Week 4

Theory

- Foreign key
  - Referential integrity constraint
  - "foreign-key constraints," assert that a value appearing in one relation must also appear in the primary-key component(s) of another relation.
  - The referenced attributes must be primary key of their relation/unique
  - Referenced values should exist in the table it references
  - Declared using either "REFERENCES <table>(<attribute>)"
  - Or "FOREIGN KEY (<attributes>) REFERENCES <table> (<attributes>)"
- Enforcing foreign-key constraint
  - System throws error in following cases:
    a) We try to insert a new `Studio` tuple whose `presC#` value is not `NULL` and is not the `cert#` component of any `MovieExec` tuple.

    b) We try to update a `Studio` tuple to change the `presC#` component to a non-`NULL` value that is not the `cert#` component of any `MovieExec` tuple.

    c) We try to delete a `MovieExec` tuple, and its `cert#` component, which is not `NULL`, appears as the `presC#` component of one or more `Studio` tuples.
  -
    d) We try to update a `MovieExec` tuple in a way that changes the `cert#` value, and the old `cert#` is the value of `presC#` of some movie studio.
  -
    - For the last two we have some options (p305-306):
      - Default policy: reject such changes
      - Cascade policy: changes to the referenced attributes are mimicked at the foreign key
      - Set-null policy: changes to the referenced relations, means the foreign-key value is set to NULL
    - ^Could be chosen independently for deletes and updates:
      - Stated together with foreign key.
      - ON DELETE/UPDATE SET NULL /CASCADE

Constraints on attributes and tuples:

- Not-null constraint:
  - Set on single attribute
- CHECK constraint:
  - Checks that some constraint is valid in regards to some attribute
  - Could also be check involving multiple attributes

1.1 p310

Exercise 1.1: Our running example movie database 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 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 `StarsIn` must also appear in `Movie`. Handle violations by rejecting the modification.

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

Obs.: In the following answers I have some left over constraints, from earlier.

a. …

```
CREATE TABLE Movies (
  title        CHAR(40),
  year         INT,
  length       INT,
  genre        CHAR(40),
  studioName   CHAR(40),
  producerCert INT REFERENCES MovieExec (cert),
  PRIMARY KEY (title, year)
);
```

b. …

```
CREATE TABLE Movies (
  title        CHAR(40),
  year         INT,
  length       INT,
  genre        CHAR(40),
  studioName   CHAR(40),
  producerCert INT REFERENCES MovieExec (cert) ON UPDATE SET NULL ON DELETE SET
NULL,
  PRIMARY KEY (title, year)
);
```

c. …

```
CREATE TABLE Movies (
  title        CHAR(40),
  year         INT,
  length       INT,
  genre        CHAR(40),
  studioName   CHAR(40),
  producerCert INT REFERENCES MovieExec (cert) ON UPDATE CASCADE ON DELETE
CASCADE ,
  PRIMARY KEY (title, year)
);
```

d. …

```
CREATE TABLE StarsIn (
  movieTitle CHAR(40),
  movieYear  INT,
  starName   CHAR(40),
  FOREIGN KEY (movieTitle, movieYear) REFERENCES Movies (title, year),
  PRIMARY KEY (movieTitle, movieYear, starName)
);
```

e. …

```
CREATE TABLE StarsIn (
  movieTitle CHAR(40),
  movieYear  INT,
  starName   CHAR(40),
  FOREIGN KEY (starName) REFERENCES MovieStar (name) ON DELETE CASCADE ON UPDATE
CASCADE,
  PRIMARY KEY (movieTitle, movieYear, starName)
);
```

1.2 p310

! **Exercise 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?

You can do it. But it will not be optimal due to we then have to insert into the relationship, before inserting into the entity. Not a behaviour we want.

2.1 p315

Exercise 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.

a. …

```
CREATE TABLE Movies (
  title         CHAR(40),
  year          INT CHECK (year > 1915),
  length        INT,
  genre         CHAR(40),
  studioName    CHAR(40) REFERENCES Studio (name),
  producerCert INT,
  PRIMARY KEY (title, year)
);
```

b. …

```
CREATE TABLE Movies (
  title         CHAR(40),
  year          INT,
  length        INT CHECK (length > 60 AND length < 250),
  genre         CHAR(40),
  studioName    CHAR(40) REFERENCES Studio (name),
  producerCert INT,
  PRIMARY KEY (title, year)
);
```

c. …

```
CREATE TABLE Movies (
  title         CHAR(40),
  year          INT,
  length        INT,
  genre         CHAR(40),
  studioName    CHAR(40) REFERENCES Studio (name) CHECK (studioName IN ('Disney',
'Fox', 'MGM', 'Paramount')),
  producerCert INT,
  PRIMARY KEY (title, year)
);
```

2.2 p316

Exercise 2.2: 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.

!! e) If a producer of a movie is also the president of a studio, then they must be the president of the studio that made the movie.

a. …

```
CREATE TABLE MovieStar (
  name      CHAR(40),
  address   CHAR(40),
  gender    CHAR(40),
  birthdate DATE,
  CHECK (birthdate > ALL (SELECT year
                          FROM movies, starsin
                          WHERE title = movieTitle)),
  PRIMARY KEY (NAME)
);
```

b. …

```
CREATE TABLE Studio (
  name           CHAR(40),
  address        CHAR(40),
  presidentCert INT,
  CHECK (address NOT IN (SELECT address
                         FROM studio)),
  PRIMARY KEY (name)
);
```

c. …

```
CREATE TABLE MovieStar (
  name      CHAR(40),
  address   CHAR(40),
  gender    CHAR(40),
  birthdate DATE,
  CHECK (name NOT IN (SELECT name
                      FROM movieexec)),
  PRIMARY KEY (NAME)
);
```

d. …

```
CREATE TABLE Studio (
  name           CHAR(40),
  address        CHAR(40),
  presidentCert INT,
  CHECK (name in (select studioName from movies)),
  PRIMARY KEY (name)
);
```

e. …

```
CREATE TABLE Movies (
  title         CHAR(40),
  year          INT,
  length        INT,
  genre         CHAR(40),
  studioName    CHAR(40),
  producerCert INTEGER,
  CHECK (
    producerCert NOT IN (SELECT presidentCert FROM studio)
    OR (
      producerCert IN (SELECT presidentCert FROM studio)
      AND
      studioName IN (SELECT name FROM Studio WHERE presidentCert = producerCert)
    )
  ),
  PRIMARY KEY (title, year)
);
```

4.1 p322

Exercise 4.1: Write the following assertions. The database schema is from the "PC" example:

```
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.

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.

a. ...

```
CREATE ASSERTION manPC CHECK ( NOT EXISTS (
      ( SELECT maker
        FROM product
        WHERE type = 'laptop')
        INTERSECT
      ( SELECT maker
        FROM product
        WHERE type = 'pc')
);
```

b. …

```
CREATE ASSERTION lapSpeed CHECK ( Select maker from product where type='pc' in (
SELECT p1.maker
FROM product p1, product p2, laptop, pc
WHERE
p1.maker = p2.maker
AND
p1.type = 'pc'
AND
p2.type = 'laptop'
AND
p1.model = pc.model
AND
p2.model = laptop.model
AND
pc.speed <= laptop.speed;
)
)
```

c. …

```
Create ASSERTION ( CHECK NOT EXISTS (
  SELECT *
  FROM pc, laptop
  WHERE laptop.ram > pc.ram AND laptop.price <= pc.price
);
```

d. …

```
CREATE ASSERTION prodInPC ( CHECK
  (NOT EXISTS
    SELECT model, TYPE FROM product WHERE TYPE ='pc' AND (model) NOT IN ( SELECT
model FROM pc);
  )
  AND
  (NOT EXISTS
    SELECT model, TYPE FROM product WHERE TYPE ='laptop' AND (model) NOT IN (
SELECT model FROM laptop);
  )
  (NOT EXISTS
    SELECT model, TYPE FROM product WHERE TYPE ='printer' AND (model) NOT IN (
SELECT model FROM printer);
  )
);
```

5.2 p329

Exercise 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:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
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.

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 `Printer`.

a. …

```
CREATE TRIGGER pcPrice
  AFTER UPDATE OF price
  ON pc
  REFERENCING
    OLD ROW AS oldTuple
    NEW ROW AS newTuple
  FOR EACH ROW
  WHEN (newTuple.price < ANY (SELECT price
                              FROM pc
                              WHERE pc.speed = newTuple.speed))
    UPDATE pc
    SET price = oldTuple.price
    WHERE pc.model = oldTuple.model
;
```

b. …

```
CREATE TRIGGER printerExist
  AFTER INSERT
  ON printer
  REFERENCING
    NEW ROW AS newTuple
  FOR EACH ROW
  WHEN (newTuple.model NOT IN (SELECT model from product))
    DELETE FROM printer
    WHERE model = newTuple.model
;
```

c. …

```
CREATE TRIGGER laptopModification
  AFTER INSERT ON laptop OR UPDATE of price on laptop
  REFERENCING
    OLD TABLE AS oldTable
    NEW TABLE AS newTable
  FOR EACH STATEMENT
  WHEN (1500 <= ALL (select avg(price) from laptop,product group by maker))
  BEGIN
    DELETE FROM laptop
    WHERE (model, speed,ram,hd,screen,price) IN newTable;
    INSERT INTO laptop (SELECT * from oldTable);
  END
;
```

d. …

```
CREATE TRIGGER ramHD
  AFTER UPDATE
  OF ram,hd ON pc
  REFERENCING
    OLD ROW AS oldTuple
    NEW ROW AS newTuple
  FOR EACH ROW
  WHEN (newTuple.ram >= 100 * newTuple.hd)
    UPDATE pc
    SET ram = oldTuple.ram, hd = oldTuple.hd
    WHERE model = newTuple.model
;
```

e. …

```
CREATE TRIGGER prodModels
  AFTER INSERT
  on printer OR
  INSERT on laptop OR
  INSERT on printer
  REFERENCING
    NEW ROW AS newTuple
  FOR EACH ROW
  WHEN (newTuple.model in (select model from printer UNION SELECT model from
laptop union SELECT model from pc))
```