øvelse 9.4.1b
UIS/Anders Lassen/20180316

The problem (ex 9.4.1b) 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 movie studio, produce the address of its president. ..

Scripts

Database provided script **movie_schema_p.sql** in **databases.zip.**

Analysis

The solution to this problem following the line of solution 9.4.1a is to create a cursor selecting all those movie studios with the provided name. In the stored procedure body a for loop outpus the addrees of the studio along with other tuple information.

An alternate solution is to place a SELECT..INTO statement in the stored procedure body and return either into variables or a tuple variable (record). This is the simple case where SELECT..INTO returns a scalar.

```
CREATE..(in_name studio.name%TYPE)
  myName  studio.name%TYPE;
  myAddress  studio.address%TYPE;
  myPresident  studio.president%TYPE;
BEGIN
  SELECT name, address, president
  FROM studio
  WHERE name = in_name
  INTO myName, myAddress, myPresident;
END;
```

Suppose the studio name is not unique. I studioname is the primary key, the all studio names must be unique. But suppose studio name is not unique, the the SELECT..INTO could return more that one row. Suppose the studioname does not exist. Then the SELECT..INTO return no rows. And the consequence is that the PL/pgSQL engine raises an exception and only exception handling can prevent the exception to stop the execution.

Situation	Exception
But suppose studio name is not unique, the the	TOO_MANY_ROWS

SELECT..INTO could return more that one row.	
Suppose the studioname does not exist. Then the SELECT..INTO return no rows.	NO_DATA_FOUND

This lead to two solution, is normal execution dictates that the stored procedure survives an exception, then the SELECT..INTO statement is placed in an anonymous block catching the exception.

If the normal execution is to halt after the exception is raise, the exception section of the stored procedure body can handle the exception, outputting an alert.

In your own work debugging involves identifying such situations where exceptions are ignored by program code, and unguarded SQL statement like these are candidates.

Solution 1

```
CREATE OR REPLACE FUNCTION ex941b1(
  in_name VARCHAR
) RETURNS void
AS
 $EX_B1$
DECLARE
BEGIN
  ..
  DECLARE
  BEGIN
    SELECT..INTO statement
  EXCEPTION
    WHEN NO_DATA_FOUND THEN
      RAISE NOTICE ”Try again, No data found”;
    WHEN TOO_MANY_ROWS THEN
      RAISE NOTICE ”Try again, check your keys, more that one studio found”;
  END;
  ..
END
$EX_B1$
LANGUAGE plpgsql
;
```

Solution 2

```
CREATE OR REPLACE FUNCTION ex941b2(
  in_name VARCHAR
) RETURNS void
AS
 $EX_B2$
DECLARE
BEGIN
  SELECT..INTO statement
EXCEPTION
```

```
  WHEN NO_DATA_FOUND THEN
    RAISE NOTICE ”Try again, No data found”;
  WHEN TOO_MANY_ROWS THEN
    RAISE NOTICE ”Try again, check your keys, more that one studio found”;
END
$EX_B2$
LANGUAGE plpgsql
;
```

It is left as an excersice to implement a solution. But there is a difference. In solution 1 the exception is purposely handled inside the body. A classical example is implementing 'upsert' or 'merge'. Insert data, if it exists, the update.. or visa versa. Update, if it fails, then insert.

SQL DML and DDL

The tables of database schema movies should exist in you home schema (MovieStar, Movies and StarsIn, studio, movieexec). 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 scripts must include modifications to metadata (neccessary INSERT- UPDATE-DELETE statement) with anonumous blocks to call needed stored procedures.

```
SELECT name, address, president
  FROM studio
  --WHERE name = in_name
  WHERE name like '%CN%'

SELECT s.name, s.address, s.presc#
, me.name, me.address, cerr#, me.netWorth
  FROM studio s
    JOIN movieExec me ON me.cert# = s.presc#
  --WHERE name = in_name
  WHERE s.name like '%CN%'
```

The last SQL coins the test. We want find CNN studio and list the address (of the president-ups)..

- we meed to join studio and movieexec to optain the presidents address.

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 (edit to suit case)**.

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

or from psql
```

øvelse 9.4.1b
UIS/Anders Lassen/20180316

```
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 one named cursor:

- (1) to select address of studio (to do join to studio president)

The record definition has no defined structure but act as tuple-variable for the FOR-loop holding the current tuple. The bodypart between **BEGIN** and **END** is one FOR-loop. Note that the record holds the current tuple and the named cursor is called with a parameter. No exit test is needed.

For every studio with **in_name**, the studio information along with its president is listed.

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

```
CREATE OR REPLACE FUNCTION ex941b(
  in_name VARCHAR
) RETURNS void
-- usefull DDL: ROLLBACK;SET SEARCH_PATH TO movie; COMMIT;
-- change the cursor and for loop to reflect the presidents address (TODO)
AS
 $PLAN_AL_EX$
DECLARE
  cu_studio_adress CURSOR (in_name studio.name%TYPE) FOR
  SELECT name, address, presc
  FROM studio
  WHERE name = in_name
  ;

  rec_studio record;
BEGIN
  RAISE NOTICE 'ex941b-START : %', in_name;
  RAISE NOTICE 'Given the name of a movie studio (%), produce the address of its
president.', in_name;

  --Find the address of the president

  FOR rec_studio IN cu_studio_adress( in_name)
  LOOP
    RAISE NOTICE 'LOOP-record : %', in_name;
    RAISE NOTICE 'studio : % address : % president : % '
    , rec_studio.name, rec_studio.address, rec_studio.presc
    ;
  END LOOP;
  RAISE NOTICE 'END-LOOP-CU : %', in_name;
  RAISE NOTICE 'ex941b-END : %', in_name;
END
$PLAN_AL_EX$
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 ex941b(
  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 to rollback if an error occurs, the script can be rerun. 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.

In PGADMIN4 selected statements can be executed isolated. I tend to select the rollback statement execute, (and then select the set seach_path statement to point to my movie schema, and execute - if needed), to reset the query tool..

```
-- exercise_9-4-1-b-script.sql
--ROLLBACK;SET SEARCH_PATH TO movie;

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

BEGIN
```

```
 RAISE NOTICE 'uis-hi';
  PERFORM ex941b('Mike Myers');

  PERFORM ex941b('CNN');
  PERFORM ex941b('Paramount');



 --SELECT uis_mm ('uis-kk') into eksempel3 ;
 SELECT concat (eksempel2, ' uis-kk ', eksempel3) into eksempel3 ;

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

END
;$$
```

Rollback to run again (or press green button).

This script (exercise_9-4-1-b-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-b-script.sql

or from psql



uis=# START TRANSACTION;
uis=# \i exercise_9-4-1-b-script.sql
uis=#
uis=#  SELECT --- from...

uis=# ROLLBACK;

uis=#  SELECT --- from..
```