# Homework 3. Constraints

In this assignment you are going to work with database Movies which is the running example in your textbook and which has the following schema:

**Movie** (**title**: string, **year**: int, **length**: int, **inColor**: int, **studioName**: string, **producer**: int)

**MovieStar** (**name**: string, **address**: string, **gender**: char, **birthdate**: date)

**StarsIn** (**movieTitle**: string, **movieYear**: int, **starName**: string)

**MovieExec** (**name**: string, **address**: string, **certN**: int, **netWorth**: int)

**Studio** (**name**: string, **address**: string, **presCertN**: int)

You should create all tables without constraints by running the following SQL script: *moviesdb_noconstraints.sql*. To run it before connecting to your database use:

**`psql < path/script_name`**.

If you are inside the database, run the command

**`\i path/script_name`**.

Note that you can access your home directory directly from database server, so you may create the directory **scripts** and put the script **moviesdb_noconstraints.sql** there.

Note that this script will also create new schema **`movies`**, and set the db environmental variable ***`path`*** to this new schema. Now you can access the tables within this schema without specifying the full qualified name of each database object.

Also note that PosgreSQL converts all object names to lower case, so in the future you either define and access your objects using lower case letters, or, if you want to keep upper or mixed case for object names, you enclose these names into double-quotas.

The goal of this assignment is to learn constraints and to practice setting them using postgreSQL.

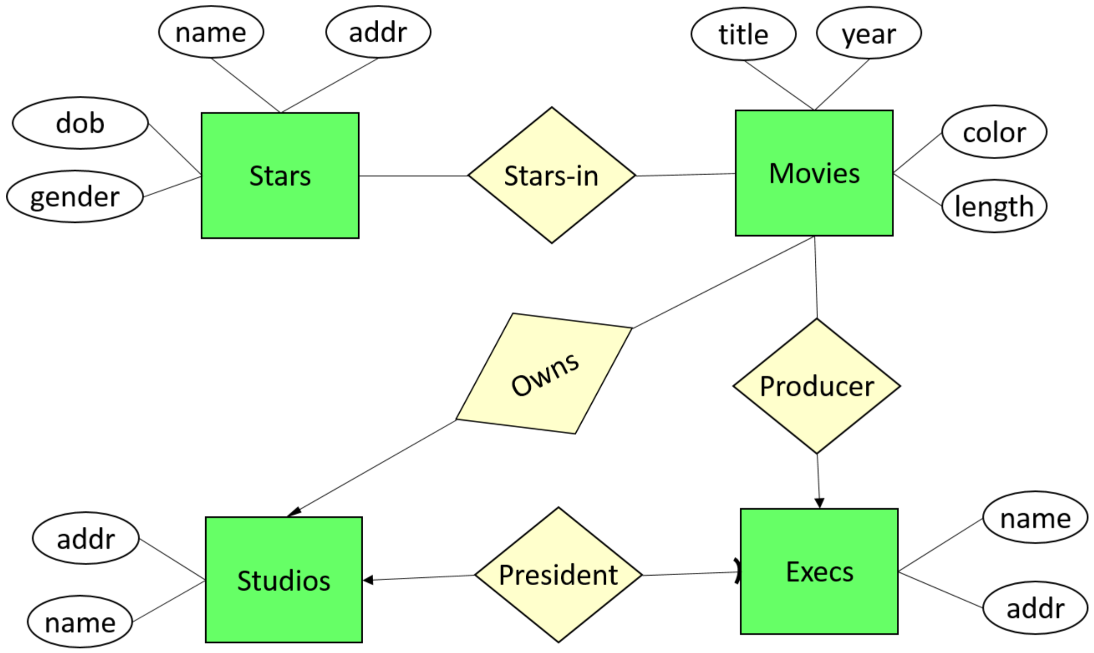
## 1. Integrity constraints

1.1. For each table, define and set primary keys. Do not introduce new surrogate keys, use instead the combination of attributes which is supposed to be unique for each tuple. Do not name your keys, leave the default names given by the DBMS. (1 point)

1.2. Based on the relationships between entities depicted in the E/R diagram below, set all foreign key constraints for table ***StarsIn***. Specify the handling of deletes in parent tables in such a way that the tuples in ***StarsIn*** will also be deleted. The updates of keys in parent tables should be rejected by the system. (1 point)

1.3. Specify foreign key constaint(s) for table ***Movie***. Certificate number of a movie producer cannot be updated if it has an entry in table ***Movie***, however if a producer is deleted from ***MovieExec***, its corresponding entry in ***Movie*** should be set to null. (1 point)

1.4. The studio name can change, and in this case we want to update the reference to the studio in table ***Movies*** accordingly. The studio cannot be deleted if it has a corresponding entry in table ***Movies***. (1 point)
1.5. Finally, insert the foreign key constraint for ***Studio***. The requirement is that each studio must have a president when this studio is created. The person cannot be updated or deleted from ***MovieExec***, if he/she is the studio president. (1.5 points)



## 2. Value constraints

2.1. The year of the movie cannot be before 1915. (1 point)
2.2. The length cannot be less than 60 nor more than 250. (1 point)
2.3. The studio name can only be Disney, Fox, MGM, or Paramount. (1.5 points)
2.4. The black-and-white movies should not be more than 90 minutes in length. (2 points)
2.5. We want to limit our collection of female stars to actresses born after ‘1960-01-01’. (2 points)

After setting all these constraints, you need to collect all new table definitions into a file **movie_constraints.txt**. To redirect all your **\d+** commands to this file, set

**\o movie_constraints.txt**.

After setting this output mode, execute **\d+** for each table in the following order:

```
\d+ Movie

\d+ “MovieStar”

\d+ “StarsIn”

\d+ “MovieExec”

\d+ Studio
```

To change the output mode back to standard output, execute `\o` without parameters.

Submit your new table definitions in file **movie_constraints.txt**.

Note: do not delete your *movies* schema, we are going to use it later in the course.

## 3. Chicken/egg problem with deferrable constraints (3 points)

Consider the following schema for Manager and Department relations:

**Manager** (eid, name, dept_id, address)
**Department** (id, name, manager_id)

In the spec for these tables we find the following requirements:

*When we insert a new manager into the Managers table, we must specify the id of the department he/she is responsible for, and the department has to exist in the Departments table. On the other hand, when we create a new department we have to specify the id of its manager, and the corresponding manager should have been already added to the Managers table.*

Specify these requirements by creating corresponding tables and setting primary/foreign key constraints.

Now, we need to insert our first manager-department values:

```
INSERT INTO manager VALUES (1, 'Bob', 1, 'address1');
INSERT INTO department VALUES (1, 'sales', 1);
```

How do we accomplish this task if with any order of insertions, one of 2 integrity constraints get violated?

Write corresponding sequence of commands for creating tables with constraints and for inserting the above values, test them in `psql` shell, and submit your entire script in file **q3.sql**. Please put

```
DROP TABLE IF EXISTS table_name CASCADE;
```

before each table creation statement.

Total: 16 points which worth only 1% of the course grade (sadly).