UV Distributed Information Management
Summer semester 2023

Assignment 01

Summary:

Deadline: April 19, 2023, 11:55 pm (aka 23:55) CET.

Extended Deadline: April 26, 2023, 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: 55% Python3 code, 45% 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).

Please submit your final Python3 code *and* the answers to the questionnaire until April 19, 2023, 11:55 pm (aka 23:55) CET via Blackboard[5] (late submission until April 26, 2023, 11:55 pm (aka 23:55) CET). Furthermore, please keep in mind that the exams contribute 46% to your final grade, hence you need to submit at least one assignment (partially) to pass the course.

1.1 Formatting Conventions

Commands for the Linux command-line tool (`terminal`), the command-line tool of PostgreSQL (`psql`), and Python3 code are written in `TrueType` font[6]. 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`:

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

```
dbtutorial@database-tutorial:~# ls -l
```

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),

[1]PostgreSQL: `https://de.wikipedia.org/wiki/PostgreSQL`
[2]The relational 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]TrueType font: `https://en.wikipedia.org/wiki/TrueType`

- the name of the machine; `database-tutorial` is the name of the given VM, and
- 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).

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;
```

For commands that are to be executed in the `psql` terminal, the non-bold prefix `dbtutorial=>` denotes the database we are currently connected to (cf. Section 2.2). Furthermore, SQL keywords like `SELECT` and `FROM` are capitalized. More information can be found in the supplementary material given in Section 4. Python3 code is simply wrapped in a box.

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:** mondays 10:15 am - 12:00 pm CET, wednesdays 01:00 - 02:30 pm CET (exception: lecture-free periods).
2. **Slack:** `https://dbteaching.slack.com/archives/C04QHH4TR7B` (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.

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 in Python3; cf. Section 2.5.
5. Answer the questionnaire; cf. Section 2.6.

2.1 PostgreSQL Setup

This section summarizes three 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 jump to Section 2.1.1.

PostgreSQL using a VM Configure a pre-installed PostgreSQL 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.

PostgreSQL using Docker Configure a pre-installed PostgreSQL in a Docker container that runs Debian Linux. For new chipsets (e.g., Apple Silicon[7]) with different architectural characteristics (compared to typical Intel and AMD chipsets), running VMs properly is still problematic, triggering quite some bugs. Therefore, you can also use an experimental setup (i.e., you are among the first students to try it out) using Docker, which basically mimics a lightweight VM without a graphical user interface (i.e., this is the most advanced option). For this option, you can jump to Section 2.1.3.

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 or Docker, you can directly jump to Section 2.1.2 and Section 2.1.3, respectively. PostgreSQL is already pre-installed in both cases.

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.

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[8] 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[9] 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

[7]Apple Silicon chipset: `https://en.wikipedia.org/wiki/Apple_silicon`
[8]Virtual machine: `https://en.wikipedia.org/wiki/Virtual_machine`
[9]VirtualBox: `https://www.virtualbox.org/manual/ch02.html`

that can be used for this assignment. The image (as `.ova` file) can be downloaded from our Nextcloud[10] and runs Debian Linux[11] as operating system with an installation of PostgreSQL.

After downloading the `debian-postgresql-vm.ova` file, this file must be imported into VirtualBox. There are many online tutorials[12] 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 should import the image and set 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[13] 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[14].

2.1.3 Docker Setup (Experimental)

Remark: This is the most advanced option (i.e., no graphical user interface is available) and still experimental (i.e., you are among the first students to try it out).

Please install Docker Desktop[15] for *your* system configuration: Installations exist for Linux, Windows, MacOS with Intel chipset, and MacOS with Apple Silicon chipset. The documentation provides information on how to install and start Docker for your particular system.

Docker[16] is a software package that also virtualizes at the level of an operating system (e.g., you can run Debian Linux within Windows or MacOS) but follows a different philosophy. Traditional VMs are *stateful*, meaning that the state of your VM can be stored and you are able to resume from it later on. Contrarily, Docker is designed to be *stateless*, meaning that it is not as easy to store the state and resume from it. This typically implies that the **effects of all your commands are lost** once you shut down Docker (or the terminal it is running in); but we will learn about a workaround to avoid this later in this section. Furthermore, Docker operates at the (finer) granularity of so-called *containers* where a container is meant to be a lightweight building block with a single responsibility, i.e., it typically runs a single service. Then, multiple of these containers can be used to implement a broader functionality (the containers can be "stacked" like in a dock). In contrast, a traditional VM is a full-fledged, virtualized machine that may run many different services at once.

[10]VM image for this assignment: `https://kitten.cosy.sbg.ac.at/index.php/s/AeBcR5kMAApm33L/download`
[11]Debian Linux: `https://en.wikipedia.org/wiki/Debian`
[12]Virtual machine import: `https://docs.oracle.com/cd/E26217_01/E26796/html/qs-import-vm.html`
[13]The `root` user: `https://en.wikipedia.org/wiki/Superuser`
[14]Shared clipboards in VirtualBox: `https://www.youtube.com/watch?v=fqrJ7qlhJu0`
[15]Getting Started with Docker: `https://www.docker.com/get-started/`
[16]The Docker software package: `https://en.wikipedia.org/wiki/Docker_(software)`

For this assignment, we will use Docker similar to our VM image: It uses Debian Linux with a pre-installed PostgreSQL server. To this end, please download a so-called *Dockerfile* from our Nextcloud[17]. You can then build and run the Docker *image* in your Linux or MacOS `terminal` as well as Windows' Powershell[18] as shown in Listing 3. Line 1 builds the Docker image and pulls a Debian image from the world wide web (i.e., you must be connect to the internet) and tags/names the resulting image as `postgresql`. In line 2, we run the built Docker image by referring to its tag/name (`run -it postgresql`) and specify a directory that is shared between Docker and your "regular" operating system (in this case, the current directory `${PWD}` can be accessed within Docker in `/home/dbtutorial`). Finally, we end up as user `dbtutorial` in the running Docker image as shown in line 3; and a (at first sight) cryptic machine name (`6fb3eb1459c7` in this example). Recall that Docker uses containers and `6fb3eb1459c7` uniquely identifies the current container (as virtual machine, if you want).

Remark: The container hash will be different on your machine.

Listing 3: **`terminal`** – Build and run the Docker image (mind the trailing dot in line 1).

```
1 dbtutorial@database-tutorial:~$ docker build --pull -f "Dockerfile" -t postgresql .
2 dbtutorial@database-tutorial:~# docker run -v ${PWD}:/home/dbtutorial/ -it postgresql
3 dbtutorial@6fb3eb1459c7:~$
```

Within this container, we can now use Debian Linux (cf. Section 2.2) but we only have a command-line interface (no graphical user interface). To this end, we briefly study a few additional Linux commands that may be helpful in the course of this assignment:

`touch assignment1.py` Creates a new, empty file named `assignment1.py` within the current working directory. If you create this file within `/home/dbtutorial`, the file will also be available on your host system through the shared directory mentioned above.

`nano assignment1.py` Opens a command-line editor named `nano`[19] to modify the file `assignment1.py`. At the bottom of `nano`, you find the most important keyboard shortcuts to interact with the editor, e.g., `CTRL+O` saves (i.e., overwrites) the file and `CTRL+X` closes `nano`. As an alternative, you may also use a command-line editor named `vim`[20].

If we close the Docker container using the `exit` command, we lose the effects of all commands that have been executed in our container (including files that we created within the container). This is also reflected by the fact that the container has a different hash (name) if we run it again (cf. line 2 in Listing 3). For convenience, we may want to save the state (although it is not in line with the Docker philosophy). We can accomplish this by opening another terminal (or Powershell) *outside* of our Docker container and executing the commands shown in Listing 4.

Listing 4: **`terminal`** – Commit and resume a container (in this case `6fb3eb1459c7`).

```
1 dbtutorial@database-tutorial:~$ docker commit 6fb3eb1459c7 postgresql-save
2 sha256:6cab............................
3 dbtutorial@database-tutorial:~$ docker images
4 REPOSITORY        TAG        IMAGE ID       CREATED           SIZE
5 postgresql-save   latest     6cab........   19 seconds ago    630MB
6 dbtutorial@database-tutorial:~# docker run -it 6cab........
7 dbtutorial@c69e4ff74ef6:~$
```

First, we use `docker commit` with the hash of our Docker container (which is still running) and a name, e.g., `postgresql-save` (line 1). This "saves" the current state of our Docker container

[17] Dockerfile for this assignment: `https://kitten.cosy.sbg.ac.at/index.php/s/ZCTiTkksrqKe4NY/download`
[18] Windows' Powershell: `https://en.wikipedia.org/wiki/PowerShell`
[19] The `nano` editor: `https://en.wikipedia.org/wiki/GNU_nano`
[20] The `vim` editor: `https://en.wikipedia.org/wiki/Vim_(text_editor)`

into an image, which has a unique image ID that is different from the hash of our running Docker container. We can confirm this by executing `docker images` (line 3), which lists all committed images. To resume this specific Docker container, we must use the image ID (*not* the hash of the container) combined with `docker run -it` (line 6). Line 7 then shows a different container hash (i.e., `c69e4ff74ef6`) but this container resembles the state that has been committed before.

Remark: Any feedback on the current Docker setup is very much appreciated; by "experts" that are familiar with Docker as well as by students that are new to Docker.

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)
- 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., within the database system). First, we open a Linux terminal and switch to the system's `root` user (cf. Listing 5).

Listing 5: **`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 `postgres=>`; cf. Listing 6):

Listing 6: **`terminal`** – Switch to the `postgres` user and start the `psql` terminal.

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

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 7.

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

Listing 7: **`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[21] with restricted privileges as shown in Listing 8:

Listing 8: **psql** – Create a new user `dbtutorial` within your database.

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

Once the database's `dbtutorial` user exists, we can switch back to the system's `dbtutorial` user and reconnect again without `root` privileges. 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 9 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 `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 10).

Listing 9: **psql** – Close the connection to PostgreSQL.

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

Listing 10: **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=>
```

The command in line 5 of Listing 10 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[22].

`postgres` Tells the `psql` terminal to connect to a database named `postgres`.

Comparing Listing 7 and 10, 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 11):

Listing 11: **psql** – Show the privileges of all database users.

```
dbtutorial=> \du

Role name   |                 Attributes              | Member of
------------+-----------------------------------------+-----------
 dbtutorial | Create DB                               | {}
 postgres   | Superuser, Create role, Create DB, ... | {}

dbtutorial=>
```

[21]Create a new user in PostgreSQL: https://www.postgresql.org/docs/current/sql-createuser.html
[22]Localhost: https://en.wikipedia.org/wiki/Localhost

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 named `assignment1` for this assignment. Therefore, we execute the command shown in Listing 12:

Listing 12: **`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 12 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 7) should print one additional database, namely `assignment1`. **Please use `assignment1` as database for this assignment.** In order to use our new database `assignment1`, you must connect to it **after** creating it (the creation itself does not 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 9), we can reconnect to our `assignment1` database as shown in Listing 13.

Listing 13: **`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 14), the database will report that no relations (i.e., tables) have been found.

Listing 14: **`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)*[23]. First, we must download the data from our Nextcloud[24] 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 15). This may be particularly useful if you use Docker or the VM without a shared directory.

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

[24]Data for this assignment: `https://kitten.cosy.sbg.ac.at/index.php/s/4jdyWRCKMxRfDF7/download`

Listing 15: **terminal** – Download the data with `curl` (`user@machine` string shortened).

```
dbtutorial(...):~$ curl https://kitten.cosy.sbg.ac.at/index.php/s/4jdyWRCKMxRfDF7/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 16), or you copy and execute each statement separately using the `psql` terminal.

Listing 16: **psql** – Create two tables using the given SQL script.

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

Remark: The commands shown in Listings 16–18 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 and execute the corresponding command from the current directory (e.g., `\i create_db.sql`).

Again, PostgreSQL provides feedback to the user: It prints `CREATE TABLE` twice, i.e., once per tables 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 17 (one after another; mind the semi-colon at the end) in your `psql` terminal.

Listing 17: **psql** – Import the two plain files into our tables.

```
assignment1=> \COPY titles FROM title.basics_no_header_array_format.tsv WITH DELIMITER E'\t';
COPY 5447872
assignment1=> \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 18 (where => is the shortened version of `assignment1=>` to fit the line width):

Listing 18: **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 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 7). 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 19:

Listing 19: **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 20–23) that can be executed out of the box by entering them into the `psql` terminal (one after another; mind the trailing semi-colon):

Listing 20: **psql** – Query **Q1**.

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

Listing 21: **psql** – Query **Q2**.

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

Listing 22: **psql** – Query **Q3**.

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

Listing 23: **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[25]: 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 condition(s) in the `WHERE` clause.

The last query, **Q4**, is basically **Q2** with a small extension: We put the `EXPLAIN` keyword[26] 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 part of this assignment), the other queries could also be extended with the `EXPLAIN` keyword.

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)

[25] Cartesian product: `https://en.wikipedia.org/wiki/Cartesian_product`

[26] PostgreSQL's EXPLAIN statement: `https://www.postgresql.org/docs/current/using-explain.html`

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 should execute the queries **Q1**, **Q2**, and **Q4** as provided in Section 2.4 and print the respective results (i.e., all tuples that are returned by PostgreSQL; 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 be accomplished by modifying the SQL command itself to count the number of tuples (Hint: Use the `COUNT` aggregate function[27]) 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[28]). 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[29] and the usage of psycopg2[30] in combination with PostgreSQL. Nonetheless, we provide a minimum template code in Python3 that can be used as a starting point.

Listing 24: Python3 template code to access our database using psycopg2.

```python
#!/usr/bin/python3

import psycopg2 as pg2

def main():
  try:
    # Connect to database 'assignment1' with user 'user' and password 'pw'.
    connection = pg2.connect(
      "user='user' password='pw' host='localhost' dbname='assignment1'")
  except:
    print("Unable to establish a connection to {}".format('assignment1'))

  # Retrieve a so-called cursor in order to interact with the database.
  cursor = connection.cursor()

  try:
    # Send a simple SELECT query to the database.
    cursor.execute("""SELECT * FROM names WHERE primaryName = 'Christian Bale'""")

    # Retrieve the entire result of the query.
    # cursor.fetchone() would retrieve only the first tuple of the result.
    records = cursor.fetchall()

    # Print the retrieved result.
    for record in records:
      print("{}".format(record))
  except:
    print("Unable to execute simple SELECT query.")
  finally: # The finally-branch is always executed (independently of an exception).
    if cursor is not None:
      cursor.close() # Close the cursor.

    if connection is not None:
      connection.close() # Close the connection.

if __name__ == "__main__":
  main()
```

[27]The `COUNT` aggregation: `https://www.postgresql.org/docs/current/functions-aggregate.html`
[28]Python's `len()` function: `https://docs.python.org/3/library/functions.html#len`
[29]Installation of the `psycopg2` module: `https://www.psycopg.org/`, `https://pypi.org/project/psycopg2/`
[30]`psycopg2` tutorial: `https://wiki.postgresql.org/wiki/Psycopg2_Tutorial`

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) the 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. The code must print the results of *all* four queries **Q1**–**Q4** (one after another) when executed *as submitted*. 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`.

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 18 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
2	Q1 is executed and the correct result is printed to the command line.
2	Q2 is executed and the correct result is printed to the command line.
2	Modified Q3 is executed and the correct result is printed to the command line.
2	Q4 is executed and the correct result is printed to the command line.
1+1	Answer 2 questions w.r.t. your submission in the after-assignment meeting.
10	

Questionnaire The questionnaire contributes at most 8 points and is evaluated based on the following criteria (taking the discussion in the after-assignment into account):

Max. Points	Criterion
2	Correctness of answer A1.
1+1	Correctness of answer A2.
2	Correctness of answer A3.
2	Correctness of answer A4.
8	