UV Distributed Information Management
Summer semester 2024

Assignment 01

Summary:

Deadline: April 30, 2024, 11:55 pm (aka 23:55) CET.

Extended Deadline: May 07, 2024, 11:55 pm (aka 23:55) CET.

Submission: Submit a compressed archive (e.g., a `zip` or a `tar.gz` file) that contains your Python3 code and the answers to the questionnaire via Blackboard.

Grading: 50% Python3 code, 50% answers (incl. meeting; cf. Section 5 for details).

---

1 General Remarks

The purpose of this assignment is to get in touch with a widely used general-purpose database system (DBS), namely **PostgreSQL.** PostgreSQL [1] is an open-source database system that is based on the relational data model [2]. It is rather intuitive to use and accessible using the Python3 programming language [3] (i.e., in combination with the `psycopg2` [4] module).

For your convenience, Section 5 contains the grading scheme that will be applied for this assignment.

Please submit your final Python3 code **and** your answers to the questionnaire until April 30, 2024, 11:55 pm (aka 23:55) CET via Blackboard [5] (late submission until May 07, 2024, 11:55 pm (aka 23:55) CET). Furthermore, please keep in mind that the exams contribute 40% to your final grade, hence you need to submit at least one assignment (partially) to pass this course.

1.1 Formatting Conventions

Commands for the Linux command-line tool (`terminal`) [6] and the command-line tool of PostgreSQL (`psql`) are written in `TrueType` font [7]. In addition, all commands are in a box that specifies the used command-line tool at the beginning of the title (separated by a dash –, i.e., **`terminal`** for Linux and **`psql`** for PostgreSQL). Listing 1 shows an example command executed in the Linux `terminal`.

The Linux `terminal` shows a prefix `dbtutorial@database-tutorial:~#` that consists of

- the name of the user that executes the command; `dbtutorial` is the default user of the virtual machine (VM) (this may be different if you do not use the given VM),
- the name of the machine; `database-tutorial` is the name of the given VM, and

[1] PostgreSQL: `https://de.wikipedia.org/wiki/PostgreSQL`
[2] The relational data model: `https://en.wikipedia.org/wiki/Relational_model`
[3] The Python programming language: `https://en.wikipedia.org/wiki/Python_(programming_language)`
[4] `https://www.psycopg.org/` and `https://pypi.org/project/psycopg2/`
[5] Blackboard: `https://elearn.sbg.ac.at`
[6] Command-line tool: `https://en.wikipedia.org/wiki/Command-line_interface`
[7] `TrueType` font: `https://en.wikipedia.org/wiki/TrueType`

- a delimiter that separates the actual command from the "`user@machine`" string; "`:~#`" is the default delimiter of the given VM (where ~ denotes the current directory).

Moreover, we use a color code for readability:

- A **command** itself (which is to be copied) is depicted in bold, solid **`black`**.
- **Prefixes**, which are *not* part of a command itself, are shown in `gray`.
- **Outputs**, which are the result of a command that has been executed, are shown in `green`.

Listing 1: **`terminal`** – Show directories.

```
dbtutorial@database-tutorial:~# ls -l
total 32
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Mar 31 15:43 Desktop
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Documents
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Downloads
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Music
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Pictures
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Public
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Templates
drwxr-xr-x 2 dbtutorial dbtutorial 4096 Feb 28  2022 Videos
```

Contrarily, Listing 2 exemplifies a command in PostgreSQL's command-line tool.

Listing 2: **`psql`** – Show all entries in table `t`.

```
dbtutorial=> SELECT * FROM t;
id
----
(0 rows)
```

For commands that are to be executed in the `psql` terminal, the prefix `dbtutorial=>` denotes the database we are currently connected to (cf. Section 2.2 for more details). Furthermore, SQL keywords like `SELECT` and `FROM` are capitalized. Additional information can be found in the supplementary material given in Section 4.

Python3 [8] code is simply wrapped in a box without specifying any command-line tool, and we provide a `.py` file that contains the Python3 code (in order to make using the template easier).

1.2 Support

Remark: Please notify the instructor as soon as possible if this assignment description is (partially) unclear or if there is a problem with the submission.

If you have trouble understanding this assignment, please use one of the following communication channels to get help (in this order):

1. **Lecture:** wednesdays 01:00 - 02:30 pm CET (exception: lecture-free periods).
2. **Slack:** `https://dbteaching.slack.com/archives/C06MFP830GH` (I will check regularly and do my best to reply fast, but please do not expect me to be available 24/7).
3. **Email:** dkocher@cs.sbg.ac.at (as a last resort).

We recommend to start the assignment early. In case of a problem, it is easier for the instructor and other students to provide help in time if you identify problems early.

[8] The Python Programming language: `https://en.wikipedia.org/wiki/Python_(programming_language)`

2 Assignment Description

We **highly recommend** that you read this section (including all subsections) to the end before you start working (this should be less error-prone).

This assignment is divided into five parts and only parts 1 (depending on your choice), 4, and 5 contribute to the overall grade of this assignment. We recommend to follow these steps and refer to the corresponding sections for more details.

1. Set up and configure PostgreSQL (PSQL); cf. Section 2.1 and Section 2.2.
2. Create tables and fill tables with data; cf. Section 2.3.
3. Familiarize yourself (i.e., play around) with SQL; cf. Section 2.4.
4. Write an example application that accesses the database using Python3; cf. Section 2.5.
5. Answer the questionnaire; cf. Section 2.6.

2.1 PostgreSQL Setup

This section summarizes two possible ways to set up PostgreSQL and solve this assignment:

PostgreSQL Installation Install PostgreSQL on your own system (or virtual machine) using an operating system of your choice. This implies that you may "pollute" your system with this installation and that the instructor may need additional information on your particular setup to provide help (since we mostly work on Linux systems). Nonetheless, we want to emphasize that it is an **excellent exercise** for students to perform the installation of such a system on their own (at least once in their career). If you choose this option, you can continue with Section 2.1.1.

PostgreSQL using a VM Configure a pre-installed PostgreSQL instance in a virtual machine (VM) that runs Debian Linux. This implies that you will not experience the process of installing PostgreSQL on your own, but it may be easier for the instructor to help since the VM runs Debian Linux. On the one hand, it may be cumbersome if you do not have any experience with Linux systems (and VMs), but on the other hand it may also be a nice opportunity to familiarize yourself and play around with a Linux system (and a VM). To continue with this option, you can jump to Section 2.1.2.

You can choose your preferred way and the choice itself will not influence your grade in any way. However, make sure that you can answer questions regarding your choice.

2.1.1 PostgreSQL Installation

Remark: If you decide to use the VM image, you can directly jump to Section 2.1.2. PostgreSQL is already pre-installed.

The first step is to install a PostgreSQL server locally on your (possibly virtual) machine. This should be possible on almost all operating systems but the specific steps may diverge for different operating systems. Once PostgreSQL is installed, you can create a database and import data. Sources on how to install PostgreSQL can be found in the supplementary material provided in Section 4. Although PostgreSQL also provides a graphical user interface named *pgAdmin*, we recommend to install PostgreSQL's **Command-Line Tools** (on some systems, they must be installed separately), which allow you to interact with the database system using the command line. In particular, the `psql` command-line tool (aka `psql` *terminal* or *shell*) provides the most basic way to use the database. Contrarily, the *Stack Builder* is not required for this assignment.

2.1.2 Virtual Machine Setup

We provide a so-called virtual machine (VM) *image*, which consists of a pre-installed Debian Linux as operating system and a pre-installed PostgreSQL instance. In this case, you may have to familiarize yourself with the concept of a VM and how to host the corresponding image (see Assignment 0 for details). Essentially, a virtual machine [9] is a software that is designed to provide you with a substitute of a real machine, that is, it runs another operating system on top of your regular operating system – it **virtualizes** the hardware of another system. For this assignment, it suffices to install a software named VirtualBox [10] that acts as a host for virtual machines. After you successfully installed VirtualBox, the VM image can be used to set up the virtual machine that can be used for this assignment. The image (as `.ova` file) can be downloaded from a Google drive [11] and runs Debian Linux [12] as operating system with an installation of PostgreSQL.

After downloading the `debian-vm.ova` file, this file must be imported into VirtualBox. There are many online tutorials [13] on how to accomplish this and it primarily consists of two steps (see also Assignment 0): (1) Choose the image to import (navigate to the downloaded file) and (2) check/modify the settings of the virtual machine (typically no modifications are required, but sometimes, for example, you may have to disable the USB port). Once you click on the **Import** button, VirtualBox imports the image and sets up the virtual machine that can be used for this assignment. VirtualBox then shows a new VM on the left-hand side, which can be started/booted with a double-click.

For this VM, there exist two users: (1) `dbtutorial` and (2) `root`. If you are not familiar with the concept of a `root` user (or superuser), please check out the corresponding article [14] on Wikipedia. In essence, the `root` user has **all privileges**, whereas the `dbtutorial` user has not. By default, we will work with the `dbtutorial` user but we will temporarily switch to the `root` user if we need additional privileges. The password for both users is the same: `dbpwd1`

Once you logged into the VM and before you start working on the actual assignment, try to familiarize yourself a bit with Debian Linux (unless you already have experience using Linux systems and/or solved Assignment 0). For example, open a Linux `terminal` (aka Linux *shell* or *command-line tool*) and try some basic commands (some links can be found in the supplementary material, cf. Section 4). In particular, we will use PostgreSQL's `psql` terminal (aka `psql` *shell* or *command-line tool*), which provides the most basic way to use PostgreSQL.

Remark: Note that you may not be able to use the typical `CTRL+C` and `CTRL+V` sequence to copy and paste between your regular system and the VM as the clipboard is not shared by default [15].

2.2 PostgreSQL Configuration

During installation, PostgreSQL creates a default database named `postgres`. Depending on your particular setup, you may be asked for a server, a database, a port, and/or credentials (username + password). In this case, use the following default configuration:

- Server: `localhost` (or leave blank)
- Database: `postgres` (or leave blank)
- Port: `5432` (or leave blank)

[9] Virtual machine: `https://en.wikipedia.org/wiki/Virtual_machine`
[10] VirtualBox: `https://www.virtualbox.org/manual/ch02.html`
[11] VM image for this assignment: `https://drive.google.com/file/d/1s6bp8TyzFfHL08EZaI1P4Qm6nnTG7mFA/view?usp=sharing`
[12] Debian Linux: `https://en.wikipedia.org/wiki/Debian`
[13] Virtual machine import: `https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html`
[14] The `root` user: `https://en.wikipedia.org/wiki/Superuser`
[15] Shared clipboards in VirtualBox: `https://www.youtube.com/watch?v=fqrJ7qlhJu0`

- Username: `postgres` (or leave blank)
- Password: The password you set during the installation (in case of doubt: `dbpwd1`).

Before we use the database, we create a database user named `dbtutorial`. There are two types of users: (i) Users of the operating system and (ii) users of PostgreSQL (i.e., they exist only within the database system). First, we open a Linux terminal and switch to the system's `root` user (cf. Listing 3).

Listing 3: **terminal** – Switch to the `root` user (mind the trailing dash).

```
dbtutorial@database-tutorial:~# su -
Password:
root@database-tutorial:~#
```

Remark: The `dbtutorial@database-tutorial:~$` (as well as `root@database-tutorial:~#`) at the beginning should already be displayed in your terminal and is not part of the command itself (which is `su -`). Do not get confused by the fact that nothing is shown when you type in the password (not even asterisks `*****`, which are often used for passwords). This is the default behavior of Linux and you will be asked again if the password is incorrect.

Then, we switch to the system's `postgres` user (which also exists in our database) and connect to the default database named `postgres` (indicated by the prefix `postgres=#`; cf. Listing 4).

Listing 4: **terminal** – Switch to the `postgres` user and start the `psql` terminal.

```
root@database-tutorial:~# su - postgres
postgres@database-tutorial:~# psql
postgres=#
```

Remark: The `postgres=#` (or `postgres=>`) at the beginning should already be displayed in your `psql` terminal and is not part of the command.

Now, we are in the `psql` terminal and can *directly* send commands to the PostgreSQL database. All available databases can be viewed by executing the `\l` command (mind the backslash) in the `psql` terminal as shown in Listing 5.

Listing 5: **psql** – List all available databases.

```
postgres=# \l
```

There should be three databases: `postgres`, `template0`, and `template1`. The next step is to create a new user [16] with restricted privileges as shown in Listing 6.

Listing 6: **psql** – Create a new user `dbtutorial` within your database.

```
postgres=# CREATE USER dbtutorial WITH CREATEDB LOGIN PASSWORD 'dbpwd1' NOSUPERUSER;
CREATE ROLE
postgres=#
```

Once the database's `dbtutorial` user exists, we can switch back to the system's `dbtutorial` user and reconnect again *without* the intermediate switch to the Linux `root` user. This is the common and more secure way to use our database (it is typically never a good idea to work with `root` privileges all the time). To this end, we first close the connection to our database using the `\q` command as shown in Listing 7 and then exit `root` mode by typing the `exit` command twice (first to switch back from the `postgres` user to `root` and then to switch back to our regular

[16] Create a new user in PostgreSQL: `https://www.postgresql.org/docs/current/sql-createuser.html`

dbtutorial user). Finally, we can start using PostgreSQL with our newly created database user (still on the default database; indicated by `-h localhost postgres`; cf. Listing 8).

Listing 7: **psql** – Close the connection to PostgreSQL.

```
1 postgres=# \q
2 postgres@database-tutorial:~$
```

The command in line 5 of Listing 8 now uses the psql command with multiple options:

-U dbtutorial tells the psql terminal to use the dbtutorial user to connect to the database (instead of PostgreSQL's default user, i.e., postgres).

-h localhost tells the psql terminal that PostgreSQL is running on *our* current machine (and not on some other machine that we want to access remotely). localhost is a defined name that refers to the machine on which the respective command is performed [17].

postgres tells the psql terminal to connect to a database named postgres.

Listing 8: **terminal** – Connect to the postgres database from our default Linux user.

```
1 postgres@database-tutorial:~$ exit
2 exit
3 root@database-tutorial:~# exit
4 logout
5 dbtutorial@database-tutorial:~$ psql -U dbtutorial -h localhost postgres
6 postgres=>
```

Comparing Listing 5 and 8, we notice that the prefix in the psql terminal changed: It shows our new user dbtutorial followed by => (instead of =#). The latter means that we do *not* have root privileges. We can check the privileges of all users with the \du command (cf. Listing 9).

Listing 9: **psql** – Show the privileges of all database users.

```
1 dbtutorial=> \du
2 Role name   |                Attributes              | Member of
3 ------------+----------------------------------------+-----------
4 dbtutorial  | Create DB                              | {}
5 postgres    | Superuser, Create role, Create DB, ... | {}
6
7 dbtutorial=>
```

Importantly, we observe that our dbtutorial user has the privilege to create a new database (Create DB). Although there is a default database, we create a dedicated database for this assignment and name it assignment1. To this end, we execute the command shown in Listing 10.

Listing 10: **psql** – Create a new database named assignment1.

```
1 dbtutorial=> CREATE DATABASE assignment1;
2 CREATE DATABASE
3 dbtutorial=>
```

Remark: Database creation can take some time, hence please wait for the command to finish. Line 2 in Listing 10 is not to be entered by you: This is a feedback from the database to you, indicating that the preceding CREATE DATABASE command has been successfully executed.

Afterwards, the \l command (cf. Listing 5) should print one additional database, namely assignment1. **Please use assignment1 as database for this assignment.** In order to use our

[17] Localhost: https://en.wikipedia.org/wiki/Localhost

new database `assignment1`, you must connect to it **after** creating it (the creation itself does not automatically connect to the newly created database; indicated by the fact that `postgres=>` is still displayed in our `psql` terminal). Therefore, we use the command `\c assignment1` to connect to the new database. After the connection is established, the `psql` terminal should display `assignment1=>` (instead of `postgres=>`). We can also connect to our `assignment1` database using the Linux `terminal` in a similar manner. After closing the connection to our database (cf. Listing 7), we can reconnect to our `assignment1` database as shown in Listing 11.

Listing 11: **`terminal`** – Connect to the `assignment1` database from our default Linux user.

```
dbtutorial@database-tutorial:~$ psql -U dbtutorial -h localhost assignment1
assignment1=>
```

2.3 Data Initialization

Once your database is ready, the database contains no data, i.e., it is empty. By executing the `\d` command (cf. Listing 12), the database will report that no relations (i.e., tables) have been found.

Listing 12: **`psql`** – List all tables in our database `assignment1`.

```
assignment1=> \d
Did not find any relations.
```

Therefore, we have to populate the database with some data. In the course of this assignment, we will use parts of the publicly available *Internet Movie Database (IMDB)* [18]. First, we must download the data from our Nextcloud [19] and unzip it. Second, we use the data definition language (DDL) of PostgreSQL to create the tables that will store the data. Finally, we fill the tables with the data of two given plain files that are to be imported into our database.

After unzipping the data file, we find a new directory named `imdb` with three files:

`name.basics_no_header_array_format.tsv` contains actor names.

`titles.basics_no_header_array_format.tsv` contains movie titles.

`create_db.sql` contains the `CREATE TABLE` statements for this assignment.

The data for this assignment can either be downloaded using a Browser (Firefox, Chrome, and the likes) or with the `curl` tool in the Linux `terminal` (cf. Listing 13). This may be particularly useful if you use the VM without a shared directory.

Listing 13: **`terminal`** – Download the data with `curl` (`user@machine` string shortened).

```
dbtut..@:~$ curl https://kitten.cosy.sbg.ac.at/index.php/s/gjcdKd4fMJn5Ypo/download \
              --output assignment1-data.zip
```

`create_db.sql` is an SQL script that contains the statements to create the tables. Please study these statements to know about the structure of our tables. There are two possible ways to execute the statements. Either you execute the script directly from the `psql` command-line tool using `\i create_db.sql` (cf. Listing 14), or you copy and execute each statement separately using the `psql` terminal.

Remark: The commands shown in Listings 14–16 must either be executed using the correct absolute/relative path to the script/file, or you must navigate into the directory where the script/file is located before connecting to the database, and only then execute the corresponding command from the current directory (e.g., `\i create_db.sql`).

[18] The Internet Movie Database (IMDB): `https://www.imdb.com/interfaces/`

[19] Data for this assignment: `https://kitten.cosy.sbg.ac.at/index.php/s/gjcdKd4fMJn5Ypo/download`

Listing 14: **psql** – Create two tables using the given SQL script.

```
assignment1=> \i /path/to/imdb/directory/create_db.sql
CREATE TABLE
CREATE TABLE
assignment1=>
```

Again, PostgreSQL provides feedback to the user: It prints `CREATE TABLE` twice, i.e., once per table that has been created. To fill the tables with data, we provide two plain files and recommend to execute the two commands shown in Listing 15 (one after another; mind the semi-colon at the end) in your `psql` terminal (where `=>` is the shortened version of `assignment1=>` to fit the line width).

Listing 15: **psql** – Import the two plain files into our tables.

```
=> \COPY titles FROM title.basics_no_header_array_format.tsv WITH DELIMITER E'\t';
COPY 5447872
=> \COPY names FROM name.basics_no_header_array_format.tsv WITH DELIMITER E'\t';
COPY 8991013
```

Remark: The file import will take some time, hence please wait for the commands to finish.

On some systems, the encoding must be specified explicitly when importing the data from file. If you run into problems during the file import, please try the commands shown in Listing 16.

Listing 16: **psql** – Import the two plain files into our tables using UTF8 encoding.

```
=> \COPY titles FROM title.basics_no_header_array_format.tsv WITH DELIMITER E'\t'
   ENCODING 'UTF8';
COPY 5447872
=> \COPY names FROM name.basics_no_header_array_format.tsv WITH DELIMITER E'\t'
   ENCODING 'UTF8';
COPY 8991013
```

After the import, PostgreSQL provides feedback on the number of tuples (lines) that have been imported, i.e., 5,447,872 and 8,991,013 tuples have been imported into the tables `titles` and `names`, respectively. Afterwards, you can see all available tables by executing `\d` (cf. Listing 5). There should be two tables in the database: `names` (of actors) and `titles` (of movies). You can also study the schema of a particular table using the command shown in Listing 17.

Listing 17: **psql** – List the schema information of table `names`.

```
assignment1=> \d names
```

2.4 Introduction to SQL

Once the data has been successfully imported into our `assignment1` database, try to further familiarize yourself with the `psql` terminal. We provide *four* SQL queries (cf. Listings 18–21) that can be executed out of the box by entering them into the `psql` terminal (one after another; mind the trailing semi-colon):

Listing 18: **psql** – Query **Q1**.

```
assignment1=> SELECT *
     FROM names
     WHERE primaryName = 'Chris Hemsworth';
```

Listing 19: **psql** – Query **Q2**.

```
assignment1=> SELECT *
    FROM titles
    WHERE primaryTitle = 'The Avengers' AND titleType = 'movie';
```

Listing 20: **psql** – Query **Q3**.

```
assignment1=> SELECT primaryTitle
    FROM names, titles
    WHERE names.birthYear = titles.startYear
        AND names.primaryName = 'Scarlett Johansson';
```

Listing 21: **psql** – Query **Q4**.

```
assignment1=> EXPLAIN SELECT *
    FROM titles
    WHERE primaryTitle = 'The Avengers' AND titleType = 'movie';
```

`SELECT`, `FROM`, and `WHERE` clauses have briefly been covered in the lecture. The `*` in the `SELECT` clause specifies that **all** columns of the table that is given in the `FROM` clause are retrieved.

In query **Q2**, the `WHERE` clause consists of two conditions that are linked using the `AND` operator. This means that every tuple that is part of the result must satisfy **both** conditions. For example, a TV series that is named "The Avengers" is not found because it is not a movie.

Query **Q3** joins the two tables based on the condition in the `WHERE` clause. A **join** links tuples of two (or more) tables. In this specific case, we link the year of birth of the actors (`names.birthYear`) with the year of publication of the movies (`titles.startYear`). Moreover, we specify that only actors with the name "Scarlett Johansson" should be considered. Intuitively, this means that we ask the database system to show all movies that started in the year of birth of Scarlett Johansson. In this case, the `FROM` clause contains two tables that are separated by a comma. Without the `WHERE` clause, this results in the Cartesian product [20]: Without condition in the `WHERE` clause, every row of the first table (`names`) is linked with every row in the second table (`titles`). This can result in a very large join result and very high runtimes; hence we restrict the join using the given condition(s) in the `WHERE` clause.

The last query, **Q4**, is basically **Q2** with a small extension: We put the `EXPLAIN` keyword [21] in front of the query. `EXPLAIN` shows the so-called **query plan**, that is, information about the steps PostgreSQL **plans** to execute in order to determine the result (without executing the actual query). As an exercise (not graded), execute queries **Q1-3** in combination with the `EXPLAIN` keyword and try to intuitively understand the query plans (on a high level; not all the details).

2.5 Access the Data Using Python3

Although the `psql` terminal can be used to execute queries, a database system is typically accessed by an application. Therefore, the fourth part of this assignment is to write a small Python3 application. We recommend to use the `psycopg2` module (or driver) for Python3 to (a) **establish** a **connection** to your local database, (b) **execute** the **queries** and **retrieve** the results, and (c) **close** the **connection** to your local database. Your application is supposed to execute the queries **Q1**, **Q2**, and **Q4** as given in Section 2.4 and print the respective results (i.e., all tuples that are returned; output format does not matter as long as it is human-readable).

For query **Q3**, you are required to do a small modification: **Q3** as given in Section 2.4 returns a list of tuples that satisfy the condition in the `WHERE` clause. In your Python3 code, **Q3** should only return the **number of tuples** that satisfy the condition in the `WHERE` clause. This can either

[20] Cartesian product: `https://en.wikipedia.org/wiki/Cartesian_product`
[21] PostgreSQL's `EXPLAIN` statement: `https://www.postgresql.org/docs/current/using-explain.html`

be accomplished by modifying the SQL command itself to count the number of tuples (*Hint:* Use the `COUNT` aggregate function [22]) or by adapting the Python3 code such that not all the tuples are printed but only the number of tuples (*Hint:* Use the `len()` function [23]). In any case, think about the consequences of your choice and the implications of the alternative option.

Remark: Please do not confuse the `psycopg2` module with the (relatively) new `psycopg3` module for Python3. We use to the `psycopg2` module in combination with Python3.

Template Code There are many tutorials regarding the installation [24] and the usage of `psycopg2` [25] in combination with PostgreSQL. Nonetheless, we provide a minimum template code in Python3 that can be used as a starting point. Unfortunately, PDF viewers tend to represent formatted code differently that cause troubles when copying the code from the PDF into another file. Therefore, we do *not* include the template code in this assignment description but as a separate `.py` file that is available via the course website [26]. For your convenience, this template code is also located at the Desktop once you successfully set up the given VM image.

2.6 Questionnaire

The questionnaire contains questions about this assignment. These questions are potentially discussed during the after-assignment meetings. The questionnaire can be found in a separate text file named `assignment1-questionnaire.txt`.

3 Submission

Please submit a single compressed archive (e.g., `.zip` or `.tar.gz`) that contains exactly two files: (a) Your Python3 code and (b) your answers to the questionnaire.

Code

Remark: Please consider removing the database credentials (i.e., username and password) before you submit your code (in case you did not use the default ones as described in Section 2.1).

Please submit a single Python3 file (`.py`) that contains the full code for this assignment, i.e., the connection to the database and the execution of the four queries. Your code **must** print the results of **all** four queries **Q1-4** (one after another) when executed **as submitted** (*Hint:* Set up another virtual machine and try whether it runs). We will **not** debug your code, for example, change some variable to make it work. Therefore, please double-check that your Python3 code works as expected and that **all** four queries are executed (recall that query **Q3** needs to be modified as described in Section 2.5).

Questionnaire

Remark: The recommended formats are `.txt` and `.pdf`.

You can answer the questions directly in the text file `assignment1-questionnaire.txt`. If you prefer to use a different application to answer the questions (e.g., Microsoft Word and the likes), you are welcome to do so. In any case, the submitted file must be in one of the following formats: `.txt`, `.pdf`, `.odt`, `.doc`, or `.docx`.

[22] The `COUNT` aggregation: `https://www.postgresql.org/docs/current/functions-aggregate.html`
[23] Python's `len()` function: `https://docs.python.org/3/library/functions.html#len`
[24] Installation of the `psycopg2` module: `https://www.psycopg.org/`, `https://pypi.org/project/psycopg2/`
[25] `psycopg2` tutorial: `https://wiki.postgresql.org/wiki/Psycopg2_Tutorial`
[26] Template code: `https://dbresearch.uni-salzburg.at/teaching/2024ss/dim/assignment1-template.py`

4 Supplementary Material

This section provides a list of pointers to material that may be helpful to solve this assignment.

- PostgreSQL: `https://www.postgresql.org/`
- Download PostgreSQL: `https://www.postgresql.org/download/`
- The full PostgreSQL documentation (in particular, Chapter 1 may be helpful): `https://www.postgresql.org/docs/current/`
- Other resources regarding installation and usage of PostgreSQL:
  - One of the many "Getting Started" guides: `https://www.youtube.com/watch?v=BLH3s5eTL4Y`
  - Installing PostgreSQL: `https://www.postgresqltutorial.com/install-postgresql/`
  - PostgreSQL on Windows: `https://www.postgresql.org/download/windows/`
- Documentation of the `EXPLAIN` statement: `https://www.postgresql.org/docs/current/using-explain.html`
- Python Modules: `https://docs.python.org/3/installing/index.html`
- The `psycopg2` module: `https://www.psycopg.org/`
- Installation of the `psycopg2` module for Python:
  - Official website: `https://www.psycopg.org/install`
  - The Python Package Index: `https://pypi.org/project/psycopg2/`
- The `psycopg2` tutorial in the PostgreSQL wiki: `https://wiki.postgresql.org/wiki/Psycopg2_Tutorial`
- One of the many "Getting Started" guides: `https://www.youtube.com/watch?v=2PDkXviEMD0`
- One of the many introductions to the Linux terminal: `https://www.digitalocean.com/community/tutorials/an-introduction-to-the-linux-terminal`

5 Grading

For the sake of transparency, this section provides more details on the grading of this assignment, i.e., which part contributes how many points to the total number of 20 points.

Code The code contributes at most 10 points and is evaluated based on the following criteria (if the code is executed as submitted; disregarding the credentials):

Max. Points	Criterion
1.5	*Q1 is executed and the correct result is printed to the command line.*
1.5	*Q2 is executed and the correct result is printed to the command line.*
1.5	*Modified Q3 is executed and the correct result is printed to the command line.*
1.5	*Q4 is executed and the correct result is printed to the command line.*
2+2	*Answer two questions on your code in the after-assignment meeting correctly.*
10	

Questionnaire The questionnaire contributes at most 10 points and is evaluated based on the following criteria (taking the discussion in the after-assignment into account):

Max. Points	Criterion
1.5	*Correctness of answer A1.*
0.75+0.75	*Correctness of answer A2.*
1.5	*Correctness of answer A3.*
1.5	*Correctness of answer A4.*
2+2	*Answer two additional questions in the after-assignment meeting correctly.*
10	