];
    $maker = $tuple[6];

    /* print fetched info */

}
```

c)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);

/* ask user for manufacturer */


/* get PCs made by the manufacturer */

$prepQuery1 = $myCon->prepare(
             "SELECT * FROM PC WHERE model IN (" .
                             "SELECT model FROM Product " .
                                 "WHERE maker = ? AND " .
                                       "type = 'pc'");
$pcs = $myCon->execute($prepQuery1, $maker);

while ($tuple = $pcs->fetchRow()) {

   $model = $tuple[0];
   $speed = $tuple[1];
   $ram = $tuple[2];
   $hd = $tuple[3];
   $price = $tuple[4];

   /* print fetched info */

}

/* get Laptops made by the manufacturer */

$prepQuery2 = $myCon->prepare(
             "SELECT * FROM Laptop WHERE model IN (" +
                             "SELECT model FROM Product " +
                                 "WHERE maker = ? AND " +
                                       "type = 'laptop'");

$laptops = $myCon->execute($prepQuery2, $maker);
```

```
while ($tuple = $laptops->fetchRow()) {

   $model = $tuple[0];
   $speed = $tuple[1];
   $ram = $tuple[2];
   $hd = $tuple[3];
   $screen = $tuple[4];
   $price = $tuple[5];

   /* print fetched info */
}

/* get Printers made by the manufacturer */

$prepQuery3 = $myCon->prepare(
             "SELECT * FROM Printer WHERE model IN (" +
                             "SELECT model FROM Product " +
                                 "WHERE maker = ? AND " +
                                       "type = 'printer'");


$printers = $myCon->execute($prepQuery3, $maker);

while ($tuple = $printers->fetchRow()) {

   $model = $tuple[0];
   $color = $tuple[1];
   $type = $tuple[2];
   $price = $tuple[3];

   /* print fetched info */

}
```

d)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);


/* ask user for budget & the minimum speed of pc */

/* get the cheapest PC of the minimum speed */
$prepQuery1 = $myCon.prepare(
                  "SELECT model, price FROM PC
                                WHERE speed >= ? AND price <= ?
                                                 ORDER BY price");

$pcs = $myCon.execute($prepQuery1, $args1);
                             /* $args1 - min_speed, total_budget */

if ($tuple = $pcs->fetchRow()) {

   $pc_model = $tuple[0];
```

```
   $pc_price = $tuple[1];

   /* print fetched info */

}
else
   /* print no pc found within the budget message */


/* get Printer within the budget */
$rest_budget = $total_budget – $pc_price;
$color = “true”;

$prepQuery2 = $myCon.prepare(
                  “SELECT model, price FROM Printer
                                      WHERE price <= ? AND color = ?
                                                ORDER BY price”);

$printers = $myCon.execute($prepQuery1, $args2);
                              /* $args2 – rest_budget, color */

if ($tuple = $pcs->fetchRow()) {

   $printer_model = $tuple[0];
   $printer_price = $tuple[1];

   /* print fetched info */

}
else {

   $color = “false”
   $printers = $myCon.execute($prepQuery1, $args2);
                                 /* $args2 – rest_budget, color */

   if ($tuple = $pcs->fetchRow()) {

      $printer_model = $tuple[0];
      $printer_price = $tuple[1];

      /* print fetched info */
   }
   else
     /* print no printer found within the budget message */
}

e)
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);


/* ask user for manufacturer, model, speed, RAM, hard-disk, */
/* & price of a new PC */

$prepQuery1 = $myCon.prepare(
```

```
             "SELECT COUNT(*) FROM PC WHERE model = ?");

$rs = $myCon.execute($prepQuery1, $pmodel);
$tuple = $rs->fetchRow();
$count = $tuple[0];

if ($count  > 0) {
      Printf("Warnning: The PC model already exists\n");
}
else {

   $prepStmt2 = $myCon.prepare(
                       "INSERT INTO Product VALUES(?, ?, ?)");
   /* pmaker, pmode, & ptype are $args1 */

   $result = $myCon->execute($prepStmt2, $args1);

   $prepStmt3 = $myCon->prepare(
                    "INSERT INTO PC VALUES(?, ?, ?, ?, ?)");

   /* pmodel, pspeed, pram, phd, & pprice are in are $args2 */

   $result = $myCon->execute($prepStmt3, $args2);

}
```

9.7.2

a)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);

$classrs = $myCon->query(
             "SELECT class, numGuns, bore FROM Classes");
$tuple = $classrs->fetchRow();

if(!$tuple)
  /* print message and exit */ ;

$cclass = $tuple[0];
$cnumGuns = $tuple[1];
$cbore = $tuple[2];

$maxFirepower = $cnumGuns * ($cbore * $cbore * $cbore);
$maxFirepowerClass = $cclass;

while($tuple = $classrs->fetchRow()) {

  $cclass = $tuple[0];
  $cnumGuns = $tuple[1];
  $cbore = $tuple[2];

  $firepower = $cnumGuns * ($cbore * $cbore * $cbore);
```

```
  if( $firepower > $maxFirepower )
  {
      $maxFirepower = $firepower;
      $maxFirepowerClass = $cclass;
  }
}

/* print maxFirepowerClass */
```

b)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);

$prepQuery = $myCon->prepare(
             "SELECT COUNTRY FROM Classes C WHERE C.class IN ( " .
                  "SELECT S.class FROM Ships S WHERE S.name IN ( " .
                       "SELECT ship FROM Outcomes WHERE battle = ?))");
/* battle in $ibattle */

$classrs = $myCon->execute($prepQuery, $ibattle);

while($tuple = $classrs->fetchRow()) {

   $ocountry = $tuple[0];

   /* print ocountry */
}
```

c)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);

/* ask user for a class and other info for Classes table */

$prepStmt1 = $myCon->prepare(
             "INSERT INTO Classes VALUES (?, ?, ?, ?, ?, ?)");

/* $iclass, $itype, $icountry, $inumGuns, $ibore, & $idisplacement in
$args1 */

$result = $myCon->execute($prepStmt1, $args1);

/* ask user for a ship and launched */

$prepStmt2 = $myCon->prepare(
              "INSERT INTO Ships VALUES (?, ?, ?)");

/* $iship, $iclass, $ilaunched in $args2 */
```

```
while(there_is_input)
{
   $result = $myCon->execute($prepStmt2, $args2);

   /* ask user for a ship and launched & get into args2*/
}
```

d)

```
include(DB.php);

$myCon = DB::connect(<vendor>:://<username>:<password>
              <hostname>/<databasename>);


$rs = $myCon->query(
             "SELECT b.name, b.date, s.name, s.launched " .
                  "FROM Battles b, Outcomes o, Ships s " .
                     "WHERE b.name = o.battle AND " .
                           "o.ship = s.name AND " .
                           "YEAR(b.date) > s.launched ");


while($tuple = $rs.fetchRow()) {

   $bname = $tuple[0];
   $bdate = $tuple[1];
   $sname = $tuple[2];
   $lyear = $tuple[3];

   /* prompt user and ask if a change is needed */

   if(change_battle)
   {
     /* get a new battle date to newbdate in $args2*/
     $prepStmt2 = $myCon->prepare(
                        "UPDATE Battles SET date = ? WHERE name = ?");
     $result = $myCon->execute($prepStmt2, $args2);
   }

   if(change_ship)
   {
     /* get a new launched year to newlyear in $args3 */
     $prepStmt3 = $myCon->prepare(
                       "UPDATE Ships SET launched = ? WHERE name= ?");

     $result = $myCon->execute($prepStmt3, $args3);

   }
}
```

Solutions Manual

Chapter 10

Section 10.1

Exercise 10.1.1

(a) SELECT on MovieStar, SELECT on MovieExec.

(b) SELECT on MovieExec, SELECT on Movies, SELECT on StarsIn.

(c) SELECT on Movies, SELECT on Studio, INSERT on Studio (or INSERT(name) on Studio).

(d) DELETE on StarsIn.

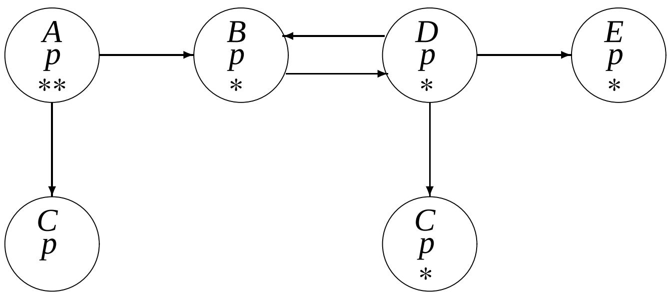
(e) UPDATE on MovieExec (or UPDATE(name) on MovieExec).

(f) REFERENCES on MovieStar (or REFERNCES(gender, name) on MovieStar).

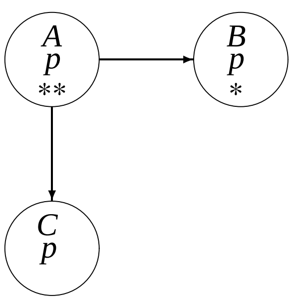
(g) REFERENCES on Studio, REFERENCES on MovieExec (or REFERENCES(name, presC#) on Studio, REFERENCES(cert#, netWorth) on MovieExec).

Exercise 10.1.2

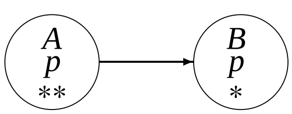
After step (4), the grant diagram is as follows:



After step (5), the grant diagram is as follows:

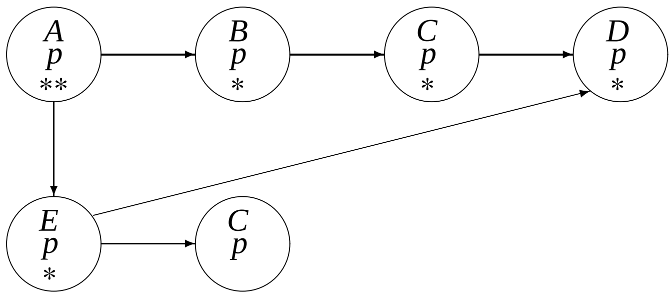


After step (6), the grant diagram is as follows:

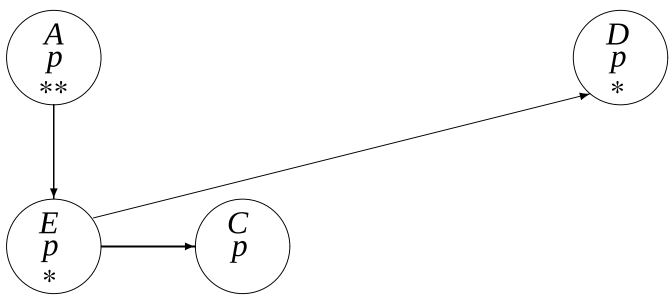


Exercise 10.1.3

After step (5), the grant diagram is as follows:

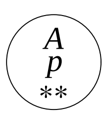


After step (6), the grant diagram is as follows:



Exercise 10.1.4

The grant diagram after the final step is as follows:



Section 10.2

Exercise 10.2.1

(a) The rules for trips that have reasonable connections are:

$$
\begin{aligned}
\texttt{Trips}(x, y, \texttt{dep}, \texttt{arr}) &\leftarrow \texttt{Flights}(\_, x, y, \texttt{dep}, \texttt{arr}) \\
\texttt{Trips}(x, y, \texttt{dep}, \texttt{arr}) &\leftarrow \texttt{Trips}(x, z, \texttt{dep1}, \texttt{arr1}) \texttt{ AND} \\
&\qquad \texttt{Trips}(z, y, \texttt{dep2}, \texttt{arr2}) \texttt{ AND} \\
&\qquad \texttt{arr1} \leqslant \texttt{dep2} - 100
\end{aligned}
$$

(b) Using the book's syntax, the SQL is:

```
WITH RECURSIVE Trips(frm, to, dep, arr) AS
  (SELECT frm, to, dep, arr
   FROM   Flights          )
  UNION
  (SELECT T.frm, F.to, T.dep, F.arr
   FROM   Trips T, Flights F
   WHERE  T.to = F.from
     AND  T.arr <= F.dep - 100     )
SELECT *
FROM   Trips;
```

Exercise 10.2.2

Because `FROM` is one of the SQL reserved words, using it as an identifier is not recommended. Note that most major vendors do not prohibit the use of reserved words when the use is not ambiguous (e.g. SELECT FROM FROM FROM is not ambiguous and will work), but such use is highly discouraged for readability and portability reasons.

Exercise 10.2.3

(a)

$$
\begin{aligned}
\mathtt{FollowOn}(x, y) &\leftarrow \mathtt{SequelOf}(x, y) \\
\mathtt{FollowOn}(x, y) &\leftarrow \mathtt{FollowOn}(x, z)\ \mathtt{AND} \\
&\quad\ \mathtt{SequelOf}(z, y)
\end{aligned}
$$

(b) Using the book's syntax, the SQL is:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT *
FROM   FollowOn;
```

(c) Using the book's syntax, the SQL is:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie, followOn
FROM   FollowOn
EXCEPT
SELECT movie, sequel
FROM   SequelOf;
```

(Similarly, NOT IN or NOT EXISTS can be used instead of EXCEPT).

(d) One of the ways is to first get all of the recursive tuples as for the original FollowOn in (a), and then subtract the those tuples that represent sequel or sequel of a sequel. Using the book's syntax, the SQL would be:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie, followOn
FROM   FollowOn
EXCEPT
(SELECT movie, sequel
 FROM   SequelOf
 UNION
 SELECT X.movie, Y.sequel
 FROM   Sequel X, Sequel Y
 WHERE  X.sequel = Y.movie);
```

Another way would be to start FollowOn tuples only from the tuples of movies that have more than two sequels (using a join similar to the one above but with three Sequel tables).

(e) We simply need to count the number of followon values per movie. Using the book's syntax, the SQL would be:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie
```

```
FROM   FollowOn
GROUP BY movie
HAVING COUNT(followon) >= 2;
```

(f) This is, in a sense, a reverse of (e) above, because to have at most one followon means that the total count of the tuples grouped by the given movie x must be no greater than 2 (one for the movie and its sequel, and the other for the sequel and its sequel). Using the book's syntax, the SQL would be:

```
WITH RECURSIVE FollowOn(movie, followOn) AS
  (SELECT movie, sequel
   FROM   SequelOf     )
  UNION
  (SELECT F.movie, S.sequel
   FROM   FollowOn F, Sequel S
   WHERE  F.followOn = S.movie)
SELECT movie, followon
FROM   FollowOn
WHERE  movie IN(SELECT movie
                FROM   FollowOn
                GROUP BY movie
                HAVING COUNT(followon) <= 2);
```

Exercise 10.2.4

(a)
```
WITH RECURSIVE Path(class, rclass) AS
  (SELECT class, rclass
   FROM   Rel          )
  UNION
  (SELECT Path.class, Rel.rclass
   FROM   Path, Rel
   WHERE  Path.rclass = Rel.class)
SELECT *
FROM   Path;
```

(b)
```
WITH RECURSIVE Path(class, rclass) AS
  (SELECT class, rclass
```

```
    FROM   Rel
    WHERE  mult = 'single')
   UNION
   (SELECT Path.class, Rel.rclass
    FROM   Path, Rel
    WHERE  Path.rclass = Rel.class
      AND  Rel.mult = 'single'    )
SELECT *
FROM   Path;
```

(c)
```
WITH RECURSIVE Path(class, rclass) AS
  (SELECT class, rclass
   FROM   Rel
   WHERE  mult = 'multi')
  UNION
  (SELECT Path.class, Rel.rclass
   FROM   Path, Rel
   WHERE  Path.rclass = Rel.class)
  UNION
  (SELECT Rel.class, Path.rclass
   FROM   Path, Rel
   WHERE  Rel.rclass = Path.class)
SELECT *
FROM   Path;
```

(d) This could be viewed as relation from (a) EXCEPT relation from (b).

```
WITH RECURSIVE PathAll(class, rclass) AS
   (SELECT class, rclass
    FROM   Rel          )
   UNION
   (SELECT PathAll.class, Rel.rclass
    FROM   PathAll, Rel
    WHERE  PathAll.rclass = Rel.class),
RECURSIVE PathSingle(class, rclass) AS
   (SELECT class, rclass
    FROM   Rel
```

```
     WHERE  mult = 'single')
    UNION
    (SELECT PathSingle.class, Rel.rclass
     FROM   PathSingle, Rel
     WHERE  PathSingle.rclass = Rel.class
       AND  Rel.mult = 'single'          )
  SELECT class, rclass
  FROM   PathAll
  EXCEPT
  SELECT class, rclass
  FROM   PathSingle
  ;
```

(e) We include the edge label as part of the recursive relation and then, basically, we build the path as in (a) except we only add edges that have an opposite label.

```
  WITH RECURSIVE Path(class, rclass, mult) AS
    (SELECT class, rclass, mult
     FROM   Rel                 )
    UNION
    (SELECT Path.class, Rel.rclass, Rel.mult
     FROM   Path, Rel
     WHERE  Path.rclass  = Rel.class
       AND  Path.mult   <> Rel.mult )
  SELECT *
  FROM   Path;
```

(f)
```
  WITH RECURSIVE Path(class, rclass) AS
    (SELECT class, rclass
     FROM   Rel
     WHERE  mult = 'single')
    UNION
    (SELECT Path.class, Rel.rclass
     FROM   Path, Rel
     WHERE  Path.rclass = Rel.class
       AND  Rel.mult = 'single'    )
```

```
SELECT *
FROM   Path X
WHERE  EXISTS(SELECT 1
              FROM   Path Y
              WHERE  Y.class  = X.rclass
                AND  Y.rclass = X.class )
;
```

Section 10.3

Exercise 10.3.1

(a)
```
Stars(name, address, birthdate)
Movies(title, year, length, stars({*Stars}))
```

(b)
```
Stars(name, address, birthdate)
Movies(title, year, length, stars({*Stars}))
Studios(name, address, movies({*Movies}))
```

(c)
```
Stars(name, address, birthdate)
Movies(title, year, length, studio(name, address), stars({*Stars}))
```

Exercise 10.3.2

```
Customers(name, address, phone, ssNo, accts({*Accounts}))
Accounts(number, type, balance, owners({*Customers}))
```

Exercise 10.3.3

```
Customers(name, address, phone, ssNo, accts({*Accounts}))
Accounts(number, type, balance, owner(*Customers))
```

Exercise 10.3.4

```
Players(name)
Teams(name, players({*Players}), captain(*Players), colors)
Fans(name, fav_teams({*Teams}), fav_players({*Players}), fav_color)
```

Exercise 10.3.5

```
People(name, mother(*People), father(*People), children({*People}))
```

Section 10.4

Exercise 10.4.1

```
Movies(
 title        TitleType,
 year         YearType,
 length       DurationType,
 genre        GenreType,
 studioName   BusinessNameType,
 producerC#   CertificateType
)

MovieStar(
 name         PersonNameType,
 address      AddressType,
 gender       GenderType,
 birthdate    DateType
)

StarsIn(
 movieTitle    TitleType,
 movieYear     YearType,
 starName      PersonNameType
)

MovieExec(
 name          PersonNameType,
 address       AddressType,
 cert#         CertificateType,
 netWorth      CurrencyType
)

Studio(
```

```
 name          BusinessNameType,
 address       AddressType,
 presC#        CertificateType
)
```

Exercise 10.4.2

(a)
```
CREATE TYPE NameType AS(
    first   VARCHAR(30),
    middle  VARCHAR(50),
    last    VARCHAR(30),
    title   VARCHAR(10)
   );
```

(b)
```
CREATE TYPE PersonType AS(
    name    NameType,
    mother  REF(PersonType),
    father  REF(PersonType)
  );
```

(c)
```
CREATE TYPE MarriageType AS(
    date     DATE,
    husband  REF(PersonType),
    wife     REF(PersonType)
  );
```

Exercise 10.4.3

```
CREATE TYPE ProductType AS(
  maker       CHAR(5),
  model       INTEGER,
  type        CHAR(8)
);

CREATE TABLE Product OF ProductType(
  REF IS ProductId SYSTEM GENERATED
);
```

```
CREATE TABLE PC(
  model       REF(ProductType) SCOPE Product,
  speed       DECIMAL(5,2),
  ram         INTEGER,
  hd          INTEGER
  price       DECIMAL(10,2)
);

CREATE TABLE Laptop(
  model       REF(ProductType) SCOPE Product,
  speed       DECIMAL(5,2),
  ram         INTEGER,
  hd          INTEGER
  screen      DECIMAL(5,2),
  price       DECIMAL(10,2)
);

CREATE TABLE Printer(
  model       REF(ProductType) SCOPE Product,
  color       CHAR(1),
  type        VARCHAR(10),
  price       DECIMAL(10,2)
);
```

Exercise 10.4.4

Model attribute in Products cannot be a reference to the tuple in the relation for that type of product because that would create a circular reference situation where the model is a reference to the relation itself which has a model attribute but is a reference, etc. There would not be a column that stores the actual model values.

Exercise 10.4.5

```
CREATE TYPE ClassType AS (
  class       VARCHAR(30),
  type        CHAR(2),
  country     VACHAR(30),
  numGuns     INTEGER,
```

```
  bore        INTEGER,
  disp        INTEGER
);

CREATE TYPE ShipType AS (
  name        VARCHAR(30),
  class       REF(ClassType),
  launched    INTEGER
);

CREATE TYPE BattleType AS (
  name        VARCHAR(30),
  date        DATE
);

CREATE TYPE OutcomeType AS (
  ship        REF(ShipType),
  battle      REF(BattleType),
  result      VARCHAR(10)
);

CREATE TABLE Classes OF ClassType (
  REF IS classID SYSTEM GENERATED
);

CREATE TABLE Ships OF ShipType(
   REF IS shipID SYSTEM GENERATED
);

CREATE TABLE Battles OF TYPE BattleType(
   REF IS battleID SYSTEM GENERATED
);

CREATE TABLE Outcomes OF TYPE OutcomeType(
   REF IS outcomeID SYSTEM GENERATED
);
```

Section 10.5

Exercise 10.5.1

(a)
```
SELECT star->name
FROM   StarsIn
WHERE  movie->title = 'Dogma';
```

(b)
```
SELECT DISTINCT movie->title, movie->year
FROM   StarsIn
WHERE  star->address.city() = 'Malibu';
```

(c)
```
SELECT movie
FROM   StarsIn
WHERE  star->name = 'Melanie Griffith';
```

(d)
```
SELECT   movie->title, movie->year
FROM     StarsIn
GROUP BY movie->title, movie->year
HAVING   COUNT(*) >= 5;
```

Exercise 10.5.2

(a)
```
SELECT model->maker
FROM   PC
WHERE  hd > 60;
```

(b)
```
SELECT DISTINCT model->maker
FORM   Printers
WHERE  type = 'laser';
```

(c)
```
WITH MaxSpeedsPerMaker(maker, maxSpeed) AS(
  SELECT   model->maker, MAX(speed)
  FROM     Laptops
  GROUP BY model->maker                   ),
MakerTopModel(maker,topModel) AS(
  SELECT M.maker, L.model->model
  FROM   Laptops L, MaxSpeedsPerMaker M
  WHERE  L.model->maker = M.maker
    AND  L.speed        = maxSpeed     )
```

```
SELECT model->model, topModel
FROM   Laptops L, MakerTopModel M
WHERE  L.model->maker = M.maker
;
```

Exercise 10.5.3

(a)
```
SELECT x.name
FROM   Ships x
WHERE  x.class->disp > 35000;
```

(b)
```
SELECT DISTINCT x.battle->name
FROM   Outcomes x
WHERE  x.result = 'sunk';
```

(c)
```
SELECT DISTINCT x.class->class
FROM   Ships x
WHERE  x.launched > 1930;
```

(d)
```
SELECT DISTINCT x.battle->name
FROM   Outcomes x
WHERE  x.result = 'damaged'
  AND  x.ship->class->country = 'USA';
```

Exercise 10.5.4

```
CREATE FUNCTION StarLEG(p1 StarType,
                        p2 StarType )
RETURNS INTEGER
  IF     p1.name < p2.name THEN RETURN(-1)
  ELSEIF p1.name > p2.name THEN RETURN( 1)
  ELSE   RETURN(AddrLEG(p1.address,p2.addres))
  ENDIF
;
CREATE ORDERING FOR StarType
  ORDERING FULL BY RELATIVE WITH StarLEG;
```

Exercise 10.5.5

```
CREATE PROCEDURE DeleteStar(IN pName VARCHAR(50))
BEGIN
  DELETE FROM StarsIn
  WHERE  star->name = pName;

  DELETE FROM MovieStar x
  WHERE  x.name = pName;
END;
```

Section 10.6

Exercise 10.6.1

(a) Dimension attributes are: cust, date, proc, memory, hd, od.
Dependent attributes are: quant, price.

(b)
```
Cust(custID, name, address, phone, creditCard)
Proc(procID, manufacturer, name, model, speed)
HD(hdID, manufacturer, name, model, capacity,
   cylinders, surfaces, speed)
OD(odID, manufacturer, type, capacity, speed)
```

Exercise 10.6.2

First we could select the number of orders that had DVD disks and the number of orders that had CD disks. This would show just the totals over all orders.

```
SELECT   D1.type, COUNT(*)
FROM     Orders F, OD D1
WHERE    F.od = D1.odID
GROUP BY D1.type
HAVING   D1.type IN('DVD','CD')
;
```

Then we could drill-down to see what the totals are per month, hopefully seeing that the numbers for DVDs increase and the numbers for CDs decrease.

```
SELECT   MONTH(F.date) MONTHS, D1.type, COUNT(*)
FROM     Orders F, OD D1
WHERE    F.od = D1.odID
GROUP BY MONTHS, D1.type
HAVING   D1.type IN('DVD','CD')
;
```

Next we could drill-up to show the totals per year.

```
SELECT   YEAR(F.date) YEARS, D1.type, COUNT(*)
FROM     Orders F, OD D1
WHERE    F.od = D1.odID
GROUP BY YEARS, D1.type
HAVING   D1.type IN('DVD','CD')
;
```

Section 10.7

Exercise 10.7.1

(a) The ratio is $\left(\frac{11}{10}\right)^{10}$, or about 2.59.

(b) The ratio is $\left(\frac{3}{2}\right)^{10}$, or about 57.66.

Exercise 10.7.2

(a) Assuming the column name for SUM(val) in SalesCube is val:

```
SELECT   dealer, val
FROM     SalesCube
WHERE    model IS NULL
  AND    color = 'blue'
  AND    date IS NULL
  AND    dealer IS NOT NULL
;
```

(b) Assuming the column name for SUM(cnt) in SalesCube is cnt:

```
SELECT   cnt
FROM     SalesCube
WHERE    model = 'Gobi'
  AND    color = 'green'
  AND    date IS NULL
  AND    dealer = 'Smilin'' Sally'
;
```

(c) Assuming the column names for SUM(cnt) and SUM(val) in SalesCube are cnt and val:

```
SELECT   val/cnt
FROM     SalesCube
WHERE    model = 'Gobi'
  AND    color IS NULL
  AND    YEAR(date) = 2007
  AND    MONTH(date) = 3
  AND    dealer IS NOT NULL
;
```

Exercise 10.7.3

The rollup would not help and would make it more difficult to ensure that we do not double count the rows and only consider the rows that are in CUBE(Sales) but not in Sales.

Exercise 10.7.4

```
CREATE MATERIALIZED VIEW OrdersCube(
  cust, date, proc, memory, hd, od, tquant, tprice)
 AS(
  SELECT cust, date, proc, memory, hd, od, SUM(quant), SUM(price)
  FROM   Orders
  GROUP BY cust, date, proc, memory, hd, od)
WITH CUBE;
```

Exercise 10.7.5

(a)
```
SELECT D1.speed, MONTH(F.date), SUM(F.tquant)
FROM   OrdersCube F, Proc D1
WHERE  F.proc = D1.procID
  AND  F.cust IS NULL
  AND  YEAR(F.date) = 2007
  AND  F.memory IS NULL
  AND  F.hd IS NULL,
  AND  F.od IS NULL
GROUP BY D1.speed, MONTH(F.date)
;
```

(b)
```
SELECT D1.type, D2.type, SUM(F.tquant)
FROM   OrdersCube F, Proc D1, HD D2
WHERE  F.proc = D1.procID
  AND  F.hd   = D2.hdID
  AND  F.cust IS NULL
  AND  F.date IS NULL
  AND  F.memory IS NULL
  AND  F.od IS NULL
GROUP BY D1.type, D2.type
;
```

(c)
```
SELECT MONTH(F.date), SUM(tprice)/SUM(F.tquant)
FROM   OrdersCube F, Proc D1
WHERE  F.proc = D1.procID
  AND  D1.speed = 3.0
  AND  F.cust IS NULL
  AND  F.date >= '01/01/2005'
  AND  F.memory IS NULL
  AND  F.hd IS NULL,
  AND  F.od IS NULL
GROUP BY MONTH(F.date)
;
```

Exercise 10.7.6

Yes, other rollups could contain these tuples. Those rollups can be formed by rearranging the group by list so that columns we need to be aggregated are at the tail of the list. For instance, to include tuple

```
('Gobi', NULL, '2001-05-21', 'Friendly Fred', 152000, 7)
```

The group by list would be:

```
GROUP BY model, date, dealer, color WITH ROLLUP
```

Exercise 10.7.7

In the worst case, the fact table could have only one row, the CUBE(F) would add an additional 2^n tuples, and so the ratio would be 2^n.