Marc Tom Thorgersen, mthorg13　　Mathias Sass Michno, mmichn13
Søren Hvidberg Frandsen, sfrand12　　Troels Beck Krøgh, tkragh13

Self study 1 - Table Description for movie database

The database contains information of movies, the people who worked on the movies, users, and reviews made by these users. There is also information about awards, and which people and movies have won or been nominated for these awards.

Tables:

- **Movies**
  - This table has an ID for the movie, along with the title of the movie, its run-time and release date. The ID is used in other tables to relate the movies with other rows of other tables.
- **People**
  - A table that contains all persons that may be related to a role.
- **Crew**
  - Relates persons to movies and their role in the movies. Note that if a person has more than one role, additional rows are needed.
- **Role**
  - Describes the different roles, persons can have in a movie, e.g director, producer, actor, camera operator, assistants etc.
- **Award**
  - This table contains different awards with specification and year relevant to the given award.
- **MovieAward**
  - Relates movies awards and optionally a list ID that represents a group of persons. Moveover a bool indicates if the award is won or if it only is a nomination. The list ID is used for award such as “Best Cast”.
- **PersonList**
  - Lists persons for use in relation to awards. A list can contain several persons, and persons may be on several lists.
- **User**
  - Contains usernames and unique ID, and additional user information which we have not written here. It can be encrypted passwords, e-mails, birthdate etc.
- **Review**
  - This table relates users to movies and provides additional information about the specific review, such as rating, review title and review text

F-key = Foreign key
P-key = Primary key
Join table = Also know as junction tables, and defines many-to-many relations between entries in other tables.
OPT = Optional, can be NULL if not used.

Movie:

Movie_ID (P-key)	Movie_Title	Movie_Runtime	Movie_Release_Date
1	Lord of The Rings: The Return of the King	201	2003
2	Deadpool	108	2016

People:

Person_ID (P-key)	Person_Name	Person_Birthdate	Person_PlaceOfBirth
1	Viggo Mortensen	1958-10-20	Denmark
2	Ryan Reynolds	1976-10-23	Canada

Crew (Join table):

Person_ID (F-key)	Movie_ID (F-Key)	Role_ID (F-Key)
1	1	1
1	2	1

Role:

Role_ID (P-key)	Role_Name
1	Actor

Award:

Award_ID (P-key)	4Award_Name	Award_Year
1	Oscar: Best Cast Ensemble	2003

MovieAward (Join table):

Movie_ID (F-key)	Award_ID (F-key)	List_ID (F-key, OPT)	Won? (bool)
1	1	1	1

PersonList (Join table):

List_ID (F-key)	Person_ID (F-key)
1	1

User:

User_ID (P-key)	User_Name	Additional User information
1	JensJensen	…

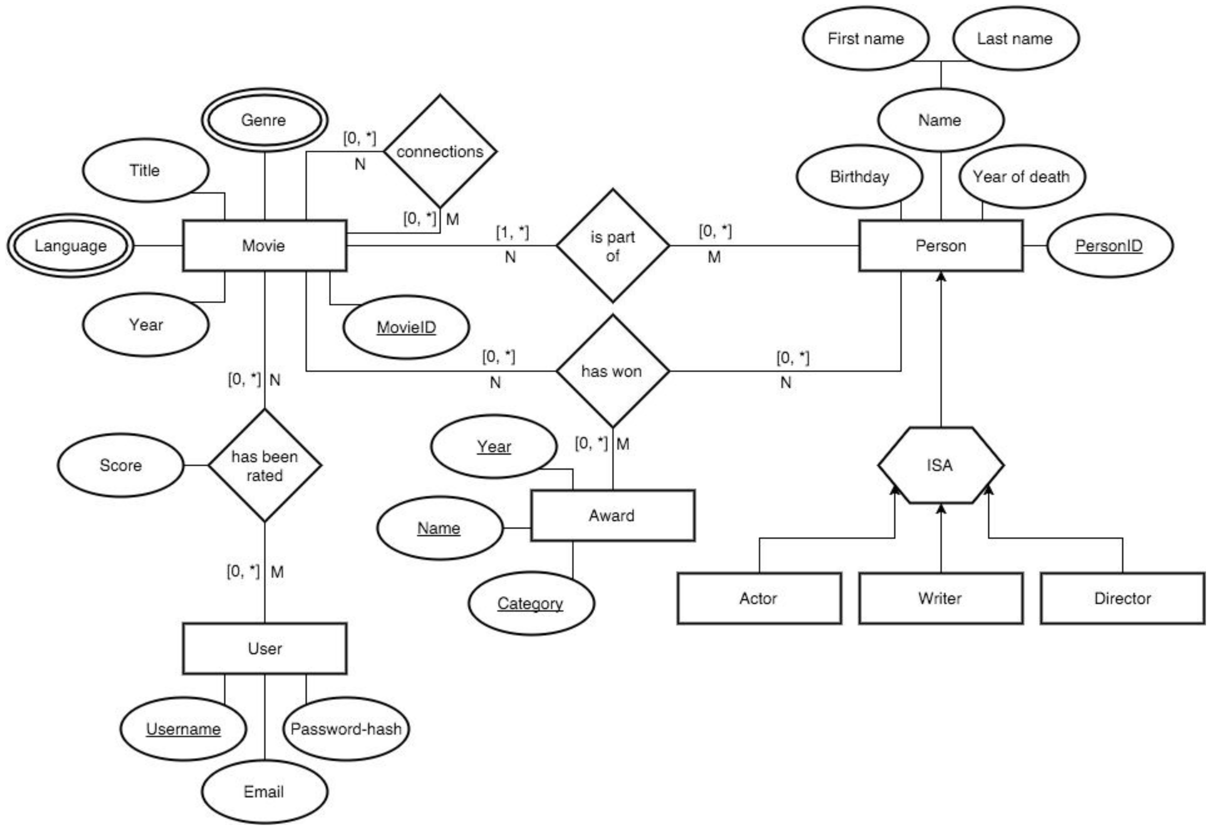
Review:

Review_ID (P-key)	User_ID (F-key)	Movie_ID (F-key)	Rating_Number	Review_Title	Review_Text
1	1	1	10	Best	Movie Ever
2	1	2	10	I like	Turtles

Self study 2 - Entity Relationship Diagram

A) & B) - We did them in one go.
We have chosen to do an ISA pattern to use the inheritance since persons can have different kind of roles, i.e. writer or actor.

We have also used a compound attribute for Name on persons.
The name on awards is e.g. “Academy Award”, or “Golden Globe”.
On the entity “Movie”, “Genre” and “Language” are multi-value attributes.



C)
User: (<u>Username</u>, Email, Password-hash)
Has been rated: (<u>MovieID</u>, <u>Username</u>)
    With foreign keys: {MovieID, Username} → {Movie.MovieID, User.Username}
Movie: (<u>MovieID</u>, Title, Language, Year, Genre}
Connections: (<u>aMovieID</u>, <u>bMovieID</u>)
    With foreign keys: {aMovieID, bMovieID} → {Movie.MovieID, Movie.MovieID}
Person: (<u>PersonID</u>, First name, Last name, Birthday, Year of death)
Is part of: (<u>MovieID</u>, <u>PersonID</u>)
    With foreign keys: {MovieID, PersonID} → {Movie.MovieID, Person.PersonID}
Award: (<u>Year</u>, <u>Name</u>, <u>Category</u>}
Has won: (<u>Category</u>, <u>Year</u>, <u>Name</u>, MovieID, PersonID}

With foreign keys {Category, Year, Name} → {Award.Category, Award.year, Award.Name}
(Partitioning is used here)
Actor: (<u>PersonID</u>)
With foreign key {PersonID} → {Person.PersonID}
Writer: (<u>PersonID</u>)
With foreign key {PersonID} → {Person.PersonID}
Director: (<u>PersonID</u>)
With foreign key {PersonID} → {Person.PersonID}

E)
We have removed the “roles” table, and instead used the ISA pattern.
We have also removed the AwardID, since we can use other attributes as keys.
We have removed the personList, and created the relationship type is part of.
We have also removed UserID, and use username as the key instead.
Reviews do not need a reviewID so that is gone as well, we have also removed some information on review title, but that is mostly because it seems the score is only relevant, so now that is just an attribute on the relationship type.

Marc Tom Thorgersen, mthorg13
Søren Hvidberg Frandsen, sfrand12

Mathias Sass Michno, mmichn13
Troels Beck Krøgh, tkragh13

Self study 3 - Refinement, normalization, and SQL-DDL

(a) Identify functional dependencies for each of the relations you created.

- User:

{ Username } → { Username, Email, Password-Hash }

- Has Been Rated:

{ MovieID, Username } → { MovieID, Username, Score }

- Movie:

{ MovieID } → { Title, Year, Genre, Language, MovieID }

- Connections:

{ aMovieID, bMovieID } → { aMovieID, bMovieID }

- Person:

{ PersonID } → { PersonID, First name, Last name, Birthday, Year of death }

- Is Part Of:

{ MovieID, PersonID } → { MovieID, PersonID}

- Award

{ Year, Name, Category } → { Year, Name, Category }

- Has Won:

{ Year, Name, Category } → { MovieID, PersonID, Year, Name, Category }

- Actor:

{ PersonID } → { PersonID }

- Writer:

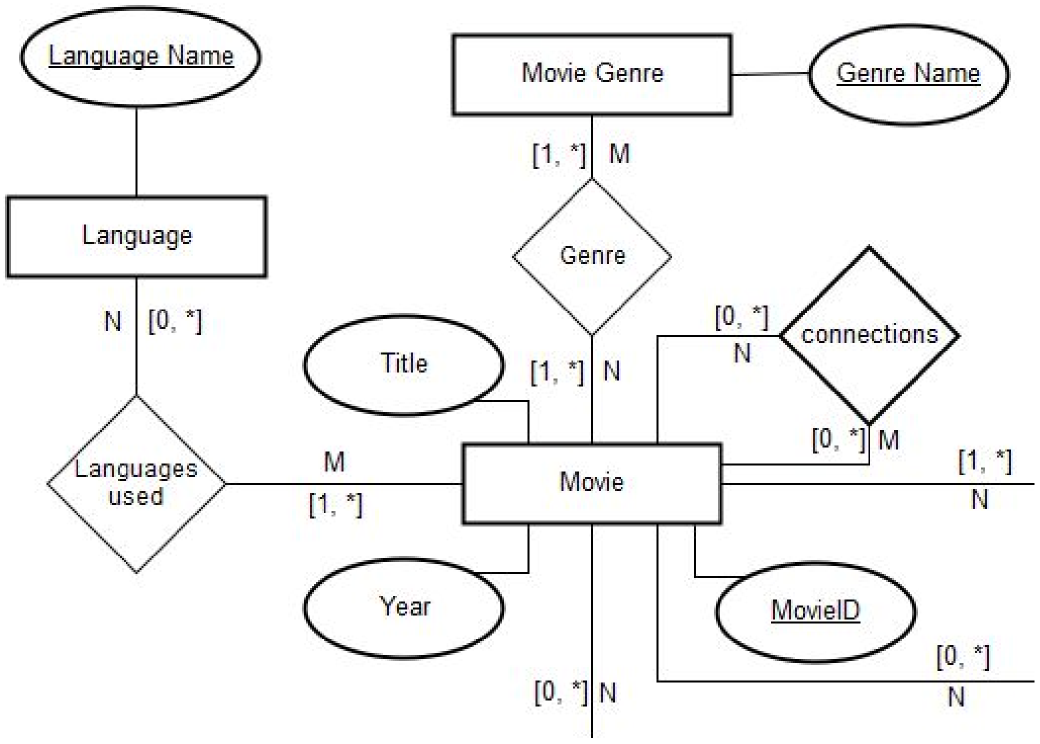
{ PersonID } → { PersonID }

- Director:

{ PersonID } → { PersonID }

(b) If necessary, transform your schema into 3NF and formally show that the result is actually in 3NF/BCNF.

Since we used multi-value attributes we need to change it in order to be in BCNF, the following ER diagram is constructed without them. Note that the picture is cropped unchanged information is omitted.



The following additions are made to the functional dependencies:

- Language

{ LanguageName } → { LanguageName }

- Languages used

{ MovieID } → { Language Name, MovieID }

- Movie Genre

{ GenreName } → { GenreName }

- Genre

{ MovieID } → { MovieID, Genre Name}

Moreover "Genre" and "Language" are removed from "Movie" in the functional dependencies listed in *(a)*

User, Movie and Person are all based on their primary key, therefore they follow the rule, "α is a super key of R".

Award, Genre, Language Name, Actor, Writer and Director are all trivial.

The left side of the functional dependency of Has Been Rated is a super key of the relation. The same holds for the rest of the relationship types.

Thusly our schema is in BCNF and therefore also in 3NF

(c) Determine for each obtained relational schema the highest normal form it still supports, i.e., check if your relations are also in BCNF.

It is, see argument above.

(d) List the SQL statements that create all your tables properly and ensure the functional dependencies that you have identified.

```
CREATE TABLE Movie(
      MovieID INTEGER PRIMARY KEY NOT NULL,
      Title VARCHAR(250),
      /* For example this (existing) movie title is very long:
      'Night Of The Day Of The Dawn Of The Son Of The Bride Of The
      Return Of The Revenge Of The Terror Of The Attack Of The
      Evil, Mutant, Hellbound, Flesh-Eating, Crawling, Alien,
      Zombified, Subhumanoid Living Dead - Part 5'
      */
      vYear DATE);

CREATE TABLE Award (
      vYear DATE NOT NULL,
      vName VARCHAR(100) NOT NULL,
      Category VARCHAR(100) NOT NULL,
      PRIMARY KEY(vYear , vName, Category));

CREATE TABLE vUser (
      Username VARCHAR(32) PRIMARY KEY NOT NULL,
      Email VARCHAR(256),
      Password_Hash CHAR(64));

CREATE TYPE vNAME AS (
      First_Name VARCHAR(32),
      Last_Name VARCHAR(32));

CREATE TABLE Person (
      vName NAME,
      Birthday DATE,
      Year_Of_Death DATE,
      PersonID INTEGER PRIMARY KEY NOT NULL);


CREATE TABLE Actor (
      PersonID INTEGER PRIMARY KEY NOT NULL);

CREATE TABLE Director (
      PersonID INTEGER PRIMARY KEY NOT NULL);


CREATE TABLE Writer (
      PersonID INTEGER PRIMARY KEY NOT NULL);

CREATE TABLE Movie_Genre (
      Genre_Name VARCHAR(32) PRIMARY KEY NOT NULL);

CREATE TABLE Language (
      Language_Name VARCHAR(32) PRIMARY KEY NOT NULL);

CREATE TABLE Is_Part_Of (
      MovieID INTEGER REFERENCES Movie NOT NULL,
      PersonID INTEGER REFERENCES Person NOT NULL);

CREATE TABLE Has_Been_Rated (
      MovieID INTEGER REFERENCES Movie NOT NULL,
      Username VARCHAR(32) REFERENCES vUser NOT NULL,
      Score NUMERIC(3,1) CHECK (Score <= 10.0));

CREATE TABLE Has_Won (
      vYear DATE NOT NULL,
      vName VARCHAR(250) NOT NULL,
      Category VARCHAR(100) NOT NULL,
      MovieID INTEGER REFERENCES Movie NOT NULL,
      PersonID INTEGER REFERENCES Person NOT NULL,
      FOREIGN KEY (vYear, vName, Category) REFERENCES Award (vYear,
vName, Category));

CREATE TABLE Languages_Used (
      MovieID INTEGER REFERENCES Movie NOT NULL,
      Language_Name VARCHAR(32) REFERENCES Language NOT NULL);

CREATE TABLE Genre (
      MovieID INTEGER REFERENCES Movie NOT NULL,
      Genre_Name VARCHAR(32) REFERENCES Movie_Genre NOT NULL);
```

```
CREATE TABLE Connections (
	aMovieID INTEGER REFERENCES Movie NOT NULL,
	bMovieID INTEGER REFERENCES Movie NOT NULL);
```

Reflections

We have changed it from multi valued attributes into relationship types using many to many relations, in order to accommodate BCNF (and 3NF).