Q1 Q2	**Q1. Write SQL Statements** **Assume a given schema. Write a set of SQL statements that will include JOIN and pattern matching operations.** **Q2. Write SQL Statements** **Write a set of advanced SQL statements that will include UNION and sub-queries.**
Exercise 6.1.1: If a query has a SELECT clause SELECT A B how do we know whether *A* and *B* are two different attributes or *B* is an alias of .4?	6.1.1 Attributes must be separated by commas. Thus here B is an alias of A.
Exercise 6.1.2: Write the following queries, based on our running movie database example M ovies(title, year, length, genre, studioName, producerC#) StarsIn(movieTitle, movieYear, starName) MovieStar(name, address, gender, birthdate) MovieExec(name, address, cert#, netWorth) Studio(name, address, presC#) a) Find the address of MGM studios. b) Find Sandra Bullock's birthdate. c) Find all the stars that appeared either in a movie made in 1980 or a movie with "Love" in the title. d) Find all executives worth at least $10,000,000. e) Find all the stars who either are male or live in Malibu (have string Malibu as a part of their address).	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) (pattern matching) `SELECT name AS Star_Name` `FROM movieStar` `WHERE gender = 'M'` `OR address LIKE '% Malibu %';`
Exercise 6.1.3: Write the following queries in SQL. They refer to the database schema of Exercise 2.4.1:	6.1.3 a) `SELECT model,` `speed,` `hd`

Product(maker,model, type)
PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price)
Show the result of your queries using the data from Exercise 2.4.1.

a) Find the model number, speed, and hard-disk size for all PC's whose price is under $1000.

```
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) Do the same as (a), but rename the speedcolumn gigahertzand the hd column gigabytes.

```
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) Find the manufacturers of printers.

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

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

  7 record(s) selected.
```

d) Find the model number, memory size, and screen size for laptops costing more than $1500.

```
d)
SELECT  model,
        ram  ,
```

e) Find all the tuples in the Printer relation for color printers. Remember that coloris a boolean-valued attribute.

f) Find the model number and hard-disk size for those PC's that have a speed of 3.2 and a price less than $2000.

```
        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.

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. ```
Exercise 6.1.4: Write the following queries based on the database schema of Exercise 2.4.3: Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched) Battles(name, date) Outcomes(ship, battle, result) and show the result of your query on the data of Exercise 2.4.3. a. Find the class name and country for all classes with at least 10 guns. b. Find the names of all ships launched prior to 1918, but call the resulting column shipName. c. Find the names of ships sunk in battle and the name of the battle in which they were sunk.	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 ```

d. Find all ships that have the same name as their class.

e. Find the names of all ships that begin with the letter “R.”

f. Find the names of all ships whose name consists of three or more words (e.g., King George V).

```
  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)(pattern matching)
SELECT  name AS shipName
FROM    Ships
WHERE   name LIKE 'R%';

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

  7 record(s) selected.


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.
```

Exercise 6.1.5: Let *a* and *b* be integer-valued attributes that may be NULL in some tuples. For each of the following conditions (as may appear in a WHERE clause), describe exactly the set of (a, 6) tuples that satisfy the condition, in cluding the case where *a* and/or *b* is NULL.
a) a = 10 OR b = 20
b) a = 10 AND b = 20

c) a < 10 OR a >= 10

```
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) a = b !e) a <==b

```
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
```

Exercise 6.1.6: In Example 6.10 we discussed the query
SELECT *
FROM Movies
WHERE length <= 120 OR length > 120;
which behaves unintuitively when the length of a movie is NULL. Find a simpler, equivalent query, one with a single condition in the WHEREclause (no AND or OR of conditions).

```
6.1.6

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

Exercise 6.2.1: Using the database schema of our running movie example

```
Movies(title, year, length, genre, studioName,
producerC#) StarsIn(movieTitle, movieYear,
starName) MovieStar(name,address, gender,
birthdate) MovieExec(name, address, cert#,
netWorth)

Studio(name, address, presC#)
```

write the following queries in SQL.

a) Who were the male stars in *Titanic*?
b) Which stars appeared in movies produced by MGM in 1995?

c) Who is the president of MGM studios?

```
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
```

! d) Which movies are longer than *Gone With the Wind*?

```
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) Which executives are worth more than Merv Griffin?

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

Exercise 6.2.2: Write the following queries, based on the database schema

```
Product(maker,model, type)
PC(model, speed, ram, hd, price) Laptop(model,
speed, ram, hd, screen, price) Printer(model,
color, type, price)
```

of Exercise 2.4.1, and evaluate your queries using the data of that exercise.

a. Give the manufacturer and speed of laptops with a hard disk of at least thirty gigabytes.

```
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. Find the model number and price of all products (of any type) made by manufacturer *B*.

```
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. Find those manufacturers that sell Laptops, but not PC's.

```
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. Find those hard-disk sizes that occur in two or more PC's.

```
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. Find those pairs of PC models that have both the same speed and RAM. A pair should be listed only once; e.g., list (*i* , *j*) but not *(j,i)*.

```
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. Find those manufacturers of at least two different computers (PC's or laptops) with speeds of at least 3.0

```
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.
```

Exercise 6.2.3: Write the following queries, based on the database schema

```
Classes(class, type, country, numGuns, bore,
displacement) Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

of Exercise 2.4.3, and evaluate your queries using the data of that exercise.

6.2.3

a. Find the ships heavier than 35,000 tons.

```
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. List the name, displacement, and number of guns of the ships engaged in the battle of Guadalcanal.

```
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. List all the ships mentioned in the database. (Remember that all these ships may not appear in the Ships relation.)

```
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
```

d. Find those countries that have both battleships and battlecruisers.

e. Find those ships that were damaged in one battle, but later fought in another.

f. Find those battles with at least three ships of the same country.

```
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;
```

! Exercise 6.2.4: A general form of relational-algebra query is

$$\pi_L\Big(\sigma_C(R_1 \times R_2 \times \cdots \times R_n)\Big)$$

Here, L is an arbitrary list of attributes, and C is an arbitrary condition. The list of relations $R_1, R_2, \ldots, R_n$ may include the same relation repeated several times, in which case appropriate renaming may be assumed applied to the R_i's. Show how to express any query of this form in SQL.

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,…

! Exercise 6.2.5: Another general form of relational-algebra query is

$$\pi_L\Big(\sigma_C(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)\Big)$$

The same assumptions as in Exercise 6.2.4 apply here; the only difference is that the natural join is used instead of the product. Show how to express any query of this form in SQL.

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.

Exercise 6.3.1: Write the following queries, based on the database schema

```
Product(maker,model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen,
price) Printer(model, color, type, price)
```

of Exercise 2.4.1. You should use at least one subquery in each of your answers and write each query in two significantly different ways (e.g., using different sets of the operators `EXISTS`, `IN`, `ALL`, and `ANY`).

a. Find the makers of PC's with a speed of at least 3.0.
b. Find the printers with the highest price.
c. Find the laptops whose speed is slower than that of any PC.
d. Find the model number of the item (PC, laptop, or printer) with the highest price.

```
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,
```

e. Find the maker of the color printer with the lowest price.

```
                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
```

f. Find the maker(s) of the PC(s) with the fastest processor among all those PC's that have the smallest amount of RAM.	(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));
Exercise 6.3.2 : Write the following queries, based on the database schema Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched) Battles(name, date) Outcomes(ship, battle, result) of Exercise 2.4.3. You should use at least one subquery in each of your answers and write each query in two significantly	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

different ways (e.g., using different sets of the operators `EXISTS`, `IN`, `ALL`, and `ANY`).

a) Find the countries whose ships had the largest number of guns.
b) Find the classes of ships, at least one of which was sunk in a battle.

```
        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) Find the names of the ships with a 16-inch bore.

```
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) Find the battles in which ships of the Kongo class participated.

```
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) Find the names of the ships whose number of guns was the largest for those ships of the same bore.

```
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) ;
Exercise 6.3.3: Write the query of Fig. 6.10 without any subqueries. SELECT title FROM Movies Old WHEREyear<ANY (SELECT year FROM Movies WHERE title = Old.title);	6.3.3 SELECT title FROM Movies GROUP BY title HAVING COUNT(title) > 1 ;
! **Exercise 6.3.4:** Consider expression $\pi_L(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_n)$ of relational algebra, where L is a list of attributes all of which belong to R_1. Show that this expression can be written in SQL using subqueries only. More precisely, write an equivalent SQL expression where no FROM clause has more than one relation in its list.	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) ...
Exercise 6.3.5 Write the following queries without using intersection or difference operators a. Intersection query of Fig. 6.5 **Example 6.16:** Suppose we wanted the names and addresses of all female movie stars who are also movie executives with a net worth over $10,000,000. Using the following two relations: MovieStar(name, address, gender, birthdate) MovieExec(name, address, cert#, netWorth) we can write the query as in Fig. 6.5. Lines (1) through (3) produce a relation whose schema is (name, address) and whose tuples are the names and addresses of all female movie stars. 1) (SELECT name, address 2) FROM MovieStar 3) WHERE gender = 'F') 4) INTERSECT 5) (SELECT name, address 6) FROM MovieExec 7) WHERE netWorth > 10000000); Figure 6.5: Intersecting female movie stars with rich executives	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. The difference query of 6.17

Example 6.17: In a similar vein, we could take the difference of two sets of persons, each selected from a relation. The query

```
(SELECT name, address FROM MovieStar)
    EXCEPT
(SELECT name, address FROM MovieExec);
```

gives the names and addresses of movie stars who are not also movie executives, regardless of gender or net worth. □

In the two examples above, the attributes of the relations whose intersection or difference we took were conveniently the same. However, if necessary to get a common set of attributes, we can rename attributes as in Example 6.3.

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

Exercise 6.3.6 We have noticed that certain operators of SQL are redun dant, in the sense that they always can be replaced by other operators. For example, we saw that *s* IN*R* can be replaced by *s* = ANY*R*. Show that EXISTS and NOT EXISTS are redundant by explaining how to replace any expression of the form EXISTS *R* or NOT EXISTS *R* by an expression that does not involve EXISTS(except perhaps in the expression *R* itself). *Hint:* Remember that it is permissible to have a constant in the SELECT clause.

```
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
        ) ;
```

Exercise 6.3.7: For these relations from our running movie database schema

```
6.3.7
(a)
```

StarsIn(movieTitle, movieYear, starName) MovieStar(name, address, gender, birthdate) MovieExec(name, address, cert#, netWorth) Studio(name, address, presC#) describe the tuples that would appear in the following SQL expressions: a) Studio CROSS JOIN MovieExec; b) Starsln NATURAL FULL OUTER JOIN MovieStar; c) Starsln FULL OUTER JOIN MovieStar ON name = starName;	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.
Exercise 6.3.8: Using the database schema Product(maker,model, type) PC(model, speed, ram, hd, rd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price) write a SQL query that will produce information about all products — PC's, laptops, and printers — including their manufacturer if available, and whatever information about that product is relevant (i.e., found in the relation for that type of product).	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;
Exercise 6.3.9: Using the two relations `Classes(class, type, country, numGuns, bore, displacement)` `Ships(name, class, launched)` from our database schema of Exercise 2.4.3, write a SQL query that will produce all available information about ships, including that information available in the `Classes` relation. You need not produce information about classes if there are no ships of that class mentioned in `Ships`.	6.3.9 SELECT * FROM Classes RIGHT NATURAL OUTER JOIN Ships ;
`Exercise6.3.10`: Repeat Exercise 6.3.9, but also include in the result, for any class C that is not mentioned in`Ships`, information about the ship that has the same name *C* as its class. You may assume that there is a ship with the class name, even if it doesn't appear in `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)) ;
`Exercise6.3.11`: The join operators (other than outerjoin) we learned in this section are redundant, in the sense that they can always be replaced by select- from-where expressions. Explain how to write expressions of the following forms using select-from-where: a) `R CROSS JOIN S;`	6.3.11 (a) SELECT * FROM R, S ; (b) Let Attr consist of AttrR = attributes unique to R

b) R NATURAL JOIN S; c) R JOIN S ON *C* where *C* is a SQL condition.	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 ;

Section 6.4	
Exercise 6.4.1: Write each of the queries in Exercise 2.4.1 in SQL, making sure that duplicates are eliminated. Product(maker,model, type) PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price) a.What PC models have a speed of at least 3.00? b. Which manufacturers make laptops with a hard disk of at least 100GB?	(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. Find the model number and price of all products (of any type) made by manufacturer B.

```
(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. Find the model numbers of all color laser printers.

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

e. Find those manufacturers that sell Laptops, but not PC's.

```
(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. Find those hard-disk sizes that occur in two or more PC's.

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

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

g. Find those pairs of PC models that have both the same speed and RAM. A pair should be listed only once; e.g., list (i, j) but not (j, i).

```
(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. Find those manufacturers of at least two different computers (PC's or laptops) with speeds of at least 2.80.

```
(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. Find the manufacturer(s) of the computer (PC or laptop) with the highest available speed.

```
(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) ;
j. Find the manufacturers of PC's with at least three different speeds.	(j) SELECT R.maker FROM Product R, PC P WHERE R.model = P.model GROUP BY R.maker HAVING COUNT(DISTINCT speed) >= 3 ;
k. Find the manufacturers who sell exactly three different models of PC.	(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 ;
Exercise 6.4.3: For each of your answers to Exercise 6.3.1, determine whether or not the result of your query can have duplicates. If so, rewrite the query to eliminate duplicates. If not, write a query without subqueries that has the same, duplicate-free answer.	(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
        );
```

Exercise 6.4.3: For each of your answers to Exercise 6.3.1, determine whether or not the result of your query can have duplicates. If so, rewrite the query to eliminate duplicates. If not, write a query without subqueries that has the same, duplicate-free answer.

```
Product(maker,model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen,
price) Printer(model, color, type, price)
```

a. Find the makers of PC's with a speed of at least 3.0.

a)

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

b. Find the printers with the highest price.

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. Find the laptops whose speed is slower than that of any PC.

c)

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

d. Find the model number of the item (PC, laptop, or printer) with the highest price.

```
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,
```

e. Find the maker of the color printer with the lowest price.

```
                        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. Find the maker(s) of the PC(s) with the fastest processor among all those PC's that have the smallest amount of RAM.

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
                )
        );
```

Exercise 6.4.4: Repeat Exercise 6.4.3 for your answers to Exercise 6.3.2.

For each of your answers to Exercise 6.3.1, determine whether or not the result of your query can have duplicates. If so, rewrite the query to eliminate duplicates. If not, write a query without subqueries that has the same, duplicate-free answer

```
Classes(class, type, country, numGuns,
bore, displacement) Ships(name, class,
launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

a) Find the countries whose ships had the largest number of guns.

a)

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

! b) Find the classes of ships, at least one of which was sunk in a battle.

```
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) Find the names of the ships with a 16-inch bore.

```
c)

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

d) Find the battles in which ships of the Kongo class participated.

```
d)

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

!! e) Find the names of the ships whose number of guns was the largest for those ships of the same bore.

```
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 ;
```

Exercise 6.4.6: Write the following queries, based on the database schema
Product(maker,model, type)
PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price)

a) Find the average speed of PC's.

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

b) Find the average speed of laptops costing over $1000.

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

c) Find the average price of PC's made by manufacturer "A."

```
(c)

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

```
(d)
```

! d) Find the average price of PC's and laptops made by manufacturer "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 ;
```

e) Find, for each different speed, the average price of a PC.

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

! f) Find for each manufacturer, the average screen size of its laptops.

```
(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 ;
```

! g) Find the manufacturers that make at least three different models of PC. !

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

h) Find for each manufacturer who sells PC's the maximum price of a PC.

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

! i) Find, for each speed of PC above 2.0, the average price.

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

!! j) Find the average hard disk size of a PC for all those manufacturers that make printers.

```
(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'
        ) ;
```

Exercise6.4.7: Write the following queries, based on the database schema

Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)

of Exercise 2.4.3, and evaluate your queries using the data of that exercise.

a. Find the number of battleship classes.

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

b. Find the average number of guns of battleship classes.

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

c. Find the average number of guns of battleships. Note the difference be- tween (b) and (c); do we weight a class by the number of ships of that class or not?

```
(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';
```

d. Find for each class the year in which the first ship of that class was launched.

```
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 ;
```

e. Find for each class the number of ships of that class sunk in battle.

```
(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 ;
```

f. Find for each class with at least three ships the number of ships of that class sunk in battle.

```
(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 ;
```

g. The weight (in pounds) of the shell fired from a naval gun is approximately one half the cube of the bore (in inches). Find the average weight of the shell for each country's ships.

```
(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 ;
```

Exercise6.4.8: In Example 5.10 we gave an example of the query: "find, for each star who has appeared in at least three movies, the earliest year in which they appeared." We wrote this query as a 7 operation. Write it in SQL.

```
      6.4.8
SELECT  starName,
        MIN(YEAR) AS minYear
FROM    StarsIn
GROUP BY starName
HAVING COUNT(title) >= 3 ;
```

! Exercise 6.4.9: The 7 operator of extended relational algebra does not have a feature that corresponds to the HAVINGclause of SQL. Is it possible to mimic a SQL query with a HAVING clause in relational algebra? If so, how would we do it in general?

```
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.
```

Section 6.5	
Exercise 6.5.1: Write the following database modifications, based on the database schema Product(maker,model, type) PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price) of Exercise 2.4.1. Describe the effect of the modifications on the data of that exercise.	
a. Using two INSERT statements, store in the database the fact that PC model 1100 is made by manufacturer C, has speed 3.2, RAM 1024, hard disk 180, and sells for $2499.	(a) INSERT INTO Product VALUES ('C' , '1100', 'pc') ; INSERT INTO PC VALUES ('1100', 3.2 , 1024,180,2499) ;
b. Insert the facts that for every PC there is a laptop with the same manu- facturer, speed, RAM, and hard disk, a 17-inch screen, a model number 1100 greater, and a price $500 more.	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 all PC's with less than 100 gigabytes of hard disk.

```
(c)
DELETE
FROM    PC
WHERE   hd < 100 ;
```

d. Delete all laptops made by a manufacturer that doesn't make printers.

```
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. Manufacturer A buys manufacturer B. Change all products made by B so they are now made by A.

```
(e)
UPDATE Product
SET     maker = 'A'
WHERE   maker = 'B' ;
```

f. For each PC, double the amount of RAM and add 60 gigabytes to the amount of hard disk. (Remember that several attributes can be changed by one UPDATEstatement.)

```
(f)
UPDATE PC
SET     ram = ram*2,
        hd  =hd  +60 ;
(g)
```

g. For each laptop made by manufacturer B, add one inch to the screen size and subtract $100 from the price.

```
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'
        ) ;
```

Exercise 6.5.2: Write the following database modifications, based on the database schema

Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)

of Exercise 2.4.3. Describe the effect of the modifications on the data of that exercise.

a) The two British battleships of the Nelson class — Nelson and Rodney — were both launched in 1927, had nine 16-inch guns, and a displacement of 34,000 tons. Insert these facts into the database.

```
(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) Two of the three battleships of the Italian Vittorio Veneto class — Vit- torio Veneto and Italia — were launched in 1940; the third ship of that class, Roma, was launched in 1942. Each had nine 15-inch guns and a displacement of 41,000 tons. Insert these facts into the database.

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

c) Delete from Ships all ships sunk in battle. d) Modify the Classes relation so that gun bores are measured in centime- ters (one inch = 2.5 centimeters) and displacements are measured in met- ric tons (one metric ton = 1.1 tons). e) Delete all classes with fewer than three ships.	); 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) ;

Q3 Write SQL constrains~chap 7	Write a CREATE table statements with constraints on Insert and Update only
Exercise 7.1.1: Our running example movie database of Section 2.2.8 has keys defined for all its relations. Movies(title , year, length, genre, studioName, producerC#) Starsln(movieTitle. movieYear. starName)	

MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)

Declare the following referential integrity constraints for the movie database as in Exercise 7.1.1.

a) The producer of a movie must be someone mentioned in MovieExec. Mod- ifications to MovieExec that violate this constraint are rejected.

```
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#)
);
```

b) Repeat (a), but violations result in the producerC# in Movie being set to NULL.

```
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) Repeat (a), but violations result in the deletion or update of the offending Movie tuple.

```
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) A movie that appears in StarsIn must also appear in Movie. Handle violations by rejecting the modification.

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

e) A star appearing in StarsIn must also appear in MovieStar. Handle violations by deleting violating tuples.	e) CREATE TABLE StarsIn (movieTitle CHAR(100) REFERENCES Movie(title) ON DELETE CASCADE, movieYear INT, starName CHAR(30), PRIMARY KEY (movieTItle, movieYear, starName));
Exercise 7.1.2: We would like to declare the constraint that every movie in the relation Movie must appear with at least one star in StarsIn. Can we do so with a foreign-key constraint? Why or why not?	**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.
Exercise 7.1.3: Suggest suitable keys and foreign keys for the relations of the PC database: Product(maker, model, type) PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price) of Exercise 2.4.1. Modify your SQL schema from Exercise 2.3.1 to include declarations of these keys.	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);
Exercise 7.1.4: Suggest suitable keys for the relations of the battleships database C lasses(class, type, country, numGuns, bore, displacement) Ships(name, class, launched) Battles(name, date) Outcomes(ship, battle, result) of Exercise 2.4.3. Modify your SQL schema from Exercise 2.3.2 to include declarations of these keys.	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);
Exercise 7.1.5: Write the following referential integrity constraints for the battleships database as in Exercise 7.1.4. Use your assumptions about keys from that exercise, and handle all violations by setting the referencing attribute value to NULL. a) Every class mentioned in Ships must be mentioned in Classes. b) Every battle mentioned in Outcomes must be mentioned in Battles. c) Every ship mentioned in Outcomes must be mentioned in Ships.	**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;
Exercise 7.2.1: Write the following constraints for attributes of the relation Movies(title, year, length, genre, studioName, producerC#) a) The year cannot be before 1915. b) The length cannot be less than 60 nor more than 250.	a) year INT CHECK (year >= 1915) b) length INT CHECK (length >= 60 AND length <= 250)

c) The studio name can only be Disney, Fox, MGM, or Paramount.

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

Exercise 7.2.2: Write the following constraints on attributes from our example schema

Product(maker, model, type)
PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(m odel, color, type, price)
of Exercise 2.4.1.

7.2.2

a) The speed of a laptop must be at least 2.0.

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

b) The only types of printers are laser, ink-jet, and bubble-jet.

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

c) The only types of products are P C 's, laptops, and printers.

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

!d) A model of a product must also be the model of a PC, a laptop, or a 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))
  …
);
```

Exercise 7.2.3: Write the following constraints as tuple-based CHECK con- straints on one of the relations of our running movies example:

Movies(title, year, length, genre, studioName, producerC#) StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)

If the constraint actually involves two relations, then you should put constraints in both relations so that whichever relation changes, the constraint will be checked on insertions and updates. Assume no deletions; it is not always pos- sible to maintain tuple-based constraints in the face of deletions.

a. A star may not appear in a movie made before they were born.

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

b. No two studios may have the same address.

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

c. A name that appears in MovieStar must not also appear in MovieExec.

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

d. A studio name that appears in Studio must also appear in at least one Movies tuple.

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

e. If a producer of a movie is also the president of a studio, then they must be the president of the studio that made the movie.

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

	WHERE presC# = producerC#)) …);
Exercise 7.2.4: Write the following as tuple-based CHECK constraints about our "PC" schema. a) A PC with a processor speed less than 2.0 must not sell for more than $600. b) A laptop with a screen size less than 15 inches must have at least a 40 gigabyte hard disk or sell for less than $1000.	a) CHECK (speed >= 2.0 OR price <= 600) b) CHECK (screen >= 15 OR hd >= 40 OR price <= 1000)
Exercise 7.2.5: Write the following as tuple-based CHECK constraints about our "battleships" schema: Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched) Battles(name, date) Outcomes(ship, battle, result) a) No class of ships may have guns with larger than a 16-inch bore. b) If a class of ships has more than 9 guns, then their bore must be no larger than 14 inches. ! c) No ship can be in battle before it is launched.	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)))
Exercise 7.2.6: In Examples 7.6 and 7.8, we introduced constraints on the gender attribute of MovieStar. What restrictions, if any, do each of these con- straints enforce if the value of gender is NULL?	The constraint in Example 7.6 does not allow NULL value for gender while the constraint in Example 7.8 allows NULL.

Section 7.3 Modification of constraints	
Exercise 7.3.1: Show how to alter your relation schemas for the movie example: Movie(title, year, length, genre, studioName, producerC#) Starsln(movieTitle, movieYear, starName) MovieStar(name, address, gender, birthdate) MovieExec(name, address, cert#, netWorth) Studio(name, address, presC#) in the following ways. a. Make title and year the key for Movie. b. Require the referential integrity constraint that the producer of every movie appear in MovieExec. c. Require that no movie length be less than 60 nor greater than 250. d. Require that no name appear as both a movie star and movie executive (this constraint need not be maintained in the face of deletions). e. Require that no two studios have the same address	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);`
Exercise 7.3.2: Show how to alter the schemas of the “battleships” database: Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched) Battles(name, date) Outcomes(ship, battle, result) to have the following tuple-based constraints.	

a. Class and country form a key for relation Classes.

```
a)
ALTER TABLE Classes ADD CONSTRAINT myKey
       PRIMARY KEY (class, country);
```

b. Require the referential integrity constraint that every battle appearing in Outcomes also appears in Battles.

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

c. Require the referential integrity constraint that every ship appearing in Outcomes appears in Ships.

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

d. Require that no ship has more than 14 guns.

```
d)
ALTER TABLE Ships ADD CONSTRAINT
classGunCheck
       CHECK (class NOT IN (SELECT class
FROM Classes
                                          WHERE
numGuns > 14));
```

e. Disallow a ship being in battle before it is launched.

```
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)))
```

Section 7.4	
Exercise 7.4.1: Write the following assertions. The database schema is from the "PC" example of Exercise 2.4.1: Product(maker, model, type) PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price)	

a) No manufacturer of PC's may also make laptops.

```
a)
CREATE ASSERTION CHECK
    (NOT EXISTS
        (
            (SELECT maker FROM Product
NATURAL JOIN PC)
         INTERSECT
            (SELECT maker FROM Product
NATURAL JOIN Laptop)
        )
    );
```

b) A manufacturer of a PC must also make a laptop with at least as great a processor speed.

```
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) If a laptop has a larger main memory than a PC, then the laptop must also have a higher price than the PC.

```
c)
CREATE ASSERTION CHECK
    (NOT EXISTS
        (SELECT model
         FROM Laptop
         WHERE price <= ALL
                (SELECT price FROM PC
                 WHERE PC.ram < Laptop.ram
                )
          )
       )
    );
```

d) If the relation Product mentions a model and its type, then this model must appear in the relation appropriate to that type.

```
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)
     );
```

Exercise 7.4.2: Write the following as assertions. The database schema is from the battleships example of Exercise 2.4.3.

Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)

a. No class may have more than 2 ships.

```
a)
CREATE ASSERTION CHECK
    ( 2 >= ALL
        (SELECT COUNT(*) FROM Ships GROUP
BY class)
     );
```

b. No country may have both battleships and battlecruisers.

```
b)
CREATE ASSERTION CHECK
    (NOT EXISTS
        (SELECT country FROM Classes
            WHERE type = 'bb'
         )
        INTERSECT
        (SELECT country FROM Classes
            WHERE type = 'bc'
         )
     );
```

c. No ship with more than 9 guns may be in a battle with a ship having fewer than 9 guns that was sunk.

```
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. No ship may be launched before the ship that bears the name of the first ship's class.

```
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. For every class, there is a ship with the name of that class.

```
e)
CREATE ASSERTION CHECK
    (ALL (SELECT class FROM Classes c)
       IN (SELECT class FROM Ships GROUP BY
class)
     );
```

Exercise 7.4.3: The assertion of Exercise 7.11 can be written as two tuple-
based constraints. Show how to do so.

```
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))
Triggers ! Question 4	Q4. Write SQL Triggers Write triggers. In each case, disallow or undo the modification if it does not satisfy the stated constraint.
Exercise 7.5.1: Write the triggers analogous to Fig. 7.6 for the insertion and deletion events on MovieExec. CREATE TRIGGER AvgNetWorthTrigger AFTER UPDATE OF netW orth ON MovieExec REFERENCING OLD TABLE AS OldStuff, NEW TABLE AS NewStuff FOR EACH STATEMENT WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec)) BEGIN DELETE FROM MovieExec WHERE (name, address, cert#, netWorth) IN NewStuff; INSERT INTO MovieExec (SELECT * FROM OldStuff); END;	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);
Exercise 7.5.2: Write the following as triggers. In each case, disallow or undo the modification if it does not satisfy the stated constraint. The database schema is from the “PC” example of Exercise 2.4.1: Product(maker, model, type) PC(model, speed, ram, hd, price) Laptop(model, speed, ram, hd, screen, price) Printer(model, color, type, price)	

a. When updating the price of a PC, check that there is no lower priced PC with the same speed.

```
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. When inserting a new printer, check that the model number exists in Product.

```
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. When making any modification to the Laptop relation, check that the average price of laptops for each manufacturer is at least $1500.

```
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. When updating the RAM or hard disk of any PC, check that the updated PC has at least 100 times as much hard disk as RAM.

```
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)
```

e. When inserting a new PC, laptop, or printer, make sure that the model number did not previously appear in any of PC, Laptop, or Printer.

```
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;
```

Exercise 7.5.3: Write the following as triggers. In each case, disallow or undo the modification if it does not satisfy the stated constraint. The database schema is from the battleships example of Exercise 2.4.3.

Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)

a. When a new class is inserted into Classes, also insert a ship with the name of that class and a NULL launch date.

```
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. When a new class is inserted with a displacement greater than 35,000 tons, allow the insertion, but change the displacement to 35,000.

```
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. If a tuple is inserted into Outcomes, check that the ship and battle are listed in Ships and B attles, respectively, and if not, insert tuples into one or both of these relations, with NULL components where necessary.

```
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. When there is an insertion into Ships or an update of the class attribute of Ships, check that no country has more than 20 ships.

(solution for d is weird !)

```
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);
```

e. Check, under all circumstances that could cause a violation, that no ship fought in a battle that was at a later date than another battle in which that ship was sunk.

```
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;
```

Exercise 7.5.4: Write the following as triggers. In each case, disallow or undo the modification if it does not satisfy the stated constraint. The problems are based on our running movie example:

Movies(title, year, length, genre, studioName, producerC#) StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
You may assume that the desired condition holds before any change to the database is attempted. Also, prefer to modify the database, even if it means inserting tuples with NULLor default values, rather than rejecting the attempted modification.

a) Assure that at all times, any star appearing in StarsIn also appears in MovieStar

a.

```
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) Assure that at all times every movie executive appears as either a studio producer of a movie, or both.

```
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) Assure that every movie has at least one male and one female star.

```
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);
```

d) Assure that the number of movies made by any studio in any year is no more than 100.

```
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) Assure that the average length of all movies made in any year is no more than 120.

```
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;
```

chapter 8 Q5. Write materialized views	Q5. Write materialized views Given set of base tables and materialized view that is based on the given base tables. What modifications to the base tables that would require changes to the Materialized View and how do you propagate the changes incrementally to the materialized view?
Exercise 8.1.1: From the following base tables of our running example MovieStar(name, address, gender, birthdate) MovieExec(name, address, cert#, netWorth) Studio(name, address, presC#) Construct the following views: a) A view RichExec giving the name, address, certificate number and net worth of all executives with a net worth of at least $10,000,000. b) A view StudioPres giving the name, address, and certificate number of all executives who are studio presidents.	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) A view ExecutiveStar giving the name, address, gender, birth date, cer- tificate number, and net worth of all individuals who are both executives and stars.	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: Write each of the queries below, using one or more of the views from Exercise 8.1.1 and no base tables. a. Find the names of females who are both stars and executives. b. Find the names of those executives who are both studio presidents and worth at least $10,000,000. c. Find the names of studio presidents who are also stars and are worth at least $50,000,000.	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#;
Exercise 8.2.1: Which of the views of Exercise 8.1.1 are updatable?	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: Suppose we create the view: CREATE VIEW DisneyComedies AS SELECTtitle, year, length FROMMovies WHERE studioName = 'Disney' AND genre = 'comedy' ; a) Is this view updatable? b) Write an instead-of trigger to handle an insertion into this view.	**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) Write an instead-of trigger to handle an update of the length for a movie (given by title and year) in this view.	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: Using the base tables Product(maker, model, type) PC(model, speed, ram, hd, price) suppose we create the view: CREATE VIEW NewPC AS SELECT maker, model, speed, ram, hd, price FROM Product, PC WHERE Product.model = PC.model AND type = 'pc'; Notice that we have made a check for consistency: that the model number not only appears in the PC relation, but the type attribute of Product indicates that the product is a PC. a) Is this view updatable? b) Write an instead-of trigger to handle an insertion into this view. c) Write an instead-of trigger to handle an update of the price.	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) Write an instead-of trigger to handle a deletion of a specified tuple from this view.	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);
Exercise 8.3.1: For our running movies example: Movies(title, year, length, genre, studioName, producerC#) StarsIn(movieTitle, movieYear, starName) MovieExec(name, address, cert#, netWorth) Studio(name, address, presC#) Declare indexes on the following attributes or combination of attributes: a) studioName. b) address of MovieExec. c) genre and length.	**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);
Exercise 8.5.1: Complete Example 8.15 by considering updates to either of the base tables.	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: Suppose the view NewPC of Exercise 8.2.3 were a materialized view. What modifications to the base tables Product and PC would require a modification of the materialized view? How would you implement those modi- fications incrementally? Using the base tables Product(maker, model, type) PC(model, speed, ram, hd, price) suppose we create the view: CREATE VIEW NewPC AS SELECT maker, model, speed, ram, hd, price FROM Product, PC WHERE Product.model = PC.model AND type = 'pc';	**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: This exercise explores materialized views that are based on aggregation of data. Suppose we build a materialized view on the base tables Classes(class, type, country, numGuns, bore, displacement) Ships(name, class, launched)	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.

from our running battleships exercise, as follows:

```
CREATE MATERIALIZED VIEW ShipStats AS
SELECT country, AVG(displacement), COUNT(*)
FROM Classes, Ships
WHERE Classes.class = Ships.class
GROUP BY country;
```

What modifications to the base tables Classes and Ships would require a modification of the materialized view? How would you implement those modi- fications incrementally?

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: In Section 8.5.3 we gave conditions under which a materialized view of simple form could be used in the execution of a query of similar form. For the view of Example 8.15, describe all the queries of that form, for which this view could be used.	**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