Assignment 01

UV Distributed Information Management

Daniel Kocher, Summer semester 2026

Summary

- **Deadline:** April 26, 2026, 11:59 p.m. (i.e., 23:59) CET.
- **Submission:** Submit a compressed archive (`zip` or `tar.gz`) that contains your Python3 code and the answers to the questionnaire via Blackboard.
- **Grading:** 25% Python3 code, 25% answers, and 50% from the after-assignment meeting (cf. Section 5 for details).

1 Introduction

This assignment provides an introduction to 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 and accessible using the Python3 programming language [3] (e.g., using the `psycopg2` [4] module).

1.1 Grading and Submission

Section 5 specifies the grading scheme for this assignment in detail. Please submit your final Python3 code **and** your answers to the questionnaire until April 26, 2026, 11:59 p.m. (i.e., 23:59) CET via Blackboard [5]. Recall that the assignments contribute 60% to your final grade, i.e., you need to participate in at least one assignment to pass this course.

1.2 Formatting Conventions

Commands for the Linux command-line tool (`terminal`) [6] and the PostgreSQL command-line tool (`psql`) are written in `TrueType` font [7]. In addition, all commands are in a box

[1] PostgreSQL: `https://de.wikipedia.org/wiki/PostgreSQL`
[2] The relational data model: `https://en.wikipedia.org/wiki/Relational_model`
[3] 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`

that specifies the command-line tool at the beginning of the title (separated by a dash –, e.g., `dbterminal` for PostgreSQL). Listing 1 shows an example for the Linux `terminal`.

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
```

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

- 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**, where `database-tutorial` is the name of the given VM.
- A **delimiter** that separates the actual command from the "`user@machine`" string. "`:~#`" is the default delimiter of the given VM (~ denotes the `home` directory).

Moreover, we color-code commands and their output for readability:

- A **command** itself (that may be copied) is depicted in solid `black`.
- **Prefixes**, which are **not** part of a command itself, are shown in `gray`.
- **Output**, i.e., the result of a command, is shown in `green`.

Python3 [8] code is wrapped in a box without specifying any command-line tool, and we provide a `.py` file that contains the Python3 code.

Listing 2 exemplifies a command in PostgreSQL's command-line tool with output.

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

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

For commands that must be executed in the `psql` terminal, the prefix `dbtutorial=>` denotes the database we are currently connected to (cf. Section 2.2 for more details). SQL keywords like `SELECT` and `FROM` are typically capitalized, but this is not mandatory. Additional information can be found in the supplementary material given in Section 4.

1.3 Support

If anything is unclear or if there is a problem with the submission, please use one of the following communication channels to get help (in this order):

1. **Lecture:** Wednesday 11:30 a.m. - 01:30 p.m. CET (exception: lecture-free periods).

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

2. **Slack:** `https://dbteaching.slack.com/archives/C08GRUMJ97H` (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. 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 to read the entire section** (incl. all subsections) before you start working (this should be less error-prone). This assignment has five parts:

1. Set up and configure PostgreSQL (PSQL) 🔗 Section 2.1 and Section 2.2.
2. Create tables and fill tables with data 🔗 Section 2.3.
3. Familiarize yourself with (P)SQL 🔗 Section 2.4.
4. Write an small Python3 application to access the database 🔗 Section 2.5.
5. Answer the questionnaire 🔗 Section 2.6.

Follow these steps and refer to the corresponding sections for details.

2.1 Options to Solve this Assignment

This section outlines four options to solve this assignment.

1. **Use our Server Infrastructure:** The Database Research Group runs a PostgreSQL server with a web interface that can be used to solve this assignment. On the one hand, you can focus on SQL and do not have to deal with technical details; on the other hand, you will not experience the PostgreSQL installation yourself. To continue with this option, you can jump to 🔗 Section 2.1.1.
2. **Use a Virtual Machine (VM):** Use a VM with Debian Linux and PostgreSQL pre-installed as you know it from Assignment 0. Again, you will not experience the PostgreSQL installation yourself, but it may be less cumbersome than setting up PostgreSQL. To continue with this option, you can jump to 🔗 Section 2.1.2.
3. **Install PostgreSQL:** Install PostgreSQL on your system. 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** to install such a system yourself (at least once). To continue with this option, you can jump to 🔗 Section 2.1.3.

4. **Use Docker (experimental; bonus point opportunity):** Use a Docker image with PostgreSQL installed. For new chipsets (e.g., Apple Silicon [9]) with different architectural characteristics (compared to typical Intel and AMD chipsets), running VMs in VirtualBox is still problematic and buggy. Docker mimics a lightweight VM without a graphical user interface (GUI), i.e., this is the most advanced option. As this option is still experimental, you have the opportunity to receive **up to 1 bonus point** if you help to improve the Docker setup by giving (constructive) feedback on it. To continue with this option, you can jump to 🔗 Section 2.1.4.

You can choose your preferred option, but **be prepared for questions** on your choice.

2.1.1 Use our Server Infrastructure

We use the server infrastructure of the Database Research Group and you receive **personalized credentials** (i.e., do not expose them to others) to use it in your preferred browser (e.g., Firefox). Once you received your credentials, the **workflow** is as follows:

1. **Connect** to our interface: `http://terminal01.dbresearch.plus.ac.at/`
2. **Log in** with your personalized credentials and set up two-factor authentication (as you probably already know it from your student email).
3. You are redirected to a **web-based terminal interface** asking for your credentials once again, i.e., enter them.
4. You are logged into a **virtual machine (VM)** named `studentdbclient01` that you can use to work on this assignment.

Remark: If you run into problems, please contact the instructor as soon as possible and try to describe the problem in detail. We try to resolve it fast.

This option does **not** require `root` privileges. Consequently, there is **no possibility** to gain them (e.g., using `su`) and you can safely skip most steps of Section 2.2 and can proceed with Section 2.3, but take the following **important differences** into account:

- You neither have to create a new database user nor to create a new database. You can connect to an existing database as shown in Listing 3. Please replace `username` with your username. Afterwards, you are connected to your personalized database.
- For security reasons, your VM is isolated and we cannot use `curl` to download the data. However, the file `assignment1-data.zip` is already stored on your VM and you just need to `unzip` it as shown in Listing 4. Afterwards, you can proceed as described in Section 2.3.

Remark: To use the copy & paste functionality in the web interface, please use the shortcut `CRTL+ALT+SHIFT` to show/hide additional options.

[9] Apple Silicon chipset: `https://en.wikipedia.org/wiki/Apple_silicon`

Listing 3: `terminal` – Connect to your personalized database on our infrastructure.

```
username@studentdbclient01:~$ psql -h studentdbserver01 -p 5432 -U username -d ss2026-username
ss2026-username=> \q
username@studentdbclient01:~$
```

Listing 4: `terminal` – Unzip the data for this assignment.

```
username@studentdbclient01:~$ unzip assignment1-data.zip
```

2.1.2 Use a Virtual Machine (VM)

Remark: In case you owe an Apple Silicon machine [10], you find online resources [11] on how to convert the `OVA` format into a format that is usable with, e.g., UTM [12].

We use the VM image of Assignment 0, and you may have to familiarize yourself with the concept of a VM and how to host the corresponding image (cf. Assignment 0 for details). In 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. Essentially, the `root` user has **all privileges**, whereas the user `dbtutorial` has only limited privileges. By default, we will work with the `dbtutorial` user, but we will temporarily switch to `root` if we need additional privileges. The password for both users is the same: `dbpwd1`

Once you logged into the VM, familiarize yourself with Debian Linux (unless you already have experience using Linux systems and/or solved Assignment 0). For example, open the **command-line tool** of Linux, i.e., the `terminal`, and try basic commands (cf. Assignment 0 and Section 4). In particular, we will use PostgreSQL's **command-line tool**, i.e., the `psql` terminal, which provides the most basic way to use PostgreSQL.

Remark: Note that you may not be able to use `CTRL+C` and `CTRL+V` to copy and paste between your regular system and the VM as the clipboard is not shared by default [14].

2.1.3 Install PostgreSQL

For this option, the first step is to install a PostgreSQL server on your machine. This should be possible on any operating system, but the specific steps may differ. Sources on how to install PostgreSQL can be found in the supplementary material provided in Section 4. While PostgreSQL provides a graphical user interface, i.e., **pgAdmin**, we recommend to install PostgreSQL's **command-line tools** (on some systems, they must be installed separately), which allow for interaction with PostgreSQL using the command line. In particular, PostgreSQL's command-line tool `psql` provides the most basic way to interact with the database. Contrarily, the **Stack Builder** is not required for this assignment. Once PostgreSQL is installed, you can create a database and import data.

[10] Apple Silicon chipset: `https://en.wikipedia.org/wiki/Apple_silicon`
[11] OVA to UTM: `https://medium.com/@d_mechraoui/how-to-run-ova-virtual-machine-on-apple-silicon-using-utm-m1-m2-m3-7a2a883697d4`
[12] UTM – Virtual machines for MAC: `https://mac.getutm.app/`
[13] The `root` user: `https://en.wikipedia.org/wiki/Superuser`
[14] Shared clipboards in VirtualBox: `https://www.youtube.com/watch?v=fqrJ7qlhJu0`

2.1.4 Use Docker — Experimental + Opportunity for Bonus Point

Remark: You have the opportunity to receive **up to 1 bonus point** if you help to improve the Docker setup by providing (constructive) feedback.

First, install Docker Desktop [15] for your system. Installations exist for Linux, Windows, MacOS on Intel chipsets, and MacOS on Apple Silicon chipsets. 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**, i.e., the state of your VM can be stored and resumed later on. Contrarily, Docker is designed to be **stateless**, i.e., it is not as easy to store the state and resume from it. Hence, follow the instructions carefully not to lose **effects of your commands and/or data** when shutting down Docker (or the terminal it is running in). Furthermore, Docker operates at the (finer) granularity of **containers**, where a container is a lightweight building block with a single responsibility, e.g., a container can run a database. Then, multiple of these containers can be used to implement a broader functionality (containers are "stacked" like in a dock). Contrastingly, traditional VMs are full-fledged VMs that may run different services at once.

For this assignment, we use Docker as follows: We build a container that runs a PostgreSQL server and connect to it to work on the assignment. To this end, download the **Dockerfile** from our Nextcloud [17], open a `terminal`, and navigate into the directory where the Dockerfile is stored. Then, build the Docker image tagged (i.e., named) `postgresql-tutorial` as shown in Listing 5.

Listing 5: `terminal` – Build the Docker image (mind the trailing dot) & confirm it.

```
1  user@machine:~$ docker build -t postgresql-tutorial .
2  user@machine:~$ docker image ls -a
3  IMAGE                      ID           DISK USAGE CONTENT SIZE EXTRA
4  postgresql-tutorial:latest cfbc844c4ec3      323MB          0B
5  user@machine:~$ mkdir -p postgresql-data
```

Line 1 builds the Docker image and tags/names the resulting image, whereas line 2 lists all Docker images on the host system. Moreover, line 3 creates a new subdirectory that is then used to store the PostgreSQL database persistently on your host system, i.e., it is **not** lost if you shut down the container, and you can reconnect to continue working.

Next, we run the container (cf. Listing 6). This is a multi-line command that is split by "\" such that the `terminal` include the next line into the command, i.e., Lines 1–7 of Listing 6 show a single command that spreads over 7 lines. Line 1 starts a Docker container named `psql-tutorial` based on the image tagged `postgresql-tutorial` (cf. end of line 8). Line 2–4 specify three environment variables `POSTGRES_USER`, `POSTGRES_PASSWORD`, and `POSTGRES_DB`, which define the user, the password, and the database, respectively. Lines 5–6 mount two directories of your host system "into" the container:

[15] Getting Started with Docker: `https://www.docker.com/get-started/`
[16] The Docker software package: `https://en.wikipedia.org/wiki/Docker_(software)`
[17] Dockerfile: `https://kitten.cosy.sbg.ac.at/index.php/s/m96XCEsfcqznTLi`

– The directory `./postgresql-data` of the host system is mounted to the container's directory `/var/lib/postgresql/data`. This ensures that the data you store in the PostgreSQL database is not lost if you shut down the container.
– The directory `./imdb` [18] of the host system is mounted to the container's directory `/home/dbtutorial/assignment1-data`. This allows you to transfer the assignment data (and other files, e.g., the Python3 code) between host system and container.

Informally, these directories can be interpreted as shared directories between your host system and the container. Finally, line 7 detaches the container, i.e., it runs in the background [19], and specifies the Docker image `postgresql-tutorial` as basis for the container. Line 8 concludes by listing the Docker containers using a separate command.

Listing 6: `terminal` – Run Docker container with two volumes & confirm it.

```
1 user@machine:~$ docker run --name psql-tutorial \
2       -e POSTGRES_USER=dbtutorial \
3       -e POSTGRES_PASSWORD=dbpwd1 \
4       -e POSTGRES_DB=assignment1 \
5       -v ./postgresql-data:/var/lib/postgresql/data \
6       -v ./imdb:/home/dbtutorial/assignment1-data \
7       -d postgresql-tutorial
8 user@machine:~$ docker container ls
```

To connect to the running container, we use the command shown in Listing 7.

Listing 7: `terminal` – Connect to the Docker container.

```
1 user@machine:~$ docker exec -it -u dbtutorial -w /home/dbtutorial psql-tutorial bash
2 dbtutorial@6fb3eb1459c7:~$
```

We observe that the `user@machine` prefix changes to `dbtutorial@6fb3eb1459c7`. That is, the user `dbtutorial` is now connected to the container with the **unique hash** `6fb3eb1459c7` (this hash is most probably different on your system), and we can continue with the command shown in Listing 18 to connect to our database `assignment1`. From this point on, we can follow the instructions provided in Section 2.3, but keep in mind that (a) data that is stored within the directory `assignment1-data` is also stored permanently on your host system, and (b) we can use this directory as shared directory between host system and Docker container. For example, you could write the Python3 code on your host system and execute it in the container. For completeness, we describe how to stop and remove the Docker container (cf. Listing 8), and remove the corresponding Docker image (cf. Listing 9) **once you solved this assignment.**

Listing 8: `terminal` – Stop and remove the Docker container and confirm it.

```
1 user@machine:~$ docker container ls -a
2 CONTAINER ID   IMAGE                COMMAND      CREATED          STATUS          PORTS       NAMES
3 6fb3eb1459c7   postgresql-tutorial  .......      10 minutes ago   Up 10 minutes   5432/tcp    psql-tutorial
4 user@machine:~$ docker container stop 6fb3eb1459c7
5 user@machine:~$ docker container rm 6fb3eb1459c7
6 user@machine:~$ docker container ls -a
7 CONTAINER ID   IMAGE                COMMAND      CREATED          STATUS          PORTS       NAMES
```

[18] Data for this assignment: `https://kitten.cosy.sbg.ac.at/index.php/s/gjcdKd4fMJn5Ypo`
[19] Daemons in computing: `https://en.wikipedia.org/wiki/Daemon_(computing)`

Listing 9: `terminal` – Remove the Docker image and confirm it.

```
user@machine:~$ docker image ls -a
IMAGE                      ID           DISK USAGE CONTENT SIZE EXTRA
postgresql-tutorial:latest cfbc844c4ec3      323MB           0B
user@machine:~$ docker rmi cfbc844c4ec3
user@machine:~$ docker image ls -a
IMAGE                      ID           DISK USAGE CONTENT SIZE EXTRA
```

2.2 PostgreSQL Configuration

PostgreSQL creates a default database named `postgres` and you may be asked for a server, a database, a port, and credentials. The default configuration is:

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

The password is set during the installation (in case of doubt: `dbpwd1`).

There are two user types: (i) Users of the operating system and (ii) users of PostgreSQL (i.e., only in the database system). To create a database user named `dbtutorial`, we open a Linux terminal and switch to the system's `root` user (cf. Listing 10). Do not be confused that nothing is shown when you type in your password (not even asterisks `*****`).

Listing 10: `terminal` – Switch to the `root` user (mind the trailing dash).

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

Then, we switch to the system's `postgres` user (which also exists in our database) and connect to the default database `postgres` (indicated by `postgres=#`; cf. Listing 11).

Listing 11: `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 12.

Listing 12: `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 [20] with restricted privileges as shown in Listing 13.

Listing 13: `psql` – Create a new user `dbtutorial` within your database.

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

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

Once the database user `dbtutorial` exists, we switch back to the system user `dbtutorial` and reconnect **without** the intermediate switch to the `root` user. This is the proper way to use our database (it is never a good idea to work with `root` privileges all the time). To this end, we close the connection to our database using the `\q` command (cf. Listing 14) and exit `root` mode by using the `exit` command **twice.** Finally, we start using PostgreSQL with our database user (still on the default database `postgres`; cf. Listing 15).

Listing 14: `psql` – Close the connection to PostgreSQL.

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

Listing 15: `terminal` – Connect to database `postgres` using the 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=>
```

Line 5 of Listing 15 uses the `psql` command with multiple options:

1. `-U dbtutorial` tells the `psql` terminal to use the `dbtutorial` user to connect to the database (instead of PostgreSQL's default user `postgres`).
2. `-h localhost` tells the `psql` terminal that PostgreSQL is running on **our** [21] current machine (i.e., not on a remote machine).
3. `postgres` tells the `psql` terminal to connect to the database named `postgres`.

In contrast to Listing 12, Listing 15 shows our new user `dbtutorial` followed by `=>` (instead of `=#`). `=#` means `root` privileges, whereas `=>` implies **no** `root` priviliges. We can check the privileges of all users with the `\du` command (cf. Listing 16).

Importantly, we observe that our `dbtutorial` user has the privilege to create a new database (`Create DB`). Despite the default database `postgres`, we recommend to create a dedicated database for this assignment named `assignment1` (cf. Listing 17).

Remark: Database creation can take some time; please wait for it to finish.

Listing 16: `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=>
```

Listing 17: `psql` – Create a new database named `assignment1`.

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

[21] Localhost: `https://en.wikipedia.org/wiki/Localhost`

Afterwards, the `\l` command (cf. Listing 12) should print one additional database, namely `assignment1`. In order to use our 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 `\c assignment1` to connect to the new database and the `psql` terminal should then display `assignment1=>` (instead of `postgres=>`). We can also connect to our database `assignment1` using the Linux `terminal` as shown in Listing 18 (after disconnection; cf. Listing 14).

Listing 18: `terminal` – Connect to database `assignment1` using the Linux user.

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

2.3 Data Initialization

Currently, your database contains no data (i.e., is empty). The `\d` command reports that no relations (i.e., tables) have been found (cf. Listing 19).

Listing 19: `psql` – List all tables in our database `assignment1`.

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

Therefore, we have to populate the database with some data. We will use parts of the publicly available **Internet Movie Database (IMDB)** [22]. Download the data from our Nextcloud [23] and unzip it. The result is a new directory named `imdb` with three files:

- `name.basics_no_header_array_format.tsv` is a tab-separated file with actor names.
- `titles.basics_no_header_array_format.tsv` is a tab-separated file with movies.
- `create_db.sql` contains the `CREATE TABLE` (i.e., DDL) statements for this assignment.

The data for this assignment can either be downloaded using a browser (e.g., Firefox) or using the `curl` tool (cf. Listing 20). `curl` is required if you use Docker.

Listing 20: `terminal` – Data download using `curl` (`user@machine` string shortened).

```
1 ~$ 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 DDL statements. Study these statements for the structure of your tables. There are two ways to execute the statements: (1) Execute the full script from the `psql` command-line tool using `\i create_db.sql` (cf. Listing 21). (2) Copy & execute the statements on your own using the `psql` terminal.

Remark: Listings 21–23 must either be executed using the correct absolute/relative path to the file, or you must navigate into the directory of the file before connecting to the database, and only then execute the command (e.g., `\i create_db.sql`).

[22] The Internet Movie Database (IMDB): `https://www.imdb.com/interfaces/`
[23] Data for this assignment: `https://kitten.cosy.sbg.ac.at/index.php/s/gjcdKd4fMJn5Ypo`

Listing 21: `psql` – Create two tables using the given SQL script.

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

To fill the tables with data, we recommend to execute the two commands as shown in Listing 22 (one after another; mind the trailing semi-colons) in your `psql` terminal.

Remark: The file import will take some time; please wait for it to finish.

Listing 22: `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
```

On some systems, the encoding must be explicitly specified for the import to succeed. In case of problems during the import, please try the commands shown in Listing 23.

Listing 23: `psql` – Import the two plain files into our tables using UTF8 encoding.

```
ass..=> \COPY titles FROM title.basics_no_header_array_format.tsv WITH DELIMITER E'\t' ENCODING 'UTF8';
COPY 5447872
ass..=> \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 were imported into table `titles` and `names`, respectively. Then, `\d` shows two tables: `names` (of actors) and `titles` (of movies). Listing 24 shows how to view the schema of a table.

Listing 24: `psql` – List the schema information of table `names`.

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

2.4 Structured Query Language (SQL)

We provide **four** SQL queries (cf. Listings 25–28) that can be executed out of the box by entering them into the `psql` terminal (one after another; mind the trailing semi-colons):

Listing 25: `psql` – Query **Q1**.

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

Listing 26: `psql` – Query **Q2**.

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

Listing 27: `psql` – Query **Q3**.

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

Listing 28: `psql` – Query **Q4**.

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

`SELECT`, `FROM`, and `WHERE` have been covered in the lecture. The `*` in the `SELECT` clause specifies to retrieve **all** columns of the respective table.

The `WHERE` clause of query **Q2** contains two conditions that are linked using the `AND` operator. All tuples of the result must satisfy **both** conditions. For instance, a TV series 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. Specifically, we link the actor's year of birth (`names.birthYear`) with the movie's year of publication (`titles.startYear`). Moreover, we specify that only actors named "Scarlett Johansson" should be considered. Intuitively, we ask for all movies that started in the year of birth of Scarlett Johansson. The `FROM` part contains two tables that are separated by a comma. Without the (join) condition in the `WHERE` clause, the Cartesian product [24] is computed, i.e., every row of table `names` is linked with every row of table `titles`. This may result in a large result and high runtimes. Hence, we restrict the join using the given conditions in the `WHERE` clause.

Query **Q4** extends **Q2** using the `EXPLAIN` keyword. `EXPLAIN` shows the **query plan**, i.e., information about the steps PostgreSQL **plans** to execute to derive the result (without executing them). As an exercise, execute queries **Q1-3** with the `EXPLAIN` keyword and try to intuitively understand the query plans (on a conceptual level) [25].

2.5 Access the Data Using Python3

Remark: We recommend to use `psycopg2` and **not** the (relatively) new `psycopg3` module.

While the `psql` terminal provides a good starting point, a database system is typically accessed using an application. Therefore, we write a small Python3 application. Using the `psycopg2` module (or driver) for Python3 we are able to (a) establish a connection to a (local) database, (b) execute the queries and retrieve the results, and (c) close the connection to the (local) database. Your application is supposed to execute the queries **Q1**, **Q2**, and **Q4** as-is (cf. Section 2.4) and print the respective results (i.e., all tuples; output format should be human-readable).

For query **Q3**, you are required to do a small modification: **Q3** 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 [26]) or by adapting the Python3 code such that it counts the tuples. Reflect on the implications of your choice and **be prepared to answer questions** on this aspect.

[24] Cartesian product: `https://en.wikipedia.org/wiki/Cartesian_product`
[25] Using `EXPLAIN`: `https://www.postgresql.org/docs/current/using-explain.html`
[26] The `COUNT` aggregation: `https://www.postgresql.org/docs/current/functions-aggregate.html`

Template Code There are many tutorials on how to install [27] and use `psycopg2` [28]. Nonetheless, we provide a minimum Python3 template code that can be used as a starting point. The template code for this assignment is available as a separate `.py` file via the course website [29] and you can use `curl` to download the file (cf. Listing 20).

2.6 Questionnaire

The questionnaire contains questions about this assignment in a separate text file called `assignment1-questionnaire.txt`, which are discussed in the after-assignment meetings.

---

3 Submission

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

Code Please submit a Python3 file (`assignment1.py`) that contains the full code for this assignment. The code **must** print the results of **all** four queries **Q1-4** (one after another) when executed **as submitted**. We will not debug your code, e.g., change a variable to make it work. Therefore, please double-check that your Python3 code works as expected.

Questionnaire Answer the questions directly in the file `assignment1-questionnaire.txt`. If you prefer to use a different application to answer the questions (e.g., Microsoft Word and the likes), you must use one of the following formats: `.txt`, `.pdf`, `.odt`, or `.docx`.

---

4 Supplementary Material

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

- 🔗 PostgreSQL: `https://www.postgresql.org/`
- 🔗 Download PostgreSQL: `https://www.postgresql.org/download/`
- 🔗 Full PostgreSQL documentation: `https://www.postgresql.org/docs/current/`
- Other resources on the installation and the usage of PostgreSQL:
  - ▶ Some "Getting Started" guide: `https://www.youtube.com/watch?v=BLH3s5eTL4Y`
  - 🔗 Installation: `https://www.postgresqltutorial.com/install-postgresql/`
  - 🔗 PostgreSQL on Windows: `https://www.postgresql.org/download/windows/`

[27] `psycopg2` installation: `https://www.psycopg.org/` or `https://pypi.org/project/psycopg2/`
[28] `psycopg2` tutorial: `https://wiki.postgresql.org/wiki/Psycopg2_Tutorial`
[29] Code: `https://dbresearch.uni-salzburg.at/teaching/2026ss/dim/assignment1-template.py`

- 🔗 Using `EXPLAIN`: `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/`
  - 🔗 Official website: `https://www.psycopg.org/install`
  - 🔗 The Python Package Index: `https://pypi.org/project/psycopg2/`
  - 🔗 Some `psycopg2` tutorial: `https://wiki.postgresql.org/wiki/Psycopg2_Tutorial`
  - ▶ Some "Getting Started" guide: `https://www.youtube.com/watch?v=2PDkXviEMD0`

---

5 Grading

For 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 10 points.

Code The code contributes at most 5 points and is evaluated based on the following criteria (if the code is executed as submitted; disregarding the credentials):

Max. Points	Criterion
1.25	The result of Q1 is correctly printed to the command line.
1.25	The result of Q2 is correctly printed to the command line.
1.25	The result of modified Q3 is correctly printed to the command line.
1.25	The result of Q4 is correctly printed to the command line.
5	

Questionnaire The questionnaire contributes at most 5 points and is evaluated based on the following criteria:

Max. Points	Criterion
1.25	Correctness of answer A1.
0.625 + 0.625	Correctness of answer A2.
1.25	Correctness of answer A3.
1.25	Correctness of answer A4.
5	

After-Assignment Meeting The discussion in the after-assignment meeting contributes at most 10 points, which are awarded based on assignment-specific questions.
