RAWDATA Assignment 2 – Querying IMDB with SQL

This assignment concerns development of functions and procedures that extract info from the available version of the IMDB database. Since the functions will be stored in the database, you need to work on your own copy of the database and the best and easiest solution is to generate it on your own computer. See instructions by the end of this text.

What to do

Create the requested functions in your database and run the test queries listed below. Collect everything (create function statements and test queries) for all questions in a single SQL script file, generate a file with output from running the script and **hand in this output file** (not the script file). See **details below on hand-in** by the end of this text. Hand in one submission from your group (from one of the members).

Important: Work individually, then work in groups, then hand-in by group

You are supposed to submit one hand in per group and we strongly recommend you to work with the questions individually as well as in groups. Discuss and decide on what you consider to be the best solutions and hand these in.

Questions

Question a)

The following SQL-query counts the numbers of movies Kevin Bacon has participated in.

```
SELECT count(distinct movie_id)
FROM casting c, person p, movie m
WHERE c.person_id = p.id
        AND c.movie_id = m.id
        AND m.kind='movie'
        AND c.role_type='actor'
AND p.name like 'Bacon, Kevin';
```

Write a function in SQL, `title_count()`, that takes the name of an actor as argument and returns the number of titles the actor has participated in. Thus the same as above, if called with `'Bacon, Kevin'`.

Test-query:

```
SELECT title_count('Bacon, Kevin');
```

Question b)

Change the function from a) such that it takes three parameters: an actor_name and a kind and a role_type[1].

[1] see description of the imdb2 data in Assignment 1.

Test-query:

```
SELECT title_count('Bacon, Kevin', 'episode', 'actor');
```

Question c)

Give a solution to a more dynamic way of using counting titles. It must be possible to ask for a count in three ways: given an actor name only, given an actor_name and a kind, and given an actor_name, a kind and a role_type.

Test-queries:

```
SELECT title_count('Bacon, Kevin');
SELECT title_count('Bacon, Kevin', 'episode');
SELECT title_count('Bacon, Kevin', 'episode', 'actor');
```

Question d)

Write a function, **`movies()`**, that takes the name of an actor as argument and returns the production_year and the titles of the movies the actor, has participated in. Thus

```
SELECT movies (Fairlie, Ann');
```

should return the list of titles that Mads Mikkelsen has acted in.

Test-query:

```
SELECT movies (Fairlie, Ann');
```

Question e)

Write a function, **`match_title(w varchar(100))`**, that takes a string w, such as a word or a phrase, as input and find the most recent movies with a title in which w appears. Return a table with title and year of at most 10 titles and ignore those with unknown year of production.

Test-query:

```
select * from match_title('Wonderful');
```

Question f)

The following SQL-query retrieves all the different role_types that Kevin Bacon has been casted with.

```
SELECT DISTINCT role_type
FROM casting, person
WHERE casting.person_id = person.id
```

```
AND name like 'Bacon, Kevin';
```

Write a function in SQL, **`collect_role_types (person_name)`**, that returns a single value: a comma-separated string, listing the role_types that the person_name has been casted with. Thus, the function call

```
SELECT collect_role_types('Bacon, Kevin');
```

should return the following:

```
                    collect_role_types
-------------------------------------------------------------
 actor, cinematographer, director, editor, producer, writer
(1 row)
```

Use a cursor and loop through the query-result to assemble the string. Use a text variable and gradually extend its value using the | | operator (concatenation).

Test-queries:

```
SELECT collect_role_types('Bacon, Kevin');
SELECT name, collect_role_types(name)
FROM person where name like 'De Niro, R%';
```

Question g)

Rewrite your solution to f) such that you use a FOR loop for looping through a query result rather than a normal cursor. Observe, if you define a variable like **`rec record;`** you can use a loop construct like **`for rec in select … loop … end loop;`**. Notice, if your select returns a single column table with column **`nn`**, **`rec`** will be the value of **`nn`** in parentheses while **`rec.nn`** will be just the value.

Test-queries:

```
SELECT collect_role_types('Bacon, Kevin');
SELECT name, collect_role_types(name)
FROM person where name like 'De Niro, R%';
```

Question h)

This last question is about maintaining redundant information using triggers.
Do the following:

- Alter the table person and add an extra column role_types (use text as column type)
- Either update all persons or just two for test purposes. You need to assign a comma-separated string, listing the role_types that the person_name has been casted with, to the role_types attribute. If you update alle persons, take a break or use your computer for something else, while this update is running – it may take ½ to 2 hours. If you prefer, you can just update the two persons with id's 1372139 and 103491.

- Now create an insert trigger that ensures to keep the (redundant) value of the role_types column in the person table up to date after an insert on the castings table.
- Show that it works with the test queries below
- Extensions of this trigger (or addition of other triggers) to take other events (especially deletion) into consideration are not required.

Test-queries (in that order):

```
select name,role_types
from person where id in (1372139, 103491);
insert into casting(id,person_id,movie_id,role_type)
values (49502700, 1372139, 2440185, 'director');
insert into casting(id,person_id,movie_id,role_type)
values (49502701, 103491, 2400814,
'production designer');
select name,role_types from person where id in
(1372139, 103491);
```

Install and use the imdb database on your own computer

To download and install the imdb database on your own computer, do the following:

1) Download the file **imdb2.backup.zip** using the link on Moodle (more than 500 MB).
2) Unzip to get the file **imdb2.backup** (maybe around 2 GB).
3) Open your command line interface and change the current directory to the directory where you placed the unzipped file **imdb2.backup**.
4) Run the two commands (leading to a 6-10 GB database in your Postgres DBMS)
   ```
   psql -U postgres -c "create database imdb2"
   psql -U postgres -d imdb2 -f imdb2.backup
   ```
5) Write a SQL script assignment2.sql with your solution to assignment 2 (see below on how to format the script file). Run your solution such that you get the output in a file **rawX-assignment2.txt** using the following command (add path info to the script file or change the current directory to where the script file i placed):
   ```
   psql -U postgres -a -d imdb2 -f assignment2.sql > rawX-assignment2.txt
   ```
6) When finished using the imdb2 database for Assignment 2 you may consider (to save space) to delete the two files and to drop the database. To drop the database use this command:
   ```
   psql -U postgres -c "drop database imdb2"
   ```

Use the database on your account on rawdata.ruc.dk

To upload and create the imdb database tables on your account on rawdata.ruc.dk, do the following. Observe that you are using the database rawX and creating the imdb tables in that (you don't have privileges to create new databases on rawdata.ruc.dk).

1) Download the file **imdb2.backup.zip** using the link on Moodle (more than 500 MB).
2) Unzip to get the file **imdb2.backup** (maybe around 2 GB).
3) Open your command line interface and change the current directory to the directory where you placed the unzipped file **imdb2.backup**.
4) Run the command
   ```
   psql  -h rawdata.ruc.dk -p 5432 -U rawX -W -f imdb2.backup
   ```
5) Write a SQL script assignment2.sql with your solution to assignment 2 (see below on how to format the script file). Run your solution such that you get the output in a file **rawX-assignment2.txt** using the following command (add path info to the script file or change the current directory to where the script file i placed):
   ```
   psql -h rawdata.ruc.dk -p 5432 -U rawX -W -a -d imdb2 -f assignment2.sql > rawX-assignment2.txt
   ```
6)
   ```
   psql -h rawdata.ruc.dk -p 5432 -U raw16 -W -a -d raw16 -f assignment2.sql > rawX-assignment2.txt
   ```
7) When finished using the imdb2 database for this assignment, you may consider to drop the tables as well as the functions, procedures and triggers from your group rawX database. You'll need to clean up befare you hand in the result of portfolio subproject 1.

How to hand in your solution

When you have tested all your solutions to questions a) to g) one by one, include all these in a single SQL script file **assignment2.sql** with content like:

```
-- GROUP: <group-name>, MEMBERS: <name1>, <name2>, ...
-- a)
create function title_count (actor_name char(50))
returns integer as $$
...
SELECT ....  – test query 1
...

SELECT ....  – test query 2
...


-- b)
drop function if exists title_count(char(50));

...
```

The dropping is not necessary, but convenient because you can run your script over and over again. For question h) you will need to do something more if you want your script to be repeatable (which obviously is not required).
To indicate "no solution" for a question, just write "-- no solution" in place of the SQL expression.
To hand in your solution do the following (**rawX** is your group name, thus one of **raw1**, **raw2**, **raw3**, **…**):

- Generate an output file **rawX-assignment2.txt** by running the command:
  psql -U postgres -a -d imdb2 -f assignment2.sql > rawX-assignment2.txt
- Hand in your output file **rawX-assignment2.txt** on the Moodle page.

References

- [DSC] chapter 5.2-5.3
- [PGTUT] http://www.postgresqltutorial.com/introduction-to-postgresql-stored-procedures/
- [PGMAN] 43.1 - 43.12 (lookup details)