Advanced Management of Data
Exercise 1 Topic 2:
Extensions of SQL

SQL/PSM vs PL/pgSQL
SQL/PSM (SQL/Persistent Stored Modules)

- initially published 1996 as ISO standard (ISO/IEC 9075-4:1996) as extension of SQL-92
- since then only the procedural language itself is defined as SQL/PSM as part 4 of the SQL standard (SQL:1999, SQL:2003, SQL:2006, SQL:2008, SQL:2011 and SQL:2016)
- standardizes syntax and semantics for control flow, exception handling, local variables, assignment of expressions to variables and parameters, and (procedural) use of cursors
- resembles and inspired by Oracle's PL/SQL, as well as PostgreSQL's PL/pgSQL (all influenced by Ada)

SQL/PSM vs PL/pgSQL
PL/pgSQL (Procedural Language/PostgreSQL)

- first introduced with PostgreSQL 6.4 in 1998
- since then standard procedural programming language in PostgreSQL
  - only this is activated by default, but there are many other options now
- more like Oracle's PL/SQL than SQL/PSM
- main features of SQL/PSM that differ from PL/pgSQL:
  - exception handlers are subroutines (continue handlers)
  - warnings can be handled like an exception
  - declaration of variables should be based on SQL query result

Get a PostgreSQL database

It is possible to solve the tasks of this and the following exercises just theoretically, but you will learn more by trying it out on a real database, so it is advised to use the PostgreSQL database service from URZ. Just get a database at **`IdM-Portal → Databases → PostgreSQL`**

➔https://idm.hrz.tu-chemnitz.de/user/service/database/postgresql/add/

- enter name (important) and description (doesn't matter) of database
- memorize (or save) read-write-username and password

URZ provides PostgreSQL version 9.6.15, but it should be possible to work with an older or more recent one, too. Hence, you can also install PostgreSQL locally on your own computer or use another database service (as long as there is PL/pgSQL support). Other databases with an implementation of SQL/PSM might also work (but sample solutions might not).

Connect to your PostgreSQL database

To access the PostgreSQL database, just use phpPgAdmin service from URZ:

➔https://wfm.hrz.tu-chemnitz.de/phppgadmin/

- you might like to set *Sprache* to **`English`** or something different (as default language is German)
- select server **`PostgreSQL`** at the left side
- enter read-write-username with corresponding password
- select your database

Otherwise set up your own phpPgAdmin for your database or any other frontend you like or be familiar with (e.g. psql, pgAdmin3 or pgAdmin4).

Hello, World!

- write the typical “Hello, World!” program and save it as stored procedure to your database
- open a SQL console (SQL in upper right corner at phpPgAdmin) and write

```
CREATE OR REPLACE FUNCTION hello() RETURNS VOID AS $$
  BEGIN
    RAISE NOTICE 'Hello, World!';
  END;
$$ LANGUAGE plpgsql;
```

- execute your function

```
SELECT hello();
```

- unfortunately you are likely to see nothing, as the notice is ignored by phpPgAdmin
- Task: therefore rewrite your function to return the text instead of raising a notice

Hello, World!
Again

- you are not allowed to change the return type of a function (or the names of input parameters or the names of the output parameters when there is more than one), as existing calls to this function might not work any longer
- therefore you have to drop the function first and recreate it

```
DROP FUNCTION hello();

CREATE OR REPLACE FUNCTION hello() RETURNS VARCHAR AS $$
  BEGIN
    RETURN 'Hello, World!';
  END;
$$ LANGUAGE plpgsql;
```

- Task: greeting the world over and over again gets boring, so change your function
  - to take a name as input parameter and
  - to output this name instead of “World“

Hello, Robert'); DROP TABLE students; --!

```
CREATE OR REPLACE FUNCTION hello(name VARCHAR) RETURNS VARCHAR AS $$
  BEGIN
    RETURN 'Hello, ' || name || '!';
  END;
$$ LANGUAGE plpgsql;
```

- execute your function with name parameter

```
SELECT hello('Robert'); DROP TABLE students; --');
```

- fortunately you didn't have a table named `students` (for more information see https://xkcd.com/327/ and if you don't get it, just look at https://www.explainxkcd.com/wiki/index.php/Little_Bobby_Tables)

Hello, Robert'); DROP TABLE students; --!
Escaped

- using unfiltered input data is always a bad idea
- therefore you should escape them with functions like `format()`, `quote_ident()`, `quote_literal()` or `quote_nullable()`
  - see manual for more information: https://www.postgresql.org/docs/current/static/functions-string.html

```
CREATE OR REPLACE FUNCTION hello(name VARCHAR) RETURNS VARCHAR AS $$
  BEGIN
    RETURN format('Hello, %s!', name);
  END;
$$ LANGUAGE plpgsql;
```

Hello, ?!

- execute your function again without parameter
- as you can see, it still returns **`Hello, World!`**
- this is because of function overloading
- basically you have two different functions with the same name
- in phpPgAdmin you could inspect them by expanding the tree on the left side at your database
  - ➔Schemas
    - ➔public
      - ➔Functions
        - ➔hello ()
        - ➔hello (character varying)

More parameters

- Task: write a new function `swap()`, that takes two integer parameters and returns them in reverse order
- hint: you also need two output parameters
- another hint: the keyword for output parameters is `OUT`

More parameters

```
CREATE OR REPLACE FUNCTION swap(INT, INT, OUT INT, OUT INT) AS 'SELECT $2, $1'
LANGUAGE sql;
```

- the function body is just a string and can be escaped by ' or the $$-construct
- in SQL this is quite easy, but we also can do this with PL/pgSQL

```
CREATE OR REPLACE FUNCTION swap(INT, INT, OUT INT, OUT INT) AS $body$
  BEGIN
    $3 := $2; -- PL/pgSQL lets you assign like Oracle's PL/SQL
    $4 = $1;  -- but you can also use the standard SQL/PSM assignment
  END;
$body$ LANGUAGE plpgsql;

SELECT * FROM swap(1, 2);
```