øvelse 9.4.1a
UIS/Anders Lassen/20180317

The problem (ex 9.4.1a) page 402

..

Movies (title, year, genre, studioName, producerC#)
StarsIn (movieTitle, movieYear, starName)
MovieStar (name, address, gender, birthdate)
MovieExec (name, address, cert#, netWorth)
Studio (name, address, presC#)

Given the name of a star, delete them from Moviestar and delete all their movies from StarsIn and Movies . ..

Analysis

With the provided scipt **movie_schema_p.sql** in **databases.zip**, foreign keys and primary keys have been placed on some, or all tables, for example StarsIn. To delete a movie from Movies, all referencing StarsIn must also be removed in order to satisfy the referential integrity constraints.

The database schema in the book is without keys so an initial solution would ignore the known constraints, that you cannot delete a tuple that is referenced by another tuple.

SQL DML and DDL

The tables of database schema movies should exist in you home schema (MovieStar, Movies and StarsIn). Run script **movie_schema_p.sql**. If the tables are in place, the POSTGRES procedure is created by installing a function with **void** RETURNS-clause (void means empty).

- **create or replace function....**

This creates a stored procedure in PL/pgSQL. Remember to commit. The function is a database element, and exists as a database element in the same manor as a table, view, trigger-function or assertion.

To test the procedure, an anonymous block can be run in PSQL or PGADMIN as a SQL statement. Usually a SQL-worksheet will contain only one anonymous block, but there is no problem in having a mixed worksheet with usefull SQL and anonymous blocks for your selection. Work practice for a DBA (Database Administrator) could state that configuration changes to the database are added to a versioned script in order to roll on changes to a fresh database instance. These scipts must include modifications to metadata (neccessary INSERT- UPDATE-DELETE statement) with anonumous blocks to call needed stored procedures.

```
SELECT m.title, m.year, m.length, m.genre, m.studioname, m.producerc
,  s.movietitle, s.movieyear, s.starname
 FROM
  starsin s
 , movies m
  WHERE s.movietitle = m.title
 AND s.movieyear = m.year
 AND s.movietitle = movietitle --in_movietitle
 AND s.movieyear = movieyear --in_movieyear
 --AND starname like '%M%'
 AND m.title like '%Wa%'
 and m.year = 1992
 ;
```

This SQL coins the test. We want to delete Mike Myers from MovieStars, along with the StarsIn references to movies he stared in, along with the movies. Mike Myers is our test subject.

- (A) To lists stars in 'Waynes way' use as is.
- (B) To list movies Mike Myers stared in uncomment the starname AND-clause and comment the two last AND-clauses.

The precondition is (B) listing Waynes War and (A) listing two stars participating.
The post condition is (B) empty.

Configuration step when your login schema is not public (UIS database).
The most challenging solution is when loading movie_schema_p.sql . This script install the movie database and data to your local login schema allong with some foreign keys.

```
$ psql -h localhost -d uis -p 5433 -U knh487 -W -f movie_schema_p.sql
```

The tables could also be created in your local schema by selecting from schema public. In this case no primary key constraint or foreign key constraints are created and no referenctial integrity constraints in place.

```
CREATE TABLE moviestar AS SELECT * FROM public.moviestar ;
CREATE TABLE movies AS SELECT * FROM public.movies ;
CREATE TABLE starsin AS SELECT * FROM public.starsin ;
```

Usefull selects

```
SELECT * FROM moviestar;
SELECT * FROM movies;
SELECT * FROM starsin;
SELECT * FROM public.starsin;
```

Usefull select- listing tables. Change knh487 to your own kuid.

```
SELECT table_schema || '.' || table_name
FROM information_schema.tables
WHERE table_type = 'BASE TABLE'
AND table_schema NOT IN ('pg_catalog', 'information_schema')
AND table_schema IN ('public', 'knh487')
ORDER by table_name
;
```

Creating a stored procedure
Creating a stored procedure of type function. The following stored procedure can be installed on the database – either by loading it into a SQL-worksheet in **PGADMIN** and run the script (and commit), or by executing a script in **psql**.

```
$ psql -h localhost -d uis -p 5433 -U knh487 -W -f exercise_9-4-1-a-procedure.sql
```

```
or from psql

uis=# \i exercise_9-4-1-a-procedure.sql
uis=#

uis=# COMMIT;
```

The sourcecode of the stored procedure has the following contents. The parameter takes a string for the star in question. The name is prefixed 'in_' for convinience. The declaration-section has two named cursors:

- (1) to select movies **in_name** has participated in; and
- (2) to select stars in the current movie.

The two record definitions has no defined structure but act as tuple-variables for the two FOR-loops holding the current tuple. The bodypart between **BEGIN** and **END** is a nested FOR-loop. Note that the records holds the current tuple and the named cursors are called with a parameter. No exit test is needed.

For every movie the **in_name** was star in, the stars of that movie is listed. When this is done, the stars in that movie are deleted, and the current movie is deleted. When the loop has finished all the movies **in_name** was stars in, along with all the StarsIn relations for that movie, have been deleted, along with the movie. **in_name**'s StartIn-tuples and **in_name**'s MovieStar tuple are deleted.

As debug listing of program flow, 'RAISE NOTICE'- statements have been used. The output is unclear in the current form and dificult to decifer, but inspect tthe statements and note the number perfix notation.

```
CREATE OR REPLACE FUNCTION delete_star_extended(
  in_name VARCHAR
) RETURNS void
AS
$PLAN_MM_AL$
  -- Given a name of a star, delete them from moviestar
  -- and delete all their movies from startsin and movies.
  -- Problem: how about movies other stars reference
  -- usefull: SET SEARCH_PATH TO movie;
  -- ROLLBACK;

DECLARE

  cu_star_starsins CURSOR (in_name moviestar.name%TYPE) FOR
  SELECT  movietitle, movieyear, starname
  FROM moviestar m
  , starsin s
  WHERE name = in_name
  AND m.name = s.starname
  ;

  cu_movies_stars CURSOR (in_movietitle starsin.movietitle%TYPE
  , in_movieyear starsin.movieyear%TYPE) FOR
  SELECT s.movietitle, s.movieyear, s.starname
  FROM  starsin s
  , movies m
  WHERE s.movietitle = m.title
  AND s.movieyear = m.year
  AND m.title = in_movietitle
  AND m.year = in_movieyear
  ;

  rec_movie_featured_in record;
  rec_star_featured record;
BEGIN
  RAISE NOTICE 'delete_star_extended-START : %', in_name;

  FOR rec_movie_featured_in IN cu_star_starsins(in_name)
  LOOP
    RAISE NOTICE 'LOOP-record : %', in_name;
    RAISE NOTICE 'Title : % year : % star : % '
    , rec_movie_featured_in.movietitle, rec_movie_featured_in.movieyear,
rec_movie_featured_in.starname
    ;

    -- slet movies
    RAISE NOTICE 'MOVIES for %', in_name;

    FOR rec_star_featured IN cu_movies_stars (rec_movie_featured_in.movietitle,
rec_movie_featured_in.movieyear)
    LOOP
      RAISE NOTICE 'STARS OF MOVIE title : % year : % star : % '
        , rec_star_featured.movietitle, rec_star_featured.movieyear,
rec_star_featured.starname
      ;

    END LOOP;

    RAISE NOTICE 'DELETING starsin % % ', rec_movie_featured_in.movieyear,
rec_movie_featured_in.starname;
    -- slet starsin
    DELETE FROM starsin
    WHERE movietitle = rec_movie_featured_in.movietitle
    AND  movieyear = rec_movie_featured_in.movieyear
    ;

    RAISE NOTICE 'DELETING MOVIES for %', in_name;
    --
    DELETE FROM movies
    WHERE title = rec_movie_featured_in.movietitle
    AND year = rec_movie_featured_in.movieyear
    ;

  END LOOP;
  RAISE NOTICE 'END-record-CU : %', in_name;

  RAISE NOTICE 'DELETING STARSINS for %', in_name;
  -- slet starsin
  DELETE FROM starsin
  WHERE starname = in_name;
  -- slet moviestjerne
  RAISE NOTICE 'DELETING MOVIESTAR %', in_name;
  DELETE
  FROM moviestar
  WHERE name = in_name
  ;

  RAISE NOTICE 'delete_star_extended-END : %', in_name;
END
$PLAN_MM_AL$
LANGUAGE plpgsql
;


Commit to save(or press red button).
```

To remove the stored procedure, which is a database item, use the DROP-statement. Notice that the function signature must be included (function name and parameters). Functions can be overloaded with any number of parameters and datatypes.

```
DROP FUNCTION delete_star_extended(
  in_name VARCHAR
);
```

Commit to save(or press red button).

Testing or running the script

To test the stored procedure run a script with an anonymous block. The script uses 'RAISE NOTICE' to output some debug - but again, this is not the optimal of solution to tracking program statements. You must carefully understand the trace (an teach your instructor to improve!!). The anonynous block calls the stored procedure with **PERFORM**. This produces a trace that loops candidate records and deletes stars for movies to be removed, the cadidate movies, and finaly the StarsIn - and MoivieStar entries for the moviestar.

Remember is to rollback, if you want to rerun the sript. Notice a SELECT-INTO statement is included as an example of integrating a SELECT in an anonymous block. The SELECT-INTO must return a scalar (a single value) or a tuple. If returning a tuple, extra return variables or a record/tuple-type variable must be specified.

```
-- Test script
-- AL20180316

-- usefull DDL:
SET SEARCH_PATH TO movie;
-- ROLLBACK;
```

øvelse 9.4.1a
UIS/Anders Lassen/20180317

```
DO $$
DECLARE
 star_t moviestar.name%TYPE:='That';
 eksempel2 varchar(100):= 'fremad';
 eksempel3 varchar(100):= 'fremad';

BEGIN
 RAISE NOTICE 'uis-hi';
  PERFORM delete_star_extended('Mike Myers');

 --SELECT uis_mm ('uis-kk') into eksempel3 ;
 SELECT 'forward' into eksempel3 ;

 RAISE NOTICE 'uis-hi-END % %',eksempel2, eksempel3;

END
;$$
```

Rollback to run again (or press green button).

This script (exercise_9-4-1-a-script.sql) can be run from pgadmin or the command-line-interpreter psql (psql-command-line).

```
$ psql -h localhost -d uis -p 5433 -U knh487 -W -f exercise_9-4-1-a-script.sql

or from psql

uis=# START TRANSACTION;
uis=# \i exercise_9-4-1-a-script.sql
uis=#
uis=#  SELECT --- from moviestars –'Mike Myers'- will not be found

uis=# ROLLBACK;

uis=#  SELECT --- from moviestars –'Mike Myers'- will be listed
```