TDT4145: Øving 3

Oppgave 1

a) Man kan tilføye en regel/restriksjon som sier at refererte tabeller skal gjøre en handling hvis foreldre-tabellen (i dette tilfellet Film-tabellen) endres eller slettes.

Dette gjør man som følger

```
filmid INTEGER NOT NULL,
CONSTRAINT fk_film
FOREIGN KEY (filmid)
    REFERENCES Film(filmid)
    ON DELETE CASCADE
    ON UPDATE CASCADE;
```

`ON DELETE` og `ON UPDATE` definerer handling som skjer når den refererte nøkkelen (filmid) slettes eller oppdateres. `CASCADE` er en handling som tilsier at alle rader som referere gjennom fremmednøkkelen til foreldre-tabellen (Film) skal automatisk slettes/oppdateres når tilsvarende rad i foreldre-tabellen slettes/oppdateres

b) Skriv nå kode som konstruerer disse tabellene i SQL. Velg selv passende datatyper for attributtene og husk primær- og fremmednøkler. Ta også med kravet presentert i a).

```sql
DROP DATABASE IF EXISTS Movies;
CREATE DATABASE Movies;
USE Movies;

CREATE Table director (
        directorId INT NOT NULL PRIMARY KEY,
    name VARCHAR(30)
);

CREATE TABLE movie (
        movieId INT NOT NULL PRIMARY KEY,
        title VARCHAR(30),
        productionYear SMALLINT,
        directorId INT,
        CONSTRAINT m_fk_director
            FOREIGN KEY (directorId) REFERENCES director(directorId)
            ON DELETE CASCADE
);

CREATE Table actor (
    actorId INT NOT NULL PRIMARY KEY,
    name VARCHAR(30),
    birthYear SMALLINT
);

CREATE TABLE actorinmovie(
    movieId INT,
    actorId INT,
    role VARCHAR(30),
    CONSTRAINT aim_fk_movie
        FOREIGN KEY (movieId) REFERENCES movie(movieId)
            ON DELETE CASCADE,
    CONSTRAINT aim_fk_actor
        FOREIGN KEY (actorId) REFERENCES actor(actorId)
            ON DELETE CASCADE
);
```

```sql
CREATE TABLE genre(
    genreId INT NOT NULL PRIMARY KEY,
    name VARCHAR(50),
    description VARCHAR(200)
);

CREATE TABLE genreinmovie (
    genreId INT,
    movieId INT,
    CONSTRAINT gim_fk_genre
        FOREIGN KEY (genreId) REFERENCES genre(genreId)
            ON DELETE CASCADE
            ON UPDATE CASCADE,
    CONSTRAINT gim_fk_movie
        FOREIGN KEY (movieId) REFERENCES movie(movieId)
            ON DELETE CASCADE
            ON UPDATE CASCADE
);

SHOW TABLES;
```

c) Skriv SQL-setninger som legger inn følgende data i databasen:

```sql
INSERT INTO director VALUES(1, "Peyton Reed"),(2, "Tom Shadyac");
INSERT INTO movie VALUES(1, "Yes Man", 2008, 1);
INSERT INTO actor VALUES(1 , "Jim Carrey", 1962);
INSERT INTO actorinmovie VALUES(1,1,"Carl");
```

d) Skriv en SQL-setning som oppdaterer navnet til Jim Carrey til James Eugene Carrey.

```sql
UPDATE actor SET name="Jim Eugene Carrey" WHERE name="Jim Carrey";
```

---

e) Skrive en SQL-setning som sletter Tom Shadyac fra databasen.

```sql
DELETE FROM director WHERE name="Tom Shadyac";
```

Oppgave 2

a) Henter ut filmID, tittel, produksjonsår og regissørID på alle filmer.
Vi kan også spesifisere hver kolonne (tittel, filmid osv.) istedenfor *

```
SELECT * FROM film;
```

b) Finner navn på alle skuespillere født senere enn 1960.

```
SELECT navn FROM skuespiller WHERE fødselsår > 1960;
```

c) Finner navn på alle skuespillere født på 80-tallet, sortert i alfabetisk rekkefølge.

```
SELECT navn FROM skuespiller WHERE fødselsår BETWEEN 1980 AND 1990 ORDER BY navn
```

d) Finner titlene på alle filmene og de tilhørende rollene som "Morgan Freeman" harspilt.

```
SELECT tittel, rolle FROM skuespillerifilm
NATURAL JOIN skuespiller
NATURAL JOIN film
WHERE navn ="Morgan Freeman";
```

e) Henter ut de distinkte titlene på filmene hvor regissøren og en av skuespillerne i filmen har likt navn

```
SELECT tittel FROM skuespillerifilm
NATURAL JOIN skuespiller s
INNER JOIN regissør r
ON (s.navn = r.navn)
NATURAL JOIN film;
```

f) Finner antallet skuespillere som har et navn som starter på “C”.

```
SELECT COUNT(navn) FROM skuespiller WHERE navn LIKE 'C%';
```

g) For hver sjanger finner sjangernavnet og antallet filmer av den sjangeren

```
SELECT navn, count(*) as antall_filmer FROM sjangerforfilm
NATURAL JOIN sjanger
GROUP BY sjangerid;
```

h) Finner navnet på skuespillere som har spilt i filmen “Ace Ventura: Pet Detective”,men aldri i filmen “Ace Ventura: When Nature Calls”.

Filtrerer vekk de som er med i begge og deretter henter ut kun de som er med i Pet Detective

```
SELECT navn FROM (
    SELECT tittel, navn FROM skuespillerifilm
    NATURAL JOIN film NATURAL JOIN skuespiller WHERE tittel LIKE 'Ace Ventura%'
    GROUP BY skuespillerid HAVING COUNT(*) < 2
) sf
WHERE sf.tittel = 'Ace Ventura: Pet Detective';
```

i) For hver film finner navnet på filmen, filmID og gjennomsnittlig fødselsår på skuespillerne i filmen. Vi ønsker kun å få med de filmene som har gjennomsnittlig fødselsår større enn gjennomsnittlig fødselsår for alle skuespillerne i databasen.(Hint: Her kan det være lurt med en underspørring i en HAVING-del etteraggregeringen).

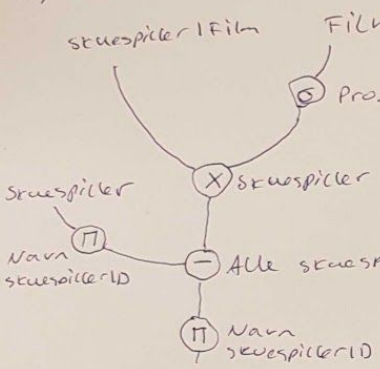
```
SELECT tittel, filmid, AVG(fødselsår) FROM skuespillerifilm
NATURAL JOIN film
NATURAL JOIN skuespiller
GROUP BY filmid HAVING AVG(fødselsår) > (
    SELECT AVG(fødselsår) FROM skuespiller
);
```

Oppgave 3: Relasjonsalgebra

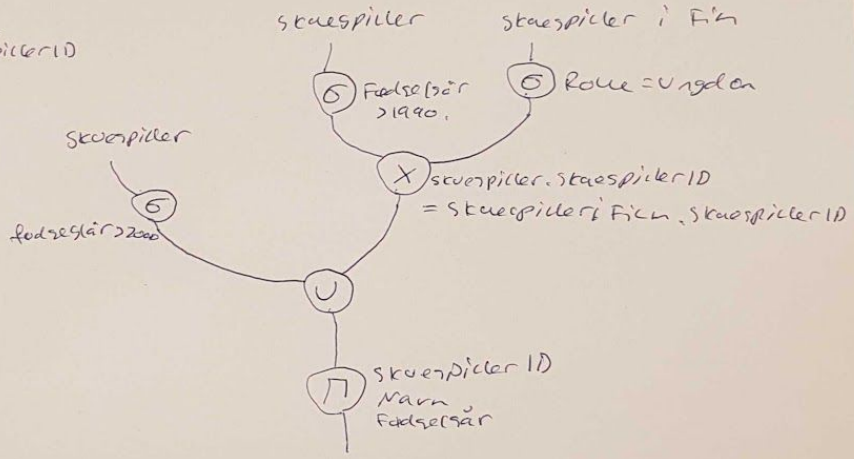
Gruppe 206

σ = select, Π = project, U = Union, ρ = rename,
− = set different, X = cartesian product

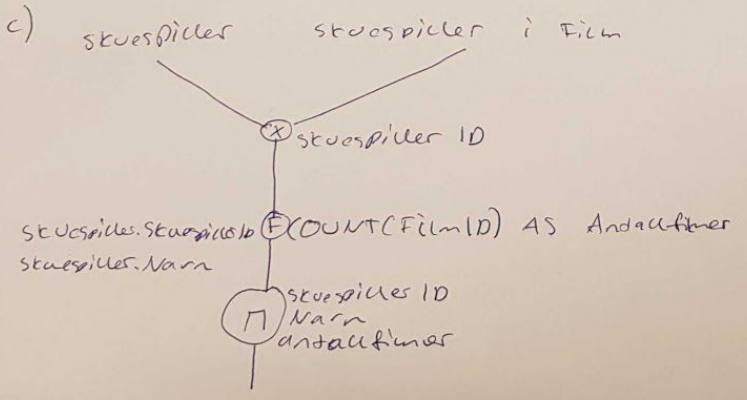
a)



b)



c)



Oppgave 4

a) 10 felter

b) Lage en egen tabell med fakultetskode, fakultetsnavn og fakultetsbygg. I orginaltabellen trenger man da kun fakultetskode.

Oppgave 5

a)
A -> C kan ikke stemme, ettersom a_1 gir både c_1 og c_2
D -> C kan ikke stemme, ettersom d_2 gir både c_2 og c_4
A kan ikke være en kandidatnøkkel, ettersom den ikke entydig definerer hver rad.

b)
A+ = AC
D+ = D
BC+ = BCD
AB+ = ABCD