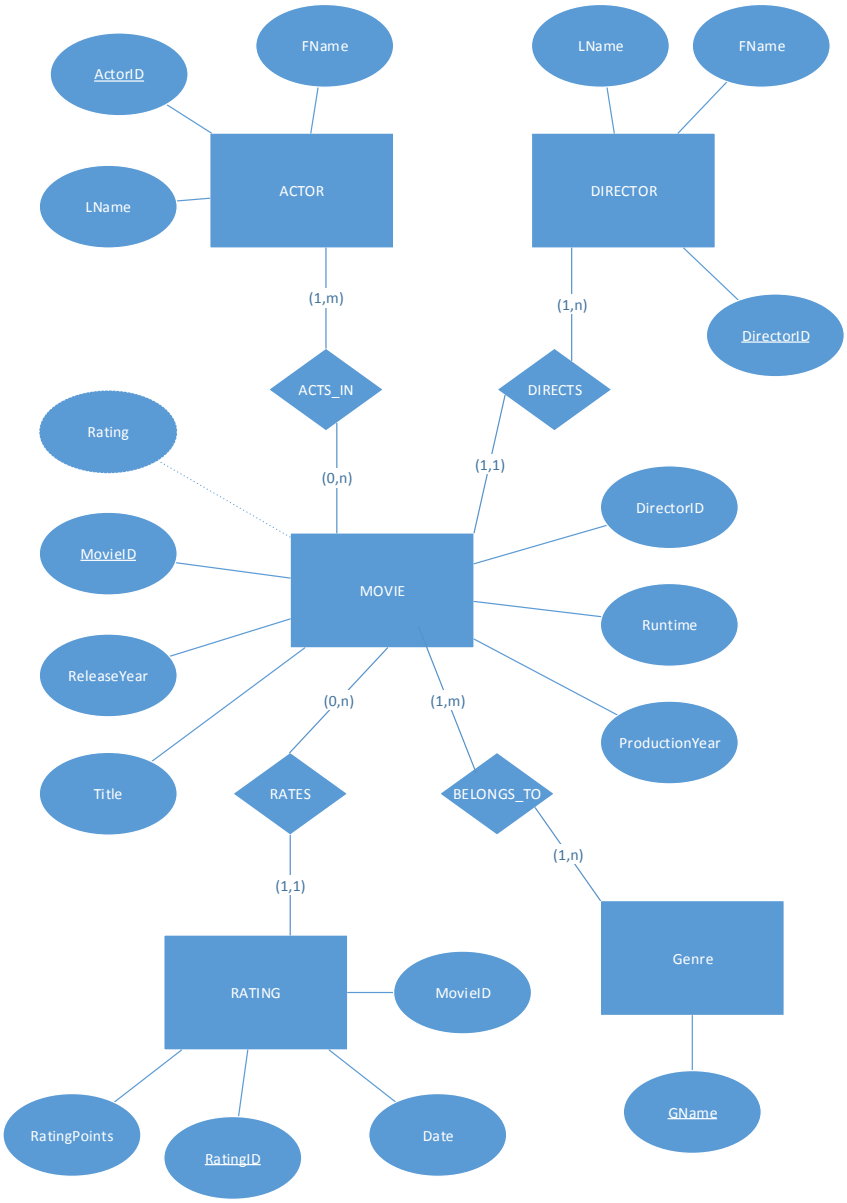
Dokumentation zum Datenbankprojekt

Katharina Chowanski / Marvin Kleinert / Marius Schidlack

21. Juli 2015

Iteration 1: Modellierung

1. Entity-Relationship-Modell



2. Relationales Modell

MOVIE (Title: string; MovieID: int; ReleaseYear: int; ProductionYear: int; Runtime: int; DirectorID: int; Rating: float)

ACTOR (ActorID: int; FName: string; Lname: string)

DIRECTOR (DirectorID: int; FName: string; LName: string)

RATING (RatingID: int; Date: date; RatingPoints: int; MovieID: int)

GENRE (GName: string)

ACTS_IN (MovieID: int; ActorID: int)

BELONGS_TO (GName: string; MovieID: int)

3. CREATE-Statements

```
CREATE TABLE MOVIE
(Title              varchar(255) NOT NULL,
MovieID             int8 NOT NULL,
ReleaseYear         interval YEAR,
ProductionYear          interval YEAR,
Runtime             int4,
DirectorID          int8 NOT NULL,
Rating              float4,
PRIMARY KEY (MovieID),
FOREIGN KEY (DirectorID) REFERENCES DIRECTOR (DirectorID)
);

CREATE TABLE ACTOR
(ActorID            int8 NOT NULL,
FName               varchar(255),
LName               varchar(255),
PRIMARY KEY (ActorID)
);

CREATE TABLE DIRECTOR
(DirectorID         int8 NOT NULL,
FName               varchar(255),
LName               varchar(255),
PRIMARY KEY (DirectorID)
);

CREATE TABLE RATING
(RatingID           int8 NOT NULL,
Date                date,
RatingPoints            int4,
MovieID             int8 NOT NULL,
PRIMARY KEY (RatingID),
FOREIGN KEY (MovieID) REFERENCES MOVIE (MovieID)
);
```

```
CREATE TABLE GENRE
(GName                varchar(255) NOT NULL,
PRIMARY KEY (GName)
);

CREATE TABLE ACTS_IN
(MovieID              int8 NOT NULL,
ActorID               int8 NOT NULL,
PRIMARY KEY (MovieID, ActorID),
FOREIGN KEY (MovieID) REFERENCES MOVIE(MovieID),
FOREIGN KEY (ActorID) REFERENCES ACTOR (ActorID)
);

CREATE TABLE BELONGS_TO
(GName                varchar(255) NOT NULL,
MovieID               int8 NOT NULL,
PRIMARY KEY (GName, MovieID),
FOREIGN KEY (GName) REFERENCES GENRE (GName),
FOREIGN KEY (MovieID) REFERENCES MOVIE (MovieID)
);
```

Iteration 2: Datentransformation und API

1. Veränderungen im Relationalen Modell

MOVIE (Title: text; MovieID: varchar(9); ReleaseYear: int2; ProductionYear: int2; Runtime: int2; DirectorID: int; Rating: numeric(3,1), RatingCount: int)

ACTOR (ActorID: varchar(9); FName: varchar(127); LName: varchar(127))

DIRECTOR (DirectorID: varchar(9); FName: varchar(127); LName: varchar(127))

RATING (RatingID: varchar(9); Date: date; RatingPoints: int2; MovieID: varchar(9))

GENRE (GName: varchar(127))

ACTS_IN (MovieID: varchar(9);
ACTOR.FName: varchar(127); ACTOR.LName: varchar(127))

DIRECTS (MovieID: varchar(9); DIRECTOR.FName: varchar(127); DIRECTOR.LName: varchar(127))

BELONGS_TO (GName: varchar(127); MovieID: varchar(9))

Die Änderungen zum ursprünglichen Relationalen Modell resultieren aus der Sichtung des Datensatzes und bestehen im Wechsel der Primärschlüssel bei den Relationen ACTOR und DIRECTOR von einer in den Daten nicht enthaltenen ID zu Vor- und Nachname, sowie einer neuen Relation DIRECTS. Diese ist nötig geworden, da in den Daten auch Filme vorkommen, die mehrere Directors haben können und somit zwischen MOVIE und DIRECTOR eine n:m-Beziehung besteht.

2. Datentransformation mit Java und JDBC

3. Datentransformation mit SQL

```
CREATE TABLE alldata
    (imdbID         VARCHAR(9),
    name            TEXT,
    year            INT2,
    rating          NUMERIC(3,1),
    votes           INT,
    runtime         INT2,
    directors       TEXT,
    actors          TEXT,
    genres          TEXT);

COPY alldata
FROM 'imdb_top100t_2015-06-18.csv'
FORMAT CSV
DELIMITER '\t';

INSERT INTO movies
(imdbID, title, rel_year, rating, rating_counts, duration)
SELECT imdbID, name, year, rating, votes, runtime
FROM alldata A
WHERE NOT EXSISTS   (SELECT imdbID
                     FROM movies
                     WHERE A.imdbID = imdbID);
```

Unsere Idee bei der Transformation der Daten ausschließlich mit SQL bestand darin, zunächst eine Tabelle für alle Attribute anzulegen, um dann mit der von SQL zur Verfügung gestellten COPY-Funktion die CSV-Datei mit ihrem Delimiter TAB in die Tabelle zu kopieren. Die gewünschten Daten für die einzelnen Tabellen unserer Datenbank erhält man schließlich durch Querys auf die neu erzeugte Tabelle.

4. API

```
public interface DatabaseAPI {
    public String bestMovies (int count);
    public String moviesRating (int from, int to);
    public String moviesRating (int from, int to, int year);
    public String actorsInMovie (String movieTitle);
    public String directorsOfMovie (String movieTitle);
    public String ratingOfMovie (String movieTitle);
    public String genresOfMovie (String movieTitle);
    public String movieInfos (String movieTitle);
    public String moviesOfActor (String fname, String lName);
    public String moviesOfActor
                    (String fname, String lName, int year);
    public String debutOfActor (String fName, String lName);
    public String  moviesOfDirector
                    (String fName, String lName);
    public String  moviesOfDirector
                    (String fName, String lName, int year);
    public String actorsWorkedForDirector
```

```
                    (String fName, String lName);
    public String directorWithMostMovies (int year);
```

Unsere API orientiert sich an den beispielanfragen in der Projektbeschreibung, umfasst aber zudem noch weitere nützliche Anfragen, wie z.B. welche Schauspieler in einem bestimmten Film mitspielen oder wer bei einem bestimmten Fil Regie geführt hat. Exemplarisch folgt nun die Implementierung von zwei Methoden des obigen Interfaces:

```
public String actorsInMovie (String movieTitle){
    String sql =    ("SELECT firstname, lastname "
                    + "FROM movies, acts_in "
                    + "WHERE title = ? "
                    + "AND id = movieID ");
    TupleQ[] pstValues = {new TupleQ(1,movieTitle)};
    ResultSet rs = manageQuery (sql, pstValues);
    String retString = resultTable(rs, new TupleR[]
            {new TupleR("firstname",String.class),
            new TupleR( "lastname", String.class) });
    return retString;
}

public String debutOfActor (String fName, String lName){
    String sql =    ("SELECT title, rel_year "
                    + "FROM movies, acts_in "
                    + "WHERE firstname = ? "
                    + "AND lastname = ? "
                    + "AND id = movieID "
                    + "AND rel_year > 0 "
                    + "ORDER BY rel_year ASC "
                    + "LIMIT 1 ");
    TupleQ[] pstValues = {new TupleQ(1,fName),
                        new TupleQ(2, lName)};
    ResultSet rs = manageQuery (sql, pstValues);
    String retString = resultTable(rs, new TupleR[]
            {new TupleR("title",String.class),
                new TupleR("rel_year",int.class)});
    return retString;
}
```

Hauptbestandteil der Methoden bildet die SQL-Abfrage, die zusammen mit den nötigen Parametern in einer separaten Funktion weiterverarbeitet wird, um sich wederholenden Code zu vermeiden. Dabei werden PreparedStatements verwendet, um mögliche SQL-Injections durch falsche Eingaben zu verhindern. Auch die Ausgabe der Daten wird in einer separaten Funktion gemanagt, sodass mögliche Änderungen in der Darstellung für die GUI leicht an nur einer Stelle vollzogen werden können.

Iteration 3: GUI und Datamining

1. Datamining

Auswertung des Projekts