FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Databases – practical course
Preparation

Prof. Dr. rer. nat. Alexander Voß
Inform-Professur

FH Aachen
University of applied Sciences

V2024

Intro

Virtualisation
&
Docker

DataGrip
IDE

Sample
Data

Intro
A few words to begin with

What is it all about?

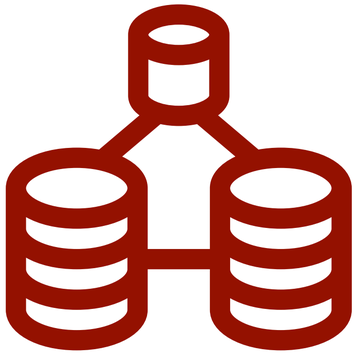
Of course, for the lectures and exercises we will need a real database, or more precisely, a **"database management system (DBMS)"**.

This will be used to **illustrate the concepts** and also to provide the data for the SQL lab and our applications.

This short tutorial will show you how to

- **set this up and**
- **import the necessary data**

so that we can work with it in the lab.



A first example

A common type of DBMS is called a **relational** DBMS (RDBMS). Here, data is organised in **tables** that have a relationship to each other - hence "relational".

id	product	category	price
4	Fish Fingers	1	1.99
5	Pasta Pan	1	3.29
6	Carrots	2	0.39
7	Onions	2	0.29
8	Bananas	2	1.29
9	Garlic	2	0.98
10	Pork Sausage	3	1.99
11	Meatballs	3	2.35
12	Milk	4	0.79
13	Quark	4	0.99
14	Yoghurt	4	0.98
15	Cola light	5	1.50

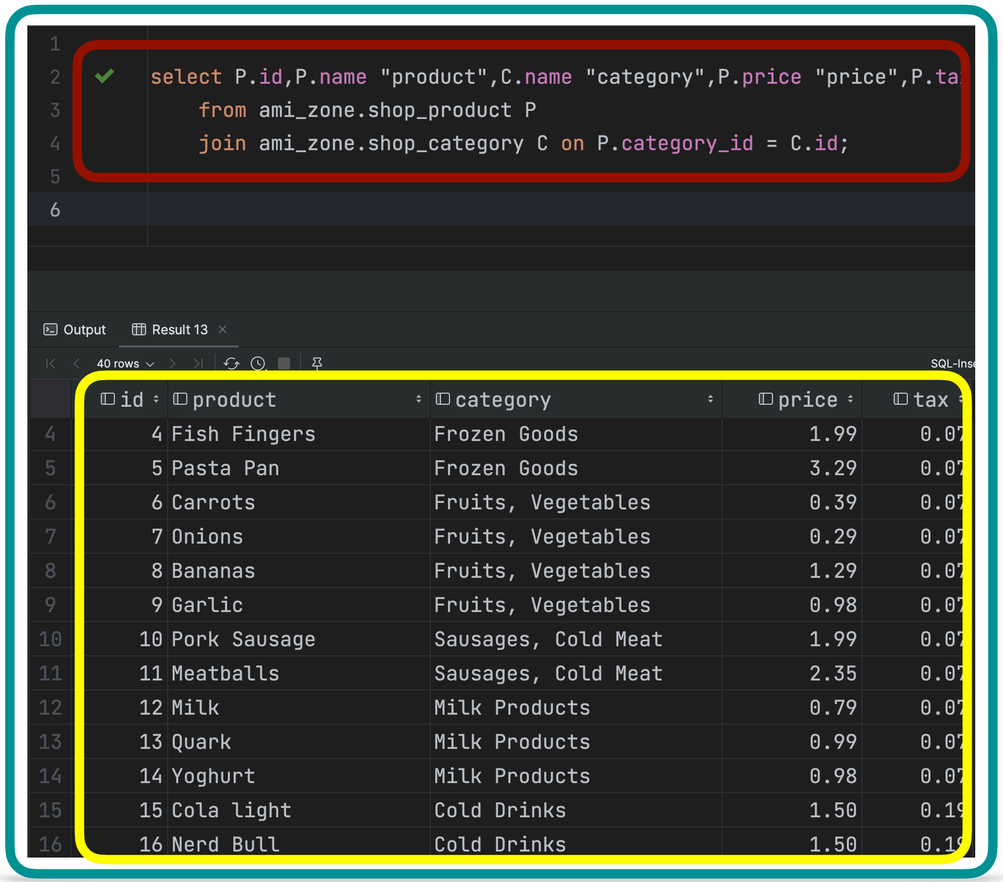
category	name
1	Frozen Goods
2	Fruits, Vegetables
3	Sausages, Cold Meat
4	Milk Products
5	Cold Drinks
6	Barbecue Products
7	Snacks
8	Hot Drinks
9	Convenience Foods
10	Baked Goods
11	Magazines
12	Services

Related product data from the ami_zone database

Although there are other systems with different organisational structures, this structured view of the data is sufficient to get you started.

We need…

a **language** in which to talk to the database.



```
select P.id,P.name "product",C.name "category",P.price "price",P.ta
    from ami_zone.shop_product P
    join ami_zone.shop_category C on P.category_id = C.id;
```

Output | Result 13

40 rows

	id	product	category	price	tax
4	4	Fish Fingers	Frozen Goods	1.99	0.07
5	5	Pasta Pan	Frozen Goods	3.29	0.07
6	6	Carrots	Fruits, Vegetables	0.39	0.07
7	7	Onions	Fruits, Vegetables	0.29	0.07
8	8	Bananas	Fruits, Vegetables	1.29	0.07
9	9	Garlic	Fruits, Vegetables	0.98	0.07
10	10	Pork Sausage	Sausages, Cold Meat	1.99	0.07
11	11	Meatballs	Sausages, Cold Meat	2.35	0.07
12	12	Milk	Milk Products	0.79	0.07
13	13	Quark	Milk Products	0.99	0.07
14	14	Yoghurt	Milk Products	0.98	0.07
15	15	Cola light	Cold Drinks	1.50	0.19
16	16	Nerd Bull	Cold Drinks	1.50	0.19

a **program** for handling commands, results and the database itself.

a **database** that provides the data.

We teach SQL, …

```
1
2   select P.id,P.name "product",C.name "category",P.price "price",P.ta
3       from ami_zone.shop_product P
4       join ami_zone.shop_category C on P.category_id = C.id;
5
```

The database language **SQL** is a major component of the exercises.

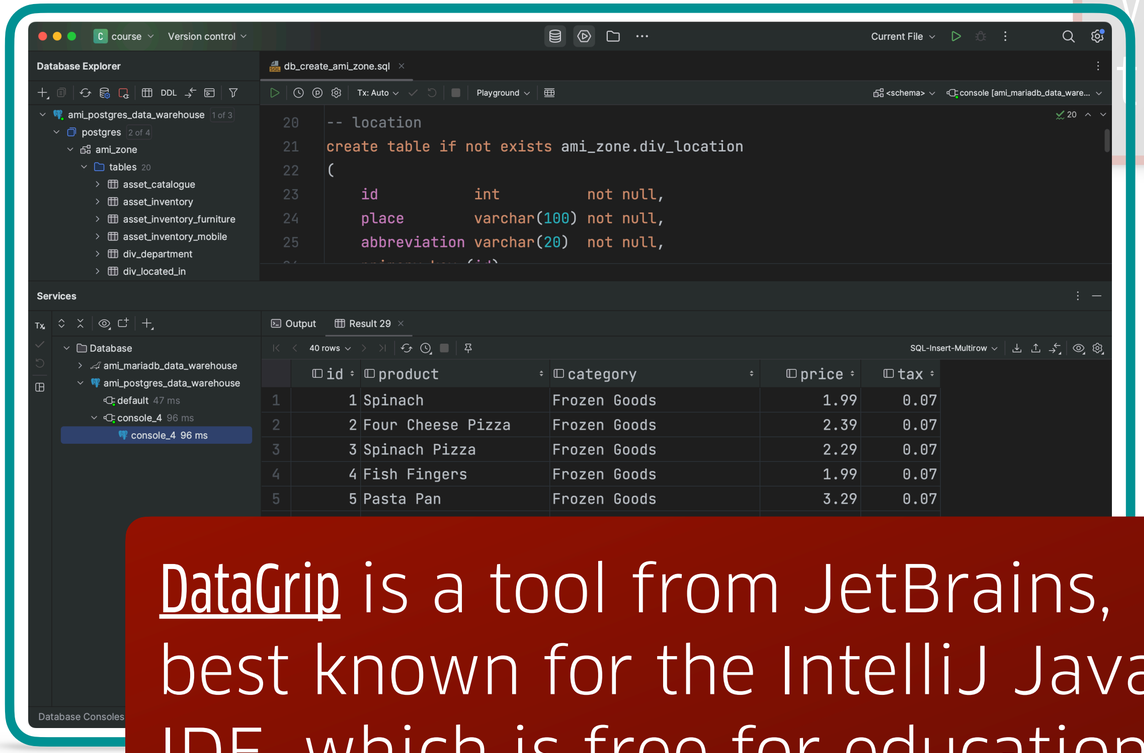
a **language** in which to talk to the database.

a program for handling commands, results and the database itself.

a database that provides the data.

We suggest DataGrip, …

a language in which to talk to the database.



a **program** for handling commands, results and the database itself.

a database that provides the data.

DataGrip is a tool from JetBrains, best known for the IntelliJ Java IDE, which is free for educational purposes. In principle, however, you can use **any other tool** or IDE that suits your DBMS.

We work with MariaDB, …

a language in which to talk to the database.

There are many DBMS, including free ones. We recommend

- PostgreSQL but also
- MariaDB.

Both are good for our purposes.

a program for handling commands, results and the database itself.

6	Carrots	Fruits, Vegetables	0.39	0.07
7	Onions	Fruits, Vegetables	0.29	0.07
8	Bananas	Fruits, Vegetables	1.29	0.07
9	Garlic	Fruits, Vegetables	0.98	0.07
10	Pork Sausage	Sausages, Cold Meat	1.99	0.07
11	Meatballs	Sausages, Cold Meat	2.35	0.07
12	Milk	Milk Products	0.79	0.07
13	Quark	Milk Products	0.99	0.07
14	Yoghurt	Milk Products	0.98	0.07
15	Cola light	Cold Drinks	1.50	0.19
16	Nerd Bull	Cold Drinks	1.50	0.19

a database that provides the data.

Some Remarks on the DBMS

- The first thing to say is that although we are mainly working with a specific relational DBMS (PostgreSQL), the **concepts** presented in the Databases module are transferable to other relational DBMSs as well as to so-called non-relational, e.g. NoSQL, databases.
- PostgreSQL is widely regarded as the most standards-compliant and compatible open-source RDBMS, especially when it comes to SQL:2011/2016.
- MariaDB would be an alternative, but we have now decided on PostgreSQL.
- Note that the the **SQL dialects** of each DBMS are different. Although the general structure of an SQL command can often be transferred to another DBMS, details such as quotation marks or upper and lower case may behave differently. If you want to use the material with another DBMS, you may need to make small adjustments.
- **Unfortunately, it is not possible to maintain the resources for all popular DBMS.**

What is to come

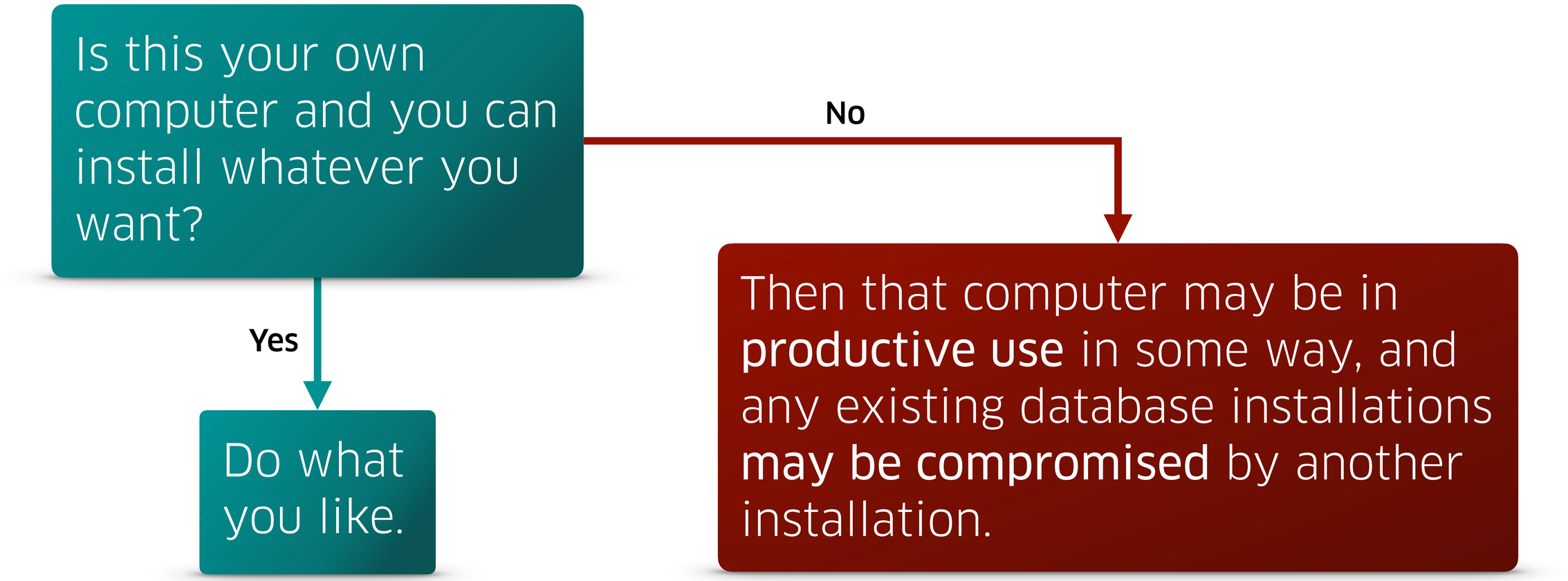
- The next step is to install **your own DBMS** while avoiding side effects on your own computer. We do this by using **virtualisation**.
- We use Podman or Docker for virtualization (with their respective UIs). You probably already have one installed – if not, please install one.

- To install, run and stop the **database services** (aka. the DBMS), we use Compose.
- We use DataGrip by JetBrains to access the DBMS. The installation is usually not critical as it is just an application. Feel free to use your favourite tool if you like, such as pgAdmin.

- You can find sample data and also SQL scripts for the SQL lab in GitHub. We assume that you have some knowledge of versioning and git. If not, let us know and we will help you.

Virtualisation and Docker
Fighting side effects

One question first



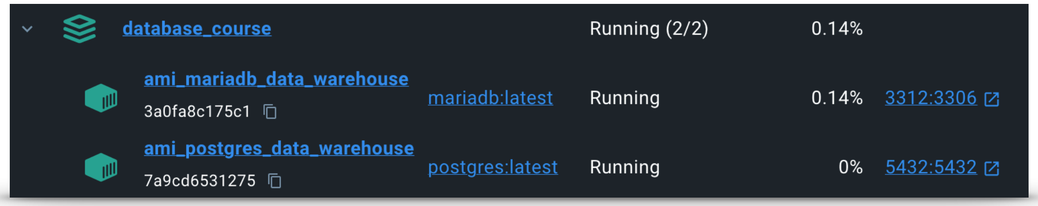
Docker

One way to install a DBMS with as few side effects as possible is to **virtualise** the database server and install it in a **Container**.

We will now **show you** how to install a system such as MariaDB or PostgreSQL in a Docker container and import our sample data.

This is done using **Compose**.

e.g.



Part of the **Docker Desktop** (free for educational use).

What does **Compose** do exactly?

*"**Compose** is a tool for defining and running multi-container **Docker** applications. With **Compose**, you use a **YAML** file to configure your application's services. Then, with a single command, you create and start all the services from your configuration."*

Docker Docs

Note: Podman with Compose is supported and works for the majority of use cases, though some advanced features may not be fully compatible.

So we provide a **compose.yml** with entries for MariaDB and PostgreSQL, and if the Docker or Podman installation is correct, that's it. Everything else, i.e. downloading the DBMS containers and launching the services, is done by Compose.

compose.yml Services

```yaml
mariadb:
  image: mariadb:latest
  container_name: ami_mariadb_data_warehouse
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_ROOT_HOST: "%"
  ports:
    - "3312:3306"
  volumes:
    - ../mariadb_data:/var/lib/mysql
  restart: always
```

```yaml
postgres:
  image: postgres:latest
  container_name: ami_postgres_data_warehouse
  environment:
    POSTGRES_USER: root
    POSTGRES_PASSWORD: root
  ports:
    - "5438:5432"
  volumes:
    - ../postgres_data:/var/lib/postgresql/data/
  restart: always
```

services (mariadb and postgres) in compose.yml

Comments follow…

Some Remarks on Compose and compose.yml

- Since a container always starts with the same initial state (feature!), no data that has been changed will survive - which is not very sexy for a DBMS. Therefore, the container is usually given special locations, i.e. "shared folders", where the data can be stored. These are the volumes-**mappings** in compose.yml.
  - These directories are automatically created on the host machine.
- In compose.yml you can also see ports – a port mapping. Depending on what is installed on your machine, you will need to **adjust these ports**.
  - 3306 and 5432 (right sides) are the default ports for these two DBMSs running in the container. They map to the ports on the host (left sides), which are here 3312 and 5438 respectively. You can choose other ports or stick with these. We need these ports to connect to the DBMS.
- Of course, you will need to **replace the passwords** you see.

Commands to copy for your convenience, but first read more about them

- git clone:

  % git clone https://github.com/codebasedlearning/database_course.git

- start a database service (Docker)

  % docker compose -f docker/compose.yml up postgres -d

  % docker compose -f docker/compose.yml up mariadb -d

- stop a service (Docker)

  % docker compose -f docker/compose.yml down postgres

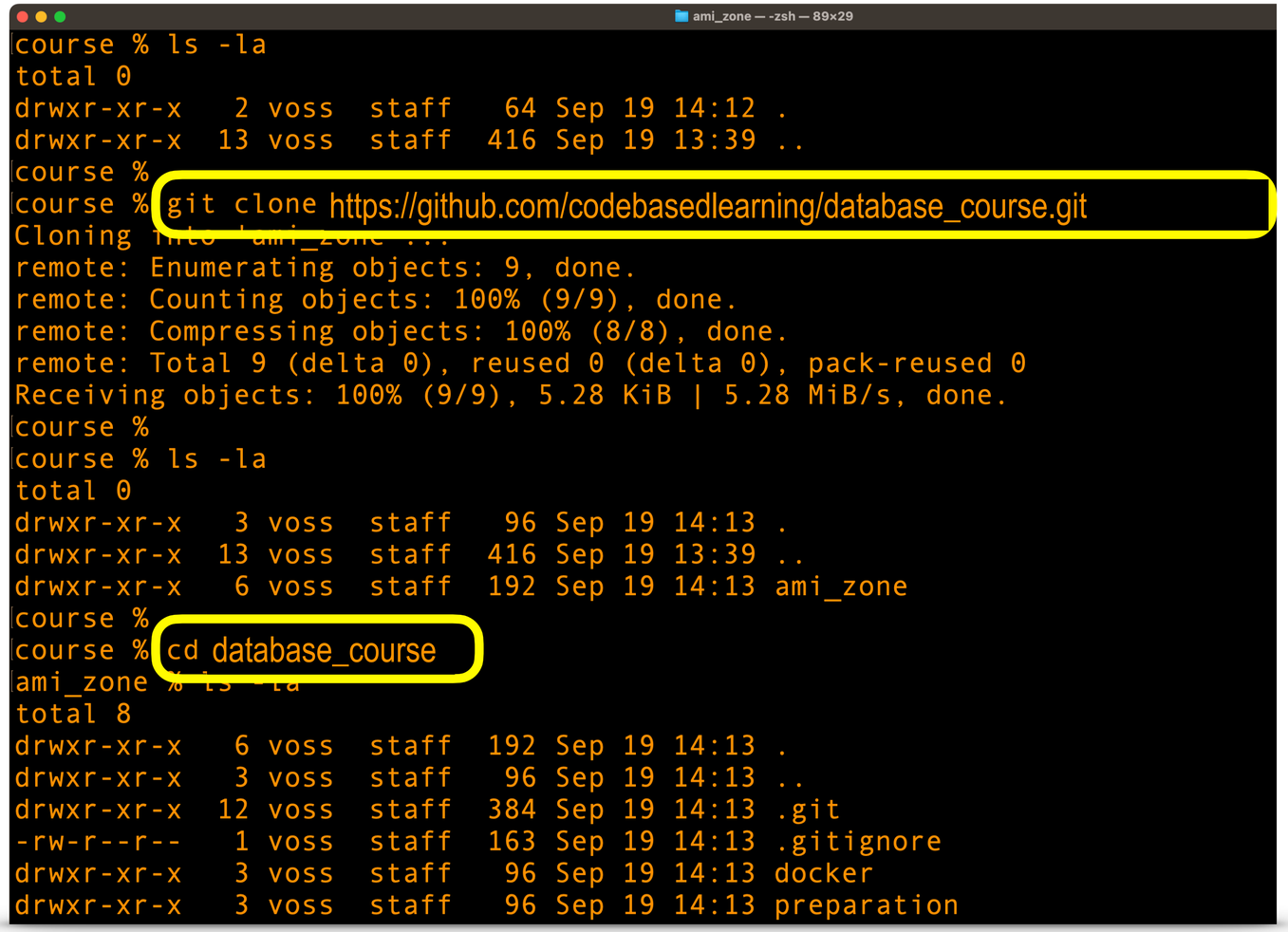
  % docker compose -f docker/compose.yml down mariadb

Commands for the copy operations in the following text (without the % prompt)

Using Compose with Podman works, but it's slightly different. If you are familiar with Podman, you will already know how to start a container.

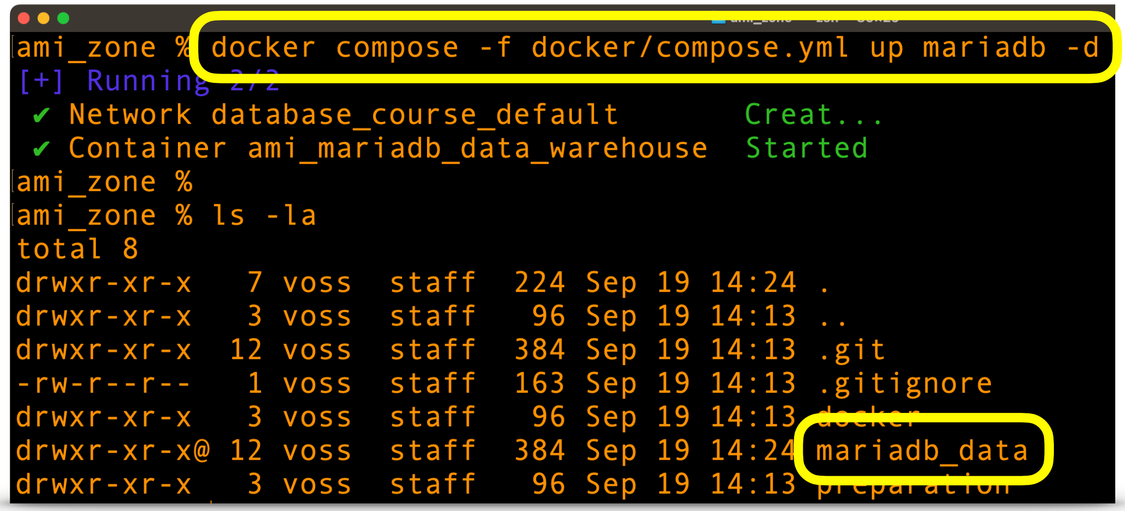
Installation Recipe Part I

- Start a terminal.
- Create or enter an empty folder, here course.
- Clone repository database_course from GitHub.
- Enter new folder database_course.

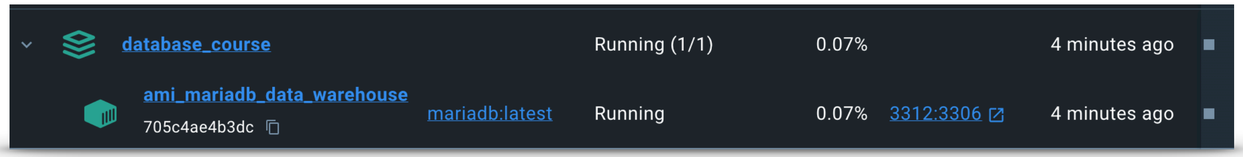


Installation Recipe Part II

- Make sure that the configured port is not being used, e.g. by another container.
- Start and stop the DBMS via `compose` (see before), you can also find the commands in `compose.yml`.
- Confirm that the container is **running**.
- You will also see that the data directory has been created.



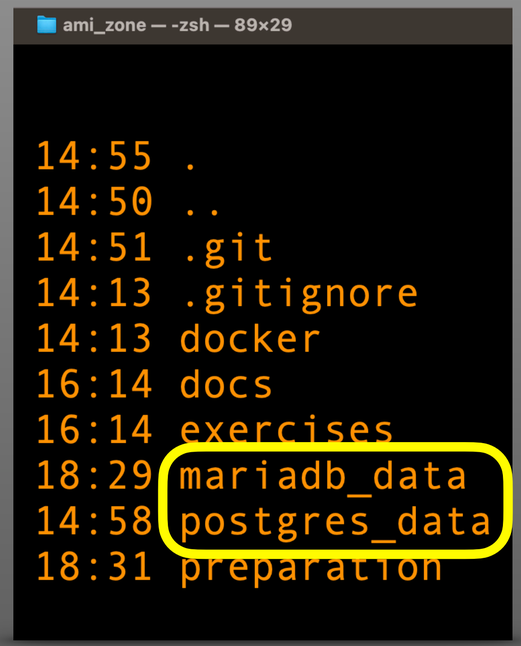
Here, MariaDB was started.



What has been, what is to come

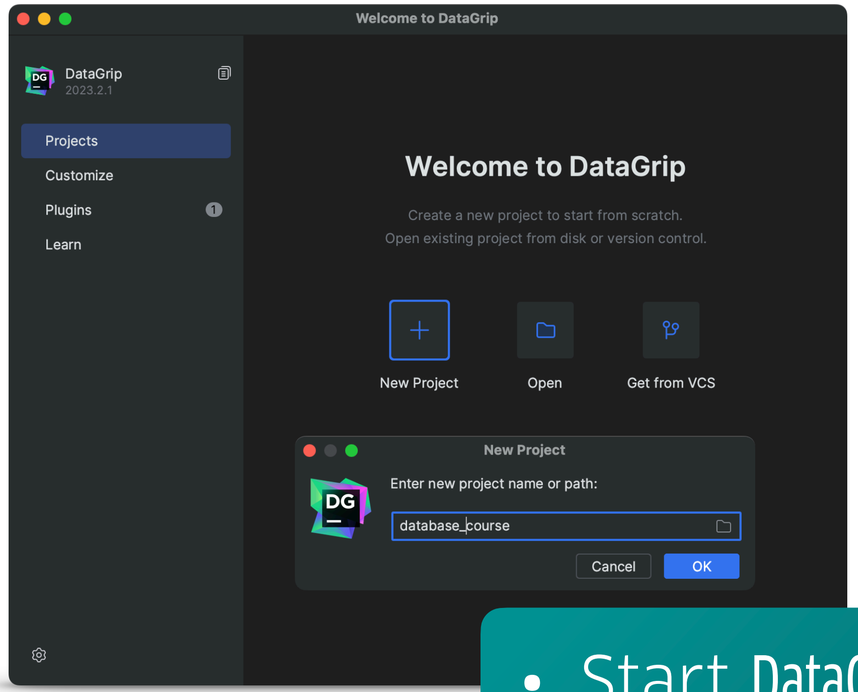
- If all went well, you should now have
  - a **running installation** of a DBMS, and
  - a **data directory** where the data resides and will still be there when we restart.
- We now **need to connect** to the DBMS so that we can
  - create data, schemes and tables, and
  - work with this data.

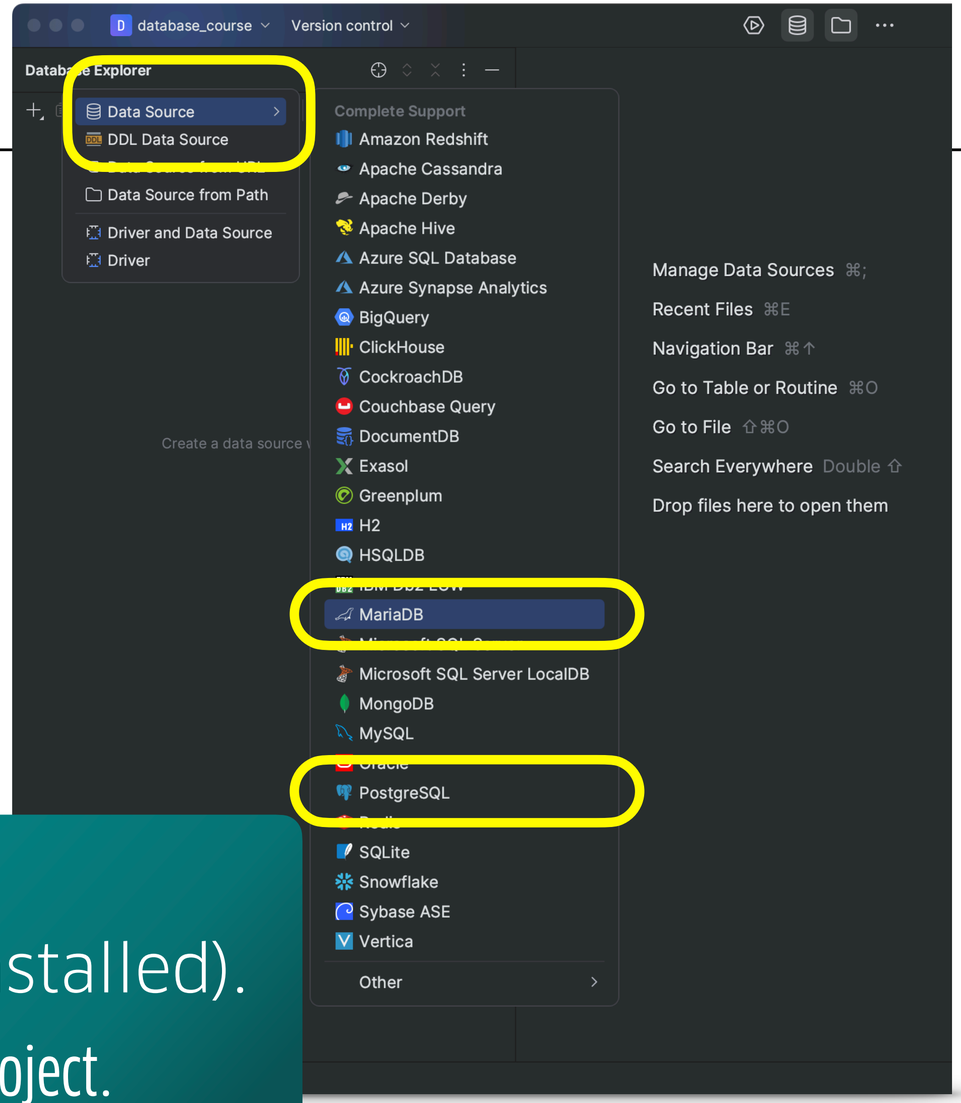
  We do this using DataGrip.



DataGrip
IDE for Databases

Connection Recipe Part I

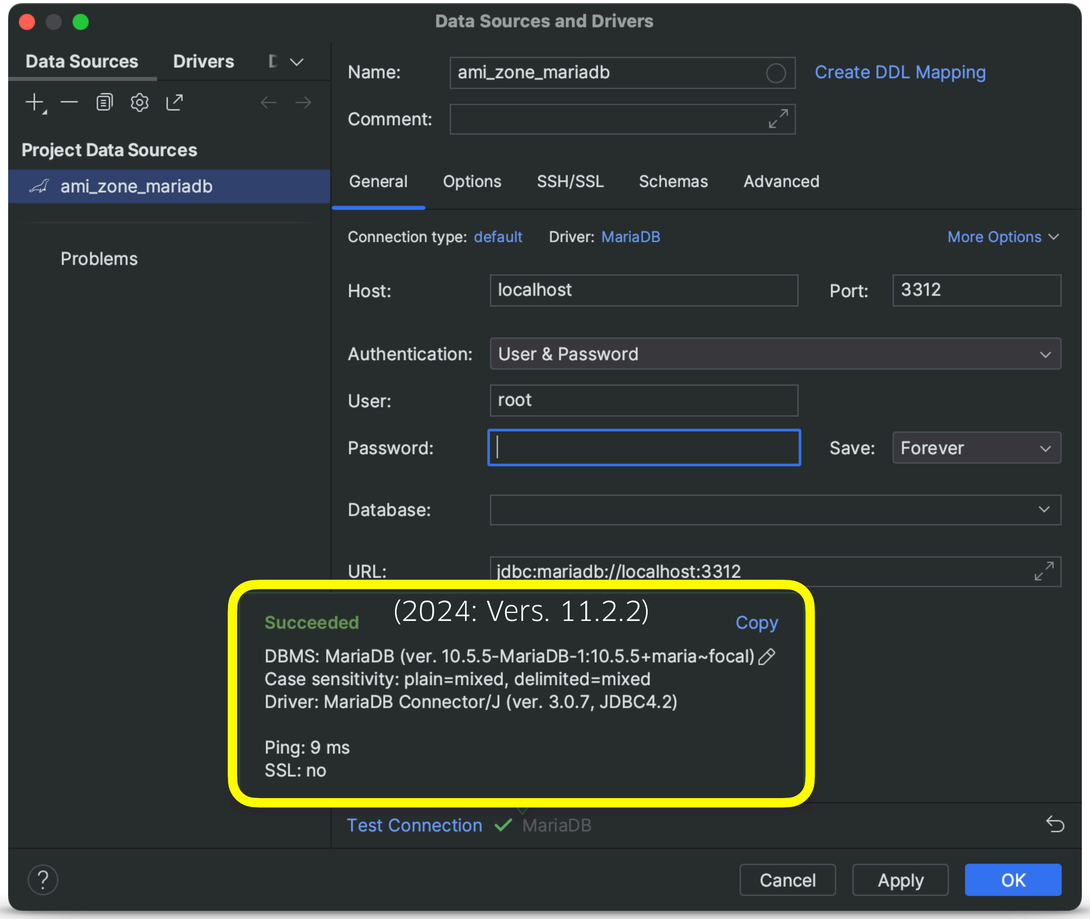




- Start DataGrip (if already installed).
- Choose New Project.
- Create new Data Source.

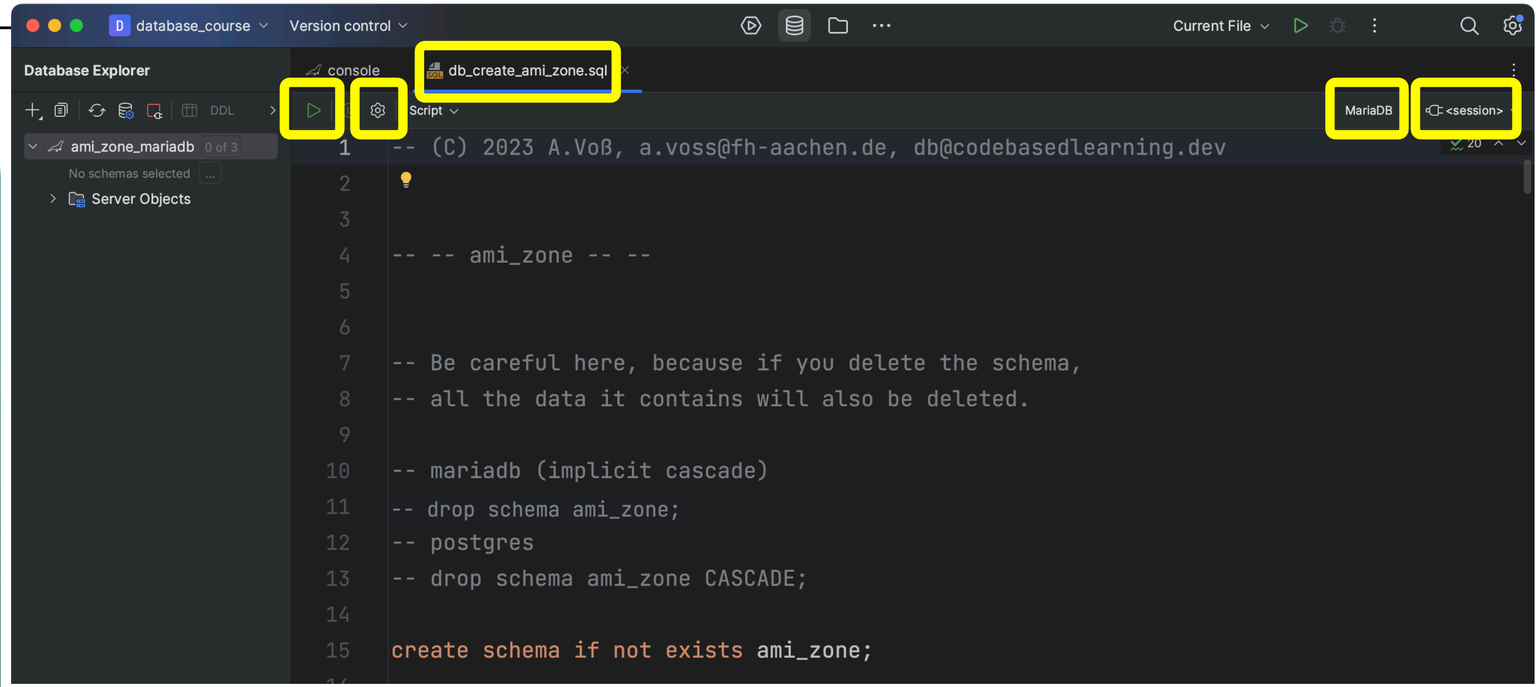
Connection Recipe Part II

- These are the project properties.
- Enter Name, Port, User, Password.
- Test Connection (maybe download missing drivers before)

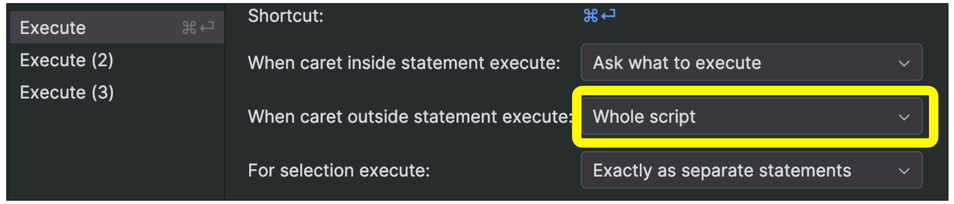


Installation Recipe Part III

- Drag and Drop db_create_ami_zone.sql from preparation into DataGrip.
- Select Dialect.
- Choose Console / New Session; or (2024) Settings
- Place the cursor on a blank line and press Execute (and hope).

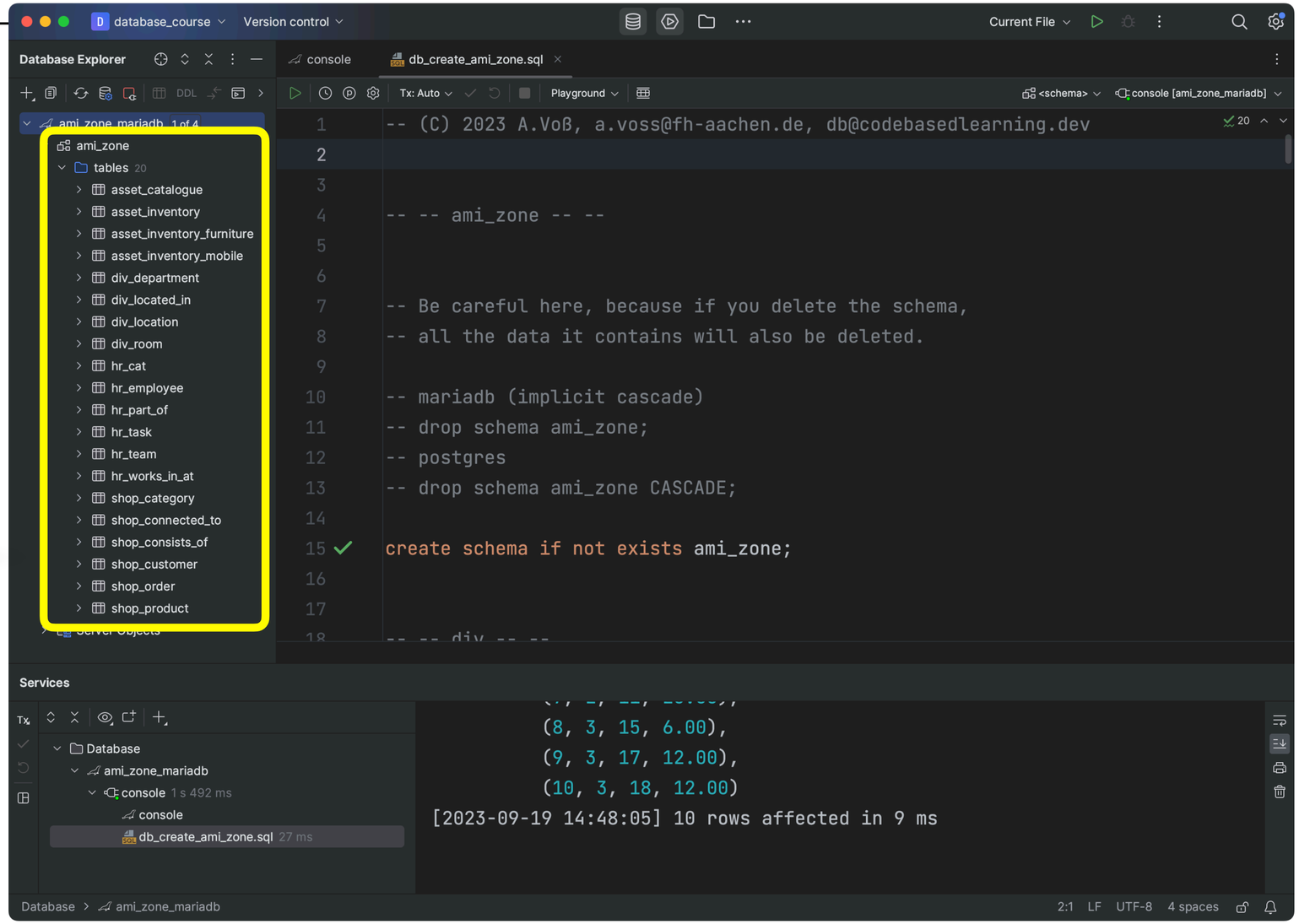


2024: Settings



Installation Recipe Part IV

Confirm that all tables have been created.

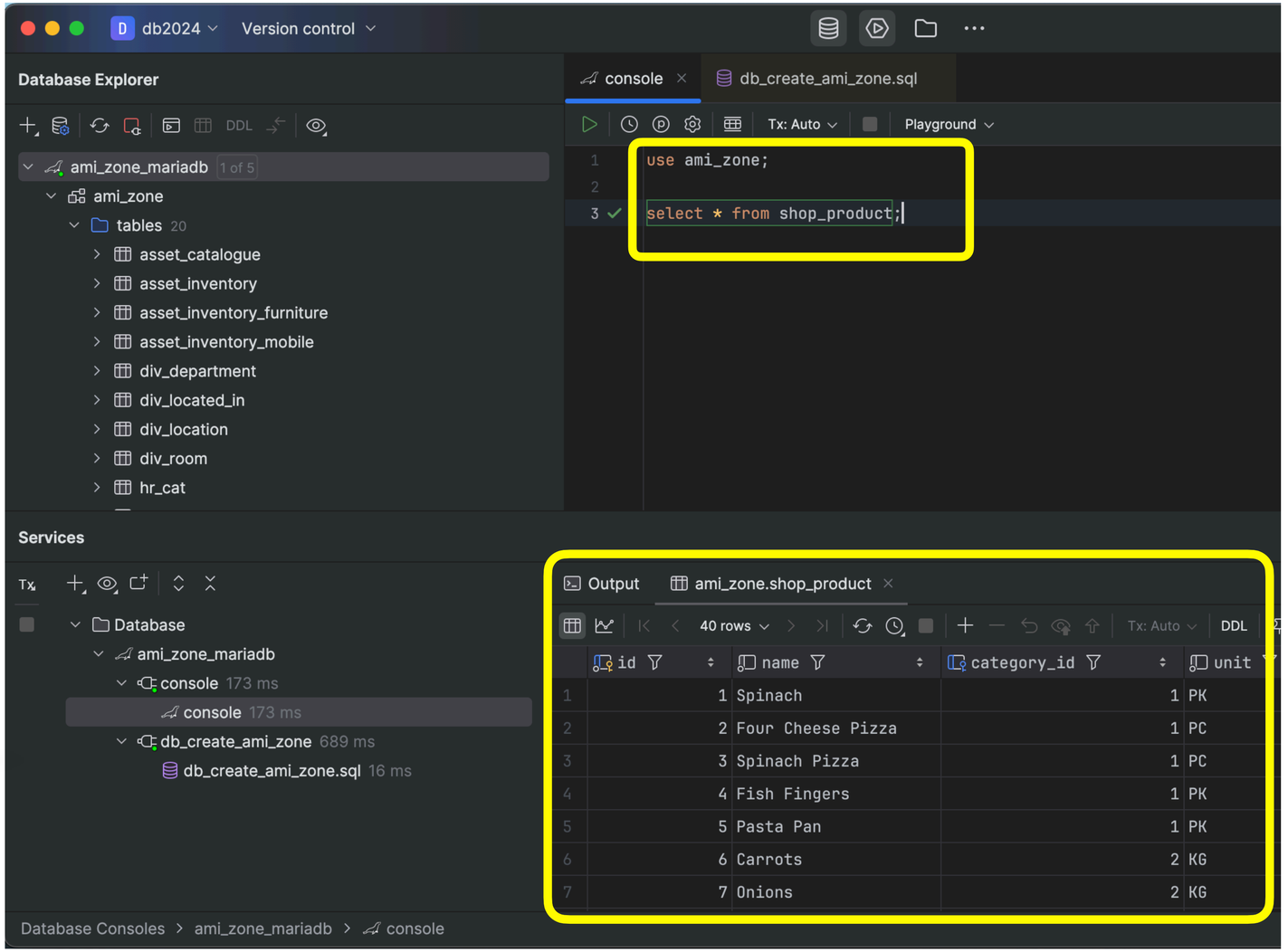


Installation Recipe Part V

Check within console:

- select * from ami_zone.shop_product;

Note: the command use (see screenshot) is MariaDB-specific

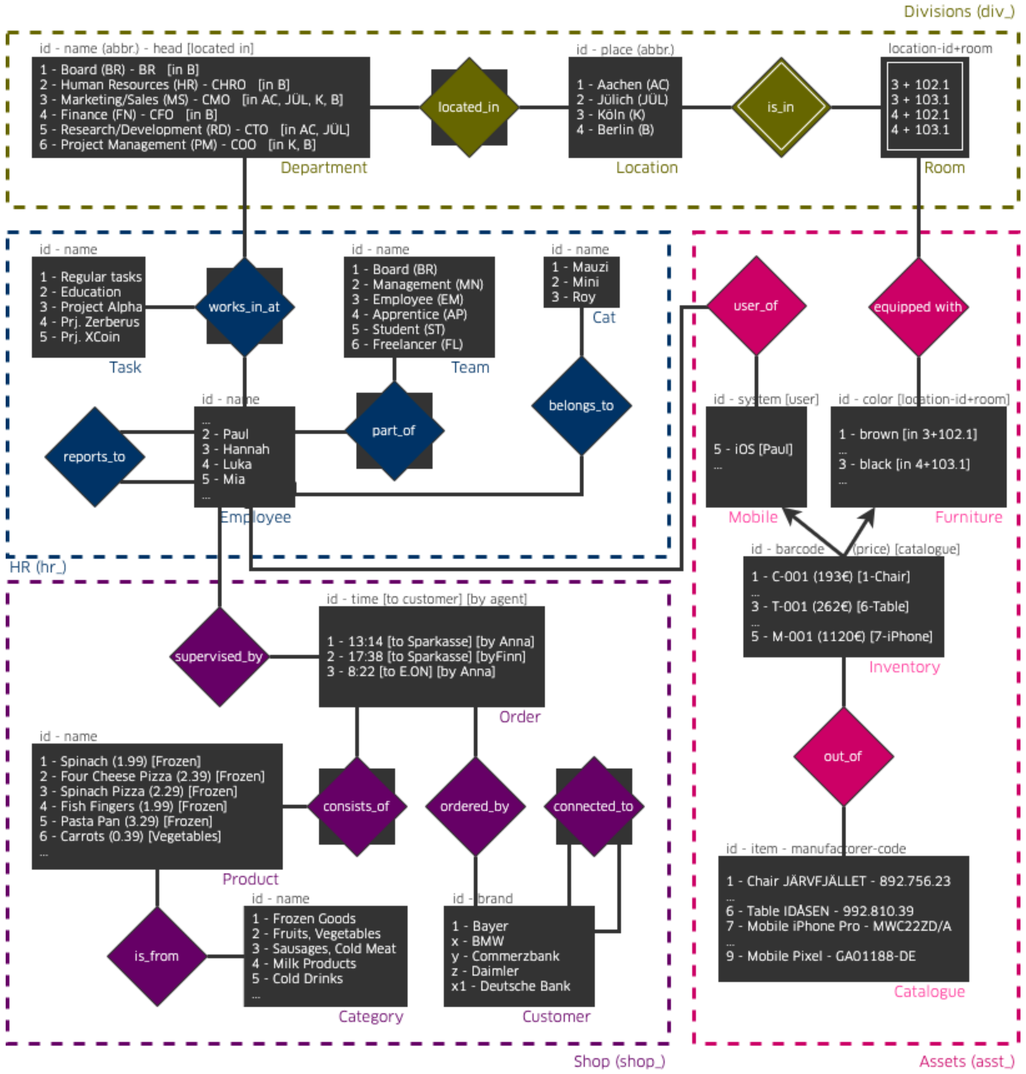


Concluding remarks on the steps taken

- The **Dialect** helps with syntax support because SQL commands (unfortunately) differ in detail.
- The execution of a single command or script takes place in a **Session** associated with a particular DBMS. You can also change this session if you have more than one DBMS connected.
- If you want to repeat the whole creation process, you can delete the so-called **schema** (or database), a kind of namespace. This will also delete all the tables.
  The command (**drop schema**) can be used in MariaDB for this, remove the comment (--) according to your DBMS.
- If any **code is selected**, the **left green arrow**  at the top left of the code will execute that selection. If the cursor is **on a blank line**, the **script will be executed**. Otherwise, the **current command is executed,** starting from the current line.

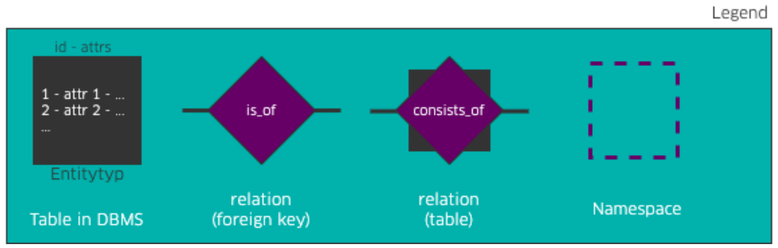
Sample Data

Main sample database

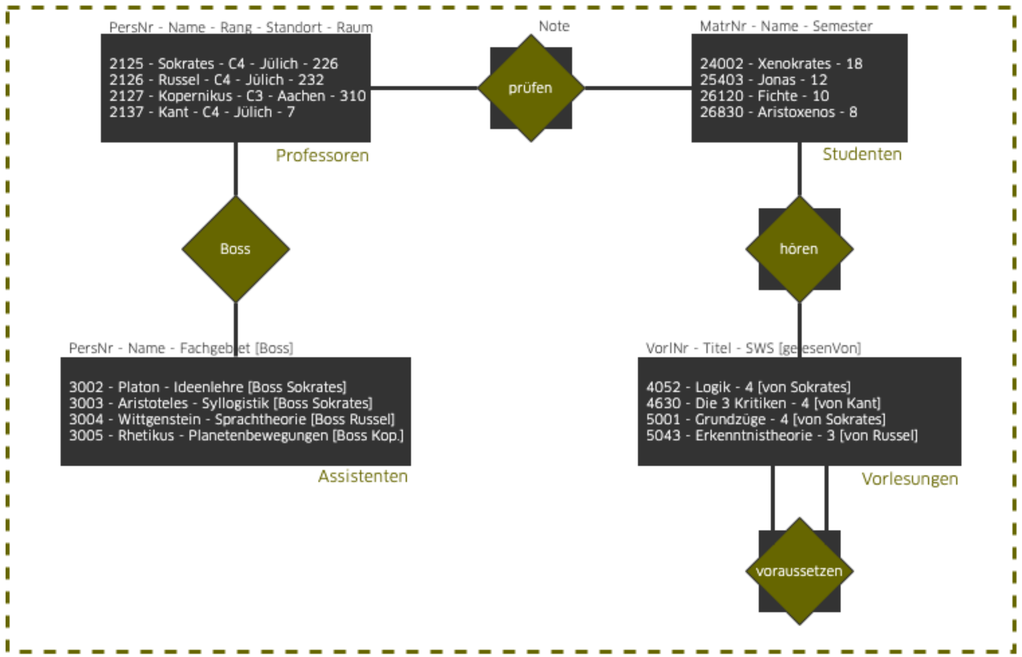


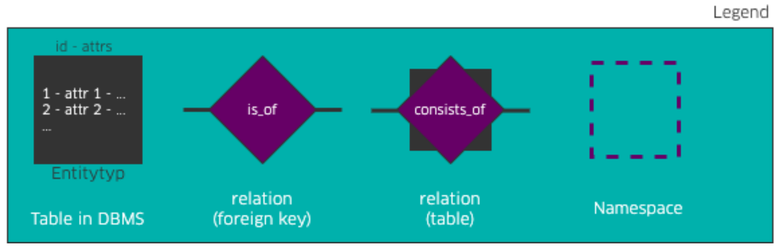
AMI_ZONE

An AMI company.



Sample database, used in exams





AMI_KEMPER

Datenbanksysteme
(Kemper, Eickler)