CSC343 Worksheet 5

June 18, 2020

1. **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#)
StarsIn(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. Modifications to MovieExec that violate this constraint are rejected.

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

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

   d) A movie that appears in Stars In must also appear in Movie. Handle violations by rejecting the modification.

   e) A star appearing in Stars In must also appear in MovieStar. Handle violations by deleting violating tuples.

2. **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?

3. **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.

4. **Exercise 7.1.4:** Suggest suitable keys for the relations of the battleships database

```
Classes(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.

5. **Exercise 7.1.5:** 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**.

6. **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.

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

7. **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(model, color, type, price)
```

of Exercise 2.4.1.

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

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

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

   d) A model of a product must also be the model of a PC, a laptop, or a printer.

8. **Exercise 7.2.3:** Write the following constraints as tuple-based CHECK constraints 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 possible to maintain tuple-based constraints in the face of deletions.

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

   b) No two studios may have the same address.

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

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

9. **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.

10. **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.

11. **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 constraints enforce if the value of gender is NULL?

12. **Exercise 7.3.1:** E xercise 7.3.1: Show how to alter your relation schemas for the 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#)
```

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.

13. **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**.

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

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

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

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

14. **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.

Comparison of Constraints

The following table lists the principal differences among attribute-based checks, tuple-based checks, and assertions.

Type of Constraint	Where Declared	When Activated	Guaranteed to Hold?
Attribute-based `CHECK`	With attribute	On insertion to relation or attribute update	Not if subqueries
Tuple-based `CHECK`	Element of relation schema	On insertion to relation or tuple update	Not if subqueries
Assertion	Element of database schema	On any change to any mentioned relation	Yes

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

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

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

15. **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.

b) No country may have both battleships and battlecruisers.

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

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

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

16. **Exercise 7.4.3:** The assertion of Exercise 7.11 can be written as two tuplebased constraints. Show how to do so.

17. **Exercise 7.5.1:** Write the triggers analogous to Fig. 7.6 for the insertion and deletion events on **MovieExec**.

18. **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:

```
1    Product(maker, model, type)
2    PC(model, speed, ram, hd, price)
3    Laptop(model, speed, ram, hd, screen, price)
4    Printer(model, color, type, price)
5
```

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

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

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

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.

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 P rin te r.

19. **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.

```
1    Classes(class, type, country, numGuns, bore, displacement)
2    Ships(name, class, launched)
3    Battles(name, date)
4    Outcomes(ship, battle, result)
5
```

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

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.

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

d) When there is an insertion into **Ships** or an update of the c la s s attribute of **Ships**, check that no country has more than 20 ships.