Solutions
Chapter 7

7.1.1

a)

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

or

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

b)

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

c)

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

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

d)

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

e)

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

7.1.2

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

7.1.3

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

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


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


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

7.1.4

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

ALTER TABLE Ships
        ADD PRIMARY KEY (name);

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

ALTER TABLE Battles
        ADD PRIMARY KEY (name);

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

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

7.1.5

a)

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

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

b)

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

c)

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

7.2.1

a)
year INT CHECK (year >= 1915)

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

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

7.2.2

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

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

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

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

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

7.2.3

a)

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

b)

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

c)

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

d)

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

e)

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

7.2.4

a)

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

b)

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

7.2.5

a)

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

b)

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

c)

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

7.2.6

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

7.3.1

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

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

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

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

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

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

7.3.2

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

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

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

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

7.4.1

a)

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

b)

```
CREATE ASSERTION CHECK
   (NOT EXISTS
      (SELECT maker
       FROM Product NATURAL JOIN PC
       WHERE speed > ALL
          (SELECT L2.speed
           FROM Product P2, Laptop L2
           WHERE P2.maker = maker AND
                P2.model = L2.model
          )
      )
   );
```

c)

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

d)

```
CREATE ASSERTION CHECK
   (EXISTS
      (SELECT p2.model FROM Product p1, PC p2
                     WHERE p1.type = ,pc? AND
                           P1.model = p2.model)
     UNION ALL
      (SELECT l.model
        FROM Product p, Laptop l
                     WHERE p.type = ,laptop? AND
                           p.model = l.model)
     UNION ALL
       (SELECT p2.model
         FROM Product p1, Printer p2
                     WHERE p1.type = ,printer? AND
                           P1.model = p2.model)
    );
```

7.4.2

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

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

c)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
            (SELECT o.battle FROM Outcomes o, Ships s, Classes c
              WHERE o.ship = s.name AND s.class = c.class AND c.numGuns > 9
             )
     INTERSECT
            (SELECT o.battle FROM Outcomes o, Ships s, Classes c
                  WHERE o.result = ,sunk? AND o.ship = s.name AND
           s.class = c.class AND c.numGuns < 9
             )
    );
```

d)
```
CREATE ASSERTION CHECK
   (NOT EXISTS
            (SELECT s1.name FROM Ships s1
            WHERE s1.launched < (SELECT s2.launched FROM Ships s2
                                                WHERE s2.name = s1.class
                        )
       )
   );
```

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

7.4.3

1)
```
presC# INT CHECK
            (presC# IN (SELECT cert# FROM MovieExec
                              WHERE netWorth >= 10000000
                        )
```

```
                )
2)
presC# INT Check
                (presC# NOT IN (SELECT cert# FROM MovieExec
                                        WHERE netWorth < 10000000
                                    )
                )
```

7.5.1

```
CREATE TRIGGER AvgNetWorthTrigger
AFTER INSERT ON MovieExec
REFERENCING
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
DELETE FROM MovieExec
        WHERE (name, address, cert#, netWorth) IN NewStuff;


CREATE TRIGGER AvgNetWorthTrigger
AFTER DELETE ON MovieExec
REFERENCING
        OLD TABLE AS OldStuff
FOR EACH STATEMENT
WHEN (500000 > (SELECT AVG(netWorth) FROM MovieExec))
INSERT INTO MovieExec
        (SELECT * FROM OldStuff);
```

7.5.2

a)

```
CREATE TRIGGER LowPricePCTrigger
AFTER UPDATE OF price ON PC
REFERENCING
        OLD ROW AS OldRow,
        OLD TABLE AS OldStuff,
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NewRow.price < ALL
        (SELECT PC.price FROM PC
         WHERE PC.speed = NewRow.speed))
BEGIN
        DELETE FROM PC
        WHERE (model, speed, ram, hd, price) IN NewStuff;
        INSERT INTO PC
                (SELECT * FROM OldStuff);
END;
```

b)

```
CREATE TRIGGER NewPrinterTrigger
AFTER INSERT ON Printer
REFERENCING
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NOT EXISTS (SELECT * FROM Product
                WHERE Product.model = NewRow.model))
DELETE FROM Printer
        WHERE (model, color, type, price) IN NewStuff;
```

c)

```
CREATE TRIGGER AvgPriceTrigger
```

```
AFTER UPDATE OF price ON Laptop
REFERENCING
        OLD TABLE AS OldStuff,
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (1500 > (SELECT AVG(price) FROM Laptop))
BEGIN
        DELETE FROM Laptop
        WHERE (model, speed, ram, hd, screen, price) IN NewStuff;
        INSERT INTO Laptop
                (SELECT * FROM OldStuff);
END;
```

d)

```
CREATE TRIGGER HardDiskTrigger
AFTER UPDATE OF hd, ram ON PC
REFERENCING
        OLD ROW AS OldRow,
        OLD TABLE AS OldStuff,
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NewRow.hd < NewRow.ram * 100)
BEGIN
        DELETE FROM PC
        WHERE (model, speed, ram, hd, price) IN NewStuff;
        INSERT INTO PC
                (SELECT * FROM OldStuff);

END;
```

e)

```
CREATE TRIGGER DupModelTrigger
BEFORE INSERT ON PC, Laptop, Printer
REFERENCING
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (EXISTS (SELECT * FROM NewStuff NATUAL JOIN PC)
            UNION ALL
            (SELECT * FROM NewStuff NATUAL JOIN Laptop)
            UNION ALL
            (SELECT * FROM NewStuff NATUAL JOIN Printer))
BEGIN
        SIGNAL SQLSTATE ,10001?
            (,Duplicate Model      –      Insert Failed?);
END;
```

7.5.3

a)

```
CREATE TRIGGER NewClassTrigger
AFTER INSERT ON Classes
REFERENCING
        NEW ROW AS NewRow
FOR EACH ROW
```

```
BEGIN
        INSERT INTO Ships (name, class, lunched)
                VALUES (NewRow.class, NewRow.class, NULL);
END;
```

b)

```
CREATE TRIGGER ClassDisTrigger
BEFORE INSERT ON Classes
REFERENCING
        NEW ROW AS NewRow,
        NEW TABLE AS NewStuff
FOR EACH ROW
WHEN (NewRow.displacement > 35000)
UPDATE NewStuff SET displacement = 35000;
```

c)

```
CREATE TRIGGER newOutcomesTrigger
AFTER INSERT ON Outcomes
REFERENCING
        NEW ROW AS NewRow
FOR EACH ROW
WHEN (NewRow.ship NOT EXISTS (SELECT name FROM Ships))
INSERT INTO Ships (name, class, lunched)
        VALUES (NewRow.ship, NULL, NULL);


CREATE TRIGGER newOutcomesTrigger2
AFTER INSERT ON Outcomes
REFERENCING
        NEW ROW AS NewRow
FOR EACH ROW
WHEN (NewRow.battle NOT EXISTS (SELECT name FROM Battles))
INSERT INTO Battles (name, date)
        VALUES (NewRow.battle, NULL);
```

d)

```
CREATE TRIGGER changeShipTrigger
AFTER INSERT ON Ships
REFERENCING
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN ( 20 < ALL
        (SELECT COUNT(name) From Ships NATURAL JOIN Classes
                GROUP BY country))
DELETE FROM Ships
WHERE (name, class, launched) IN NewStuff;

CREATE TRIGGER changeShipTrigger2
AFTER UPDATE ON Ships
REFERENCING
        OLD TABLE AS OldStuff,
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN ( 20 < ALL
```

```
                (SELECT COUNT(name) From Ships NATURAL JOIN Classes
                        GROUP BY country))
BEGIN
                DELETE FROM Ships
                WHERE (name, class, launched) IN NewStuff;
                INSERT INTO Ships
                        (SELECT * FROM OldStuff);
END;

e)
CREATE TRIGGER sunkShipTrigger
AFTER INSERT ON Outcomes
REFERENCING
                NEW ROW AS NewRow
                NEW TABLE AS NewStuff
FOR EACH ROW
WHEN ( (SELECT date FROM Battles WHERE name = NewRow.battle)
      < ALL
                (SELECT date FROM Battles
                    WHERE name IN (SELECT battle FROM Outcomes
                            WHERE ship = NewRow.ship AND
                                                                    result =       “ sunk ”
                        )
       )
     )

DELETE FROM Outcomes
WHERE (ship, battle, result) IN NewStuff;

CREATE TRIGGER sunkShipTrigger2
AFTER UPDATE ON Outcomes
REFERENCING
                NEW ROW AS NewRow,
                NEW TABLE AS NewStuff
FOR EACH ROW
FOR EACH ROW
WHEN ( (SELECT date FROM Battles WHERE name = NewRow.battle)
      < ALL
                (SELECT date FROM Battles
                    WHERE name IN (SELECT battle FROM Outcomes
                            WHERE ship = NewRow.ship AND
                                                                    result =       “ sunk ”
                        )
       )
     )
BEGIN
                DELETE FROM Outcomes
                WHERE (ship, battle, result) IN NewStuff;
                INSERT INTO Outcomes
                        (SELECT * FROM OldStuff);
END;


7.5.4

CREATE TRIGGER changeStarsInTrigger
AFTER INSERT ON StarsIn
```

```
REFERENCING
            NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.starName NOT EXISTS
                        (SELECT name FROM MovieStar))
INSERT INTO MovieStar(name)
            VALUES(NewRow.starName);


CREATE TRIGGER changeStarsInTrigger2
AFTER UPDATE ON StarsIn
REFERENCING
            NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.starName NOT EXISTS
                        (SELECT name FROM MovieStar))
INSERT INTO MovieStar(name)
            VALUES(NewRow.starName);

b)
CREATE TRIGGER changeMovieExecTrigger
AFTER INSERT ON MovieExec
REFERENCING
              NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.cert# NOT EXISTS
                        (SELECT presC# FROM Studio)
          UNION ALL
            SELECT producerC# FROM Movies)
    )
INSERT INTO Movies(procucerC#)
            VALUES(NewRow.cert#);

* insert into the relation Movies rather than Studio since there?s no
associated info with Studio.


CREATE TRIGGER changeMovieExecTrigger2
AFTER UPDATE ON MovieExec
REFERENCING
            NEW ROW AS NewRow,
FOR EACH ROW
WHEN (NewRow.cert# NOT EXISTS
                        (SELECT presC# FROM Studio)
          UNION ALL
            SELECT producerC# FROM Movies)
    )
INSERT INTO Movies(procucerC#)
            VALUES(NewRow.cert#);

c)
CREATE TRIGGER changeMovieTrigger
AFTER DELETE ON MovieStar
REFERENCING
              OLD TABLE AS OldStuff,
FOR EACH STATEMENT
WHEN ( 1 > ALL (SELECT COUNT(*) FROM StarIn s, MovieStar m
```

```
                        WHERE s.starName = m.name
                                GROUP BY s.movieTitle, m.gendar)
    )
INSERT INTO MovieStar
        (SELECT * FROM OldStuff);
```

* only considering DELETE from MovieStar since the assumption was the desired condistion was satisfied before any change.
** not considering INSERT into StarsIn since no gender info can be extracted from a new row for StarsIn.

d)

```
CREATE TRIGGER numMoviesTrigger
AFTER   INSERT   ON Movies
REFERENCING
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (100 < ALL
        (SELECT COUNT(*) FROM Movies
                GROUP BY studioName, year))
DELETE FROM Movies
WHERE (title, year, length, genre, StudioName, procedureC#)IN NewStuff;


CREATE TRIGGER numMoviesTrigger2
AFTER   UPDATE ON Movies
REFERENCING
        OLD TABLE AS OldStuff
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (100 < ALL
        (SELECT COUNT(*) FROM Movies
                GROUP BY studioName, year))
BEGIN
        DELETE FROM Movies
        WHERE (title, year, length, genre, StudioName, procedureC#)
        IN NewStuff;
        INSERT INTO Movies
            (SELECT * FROM OldStuff);
END;
```

e)

```
CREATE TRIGGER avgMovieLenTrigger
AFTER   INSERT   ON Movies
REFERENCING
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (120 < ALL
        (SELECT AVG(length) FROM Movies
                GROUP BY year))
DELETE FROM Movies
WHERE (title, year, length, genre, StudioName, procedureC#)IN NewStuff;


CREATE TRIGGER avgMovieLenTrigger2
AFTER   UPDATE ON Movies
```

```
REFERENCING
        OLD TABLE AS OldStuff
        NEW TABLE AS NewStuff
FOR EACH STATEMENT
WHEN (120 < ALL
        (SELECT AVG(length) FROM Movies
                GROUP BY year))
BEGIN
        DELETE FROM Movies
        WHERE (title, year, length, genre, StudioName, procedureC#)
        IN NewStuff;
        INSERT INTO Movies
            (SELECT * FROM OldStuff);
END;
```