Section 1

Exercise 8.1.1
a)
CREATE VIEW RichExec AS
      SELECT * FROM MovieExec WHERE netWorth >= 10000000;

b)
CREATE VIEW StudioPres (name, address, cert#) AS
      SELECT MovieExec.name, MovieExec.address, MovieExec.cert# FROM MovieExec,
      Studio WHERE MovieExec.cert# = Studio.presC#;

c)
CREATE VIEW ExecutiveStar (name, address, gender, birthdate, cert#, netWorth) AS
      SELECT star.name, star.address, star.gender, star.birthdate, exec.cert#, exec.netWorth
      FROM MovieStar star, MovieExec exec WHERE star.name = exec.name AND
      star.address = exec.address;

Exercise 8.1.2
a)
SELECT name from ExecutiveStar WHERE gender = 'f';

b)
SELECT RichExec.name from RichExec, StudioPres where RichExec.name = StudioPres.name;

c)
SELECT ExecutiveStar.name from ExecutiveStar, StudioPres
      WHERE ExecutiveStar.netWorth >= 50000000 AND
      StudioPres.cert# = RichExec.cert#;

Section 2

Exercise 8.2.1

The views RichExec and StudioPres are updatable; however, the StudioPres view needs to be created with a subquery.

CREATE VIEW StudioPres (name, address, cert#) AS
      SELECT MovieExec.name, MovieExec.address, MovieExec.cert# FROM MovieExec
      WHERE MovieExec.cert# IN (SELECT presCt# from Studio);

Exercise 8.2.2

a) Yes, the view is updatable.

b)

CREATE TRIGGER DisneyComedyInsert
INSTEAD OF INSERT ON DisneyComedies
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
INSERT INTO Movies(title, year, length, studioName, genre)
VALUES(NewRow.title, NewRow.year, NewYear.length, 'Disney', 'comedy');

c)
CREATE TRIGGER DisneyComedyUpdate
INSTEAD OF UPDATE ON DisneyComedies
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
UPDATE Movies SET length NewRow.length
WHERE title = NewRow.title AND year = NEWROW.year AND
studionName = 'Disney' AND genre = 'comedy';

Exercise 8.2.3

a) No, the view is not updatable since it is constructed from two different relations.

b)
CREATE TRIGGER NewPCInsert
INSTEAD OF INSERT ON NewPC
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
(INSERT INTO Product VALUES(NewRow.maker, NewRow.model, 'pc'))
(INSERT INTO PC VALUES(NewRow.model, NewRow.speed, NewRow.ram, NewRow.hd,
NewRow.price));

c)
CREATE TRIGGER NewPCUpdate
INSTEAD OF UPDATE ON NewPC
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
UPDATE PC SET price = NewPC.price where model = NewPC.model;

d)
CREATE TRIGGER NewPCDelete
INSTEAD OF DELETE ON NeePC
REFERENCING OLD ROW AS OldRow
FOR EACH ROW
(DELETE FROM Product WHERE model = OldRow.model)
(DELETE FROM PC where model = OldRow.model);

Section 3

Exercise 8.3.1

a)
CREATE INDEX NameIndex on Studio(name);

b)
CREATE INDEX AddressIndex on MovieExec(address);

c)
CREATE INDEX GenreIndex on Movies(genre, length);

Section 4

Exercise 8.4.1

Action	No Index	Star Index	Movie Index	Both Indexes
Q1	100	4	100	4
Q2	100	100	4	4
I	2	4	4	6
Average	$2 + 98p_1 + 98p_2$	$4 + 96\,p_2$	$4 + 96\,p_1$	$6 - 2\,p_1 - 2\,p_2$

Exercise 8.4.2

Q1 = SELECT * FROM Ships WHERE name = n;
Q2 = SELECT * FROM Ships WHERE class = c;
Q3 = SELECT * FROM Ships WHERE launched = y;
I = Inserts

Indexes Actions	None	Name	Class	Launched	Name & Class	Name & Launched	Class & Launched	Three Indexes
Q1	50	2	50	50	2	2	50	2
Q2	1	1	2	1	2	1	2	2
Q3	50	50	50	26	50	26	26	26
I	2	4	4	4	6	6	6	8
Average	$2 + 48p_1 - p_2 + 48p_3$	$4 + 46p_3 - 2p_1 - 3p_2$	$4 + 46p_1 - 2p_2 + 46p_3$	$4 + 46p_1 - 3p_2 + 22p_3$	$6 - 4p_1 - 4p_2 + 44p_3$	$6 - 4p_1 - 5p_2 + 20p_3$	$6 - 44p_1 - 4p_2 + 20p_3$	$8 - 6p_1 - 6p_2 + 18p_3$

The best choice of indexes (name and launched) has an average cost of $6 - 4p_1 - 5p_2 + 20p_3$ per operation.

Section 5

Exercise 8.5.1

Updates to movies that involves title or year

UPDATE MovieProd SET title = 'newTitle' where title='oldTitle' AND year = oldYear;

UPDATE MovieProd SET year = newYear where title='oldYitle' AND year = oldYear;

Update to MovieExec involving cert#

DELETE FROM MovieProd
WHERE (title, year) IN (
     SELECT title, year
     FROM Movies, MovieExec
     WHERE cert# = oldCert# AND cert# = producerC#
     );

INSERT INTO MovieProd
     SELECT title, year, name
     FROM Movies, MovieExec
     WHERE cert# = newCert# AND cert# = producerC#;

Exercise 8.5.2

Insertions, deletions, and updates to the base tables Product and PC would require a modification of the materialized view.

Insertions into Product with type equal to 'pc':
     INSERT INTO NewPC
          SELECT maker, model, speed, ram, hd, price FROM Product, PC WHERE
          Product.model = newModel and Product.model = PC.model;

Insertions into PC:
     INSERT INTO NewPC
          SELECT maker, 'newModel', 'newSpeed', 'newRam', 'newHd', 'newPrice'
          FROM Product WHERE model = 'newModel';

Deletions from Product with type equal to 'pc':
     DELETE FROM NewPC WHERE maker = 'deletedMaker' AND model='deletedModel';

Deletions from PC:

DELETE FROM NewPC WHERE model = 'deletedModel';

Updates to PC:
Update NewPC SET speed=PC.speed, ram=PC.ram, hd=PC.hd, price=PC.price FROM PC where model=pc.model;

Update to the attribute 'model' needs to be treated as a delete and an insert.

Updates to Product:
Any changes to a Product tuple whose type is 'pc' need to be treated as a delete or an insert, or both.

Exercise 8.5.3

Modifications to the base tables that would require a modification to the materialized view: inserts and deletes from Ships, deletes from class, updates to a Class' displacement.

Deletions from Ship:

UPDATE ShipStats SET
        displacement=((displacement * count) –
                (SELECT displacement
                FROM Classses
                WHERE class = 'DeletedShipClass')
) / (count – 1),
        count = count – 1
WHERE
        country = (SELECT country FROM Classes WHERE class='DeletedShipClass');

Insertions into Ship:

Update ShipStat SET
        displacement=((displacement*count) +
                (SELECT displacement FROM Classes
                WHERE class='InsertedShipClass')
) / (count + 1),
        count = count + 1
WHERE
        country = (SELECT country FROM Classes WHERE classes='InsertedShipClass);

Deletes from Classes:

NumRowsDeleted = SELECT count(*) FROM ships WHERE class = 'DeletedClass';

UPDATE ShipStats SET
        displacement = (displacement * count) - (DeletedClassDisplacement *

NumRowsDeleted)) / (count – NumRowsDeleted),
count = count – NumRowsDeleted
WHERE country = 'DeletedClassCountry';

Update to a Class' displacement:

N = SELECT count(*) FROM Ships where class = 'UpdatedClass';

UPDATE ShipsStat SET
displacement = ((displacement * count) + ((oldDisplacement – newDisplacement) *
N))/count
WHERE
country = 'UpdatedClassCountry';

Exercise 8.5.4

Queries that can be rewritten with the materialized view:

Names of stars of movies produced by a certain producer

SELECT starName
FROM StarsIn, Movies, MovieExec
WHERE movieTitle = title AND movieYear = year AND producerC# = cert# AND
name = 'Max Bialystock';

Movies produced by a certain producer

SELECT title, year
FROM Movies, MovieExec
Where producerC# = cert# AND name = 'George Lucas';

Names of producers that a certain star has worked with

SELECT name
FROM Movies, MovieExec, StarsIn
Where producerC#=cert# AND title=movieTitle AND year=movieYear AND
starName='Carrie Fisher';

The number of movies produced by given producer

SELECT count(*)
FROM Movies, MovieExec
WHERE producerC#=cert# AND name = 'George Lucas';

Names of producers who also starred in their own movies

SELECT name
FROM Movies, StarsIn, MovieExec
WHERE producerC#=cert# AND movieTitle = title AND movieYear = year AND
        MovieExec.name = starName;

The number of stars that have starred in movies produced by a certain producer

SELECT count(DISTINCT starName)
FROM Movies, StarsIn, MovieExec
WHERE producerC#=cert# AND movieTitle = title AND movieYear = year AND
        name 'George Lucas';

The number of movies produced by each producer

SELECT name, count(*)
FROM Movies, MovieExec
WHERE producerC#=cert# GROUP BY name