CSC343 Worksheet 5 Solution

June 19, 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 anttribute in another table
  - Added when creating table

  Example:

```
// Example 1
CREATE TABLE Studio (
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    presC# INT REFERENCES MovieExeC(cert#)
);

// Example 2
CREATE TABLE Studio (
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
    presC# INT,
    FOREIGN KEY (presC#) REFERENCES MovieExec(cert#)
);

```

b)
```
CREATE TABLE Movies (
    title CHAR(30) PRIMARY KEY,
    year INT PRIMARY KEY,
    length INT,
    genre VARCHAR(255),
    studioName VARCHAR(255),
    producerC# PRIMARY KEY
);
```

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

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

Correct Solution:

```
CREATE TABLE MovieExec (
    name CHAR(30),
    address VARCHAR(255),
    cert# INT PRIMARY KEY,
    FOREIGN KEY (cert#) REFERENCES Movies(producerC#)
        ON UPDATE CASCADE // Correction
        ON DELETE CASCADE // Correction
);
```

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 constant
    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:

```
    CREATE TABLE Movies (
        title CHAR(30) PRIMARY KEY,
        year INT PRIMARY KEY,
        length INT,
        genre VARCHAR(255),
        studioName VARCHAR(255),
        producerC# REFERENCES MovieExec(cert#)
            ON DELETE SET NULL
            ON UPDATE CASCADE
    );
```

d)

```
    CREATE TABLE Movies (
        title CHAR(30) PRIMARY KEY,
        year INT PRIMARY KEY,
        length INT,
        genre VARCHAR(255),
        studioName VARCHAR(255),
        producerC# VARCHAR(255)
        FOREIGN KEY (title) REFERENCES StarsIn(movieTitle)
    );
```

e)

```
    CREATE TABLE StarsIn (
        movieTitle CHAR(30) PRIMARY KEY,
        movieYear INT PRIMARY KEY,
        starName VARCHAR(255) PRIMARY KEY,
        FOREIGN KEY (starName) REFERENCES MovieStar(name)
            ON DELETE CASCADE
    );
```

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

```
    CREATE TABLE Movies (
        title CHAR(30) PRIMARY KEY,
        year INT PRIMARY KEY,
        length INT,
        genre VARCHAR(255),
        studioName VARCHAR(255),
        producerC# VARCHAR(255),
        FOREIGN KEY (title) REFERENCES StarsIn(movieTitle)
    );
```

3.
```sql
CREATE TABLE Product (
    maker CHAR(30),
    model INT PRIMARY KEY,
    type VARCHAR(255)
);

CREATE TABLE PC (
    model INT PRIMARY KEY,
    speed FLOAT,
    ram INT,
    hd INT,
    price FLOAT,
    FOREIGN KEY (model) REFERENCES Product(model)
);

CREATE TABLE Laptop (
    model INT PRIMARY KEY,
    speed FLOAT,
    ram INT,
    hd INT,
    screen INT,
    price FLOAT,
    FOREIGN KEY (model) REFERENCES Product(model)
);

CREATE TABLE Printer (
    model INT PRIMARY KEY,
    color BOOLEAN,
    type VARCHAR(255),
    price FLOAT,
    FOREIGN KEY (model) REFERENCES Product(model)
);


```

Correct Solution:

```sql
CREATE TABLE Product (
    maker CHAR(30),
    model INT PRIMARY KEY,
    type VARCHAR(255)
);

CREATE TABLE PC (
    model INT PRIMARY KEY,
    speed FLOAT,
    ram INT,
    hd INT,
    price FLOAT,
    FOREIGN KEY (model) REFERENCES Product(model)
        ON DELETE CASCADE
```

```
            ON UPDATE CASCADE
    );

    CREATE TABLE Laptop (
        model INT PRIMARY KEY,
        speed FLOAT,
        ram INT,
        hd INT,
        screen INT,
        price FLOAT,
        FOREIGN KEY (model) REFERENCES Product(model)
            ON DELETE CASCADE
            ON UPDATE CASCADE
    );

    CREATE TABLE Printer (
        model INT PRIMARY KEY,
        color BOOLEAN,
        type VARCHAR(255),
        price FLOAT,
        FOREIGN KEY (model) REFERENCES Product(model)
            ON DELETE CASCADE
            ON UPDATE CASCADE
    );


```

4.

```
    CREATE TABLE Classes (
        class CHAR(255) PRIMARY KEY,
        type CHAR(2),
        country CHAR(255),
        numGuns INT,
        bore INT,
        displacement INT
    );

    CREATE TABLE Ships (
        name CHAR(255) PRIMARY KEY,
        class CHAR(255),
        launched DATE,
        FOREIGN KEY (class) REFERENCES Classes(class)
            ON DELETE CASCADE
            ON UPDATE CASCADE
    );

    CREATE TABLE Battles (
        name CHAR(255) PRIMARY KEY,
        date DATE
    );

    CREATE TABLE Outcome (
        ship CHAR(255),
        battle CHAR(255),
```

```
        result CHAR(7),
        PRIMARY KEY (ship, battle, result),
        FOREIGN KEY (battle) REFERENCES Battles(name),
            ON DELETE CASCADE
            ON UPDATE CASCADE
        FOREIGN KEY (ship) REFERENCES Ships(name),
            ON DELETE CASCADE
            ON UPDATE CASCADE
    );


```

5. a)

```
    CREATE TABLE Classes (
        class CHAR(255) PRIMARY KEY,
        type CHAR(2),
        country CHAR(255),
        numGuns INT,
        bore FLOAT(3),
        displacement INT
    );

    CREATE TABLE Ships (
        name CHAR(255) PRIMARY KEY,
        class CHAR(255),
        launched DATE,
        FOREIGN KEY (class) REFERENCES Classes(class)
            ON DELETE CASCADE
            ON UPDATE CASCADE
    );

```

b)

```
    CREATE TABLE Battles (
        name CHAR(255) PRIMARY KEY,
        date DATE
    );

    CREATE TABLE Outcome (
        ship CHAR(255),
        battle CHAR(255),
        result CHAR(7),
        PRIMARY KEY (ship, battle, result),
        FOREIGN KEY (battle) REFERENCES Battles(name),
            ON DELETE CASCADE
            ON UPDATE CASCADE
    );

```

c)

```
    CREATE TABLE Ships (
        name CHAR(255) PRIMARY KEY,
        class CHAR(255),
        launched DATE,
        FOREIGN KEY (class) REFERENCES Classes(class)
            ON DELETE CASCADE
```

```
            ON UPDATE CASCADE
    );

    CREATE TABLE Outcome (
        ship CHAR(255),
        battle CHAR(255),
        result CHAR(7),
        PRIMARY KEY (ship, battle, result),
        FOREIGN KEY (battle) REFERENCES Battles(name),
            ON DELETE CASCADE
            ON UPDATE CASCADE
        FOREIGN KEY (ship) REFERENCES Ships(name),
            ON DELETE CASCADE
            ON UPDATE CASCADE
    );

```

6. a)

```
    Movies(title, year, length, genre, studioName, producerC#)

    CREATE TALBE Movies (
        ...
        year INT PRIMARY KEY CHECK (year >= 1915),
        ...
    );

```

Notes:

- CHECK Constraints
  - sets conditions that must hold for every value of an attribute

```
        // Example 1
        Studio(name, address, pressC#)


        CREATE TALBE Studio (
            ...
            presC# INT REFERENCES MovieExec(cer#)
                CHECK (presC# >= 10000)
        );

        // Example 2
        MovieStar(name, address, gender, birthdate)

        CREATE TALBE MovieStar (
            ...
            gender CHAR(1) CHECK (gender IN ('F', 'M')),
            ...
        );

```

b)
```
Movies(title, year, length, genre, studioName, producerC#)

CREATE TALBE Movies (
    ...
    length INT CHECK (length > 250 AND length < 60),
    ...
);

```

c)
```
Movies(title, year, length, genre, studioName, producerC#)

CREATE TALBE Movies (
    ...
    studioName VARCHAR(255) CHECK (studioName IN ('Disney', 'Fox', 'MGM', 'Paramount')),
    ...
);

```

7. a)
```
CREATE TABLE Laptop (
    ...
    speed FLOAT CHECK (speed >= 2.0),
    ...
);

```

b)
```
CREATE TABLE Printer (
    ...
    type VARCHAR(255) CHECK (type IN ('laser', 'ink-jet', 'bubble-jet')),
    ...
);

```

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

```

d)
```
CREATE TABLE Product (
    ...
    model INT PRIMARY KEY CHECK (type IN (
        (SELECT model FROM PC)
        UNION
        (SELECT model FROM Laptop)
        UNION
        (SELECT model FROM Printer)
    )),
    ...
);

```

8. a)

```
CREATE TABLE MovieStar (
    name CHAR(255) PRIMARY KEY,
    address VARCHAR(255),
    gender CHAR(1),
    birthdate DATE,
    CHECK (strftime('%Y', birthdate) <= (
        SELECT movieYear FROM StarsIn WHERE starName = name
    ))
);

CREATE TABLE StarsIn (
    movieTitle VARCHAR(255) PRIMARY KEY,
    movieYear INT PRIMARY KEY,
    starName VARCHAR(255) PRIMARY KEY,
    CHECK (movieYear >= (
        SELECT strftime('%Y', birthdate) FROM MovieStar WHERE
starName = name
    ))
);

```

b)

```
CREATE TABLE Studio (
    name CHAR(255) PRIMARY KEY,
    address VARCHAR(255),
    presC# INT,
    UNIQUE KEY (address)
);

```

c)

```
CREATE TABLE MovieStar (
    name CHAR(255) PRIMARY KEY,
    address VARCHAR(255),
    gender CHAR(1),
    birthdate DATE,
    CHECK (name NOT IN (
        SELECT name FROM MovieExec
    ))
);

CREATE TABLE MovieExec (
    name CHAR(255) PRIMARY KEY,
    address VARCHAR(255),
    cert# INT,
    netWorth INT,
    CHECK (name NOT IN (
        SELECT name FROM MovieStar
    ))
);

```

d)

```
CREATE TABLE Studio (
    name CHAR(30) PRIMARY KEY,
    address VARCHAR(255),
```

```
        presC# INT REFERENCES MovieExeC(cert#),
        CHECK (name IN (
            SELECT studioName FROM Movies
        ))
    );

    CREATE TABLE Movies (
        title CHAR(30) PRIMARY KEY,
        year INT PRIMARY KEY,
        length INT,
        genre VARCHAR(255),
        studioName VARCHAR(255),
        producerC# PRIMARY KEY,
        CHECK (studioName IN (
            SELECT name FROM Studio
        ))
    );
```

9. a)

```
    CREATE TABLE PC (
        model INT PRIMARY KEY,
        speed FLOAT,
        ram INT,
        hd INT,
        price FLOAT,
        CHECK (speed < 2.0 AND price <= 600)
    );
```

b)

```
    CREATE TABLE Laptop (
        model INT PRIMARY KEY,
        speed FLOAT,
        ram INT,
        hd INT,
        screen INT,
        price FLOAT,
        CHECK (screen < 15 AND (hd >= 40 OR price < 1000))
    );
```

10. a)

```
    CREATE TABLE Classes (
        class CHAR(255) PRIMARY KEY,
        type CHAR(2),
        country CHAR(255),
        numGuns INT,
        bore INT,
        displacement INT,
        CHECK (bore <= 16)
    );
```

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:

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

```

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

```

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);

```

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

```

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

```

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

```

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

```

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

```

14. a)
```
CREATE ASSERTION PCnoLaptop CHECK
    (NOT EXISTS
        (
            (SELECT maker FROM Product type='laptop')
            INTERSECT
            (SELECT maker FROM Product type='pc')
        )
    );

```

Notes:

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

Example:

```
    CREATE ASSERTION RichPres CHECK
        (NOT EXISTS
            (
                SELECT Studio.name
                FROM Studio, MovieExec
                WHERE presC# = cert# AND netWorth < 10000000
            )
        );

```

b)

```
    CREATE ASSERTION laptopGteSpeed (
        EXISTS(
            SELECT p2.model FROM PC NATURAL JOIN Product AS p1,
            Laptop NATURAL JOIN Product AS p2
            WHERE p1.speed >= p2.speed AND
                  p1.maker = p2.maker
        )
    );

```

c)

```
    CREATE ASSERTION higherPrice (
        EXISTS (
            SELECT Laptop.model FROM Laptop, PC
            WHERE Laptop.ram > PC.ram AND
                  Laptop.price > PC.price
        )
    ));

```

d)

```
    CREATE ASSERTION pcExists (
        EXISTS (
            (
                SELECT * FROM Product
                WHERE type='pc' AND model IN (SELECT model FROM PC)
            )
    );

    CREATE ASSERTION laptopExists (
        EXISTS (
            (
                SELECT * FROM Product
                WHERE type='laptop' AND model IN (SELECT model FROM
    Laptop)
            )
     );

     CREATE ASSERTION printerExists (
        EXISTS (
            (
                SELECT * FROM Product
                WHERE type='printer' AND model IN (SELECT model FROM
    Printer)
            )
```

```
);
```

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.launched <= ALL (
            SELECT launched FROM Ships AS s2 WHERE s2.class = s1.
class
        )
    )
);
```

```

```

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:

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

```

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:

```
    CREATE TRIGGER NetWorthTrigger
    AFTER UPDATE OF netWorth ON MovieExec
    REFERENCING
        OLD ROW AS OldTuple,
        NEW ROW AS NewTuple
    FOR EACH ROW
    WHEN (OldTuple.netWorth > NewTuple.netWorth)
        UPDATE MovieExec
        SET netWorth = OldTuple.netWorth
        WHERE cer# = NewTuple.cert#;

```