Daniel S. Fridjonsson, Lars Andersen, Søren S. Als & Mathias W. Pedersen.
sw608f14 - Room: 2.1.10

Self study 1 - miniproject part 1

Actors, directors, and writers are decided to be stored only as person, since they can preform multiple roles throughout their career. A person has a name, birthday, and additional information. The table for persons will be:

PERSON ID	PERSON NAME	BIRTHDAY
123	Johnny Known	03-09-1934

A movie can contain movie name, movie id, release data, and rating. The rating is calculated by taking the average of the user ratings, see last table, for a specific movie.

MOVIE ID	MOVIE NAME	RELEASE DATE	RATING
123	The Hobbit	05-12-2014	9.2

To connect persons to movies another table is used. Where the role in this table is an enum that contains actor, director, and writer.

MOVIE ID	PERSON ID	ROLE
637	463	Actor
563	563	Director
637	674	Writer

Awards can be given to both persons and movies, therefore it is necessary to have two tables, since the id's can be the same for both persons and movies. The tables will be as follows:

MOVIE ID	AWARD	YEAR
542	Best Picture	2013

PERSON ID	MOVIE ID	AWARD	YEAR
524	542	Best Actor	2013

To store a user of the system it is necessary to store a username and password. The password should be encrypted, but security is not the focus of this exercise.

USER ID	USERNAME	PASSWORD
546	NiceName	123456

Furthermore, a user can rate movies, and a table for this is created. This table contains user id, movie id, and rating

MOVIE ID	USER ID	RATING
634	342	8

Daniel S. Fridjonsson, Lars Andersen, Søren S. Als & Mathias W. Pedersen.
sw608f14 - Room: 2.1.10

Self study 2 - miniproject part 2

ER Diagram

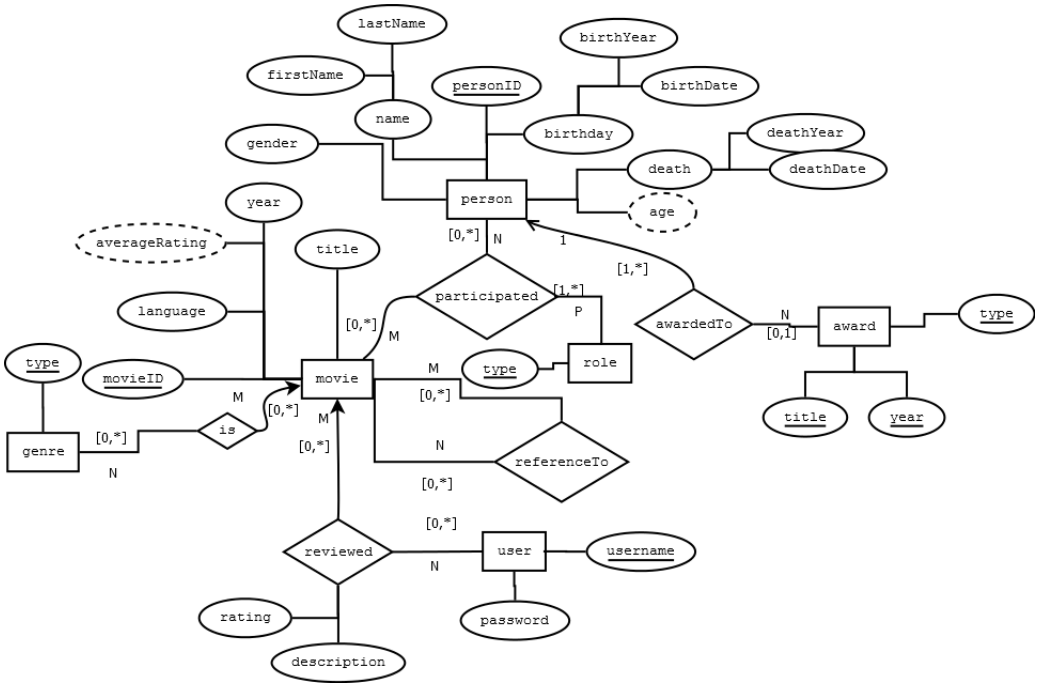


Figure 1: ER Diagram selfstudy 2

Relational Schema

genre

$\{[\underline{type : String}]\}$

movie

$\{[\underline{movieID : Int}, language : String, year : Int, title : String]\}$

user

$\{[\underline{username : String}, password : String]\}$

award

$\{[\underline{title : String, year : Int, type : String}, receiver \rightarrow person]\}$

person

$\{[\underline{personID : Int}, firstName : String, lastName : String, gender : String, birthYear : Int, birthDate : String, deathYear : Int, deathDate : String]\}$

participated

$\{[\underline{movieID \rightarrow movie, personID \rightarrow person, type \rightarrow role}]\}$

is $\{[\underline{type \rightarrow genre, movieID \rightarrow movie}]\}$

reviewed

$\{[\underline{movieID \rightarrow movie, username \rightarrow user}, rating : Int, description : String]\}$

referenceTo

$\{[\underline{from \rightarrow movie, to \rightarrow movie}]\}$

Reflections

The design is different from the initial design in that the initial design was at a mere table level. Furthermore, several additional attributes have been added to accommodate the expected information requirements. Another change is that there was no primary and foreign keys highlighted in the original answer. Average rating in the initial database was intended to be a calculated attribute, but was not listed as such, contrary to the new design. Additionally we have decided that an award only can be given to a person, as that made the design simpler. Finally we added the $referenceTo$ relation, as that was a requirement we missed in the first selfstudy.

Selfstudy 4

Revised ER Diagram

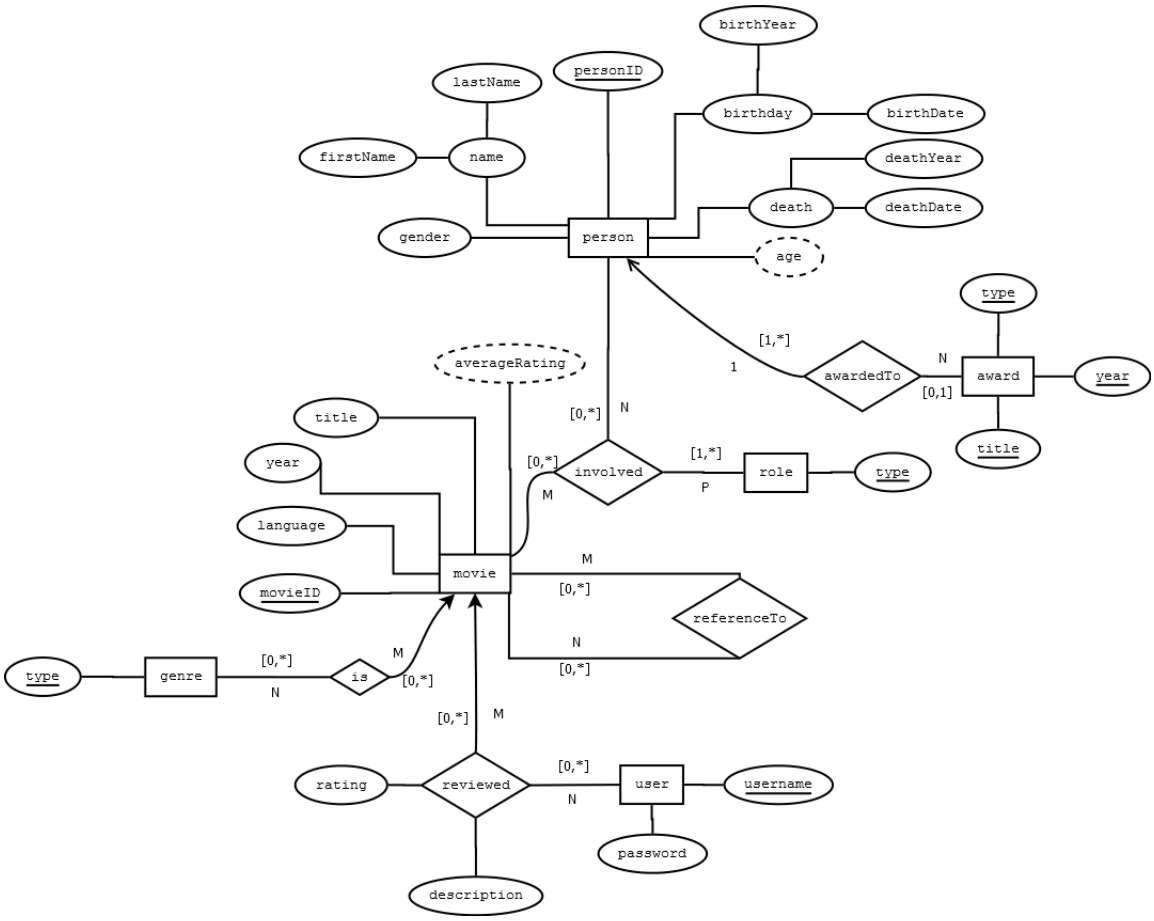


Figure 2: Revised ER Diagram selfstudy 4

Revised relational schema

ER Diagram

Relational Schema

genre

$\{[\underline{type} : String]\}$

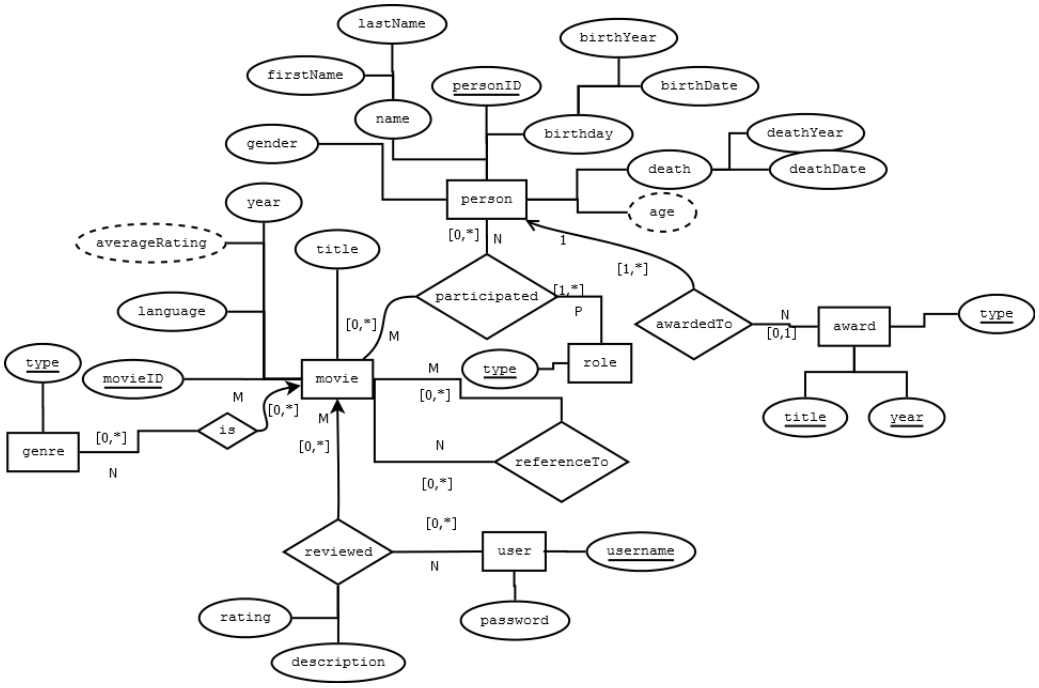
role $\{[\underline{type} : String]\}$

movie

$\{[\underline{movieID} : Int, language : String, year : Int, title : String]\}$

user

$\{[\underline{username} : String, password : String]\}$



award

$\{[\underline{title : String, year : Int, type : String}, receiver : Int \rightarrow person]\}$

person

$\{[\underline{personID : Int}, firstName : String, lastName : String, gender : String, birthYear : Int, birthDate : String, deathYear : Int, deathDate : String]\}$

involved

$\{[\underline{movieID : Int \rightarrow movie, personID : Int \rightarrow person, type : String \rightarrow role}]\}$

is $\{[\underline{type : String \rightarrow genre, movieID : Int \rightarrow movie}]\}$

reviewed

$\{[\underline{movieID : Int \rightarrow movie, username : String \rightarrow user}, rating : Int, description : String]\}$

referenceTo

$\{[\underline{from : Int \rightarrow movie, to : Int \rightarrow movie}]\}$

Functional Dependencies

$$
\begin{aligned}
genre.type &\rightarrow \emptyset \\
role.type &\rightarrow \emptyset \\
movieID &\rightarrow language, movie.year, movie.title \\
username &\rightarrow password \\
award.title, award.year, award.type &\rightarrow personID \\
personID &\rightarrow firstName, lastName, gender, birthYear, birthDate, deathYear, de \\
movieID, personID, role.type &\rightarrow \emptyset \\
genre.type, movieID &\rightarrow \emptyset \\
movieID, username &\rightarrow rating, description \\
movieID, movieID &\rightarrow \emptyset
\end{aligned}
$$

It is 3NF as rule 2 applies for all the functional dependencies, since the primary key in each relation is also a super key and due to it being 2NF, as it is fully functionally dependent. As it is rule 2 that applies for all the functional dependencies, it is also BCNF.

As the schema is already 3NF, there is no need to normalize the relations.

SQL Statements

```
CREATE TABLE genre
{
    type varchar(15) PRIMARY KEY
};

CREATE TABLE role
{
    type varchar(15) PRIMARY KEY
};


CREATE SEQUENCE serialmovie START 0;

CREATE TABLE movie
{
    movieID integer PRIMARY KEY DEFAULT nextval('serialmovie'),
    language varchar(20) DEFAULT 'English',
    year integer NOT NULL,
    title varchar(100) NOT NULL
};

CREATE TABLE user
{
    username varchar(20) PRIMARY KEY,
    password varchar(50) NOT NULL
};
```

```
CREATE TABLE award
{
      title varchar(30),
      year integer,
      type varchar(15),
      receiver integer NOT NULL,
      PRIMARY KEY(title,year,type),
      FOREIGN KEY(receiver) REFERENCES person(personID)
};

CREATE SEQUENCE serialperson START 0;

CREATE TABLE person
{
      personID integer PRIMARY KEY DEFAULT nextval('serialperson'),
      firstName varchar(20) NOT NULL,
      lastName varchar(50) NOT NULL,
      gender varchar(6),
      birthYear integer,
      birthDate varchar(7),
      deathYear integer,
      deathDate varchar(7)
};

CREATE TABLE involved
{
      movieID integer,
      personID integer,
      type varchar(15),
      PRIMARY KEY(movieID, personID, type),
      FOREIGN KEY(movieID) REFERENCES movie(movieID),
      FOREIGN KEY(personID) REFERENCES person(personID),
      FOREIGN KEY(type) REFERENCES role(type)
}

CREATE TABLE is
{
      type varchar(15),
      movieID integer,
      PRIMARY KEY(type,movieID),
      FOREIGN KEY(type) REFERENCES genre(type),
      FOREIGN KEY(movieID) REFERENCES movie(movieID)
};

CREATE TABLE reviewed
{
      movieID integer,
      username varchar(20),
      rating integer NOT NULL,
      description varchar(MAX),
```

```
    PRIMARY KEY(movieID, username),
    FOREIGN KEY(movieID) REFERENCES movie(movieID),
    FOREIGN KEY(username) REFERENCES user(username)
};

CREATE TABLE referenceTo
{
    from integer,
    to integer,
    PRIMARY KEY(from, to),
    FOREIGN KEY(from,to) REFERENCES movie(movieID)
};
```

Reflections

Differences from the original design is that we changed the layout of our ER Diagram and renamed the *participated* relation to *involved*. Considerations we have to take in the future is our encoding of dates, which could be changed to the *DATETIME* datatype. Also, we forgot the role relation in our first schema, and was thus added to the revised schema.

Selfstudy 5

Setup of the Database

We already provided the given sql statements for setting up our database, however, some slight modifications were performed. Curly brackets were changed to ordinary parentheses. Additionally to better fit the imdb data, the following was changed:

- Removed birthyear and deathyear, so it was included in birthdate and deathdate
- Made birthdate and deathdate 20 in size
- Merged firstname and lastname into name
- Made name 500 in size (there was some really long names)
- Title of movie changed to 500 length

Here is the new table creation commands:

```
CREATE TABLE genre
(
    type varchar(15) PRIMARY KEY
);

CREATE TABLE role
(
    type varchar(15) PRIMARY KEY
);
```

```
CREATE SEQUENCE serialmovie START 1;

CREATE TABLE movie
(
    movieID integer PRIMARY KEY DEFAULT nextval('serialmovie'),
    language varchar(50) DEFAULT 'English',
    year integer NOT NULL,
    title varchar(500) NOT NULL
);

CREATE TABLE users
(
    username varchar(20) PRIMARY KEY,
    password varchar(50) NOT NULL
);

CREATE SEQUENCE serialperson START 1;

CREATE TABLE person
(
    personID integer PRIMARY KEY DEFAULT nextval('serialperson'),
    name varchar(500) NOT NULL,
    gender varchar(6),
    birthDate varchar(20),
    deathDate varchar(20)
);

CREATE TABLE award
(
    title varchar(30),
    year integer,
    type varchar(15),
    receiver integer NOT NULL,
    PRIMARY KEY(title,year,type),
    FOREIGN KEY(receiver) REFERENCES person(personID)
);


CREATE TABLE involved
(
    movieID integer,
    personID integer,
    type varchar(15),
    PRIMARY KEY(movieID, personID, type),
    FOREIGN KEY(movieID) REFERENCES movie(movieID),
    FOREIGN KEY(personID) REFERENCES person(personID),
    FOREIGN KEY(type) REFERENCES role(type)
);
```

```
CREATE TABLE isA
(
      type varchar(15),
      movieID integer,
      PRIMARY KEY(type,movieID),
      FOREIGN KEY(type) REFERENCES genre(type),
      FOREIGN KEY(movieID) REFERENCES movie(movieID)
);

CREATE TABLE reviewed
(
      movieID integer,
      username varchar(20),
      rating integer NOT NULL,
      description text,
      PRIMARY KEY(movieID, username),
      FOREIGN KEY(movieID) REFERENCES movie(movieID),
      FOREIGN KEY(username) REFERENCES users(username)
);

CREATE TABLE referenceTo
(
      froma integer,
      toa integer,
      PRIMARY KEY(froma, toa),
      FOREIGN KEY(froma) REFERENCES movie(movieID),
      FOREIGN KEY(toa) REFERENCES movie(movieID)
);
```

Modifications to imdb dump

- int to integer
- removed ENGINE=InnoDB **DEFAULT** CHARSET=latin1;
- Changed indexes from **KEY** idx5 (movieId,genre)

  to **CREATE INDEX** idx5 **ON** genre (movieId, genre); and similar for the other indexes.
- keywords like *user* was changed to *üser¨* as it was a keyword.
- ' was changed to "

Importing Data from IMDB_db to our Database

In general we used a copy command to copy the desired data fra IMDB_db to a .csv file and then copying that information to our database. Also, sorry for a lot of commands :).

genre

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT genre.genre as type FROM genre)
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY genre(type) FROM 'PATH\genre.csv'
```

person

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT id, name, gender, birthdate, deathdate FROM person)
To 'PATH\person.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY person(personID,name,gender,birthDate,deathDate)
FROM 'PATH\person.csv'"
```

role

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT role FROM involved)
To 'PATH\role.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY role(type)
FROM 'PATH\role.csv'"
```

movie

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT id, language, year, title
FROM movie) To 'PATH\movie.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY movie(movieID, language, year, title)
FROM 'PATH\movie.csv'"
```

danishmovies → movie

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT 'Danish', year, title FROM danishmovies)
To 'PATH\danishmovie.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY movie(language, year, title)
FROM 'PATH\danishmovie.csv'"
```

referenceTo

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT fromid, toid FROM movieref)
To 'PATH\referemceto.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY referenceTo(froma, toa)
FROM 'PATH\referemceto.csv'"
```

involved - part1

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT movieid, personid, role FROM involved)
To 'PATH\involved.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY involved(movieID, personID, type)
FROM 'PATH\involved.csv'"
```

involved - part1 get directors from danishmovies

We found that postgres does not allow queries over multiple databases. As a solution we made a wrapper table in our database.

```
CREATE TABLE wrapperdanishdirector(
    title varchar(500) NOT NULL,
    director varchar(500) NOT NULL,
);
```

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT title, director FROM danishmovies)
To 'PATH\involvedwrapper.csv'"
```

To our database - in wrapper table

```
psql -d ourmovieDB -U postgres
-c "\COPY wrapperdanishdirector(title,director)
FROM 'PATH\involvedwrapper.csv'"
```

Adding to involved table

```
INSERT INTO involved SELECT DISTINCT movie.movieID, person.personID,'director'
FROM movie, wrapperdanishdirector, person
WHERE movie.title = wrapperdanishdirector.title
AND wrapperdanishdirector.director = person.name
AND (movie.movieID, person.personID,'director') NOT IN (SELECT *
FROM involved)
```

users

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT ratings.user, '1234' FROM ratings)
To 'PATH\users.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY users(username,password)
FROM 'PATH\users.csv'"
```

isa

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT genre.genre, movieid FROM genre)
To 'PATH\is.csv'"
```

To our database

```
psql -d ourmovieDB -U postgres
-c "\COPY isa(type, movieid)
FROM 'PATH\is.csv'"
```

reviewed

It seemed like some movies that was reviewed was not part of the set of movies from the imdb database. For that reason we made a temp reviewed table:

```
CREATE TABLE tempreviewed(
      movieID integer,
      username varchar(20),
      rating integer NOT NULL
);
```

To csv file

```
psql -d IMDB_db -U postgres
-c "\COPY (SELECT DISTINCT movieid, ratings.user, rating  FROM ratings)
To 'PATH\reviewed.csv'"
```

To our tempreviewed table in database

```
psql -d ourmovieDB -U postgres
-c "\COPY tempreviewed(movieid, username, rating)
FROM 'PATH\reviewed.csv'"
```

Final insertion into reviewed table

```
INSERT INTO reviewed SELECT * FROM tempreviewed
WHERE tempreviewed.movieid IN (SELECT DISTINCT movieid from movie)
```

SQL Statements and results

1.

Query:

```
SELECT count(*) FROM movie WHERE language = 'Danish';
```

Result:
670

2.

Query:

```
SELECT year, count(reviewed.rating) FROM movie,reviewed
WHERE movie.movieID = reviewed.movieID GROUP BY year;
```

Result:

2000	24
1962	1
2007	25
2002	19
1992	3
2008	40
2003	27
1999	48
2005	30
2004	13

3..

Query:

```
SELECT title FROM movie WHERE movieID IN
    (SELECT i1.movieID FROM involved i1, involved i2
    WHERE i1.movieID = i2.movieID AND i1.personID IN
    (SELECT personID FROM person WHERE name = 'John Travolta')
     AND i2.personID IN (SELECT personID FROM person WHERE name = 'Uma Thurman')
     AND i1.type = 'actor' AND i2.type = 'actor');
```

Result:

Good Morning America
The Rosie O'Donnell Show
The Oprah Winfrey Show
The View
Wetten, dass..?
Late Show with David Letterman
The Tonight Show with Jay Leno
You're Still Not Fooling Anybody
HBO First Look
Boffo! Tinseltown's Bombs and Blockbusters

4.

Query:

```
SELECT count(*) FROM person WHERE personID IN (SELECT DISTINCT personID
FROM involved WHERE type = 'actor' OR type = 'director') AND name LIKE 'Q%';
```

Result: 153

5.

Query:

```
SELECT count(*) FROM (SELECT username FROM reviewed
GROUP BY username HAVING count(movieID) >= 3) as alias;
```

Result:
34

6.

Query:

```
SELECT name, substring(birthdate from 1 for 4) FROM person
WHERE personID IN (SELECT personID FROM involved WHERE movieID
IN (SELECT movieID FROM movie WHERE title = 'Pulp Fiction') AND type = 'actor')
ORDER BY birthdate ASC;
```

Result:

Emil Sitka	1914
Harvey Keitel	1939
Rene Beard	1941
Christopher Walken	1943
Joseph Pilato	1949
Brenda Hillhouse	1953
John Travolta	1954
Bruce Willis	1955
Amanda Plummer	1957
Lawrence Bender	1957

7.

Query:

```
SELECT title, year FROM movie WHERE movieID IN
(SELECT movieID FROM involved WHERE personID IN
    (SELECT personID FROM person WHERE name = 'John Travolta')
   AND type ='actor') AND year >= 1980 AND year < 1990;
```

Result:

Wetten, dass..?	1981
Larry King Live	1985
That's Dancing!	1985
Perfect	1985
Biography	1987
Two of a Kind	1983
Staying Alive	1983
Live with Regis and Kathie Lee	1988
Entertainment Tonight	1981
Urban Cowboy	1980

8.

Query:

```
SELECT title FROM movie WHERE movieID IN
    (SELECT movieID FROM reviewed GROUP BY movieID ORDER BY AVG(rating)
       DESC LIMIT 2)
   AND year >= 1990 AND year < 2000;
```

Result:
The Usual Suspects

9.

Query:

```
SELECT title FROM movie WHERE movieID IN
    (SELECT movieID FROM reviewed WHERE movieID IN
        (SELECT movieID FROM reviewed
       GROUP BY movieID HAVING count(rating) >= 2)
   GROUP BY movieID ORDER BY AVG(rating) DESC LIMIT 2)
AND year >= 1990 AND year < 2000;
```

Result:
The Usual Suspects

10.

Query:

```
SELECT language, AVG(rating) FROM reviewed, movie
WHERE reviewed.movieID = movie.movieID AND year = 1994 GROUP BY language
```

Result:

NULL	7.0
French	9.0
English	8.304

11.

Query:

```
SELECT name FROM person WHERE personID IN
    (SELECT AVG(personID) FROM involved WHERE personID in
        (SELECT personID FROM involved WHERE movieID in
            (SELECT movieID FROM movie WHERE title = 'Pulp Fiction')
        AND type = 'actor')
GROUP BY movieID HAVING count(personID) = 1);
```

Result:

Caleb Allen
Rosana Arquette
Steve Buscemi
Maria de Medeiros
Karen Maruyama
Burr Steers
Eric Stoltz
Julia Sweeney
Quentin Tarantino
Christopher Walken

12.

Query:

```
SELECT title FROM movie WHERE movieID IN
    (SELECT movieID FROM reviewed WHERE movieID IN
        (SELECT movie.movieID FROM movie,involved, person
        WHERE name = 'John Travolta'
        AND person.personID = involved.personID AND type = 'actor'
        AND movie.movieID = involved.movieID)
GROUP BY movieID ORDER BY AVG(rating) DESC LIMIT 1);
```

Result:

Pulp Fiction

13.

Query:

```
SELECT p1.name FROM person p1, person p2 WHERE p1.personID IN
    (SELECT personID FROM involved WHERE type = 'actor')
AND p2.name = 'Charles Chaplin'
AND (p1.birthdate > p2.deathdate
OR p2.birthdate > p1.deathdate);
```

Result:

Elisha Cuthbert
Abe Forsyhte
Manique Ganderton
Raushan Hammond
Jason Momoa
Josh Negrin
Colin Platt
Usher Raymond
Sebastian Urzendowsky
Breven Angaelica Warren

14.

Query:

```
SELECT type,AVG(rating) FROM isa, reviewed
WHERE isa.movieID = reviewed.movieID GROUP BY type;
```

Result:

Comedy	7.2381
Drama	7.8333
Fantasy	7.2430
Biography	8.1333
Thriller	7.7824
Crime	8.3950
Muscial	7
War	8.692
History	8.1250
Adventure	7.4211

15.

Query:

```
SELECT type,AVG(rating) FROM isa, reviewed
WHERE isa.movieID = reviewed.movieID
GROUP BY type HAVING count(rating) >= 2;
```

Result:

Comedy	7.2381
Drama	7.8333
Fantasy	7.2430
Biography	8.1333
Thriller	7.7824
Crime	8.3950
War	8.2692
History	8.1250
Adventure	7.4211
Sci-Fi	7.4853

16.

Query:

```
SELECT title FROM movie
WHERE movieID IN
    (SELECT r1.froma FROM referenceTo r1, referenceTo r2
       WHERE r1.toa = r2.froma
       GROUP BY r1.froma ORDER BY count(r2.toa) DESC LIMIT 1);
```

Result:
Saturday Night Live

17.

Query:

```
SELECT count(foo.personID) FROM
    (SELECT DISTINCT i1.personID FROM involved i1, involved i2
   WHERE i1.personID = i2.personID AND i1.type = 'actor'
       AND i2.type = 'director') AS foo;
```

Result:
5930

18.

Query:

```
SELECT i1.type,i2.type FROM isa i1, isa i2
WHERE i1.type != i2.type AND i1.movieID = i2.movieID
GROUP BY i1.type, i2.type ORDER BY count(i1.movieID) DESC LIMIT 1;
```

Result:

Romance	Drama