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
---------------------------------
      2.48461538461538461538461538

  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        15.233333333333333333333333333
B        13.300000000000000000000000000
E        17.500000000000000000000000000
F        14.750000000000000000000000000
G        15.400000000000000000000000000

  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.