Santa Jose State University

Department of Computer Science

Database Management Systems - I (CS 157A)

Homework #3 (5 pts)

Answers are in blue text

Questions on Chapters (7 + 8):

Please answer the following questions:

• Q1 [1 pt]: using the same Movie schema we discussed in HW #2:

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:

a) The producer of a movie must be someone mentioned in MovieExec. Modification to MovieExec that violate this constraint are rejected?

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

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

NULL.

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

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

d) A movie that appears in StarIn must also appear in Movie. Handle violations by rejecting the modifications?

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

• Q2 [1 pt]: Write the following assertion to this schema:

Product(maker, model, type)
PC(model, speed, ram, hd, price) ← hd: hard disk
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)

a) No manufacturer of PC's may also make laptops?

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

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

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

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

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

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

```
 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)

 );
```

• Q3 [1 pt]: Write the following as triggers for the following schema. In each case disallow or undo the modification if it does not satisfy the stated constraint.

Product(maker, model, type)
PC(model, speed, ram, hd, price) ← hd: hard disk
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)

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

```
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 (model1, speed, ram, hd, price) IN NewStuff;
        INSERT INTO PC
                (SELECT * FROM OldStuff);
END;
```

b) When inserting a new printer, check that the model number exists in product?

```
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) When making any modification to the laptop relation, check that the average price of laptop for each manufacturer is at least $1,500?

```
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) 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?

```
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 < NeweRow.ram * 100)
BEGIN
        DELETE FROM PC
WHERE (model, speed, ram, hd, price) IN Newstuff;
INSERT INTO PC
                        (SELECT * FROM OldStuff);
END;
```

• Q4 [1 pt]: Construct the following Views from the Schema below:

MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)

a) A view RichExec giving the name, address, certificate number, and net worth of all executives with a net worth of at least $10,000,000?

```
CREATE VIEW RichExec AS
SELECT * FROM MovieExec WHERE netWorth >= 10000000;
```

b) A view StudioPress giving the name, address, and certificate number of all executives who are studio presidents?

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

c) A view ExecutiveStar giving the name, address, gender, birth date, certificate number, and net worth of all individuals who are both executives and stars?

CREATE VIEW ExecutiveStar (name, address, gender,
birthdate, cert#, newWorth) 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;

• Q5 [1 pt]: Using the following base Tables:

Product(maker, model, type)

PC(model, speed, ram, hd, price)

Suppose we create the following View:

```
 CREAT  VIEW  NewPC  AS
                SELECT  maker, model, speed, ram, hd, price
            FROM  Product, PC
            WHERE  Product,model = PC.model  AND type = 'PC';
```

Notice that we have made a check for consistency: that the model number not only appears in the PC relation, but the type attribute of Product indicates that the product is a PC.

a) Is this View updatable?

No, becuase it is constructed from two different relations

b) Write an instead-of trigger to handle an insertion into the view?

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

c) Write an instead-of trigger to handle an update of the price?

```
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) Write an instead-of trigger to handle a deletion of a specified tuple from this view?

```
CREATE TRIGGER NewPCUpdate
INSTEAD OF DELETE ON NewPC
REFERENCING OLD ROW AS OldRow
FOR EACH ROW
 (DELETE FROM Product WHERE model = OldModel.model)
 (DELETE FROM PC WHERE model = OldRow.model);
```