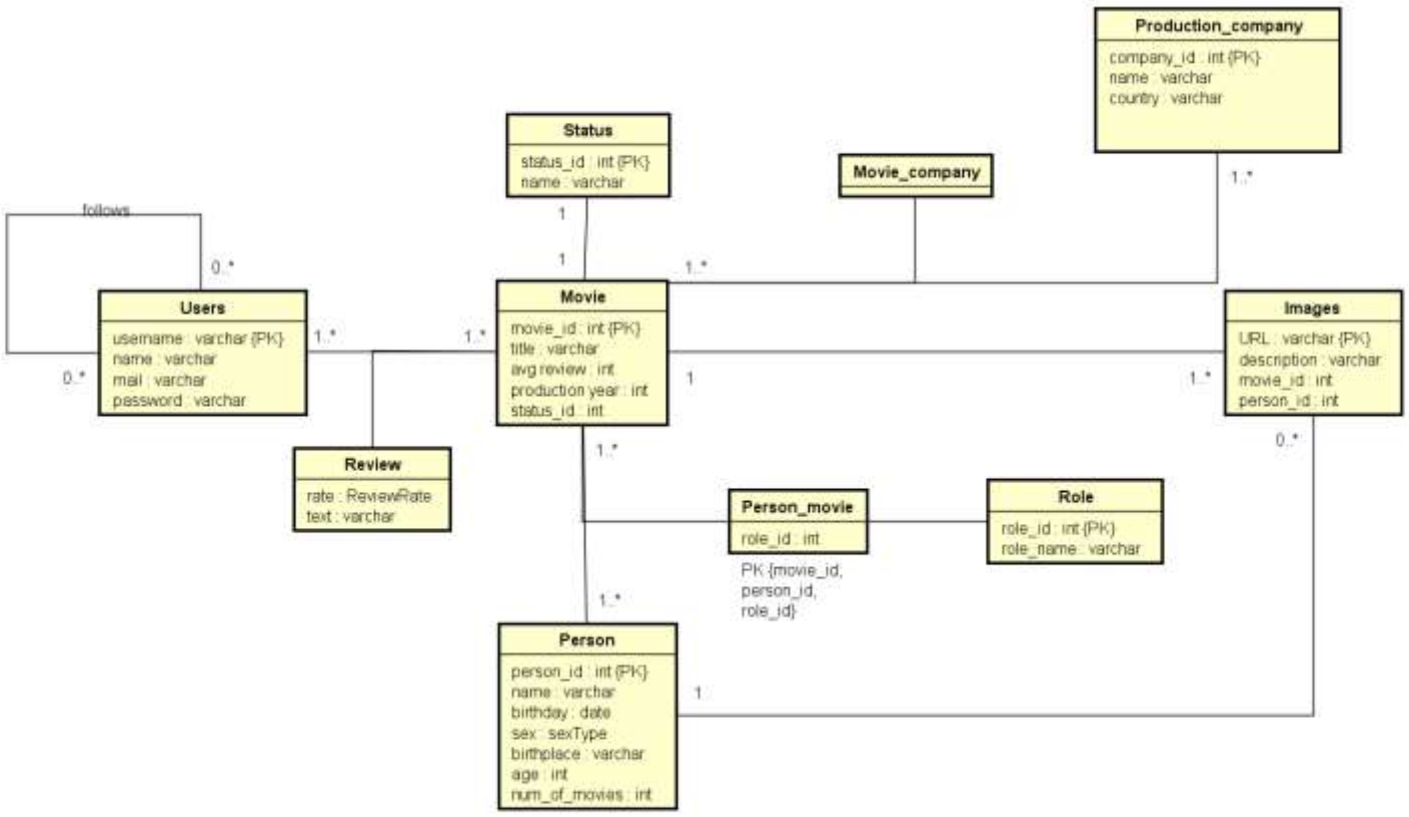
DBS1 S17 COURSE ASSIGNMENT – PART 1

JAKUB LEMKA – 249817

1. CONCEPTUAL MODEL



2. LOGICAL MODEL

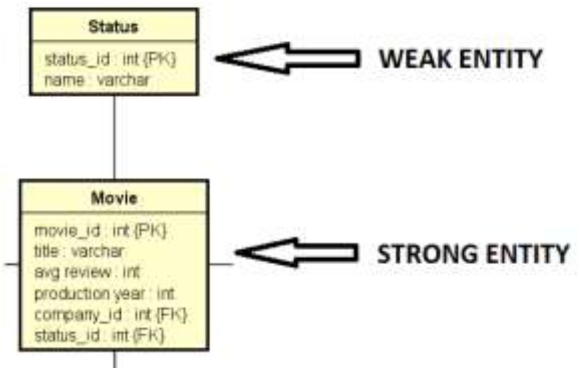
Movie (movie_id, title, avg_review, production_year, status_id) **Primary Key** movie_id **Foreign Key** status_id **references** Status (status_id) **Derived** avg_review (average rate from Review)	**Status** (status_id, name) **Primary Key** status_id **Role** (role_id, role_name) **Primary Key** role_id
Person (person_id, name, birthday, sex, birthplace, age, num_of_movies) **Primary Key** person_id **Derived** age (num of years from birthday) **Derived**	**Users** (username, name, mail, password) **Primary Key** username
Production_company (company_id, name, country) **Primary Key** company_id	**Images** (url, description, movie_id, person_id) **Primary Key** url **Foreign Key** movie_id **references** movie (movie_id) **Foreign Key** person_id **references** person (person_id)
Review (username, movie_id, rate, text) **Primary Key** username, movie_id **Foreign Key** username **references** User (username) **Foreign Key** movie_id **references** movie (movie_id)	**Person_movie** (movie_id, person_id, role_id) **Primary Key** movie_id, person_id, role_id **Foreign Key** movie_id **references** Movie (movie_id) **Foreign Key** person_id **references** Person (person_id) **Foreign Key** role_id **references** Role (role_id)
Movie_company (movie_id, company_id) **Primary Key** movie_id, company_id **Foreign Key** movie_id **references** Movie (movie_id) **Foreign Key** company_id **references** Production_company (company_id)	**Users_followers** (username_follower, username_followed) **Primary Key** username_follower, username_followed **Foreign Key** username_follower **references** Users (username)**Foreign Key** company_id **references** Users (username)

3. CREATING DOMAINS, TABLES

```sql
/* DOMAINS: */

CREATE DOMAIN SexType AS CHAR
DEFAULT 'M'
CHECK (VALUE IN ('M', 'F'));

CREATE DOMAIN ReviewRate AS smallint
DEFAULT null
CHECK (VALUE IN (0, 1, 2, 3, 4, 5));

CREATE DOMAIN prod_year AS integer
DEFAULT null
CHECK (VALUE BETWEEN 1896 AND 9999);


/* TABLES: */

CREATE TABLE images
(
   URL            VARCHAR (100),
   Description    VARCHAR (50),
   movie_id       integer,
   person_id      integer,
   PRIMARY KEY (URL)
);

CREATE TABLE movie
(
   movie_ID           Integer,
   title              varchar (30) NOT NULL,
   avg_review         ReviewRate,
   production_year    prod_year,
   company_id         integer,
   status_id          integer,
   PRIMARY KEY (movie_id)
);

CREATE TABLE status
(
   status_id    integer,
   name         varchar (30) NOT NULL,
   PRIMARY KEY (status_id)
);

CREATE TABLE production_company
(
   production_id    integer,
   name             varchar (30) NOT NULL,
   country          varchar (30),
   PRIMARY KEY (production_id)
);



CREATE TABLE Person
(
   person_id    integer,
   name         varchar (30) NOT NULL,
   birthday     date,
   sex          SexType,
   birhplace    varchar (30),
   age         integer,
   num_of_movies integer
   PRIMARY KEY (person_id)
);

CREATE TABLE person_movie
(
   movie_id    integer,
   person_id   integer,
   role_id   integer,
   PRIMARY KEY (movie_id, person_id, role_id)
);

CREATE TABLE movie_company
(
   movie_id    integer,
   company_id   integer,
   PRIMARY KEY (movie_id, company_id)
);

CREATE TABLE review
(
   username    varchar(50),
   movie_id   integer,
   rate       ReviewRate,
   text       varchar (100),
   PRIMARY KEY (username, movie_id)
);

CREATE TABLE users
(
   username   varchar (50),
   name       varchar (30),
   mail       varchar (30) NOT NULL UNIQUE,
   password   varchar (20) NOT NULL,
   PRIMARY KEY (username)
);

create table users_followers
(
   username_follower varchar(50),
   username_followed varchar(50),
   PRIMARY KEY (username_follower, username_followed)
);


ALTER TABLE images
   ADD CONSTRAINT FK_movie_images FOREIGN KEY (movie_id)
       REFERENCES movie (movie_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE images
   ADD CONSTRAINT FK_person_images FOREIGN KEY (person_id)
       REFERENCES person (person_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE movie_company
   ADD CONSTRAINT FK_prod_company_company FOREIGN KEY (company_id)
       REFERENCES production_company (production_id)
         ON UPDATE CASCADE ON DELETE SET NULL;
```

```sql
ALTER TABLE movie_company
   ADD CONSTRAINT FK_prod_company_movie FOREIGN KEY (movie_id)
       REFERENCES movie (movie_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE movie
   ADD CONSTRAINT FK_status FOREIGN KEY (status_id)
       REFERENCES status (status_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE review
   ADD CONSTRAINT FK_review_user FOREIGN KEY (username)
       REFERENCES users (username)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE review
   ADD CONSTRAINT FK_review_movie FOREIGN KEY (movie_id)
       REFERENCES movie (movie_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE person_movie
   ADD CONSTRAINT FK_pers_movie FOREIGN KEY (movie_id)
       REFERENCES movie (movie_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE person_movie
   ADD CONSTRAINT FK_movie_person FOREIGN KEY (person_id)
       REFERENCES person (person_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE person_movie
   ADD CONSTRAINT FK_movie_person_role FOREIGN KEY (role_id)
       REFERENCES role (role_id)
         ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE users_followers
   ADD CONSTRAINT FK_user_follower FOREIGN KEY (username_follower)
         REFERENCES users (username) ON UPDATE CASCADE ON DELETE SET NULL;

ALTER TABLE users_followers
   ADD CONSTRAINT FK_user_followed FOREIGN KEY (username_followed)
         REFERENCES users (username) ON UPDATE CASCADE ON DELETE SET NULL;
```

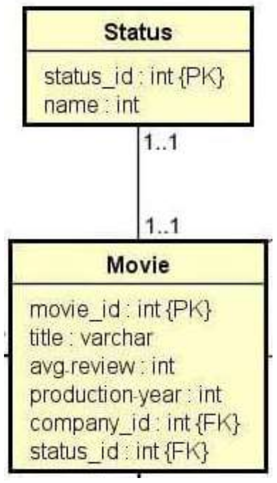
<u>EXAMPLES OF:</u>



- STRONG AND WEAK ENTITIES

<u>Weak</u> entity is for example **table 'Status'**, while on the other hand **table 'Movie'** is a <u>strong</u> entity.

- RELATIONSHIP TYPES



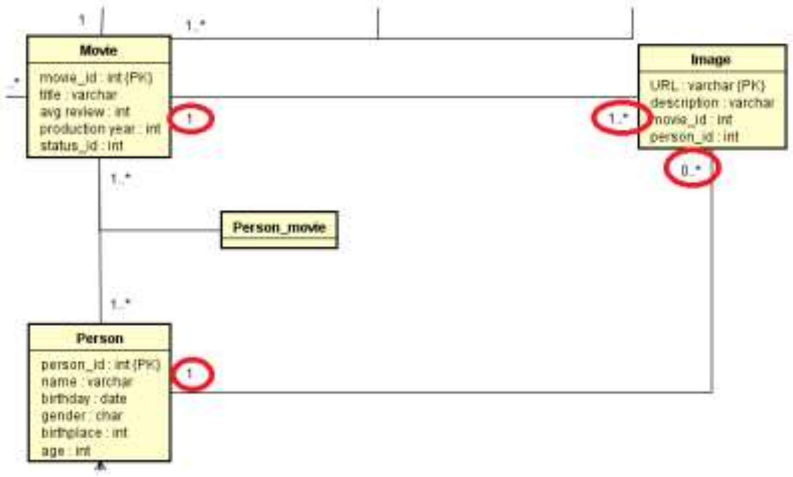
One-to-one relationship

  - One-to-one relationship

The example of one to one relationship is between table 'Movie' and table 'Status' where one movie can have only one status (for example "released", "post-production" etc.)

  - One-to-many relationship

Examples of one-to-many relationships can be found between tables 'Movie' and 'Image' or between 'Person' and 'Image'. In both cases an image cannot be associated with more than one movie/person, but movie is associated with number of images.

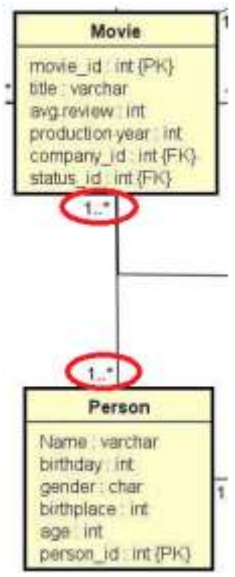


One-to-many relationship

- Many-to-many relationship

Many-to-many relationship can be distinguished between following tables:

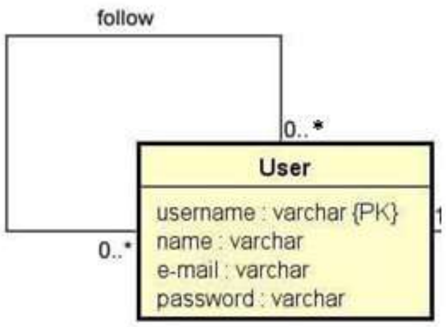
- Movie & Production_company
- Movie & Person
- Movie & User



Many-to-many relationship

- Recursive relationship

The table 'User' has a recursive relationship to itself that indicates the following of other users.



Recursive relationship

- ATTRIBUTES

  - Single-valued attribute
    - *movie_id* in table ‘Movie’
    - *username* in table ‘User’
    - etc…
  - Composite attribute
    - *name* in table ‘Person’ (could be subdivided into other two attributes containing first name and last name)
  - Multi-valued attribute
    - *description* in table ‘Image’
  - Derived attribute
    - age in table ‘Person’ (calculated based on the attribute *birthday* containing date of birth)
  - Attributes on relationships
    - *text* in table ‘Review’

- KEYS:
    - PRIMARY KEYS

TABLE	Primary key
Movie	*movie_id*
Images	*url*
Status	*status_id*
Production_company	*production_id*
Person	*person_id*
Movie_company	*movie_id, company_id (composite)*
Person_movie	*movie_id, person_id, role_id (composite)*
Review	*user_id, movie_id (composite)*
Users	*username*
Users_followers	*username_follower, username_followed(composie)*
Role	*role_id*

    - CANDIDATE KEYS

TABLE	Candidate key(s)
Movie	*movie_id*
Images	*url*
Status	*status_id, name*
Production_company	*production_id, name*
Person	*person_id*
Movie_company	*movie_id, company_id*
Person_movie	*movie_id, person_id*
Review	*user_id, movie_id*
Users	*username, mail*
Users_followers	*username_follower, username_followed*
Role	*role_id*

- COMPOSITE KEYS

TABLE	**Composite key(s)**
Movie	-
Images	-
Status	-
Production_company	-
Person	-
Movie_company	*movie_id, company_id*
Movie_person	*movie_id, person_id, role_id*
Review	*user_id, movie_id*
Users	-
Users_followers	*username_follower, username_followed*
Role	-

- Specialization/Generalization

Four entities: Director, Actor, Producer and Writer are generalized to create the superclass Person that contains common attributes *person_id, name, birthday, birthplace, age, gender.* The relationship that the Person superclass has with its subclasses is mandatory and nondisjoint, denoted as {Mandatory, And}; each member of the 'Person' superclass must be a member of one or more of the subclasses.

- SUPERCLASS: **Person**
- SUBCLASSES:
  - **Director**
  - **Actor**
  - **Producer**
  - **Writer**
- PARTICIPATION CONSTRAINT: **Mandatory**
- DISJOINT CONSTRAINT: **Nondisjoint {And}**