

CSC343 Worksheet 5 Solution

June 20, 2020

1. a)

```
CREATE TABLE MovieExec (  
    name CHAR(30),  
    address VARCHAR(255),  
    cert# INT PRIMARY KEY,  
    FOREIGN KEY (cert#) REFERENCES Movies(producerC#)  
);
```

Example:

- Foreign-key
 - **Syntax 1:** FOREIGN KEY (< attributes >) REFERENCES < table >(< attributes >)
 - **Syntax 2:** REFERENCES < table >(< attributes >)
 - Binds an attribute of one relation to an attribute in another table
 - Added when creating table

Example:

```
1 // Example 1  
2 CREATE TABLE Studio (  
3     name CHAR(30) PRIMARY KEY,  
4     address VARCHAR(255),  
5     presC# INT REFERENCES MovieExec(cert#)  
6 );  
7  
8 // Example 2  
9 CREATE TABLE Studio (  
10     name CHAR(30) PRIMARY KEY,  
11     address VARCHAR(255),  
12     presC# INT,  
13     FOREIGN KEY (presC#) REFERENCES MovieExec(cert#)  
14 );  
15
```

b)

```

1  CREATE TABLE Movies (
2      title CHAR(30) PRIMARY KEY,
3      year INT PRIMARY KEY,
4      length INT,
5      genre VARCHAR(255),
6      studioName VARCHAR(255),
7      producerC# PRIMARY KEY
8  );
9

```

c) No change required. Violation occurs by the default policy.

```

1  CREATE TABLE MovieExec (
2      name CHAR(30),
3      address VARCHAR(255),
4      cert# INT PRIMARY KEY,
5      FOREIGN KEY (cert#) REFERENCES Movies(producerC#)
6  );
7

```

Correct Solution:

```

1  CREATE TABLE MovieExec (
2      name CHAR(30),
3      address VARCHAR(255),
4      cert# INT PRIMARY KEY,
5      FOREIGN KEY (cert#) REFERENCES Movies(producerC#)
6          ON UPDATE CASCADE // Correction
7          ON DELETE CASCADE // Correction
8  );
9

```

Notes:

- Maintaining Referential Integrity
 - Three different types of policies exist on Foreign Key
 1. *The Default Policy: Reject Violating Modifications.*
 - * Is default policy
 - * Rejects any modification violating referential integrity constraint
 2. *The Cascade Policy*
 - * Changes to the referenced attributes are mimicked at foreign key.
 - * e.g. delete a tuple in **MovieExec**, deletes related referencing tuple(s) from **Studio**
 3. *The Set-Null Policy*
 - * When a modification to the referenced relation affects a foreign-key value, the latter is changed to NULL.

* This applies to both UPDATE and DELETE

Example:

```

1  CREATE TABLE Movies (
2      title CHAR(30) PRIMARY KEY,
3      year INT PRIMARY KEY,
4      length INT,
5      genre VARCHAR(255),
6      studioName VARCHAR(255),
7      producerC# REFERENCES MovieExec(cert#)
8          ON DELETE SET NULL
9          ON UPDATE CASCADE
10 );
11

```

d)

```

2  CREATE TABLE Movies (
3      title CHAR(30) PRIMARY KEY,
4      year INT PRIMARY KEY,
5      length INT,
6      genre VARCHAR(255),
7      studioName VARCHAR(255),
8      producerC# VARCHAR(255)
9      FOREIGN KEY (title) REFERENCES StarsIn(movieTitle)
10 );

```

e)

```

2  CREATE TABLE StarsIn (
3      movieTitle CHAR(30) PRIMARY KEY,
4      movieYear INT PRIMARY KEY,
5      starName VARCHAR(255) PRIMARY KEY,
6      FOREIGN KEY (starName) REFERENCES MovieStar(name)
7          ON DELETE CASCADE
8  );

```

2. Yes. Set foreign-key constraint on StarsIn's movietitle to Movie's title.

```

1  CREATE TABLE Movies (
2      title CHAR(30) PRIMARY KEY,
3      year INT PRIMARY KEY,
4      length INT,
5      genre VARCHAR(255),
6      studioName VARCHAR(255),
7      producerC# VARCHAR(255),
8      FOREIGN KEY (title) REFERENCES StarsIn(movieTitle)
9  );
10

```

```
31 CREATE TABLE Product (  
32     maker CHAR(30),  
33     model INT PRIMARY KEY,  
34     type VARCHAR(255)  
35 );  
36  
37 CREATE TABLE PC (  
38     model INT PRIMARY KEY,  
39     speed FLOAT,  
40     ram INT,  
41     hd INT,  
42     price FLOAT,  
43     FOREIGN KEY (model) REFERENCES Product(model)  
44 );  
45  
46 CREATE TABLE Laptop (  
47     model INT PRIMARY KEY,  
48     speed FLOAT,  
49     ram INT,  
50     hd INT,  
51     screen INT,  
52     price FLOAT,  
53     FOREIGN KEY (model) REFERENCES Product(model)  
54 );  
55  
56 CREATE TABLE Printer (  
57     model INT PRIMARY KEY,  
58     color BOOLEAN,  
59     type VARCHAR(255),  
60     price FLOAT,  
61     FOREIGN KEY (model) REFERENCES Product(model)  
62 );  
63  
64
```

Correct Solution:

```
1 CREATE TABLE Product (  
2     maker CHAR(30),  
3     model INT PRIMARY KEY,  
4     type VARCHAR(255)  
5 );  
6  
7 CREATE TABLE PC (  
8     model INT PRIMARY KEY,  
9     speed FLOAT,  
10    ram INT,  
11    hd INT,  
12    price FLOAT,  
13    FOREIGN KEY (model) REFERENCES Product(model)  
14        ON DELETE CASCADE
```

```
15         ON UPDATE CASCADE
16     );
17
18     CREATE TABLE Laptop (
19         model INT PRIMARY KEY,
20         speed FLOAT,
21         ram INT,
22         hd INT,
23         screen INT,
24         price FLOAT,
25         FOREIGN KEY (model) REFERENCES Product(model)
26             ON DELETE CASCADE
27             ON UPDATE CASCADE
28     );
29
30     CREATE TABLE Printer (
31         model INT PRIMARY KEY,
32         color BOOLEAN,
33         type VARCHAR(255),
34         price FLOAT,
35         FOREIGN KEY (model) REFERENCES Product(model)
36             ON DELETE CASCADE
37             ON UPDATE CASCADE
38     );
39
40
```

```
41     CREATE TABLE Classes (
42         class CHAR(255) PRIMARY KEY,
43         type CHAR(2),
44         country CHAR(255),
45         numGuns INT,
46         bore INT,
47         displacement INT
48     );
49
50     CREATE TABLE Ships (
51         name CHAR(255) PRIMARY KEY,
52         class CHAR(255),
53         launched DATE,
54         FOREIGN KEY (class) REFERENCES Classes(class)
55             ON DELETE CASCADE
56             ON UPDATE CASCADE
57     );
58
59     CREATE TABLE Battles (
60         name CHAR(255) PRIMARY KEY,
61         date DATE
62     );
63
64     CREATE TABLE Outcome (
65         ship CHAR(255),
66         battle CHAR(255),

```

```

27     result CHAR(7),
28     PRIMARY KEY (ship, battle, result),
29     FOREIGN KEY (battle) REFERENCES Battles(name),
30         ON DELETE CASCADE
31         ON UPDATE CASCADE
32     FOREIGN KEY (ship) REFERENCES Ships(name),
33         ON DELETE CASCADE
34         ON UPDATE CASCADE
35 );
36
37

```

5. a)

```

2     CREATE TABLE Classes (
3         class CHAR(255) PRIMARY KEY,
4         type CHAR(2),
5         country CHAR(255),
6         numGuns INT,
7         bore FLOAT(3),
8         displacement INT
9     );
10
11     CREATE TABLE Ships (
12         name CHAR(255) PRIMARY KEY,
13         class CHAR(255),
14         launched DATE,
15         FOREIGN KEY (class) REFERENCES Classes(class)
16             ON DELETE CASCADE
17             ON UPDATE CASCADE
18     );

```

b)

```

2     CREATE TABLE Battles (
3         name CHAR(255) PRIMARY KEY,
4         date DATE
5     );
6
7     CREATE TABLE Outcome (
8         ship CHAR(255),
9         battle CHAR(255),
10        result CHAR(7),
11        PRIMARY KEY (ship, battle, result),
12        FOREIGN KEY (battle) REFERENCES Battles(name),
13            ON DELETE CASCADE
14            ON UPDATE CASCADE
15    );

```

c)

```

2     CREATE TABLE Ships (
3         name CHAR(255) PRIMARY KEY,
4         class CHAR(255),
5         launched DATE,
6         FOREIGN KEY (class) REFERENCES Classes(class)
7             ON DELETE CASCADE

```

```

7         ON UPDATE CASCADE
8     );
9
10    CREATE TABLE Outcome (
11        ship CHAR(255),
12        battle CHAR(255),
13        result CHAR(7),
14        PRIMARY KEY (ship, battle, result),
15        FOREIGN KEY (battle) REFERENCES Battles(name),
16            ON DELETE CASCADE
17            ON UPDATE CASCADE
18        FOREIGN KEY (ship) REFERENCES Ships(name),
19            ON DELETE CASCADE
20            ON UPDATE CASCADE
21    );
22

```

6. a)

```

2    Movies(title, year, length, genre, studioName, producerC#)
3
4    CREATE TABLE Movies (
5        ...
6        year INT PRIMARY KEY CHECK (year >= 1915),
7        ...
8    );

```

Notes:

- CHECK Constraints

– sets conditions that must hold for every value of an attribute

```

1    // Example 1
2    Studio(name, address, pressC#)
3
4
5    CREATE TABLE Studio (
6        ...
7        presC# INT REFERENCES MovieExec(cer#)
8            CHECK (presC# >= 10000)
9    );
10
11    // Example 2
12    MovieStar(name, address, gender, birthdate)
13
14    CREATE TABLE MovieStar (
15        ...
16        gender CHAR(1) CHECK (gender IN ('F', 'M')),
17        ...
18    );
19

```

```

b)  Movies(title, year, length, genre, studioName, producerC#)
    2
    3  CREATE TABLE Movies (
    4      ...
    5      length INT CHECK (length > 250 AND length < 60),
    6      ...
    7  );
    8

```

```

c)  Movies(title, year, length, genre, studioName, producerC#)
    2
    3  CREATE TABLE Movies (
    4      ...
    5      studioName VARCHAR(255) CHECK (studioName IN ('Disney', 'Fox',
    6      'MGM', 'Paramount')),
    7      ...
    8  );
    9

```

```

7. a) CREATE TABLE Laptop (
    2      ...
    3      speed FLOAT CHECK (speed >= 2.0),
    4      ...
    5  );
    6

```

```

b)  CREATE TABLE Printer (
    2      ...
    3      type VARCHAR(255) CHECK (type IN ('laser', 'ink-jet', 'bubble
    4      -jet')),
    5      ...
    6  );
    7

```

```

c)  CREATE TABLE Product (
    2      maker CHAR(30),
    3      model INT PRIMARY KEY,
    4      type VARCHAR(255) CHECK (type IN ('pc', 'laptop', 'printer'))
    5  );
    6

```

```

d)  CREATE TABLE Product (
    2      ...
    3      model INT PRIMARY KEY CHECK (type IN (
    4          (SELECT model FROM PC)
    5          UNION
    6          (SELECT model FROM Laptop)
    7          UNION
    8          (SELECT model FROM Printer)
    9      )),
    10     ...
    11 );
    12

```



```

8. a) CREATE TABLE MovieStar (
2     name CHAR(255) PRIMARY KEY,
3     address VARCHAR(255),
4     gender CHAR(1),
5     birthdate DATE,
6     CHECK (strftime('%Y', birthdate) <= (
7         SELECT movieYear FROM StarsIn WHERE starName = name
8     ))
9 );

10
11 CREATE TABLE StarsIn (
12     movieTitle VARCHAR(255) PRIMARY KEY,
13     movieYear INT PRIMARY KEY,
14     starName VARCHAR(255) PRIMARY KEY,
15     CHECK (movieYear >= (
16         SELECT strftime('%Y', birthdate) FROM MovieStar WHERE
starName = name
17     ))
18 );
19

```

```

b) CREATE TABLE Studio (
2     name CHAR(255) PRIMARY KEY,
3     address VARCHAR(255),
4     presC# INT,
5     UNIQUE KEY (address)
6 );
7

```

```

c) CREATE TABLE MovieStar (
2     name CHAR(255) PRIMARY KEY,
3     address VARCHAR(255),
4     gender CHAR(1),
5     birthdate DATE,
6     CHECK (name NOT IN (
7         SELECT name FROM MovieExec
8     ))
9 );

10
11 CREATE TABLE MovieExec (
12     name CHAR(255) PRIMARY KEY,
13     address VARCHAR(255),
14     cert# INT,
15     netWorth INT,
16     CHECK (name NOT IN (
17         SELECT name FROM MovieStar
18     ))
19 );
20

```

```

d) CREATE TABLE Studio (
2     name CHAR(30) PRIMARY KEY,
3     address VARCHAR(255),

```

```

4      presC# INT REFERENCES MovieExeC(cert#),
5      CHECK (name IN (
6          SELECT studioName FROM Movies
7      ))
8  );
9
10     CREATE TABLE Movies (
11         title CHAR(30) PRIMARY KEY,
12         year INT PRIMARY KEY,
13         length INT,
14         genre VARCHAR(255),
15         studioName VARCHAR(255),
16         producerC# PRIMARY KEY,
17         CHECK (studioName IN (
18             SELECT name FROM Studio
19         ))
20     );
21

```

9. a)

```

2      model INT PRIMARY KEY,
3      speed FLOAT,
4      ram INT,
5      hd INT,
6      price FLOAT,
7      CHECK (speed < 2.0 AND price <= 600)
8  );
9

```

b)

```

2      model INT PRIMARY KEY,
3      speed FLOAT,
4      ram INT,
5      hd INT,
6      screen INT,
7      price FLOAT,
8      CHECK (screen < 15 AND (hd >= 40 OR price < 1000))
9  );
10

```

10. a)

```

2      class CHAR(255) PRIMARY KEY,
3      type CHAR(2),
4      country CHAR(255),
5      numGuns INT,
6      bore INT,
7      displacement INT,
8      CHECK (bore <= 16)
9  );
10

```

```

b) CREATE TABLE Classes (
    class CHAR(255) PRIMARY KEY,
    type CHAR(2),
    country CHAR(255),
    numGuns INT,
    bore INT,
    displacement INT,
    CHECK (numGuns > 9 AND bore <= 14)
);

```

```

c) CREATE TABLE Ships (
    name CHAR(255) PRIMARY KEY,
    class CHAR(255),
    launched INT,
    CHECK (launched <= (
        SELECT strftime('%Y', date) FROM Battles
        INNER JOIN Outcomes ON Battles.name = Outcomes.battle
        WHERE ship = Ships.name
    ))
);

```

11. *gender* \neq NULL

12. a) ALTER TABLE Movies ADD PRIMARY KEY(title, year);

```

b) ALTER TABLE Movies ADD CHECK (producerC# IN (
    SELECT cert# IN MovieExec
));

```

Correct Solution:

```

1 ALTER TABLE MovieExec ADD FOREIGN KEY (producerC#) REFERENCES
2 MovieExec(cert#)

```

```

c) ALTER TABLE Movies ADD CHECK (NOT (length < 60 OR length > 250));
2

```

```

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

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

```

e) `ALTER TABLE Studios ADD UNIQUE (address);`
2

13. a) `ALTER TABLE Classes ADD PRIMARY KEY (class, country);`
2

b) `ALTER TABLE Outcomes ADD FOREIGN KEY (battle) REFERENCES Battles(name);`
2

c) `ALTER TABLE Outcomes ADD FOREIGN KEY (ship) REFERENCES Ships(name);`
2

d) `ALTER TABLE Classes ADD CHECK (NOT (numGuns > 14));`
2

e) `ALTER TABLE Classes ADD CHECK (launched <= (`
2 `SELECT strftime('%Y', date) FROM Battles`
3 `INNER JOIN Outcomes ON Battles.name = Outcomes.battle`
4 `WHERE ship = Ships.name`
5 `));`
6

14. a) `CREATE ASSERTION PCnoLaptop CHECK`
2 `(NOT EXISTS`
3 `(`
4 `(SELECT maker FROM Product type='laptop')`
5 `INTERSECT`
6 `(SELECT maker FROM Product type='pc')`
7 `)`
8 `);`
9

Notes:

- Assertion
 - **Syntax:** CREATE ASSERTION < assertion-name > CHECK (< condition >)
 - Creates check conditions
 - Can include subqueries
 - Covers all conditions

Example:

```

1      CREATE ASSERTION RichPres CHECK
2          (NOT EXISTS
3              (
4                  SELECT Studio.name
5                  FROM Studio, MovieExec
6                  WHERE presC# = cert# AND netWorth < 10000000
7              )
8          );
9

```

b)

```

1      CREATE ASSERTION laptopGteSpeed (
2          EXISTS (
3              SELECT p2.model FROM PC NATURAL JOIN Product AS p1,
4                  Laptop NATURAL JOIN Product AS p2
5              WHERE p1.speed >= p2.speed AND
6                  p1.maker = p2.maker
7          )
8      );
9

```

c)

```

1      CREATE ASSERTION higherPrice (
2          EXISTS (
3              SELECT Laptop.model FROM Laptop, PC
4              WHERE Laptop.ram > PC.ram AND
5                  Laptop.price > PC.price
6          )
7      );
8

```

d)

```

1      CREATE ASSERTION pcExists (
2          EXISTS (
3              (
4                  SELECT * FROM Product
5                  WHERE type='pc' AND model IN (SELECT model FROM PC)
6              )
7          );
8
9      CREATE ASSERTION laptopExists (
10         EXISTS (
11             (
12                 SELECT * FROM Product
13                 WHERE type='laptop' AND model IN (SELECT model FROM
Laptop)
14             )
15         );
16
17         CREATE ASSERTION printerExists (
18             EXISTS (
19                 (
20                     SELECT * FROM Product
21                     WHERE type='printer' AND model IN (SELECT model FROM
Printer)
22                 )

```

```

23 );
24

```

15. a)

```

CREATE ASSERTION twoShipsCantExist (
  NOT EXISTS (
    (
      SELECT * FROM Classes
      NATURAL JOIN Ships
      GROUP BY class
      HAVING COUNT(name) > 2
    )
  )
);

```

b)

```

CREATE ASSERTION bbAndbcBothCantExist (
  NOT EXISTS (
    (
      SELECT country FROM Classes
      where type='bb'
    ) INTERSECT
    (
      SELECT country FROM Classes
      where type='bc'
    )
  )
);

```

c)

```

CREATE ASSERTION noMoreThan9GunsInBattle (
  NOT EXISTS (
    (
      SELECT * FROM Classes NATURAL JOIN (
        SELECT * FROM Ships INNER JOIN Outcome.ship =
Ships.name
      ) WHERE numGuns > 9
    ) INTERSECT
    (
      SELECT * FROM Classes NATURAL JOIN (
        SELECT * FROM Ships INNER JOIN Outcome.ship =
Ships.name
      ) WHERE numGuns > 9 AND result = 'sunk'
    )
  )
);

```

d)

```

CREATE ASSERTION shipMustHaveFirstShipClassNameFirst (
  NOT EXISTS (
    SELECT * FROM Ships AS s1
    WHERE s1.class <> s1.name AND s1.launches <= ALL (
      SELECT launches FROM Ships AS s2 WHERE s2.class = s1.
class
    )
  )
);

```

9

```

e)  CREATE ASSERTION classWithSameName (
      EXISTS (
        SELECT * FROM Ships WHERE class = name
      )
    );

```

```

16 CREATE Table MovieExec (
    ...
    FOREIGN KEY (cert#) REFERENCES Studio(presC#)
    CHECK (presC# = cert#),
    CHECK (networth < 10000000)
);

```

Correct Solution:

```

1  CREATE Table MovieExec (
2      ...
3      FOREIGN KEY (cert#) REFERENCES Studio(presC#)
4          CHECK (presC# = cert#),
5      CHECK (networth < 10000000)
6  );
7

```

```

17 CREATE TRIGGER NetWorthTrigger
    AFTER UPDATE OF netWorth ON MovieExec
    REFERENCING
        OLD ROW AS OldTuple,
        NEW ROW AS NewTuple
    FOR EACH STATEMENT
    WHEN (5000000 > (SELECT SUM(netWorth) FROM MovieExec))
    BEGIN
        DELETE FROM MovieExec
        WHERE (name, address, cert#, netWorth) IN NewStuff;
        INSERT INTO MovieExec (SELECT * FROM OldStuff);
    END;

```

Notes:

- Trigger
 - Is also called *event-condition-action* rules
 - are only awakened when certain events, specified by the database programmer occur

- If condition is satisfied, the *action* associated with the trigger is performed

Example:

```

1  CREATE TRIGGER NetWorthTrigger
2  AFTER UPDATE OF netWorth ON MovieExec
3  REFERENCING
4      OLD ROW AS OldTuple,
5      NEW ROW AS NewTuple
6  FOR EACH ROW
7  WHEN (OldTuple.netWorth > NewTuple.netWorth)
8      UPDATE MovieExec
9      SET netWorth = OldTuple.netWorth
10     WHERE cer# = NewTuple.cert#;
11

```

18. a)

```

2  CREATE TRIGGER CheckPriceTrigger
3  AFTER UPDATE OF price ON PC
4  REFERENCING
5      OLD ROW AS OldTuple,
6      OLD TABLE AS OldStuff,
7      NEW ROW AS NewTuple
8  FOR EACH STATEMENT
9  WHEN (NewTuple.price <> OldTuple.price AND
10      NOT NewTuple.speed IN (
11          SELECT * FROM PC AS pc
12          WHERE NewTuple.price < pc.price AND
13              NewTuple.speed = pc.speed
14      ))
15      UPDATE PC
16      SET price = NewTuple.price
17      WHERE model = NewTuple.model;

```

Correct Solution:

```

1  CREATE TRIGGER CheckPriceTrigger
2  BEFORE UPDATE OF price ON PC
3  REFERENCING
4      OLD ROW AS OldTuple,
5      OLD TABLE AS OldStuff,
6      NEW ROW AS NewTuple
7  FOR EACH ROW
8  WHEN (NewTuple.price <> OldTuple.price AND
9      NOT EXISTS (
10         SELECT pc.speed FROM PC AS pc
11         WHERE NewTuple.price < pc.price AND
12             NewTuple.speed = pc.speed
13     ))
14      UPDATE PC
15      SET price = NewTuple.price
16      WHERE model = NewTuple.model;

```



```
17
b) CREATE TRIGGER CheckPriceTrigger
2 BEFORE INSERT ON Printer
3 REFERENCING
4 NEW ROW AS NewTuple
5 FOR EACH ROW
6 WHEN (NewTuple.model IN (
7 SELECT model FROM Product
8 ))
9 INSERT INTO Printer
10 NewTuple;
11
```

Correct Solution:

```
1 CREATE TRIGGER modelExists
2 BEFORE INSERT ON Printer
3 REFERENCING
4 NEW ROW AS NewTuple
5 FOR EACH STATEMENT
6 WHEN (NewTuple.model IN (
7 SELECT model FROM Product
8 ))
9 BEGIN
10 INSERT INTO Printer
11 NewTuple;
12 END;
13
```

```
c) CREATE TRIGGER CheckAVGPriceTrigger
2 BEFORE UPDATE ON Laptop
3 REFERENCING
4 OLD ROW AS OldTuple
5 NEW ROW AS NewTuple
6 NEW TABLE AS NewStuff
7 FOR EACH STATEMENT
8 WHEN (1500 >= (
9 SELECT AVG(price) FROM NewStuff
10 ))
11 BEGIN
12 UPDATE Laptop SET
13 model=NewTuple.model,
14 speed=NewTuple.speed,
15 ram=NewTuple.ram,
16 hd=NewTuple.hd,
17 screen=NewTuple.screen
18 price=NewTuple.price
19 WHERE model = OldTuple.model;
20 END;
21
```

```
d) CREATE TRIGGER CheckAVGPriceTrigger
2 BEFORE UPDATE ON Laptop
3 REFERENCING
4     OLD ROW AS OldTuple
5     NEW ROW AS NewTuple
6     NEW TABLE AS NewStuff
7 FOR EACH STATEMENT
8 WHEN (1500 >= (
9     SELECT AVG(price) FROM NewStuff
10 ))
11 BEGIN
12     UPDATE Laptop SET
13         model=NewTuple.model,
14         speed=NewTuple.speed,
15         ram=NewTuple.ram,
16         hd=NewTuple.hd,
17         screen=NewTuple.screen,
18         price=NewTuple.price
19     WHERE model = OldTuple.model;
20 END;
21
```

```
e) CREATE TRIGGER updatingRAM
2 AFTER UPDATE OF hd, ram ON PC
3 REFERENCING
4     OLD ROW AS OldTuple
5     NEW ROW AS NewTuple
6     NEW TABLE AS NewStuff
7 FOR EACH ROW
8 WHEN (100 < (
9     NewTuple.hd / NewTuple.ram
10 ))
11
12 UPDATE PC
13 SET ram=OldTuple.ram,
14     hd=OldTuple.hd
15 WHERE model = OldTuple.model;
16
```

```
f) CREATE TRIGGER checkInsertPC
2 BEFORE INSERT ON PC
3 REFERENCING
4     NEW ROW AS NewTuple
5 FOR EACH ROW
6 BEGIN
7 WHEN (NOT NewTuple.model IN (
8     (SELECT model FROM PC)
9     UNION
10    (SELECT model FROM Laptop)
11    UNION
12    (SELECT model FROM Printer)
13 ))
14
15 INSERT INTO PC NewTuple;
```

```
16  END;
17
18  /
19
20  CREATE TRIGGER checkInsertLaptop
21  BEFORE INSERT ON Laptop
22  REFERENCING
23      NEW ROW AS NewTuple
24  FOR EACH ROW
25  BEGIN
26  WHEN (NOT NewTuple.model IN (
27      (SELECT model FROM PC)
28      UNION
29      (SELECT model FROM Laptop)
30      UNION
31      (SELECT model FROM Printer)
32  ))
33
34  INSERT INTO Laptop NewTuple;
35  END;
36
37  /
38
39  CREATE TRIGGER checkInsertPrinter
40  BEFORE INSERT ON Laptop
41  REFERENCING
42      NEW ROW AS NewTuple
43  FOR EACH ROW
44  BEGIN
45  WHEN (NOT NewTuple.model IN (
46      (SELECT model FROM PC)
47      UNION
48      (SELECT model FROM Laptop)
49      UNION
50      (SELECT model FROM Printer)
51  ))
52
53  INSERT INTO Printer NewTuple;
54  END;
55
```

19. a)

```
2  CREATE TRIGGER addShip
3  AFTER INSERT ON Classes
4  REFERENCING
5      NEW ROW AS NewTuple
6      NEW TABLE AS NewStuff
7  FOR EACH STATEMENT
8  WHEN (NewTuple.class IS NOT NULL)
9  BEGIN
10     INSERT INTO Ships (NewTuple.class, NewTuple.class, NULL)
11  END;
```

```

b) CREATE TRIGGER limitDisplacement
2 AFTER INSERT ON Classes
3 REFERENCING
4     NEW ROW AS NewTuple
5     NEW TABLE AS NewStuff
6 FOR EACH ROW
7 WHEN (NewTuple.displacement > 35000)
8 BEGIN
9     UPDATE Classes
10    SET displacement = 35000
11    WHERE class = NewTuple.class;
12 END;
13

```

```

c) CREATE TRIGGER updateMissingParts
2 AFTER INSERT ON Outcomes
3 REFERENCING
4     NEW ROW AS NewTuple
5 FOR EACH ROW
6 WHEN (NOT (NewTuple.ship IN (SELECT name FROM Ships) OR
7     NewTuple.battle IN (SELECT name FROM Battles))
8 BEGIN
9     IF (NewTuple.ship NOT IN (SELECT name FROM Ships))
10    THEN
11        INSERT INTO Ships (NewTuple.ship, NewTuple.battle, NULL);
12    ELSE
13        INSERT INTO Battles (NewTuple.battle, NULL);
14    ENDIF;
15 END;
16

```

```

d) CREATE TRIGGER updateMissingParts
2 AFTER INSERT, UPDATE OF class ON Ships
3 REFERENCING
4     NEW ROW AS NewTuple
5     OLD ROW AS OldTuple
6 FOR EACH ROW
7 WHEN (20 > (SELECT( MAX(shipCount) FROM
8     ( SELECT COUNT(name) AS shipCount FROM Ships
9     NATURAL JOIN Classes
10    GROUP BY country)
11    )
12 BEGIN
13     IF (UPDATE(class))
14     THEN
15         UPDATE Ships SET class = OldTuple.class
16         WHERE name = OldTuple.name;
17     ELSE
18         DELETE FROM Ships WHERE name = NewTuple.name;
19     ENDIF;
20 END;
21

```