Exercise 3: TDT4145 - Data Modelling, Databases and Database Management Systems

alexaoh, jimoskar

February 2021

Task 1: Construct tables and insert data in SQL

a)

When creating each table with SQL one can specify an action for ON DELETE and ON UPDATE. These are called referential triggered actions. Some alternatives are CASCADE, SET NULL and SET DEFAULT. In this case, specifying CASCADE for both ON DELETE and UPDATE on 'SkuespillerIFilm' and 'SjangerForFilm' would ensure that the right updates are made, i.e. that all tuples in these tables that reference a deleted movie-tuple also get deleted.

b)

```
create table regissør(
RegissørID INT primary key,
Navn VARCHAR(30));

create table sjanger(
SjangerID INT primary key,
Navn VARCHAR(50),
Beskrivelse VARCHAR(100));

create table skuespiller(
SkuespillerID INT primary key,
Navn VARCHAR(30),
Fødselsår INT);

create table film(
FilmID INT primary key,
Tittel VARCHAR(30),
Produksjonsår INT,
RegissørID INT,
CONSTRAINT FK
FOREIGN KEY (RegissørID) references regissør(RegissørID));

create table skuespillerifilm(
FilmID INT,
SkuespillerID INT,
Rolle VARCHAR(30),
PRIMARY KEY (FilmID, SkuespillerID),
FOREIGN KEY (FilmId) REFERENCES film(FilmId)
```

```
        ON DELETE CASCADE  ON UPDATE CASCADE);

create table sjangerforfilm(
SjangerID INT,
FilmID INT,
CONSTRAINT PK PRIMARY KEY (FilmID, SjangerID),
CONSTRAINT FKFIlm FOREIGN KEY (FilmID) references film(FilmID)
        ON DELETE CASCADE  ON UPDATE CASCADE,
CONSTRAINT FKSjanger FOREIGN KEY (SjangerID) references sjanger(SjangerID));
```

c)

```
INSERT INTO regissør
VALUES (1, 'Peyton Reed'), (2, 'Tom Shadyac');

INSERT INTO film
VALUES (1, 'Yes Man', 2008, 1);

INSERT INTO skuespiller
VALUES (1, 'Jim Carrey', 1962);

INSERT INTO skuespillerifilm
VALUES (1, 1, 'Carl');
```

d)

```
UPDATE skuespiller
SET Name = 'Jim Eugene Carrey'
Where SkuespillerID = 1;
```

e)

```
DELETE FROM regissør
WHERE Navn = 'Tom Shadyac';
```

Task 2: Writing select statements in SQL

a)

```
SELECT *
FROM film;
```

b)

```sql
SELECT Navn
FROM skuespiller AS s
WHERE s.Fødselsår > 1960;
```

c)

```sql
SELECT Navn
FROM skuespiller AS s
WHERE (s.Fødselsår >= 1980 AND s.Fødselsår <= 1989)
ORDER BY Navn;
```

d)

```sql
SELECT Tittel, Rolle
FROM (film AS f NATURAL JOIN skuespillerifilm AS sif) NATURAL JOIN
    skuespiller as sk
WHERE Navn = 'Morgan Freeman';
```

e)

```sql
SELECT DISTINCT
    dt2.Tittel
FROM
    (SELECT
        s.Navn, sif.FilmID
    FROM
        skuespiller AS s
    NATURAL JOIN skuespillerifilm AS sif) dt1
        INNER JOIN
    (SELECT
        r.Navn, f.Tittel, f.FilmID
    FROM
        regissør AS r
    NATURAL JOIN film AS f) dt2 ON (dt1.FilmID = dt2.FilmID)
WHERE
    dt1.Navn = dt2.Navn
```

f)

```sql
SELECT
    COUNT(s.SkuespillerID)
FROM
    skuespiller AS s
WHERE s.Navn LIKE 'C%';
```

g)

```sql
SELECT
    sj.navn, COUNT(*) AS filmer
FROM
    sjanger AS sj
        INNER JOIN
    sjangerforfilm AS sjf ON sj.SjangerID = sjf.SjangerID
GROUP BY sj.navn
```

h)

```sql
SELECT
    s.Navn
FROM
    skuespiller AS s
        INNER JOIN
    (SELECT
        *
    FROM
        skuespillerifilm AS sif
    NATURAL JOIN film AS f) dt1 ON (s.SkuespillerID = dt1.SkuespillerID)
WHERE
    dt1.Tittel = 'Ace Ventura: Pet Detective'
        AND s.SkuespillerID NOT IN (SELECT
            sif2.SkuespillerID
        FROM
            skuespillerifilm AS sif2
                INNER JOIN
            film AS f2 ON f2.FilmID = sif2.FilmID
        WHERE
            f2.Tittel = 'Ace Ventura: When Nature Calls');
```

i)

```sql
SELECT
    f.Tittel, f.FilmID, AVG(dt.Fødselsår) AS mean
FROM
    film AS f
        NATURAL JOIN
    (SELECT
```

```
        s.Fødselsår, sif.FilmID
    FROM
        skuespillerifilm AS sif
    NATURAL JOIN skuespiller AS s) dt
GROUP BY f.Tittel , f.FilmID
HAVING mean > (SELECT
        AVG(Fødselsår)
    FROM
        skuespiller);
```

Task 3: More queries in relational algebra

$Joined_1 \leftarrow \pi_{ActorID,Name}(Actor) \star \pi_{MovieID,ActorID}(ActorInMovie)$
$Joined_2 \leftarrow \pi_{MovieID,ProductionYear}(Movie) \star Joined_1$
$ActorAfter2014 \leftarrow \pi_{ActorID,Name}(\sigma_{ProductionYear>2014}(Joined2))$
$\pi_{ActorID,Name}(Actor) - ActorAfter2014$

b)

$Actor_{2000} \leftarrow \sigma_{BirthYear>2000}(Actor)$
$Actor_{1990} \leftarrow \sigma_{BirthYear>1990}(Actor)$

$Teenager \leftarrow Actor_{1990} \star \sigma_{Role='Teenager'}(ActorInMovie)$
$Actor_{2000} \cup \pi_{ActorID,Name,BirthYear}(Teenager)$

c)

$ActorIn \leftarrow Actor \star ActorInMovie$
${}_{ActorID,Name}\,\Im\,_{COUNT(*)}(ActorIn)$

Task 4: Introduction to database normalization

a)

If the faculty EDI changes name to DI, such that both the FacultyCode and the FacultyName must be renamed, we need to update 10 cells in the table.

b)

Without having learned about the principles in normalization theory, I would propose to split the table into two, such that one table has information about the person and the other table has the information about the faculty. The person-table would have a foreign key which points to the correct faculty code, which would work as a primary key in the faculty table (which is an assumption that can be made since we have a functional dependency from FacultyCode to FacultyName and FacultyBuilding). In this way, the faculty information from this table would have two rows in the new table (one for EDI and one for IØA), and each person would point to their respective faculty code. Hence, one would only need to update two cells in this case, when wanting to change the name of EDI to DI.

Task 5: Functional dependencies, keys and closures

a)

1. Correct; A $\rightarrow$ A is trivial.
2. Perhaps; A $\rightarrow$ B could be correct, since we have no tuples that tell us otherwise.
3. Wrong; A $\rightarrow$ C cannot hold, since the first two rows show that this rule cannot hold.
4. Wrong; AB $\rightarrow$ C cannot hold, also shown by the values in the first two rows.
5. Perhaps; C $\rightarrow$ D could be correct, since we have no tuples that tell us otherwise.
6. Wrong; D $\rightarrow$ C cannot holds, since d2 $\rightarrow$ ci has two different values for ci in the table.
7. Correct; ABCD is a superkey for the table, since all tuples in the set are unique.
8. Perhaps; since ABC could have the correct functional dependencies to be a superkey of the table.
9. Wrong; A $\not\rightarrow$ ABCD as we can see from the first two rows.
10. Perhaps; since AC could be a superkey for the table.

b)

$A^+ = AC$, $D^+ = D$, $BC^+ = BCD$ and $AB^+ = ABCD = R$.