

**Databases Project**

Business Informatics – Data Science

Students: Philipp Becht (9443009)

Pascal Schmidt (8133405)

Simon Wrigg (5874903)

Course: WWI19DSB

Lecturer: Petko Rutesic

**Table of Contents**

[Project Description 1](#_Toc72136907)

[Preliminary Specification 1](#_Toc72136908)

[ER- Diagram 3](#_Toc72136909)

[Create Tables (Examples) 4](#_Toc72136910)

[Insert Values (Examples) 4](#_Toc72136911)

[SQL Queries 5](#_Toc72136912)

[Transaction 7](#_Toc72136913)

[Stored Procedure 7](#_Toc72136914)

[Database Normalization 8](#_Toc72136915)

[Application 9](#_Toc72136916)

# Project Description

The goal of the project was to foster the basic skills to design and implement a relational database. First, we had to prepare a preliminary specification, which contains the general setup of our database. The group decided to design a database for a gym-company, which keeps track of their members, trainers, locations, and more.

Based on the preliminary specification, we created an ER-Diagram, that represents all relations between the entities as well as important indications, such as the foreign key indication.

# Preliminary Specification

The gym keeps track of each members first and last name, birthdate, email-address, sex and the chosen subscription. Furthermore, it contains the start date and end date of contract of each member, as well as the gym-location and, if chosen, also the ID of a personal trainer. We allocate each member an unique Member\_id.

For the gym’s trainers, the database manages similarly to the member table some personal data such as first name, last name, birthdate, sex and email-address. We allocate each trainer an unique Trainer\_id. Some trainers are responsible for different courses with a Course\_id. Each trainer is based in a specific gym location with an unique Base\_id.

Each offered course has a specific Course\_id. The database contains information about the course-name, when the course takes place and its duration. Also, every course takes place in a specific room marked with a Room\_id.

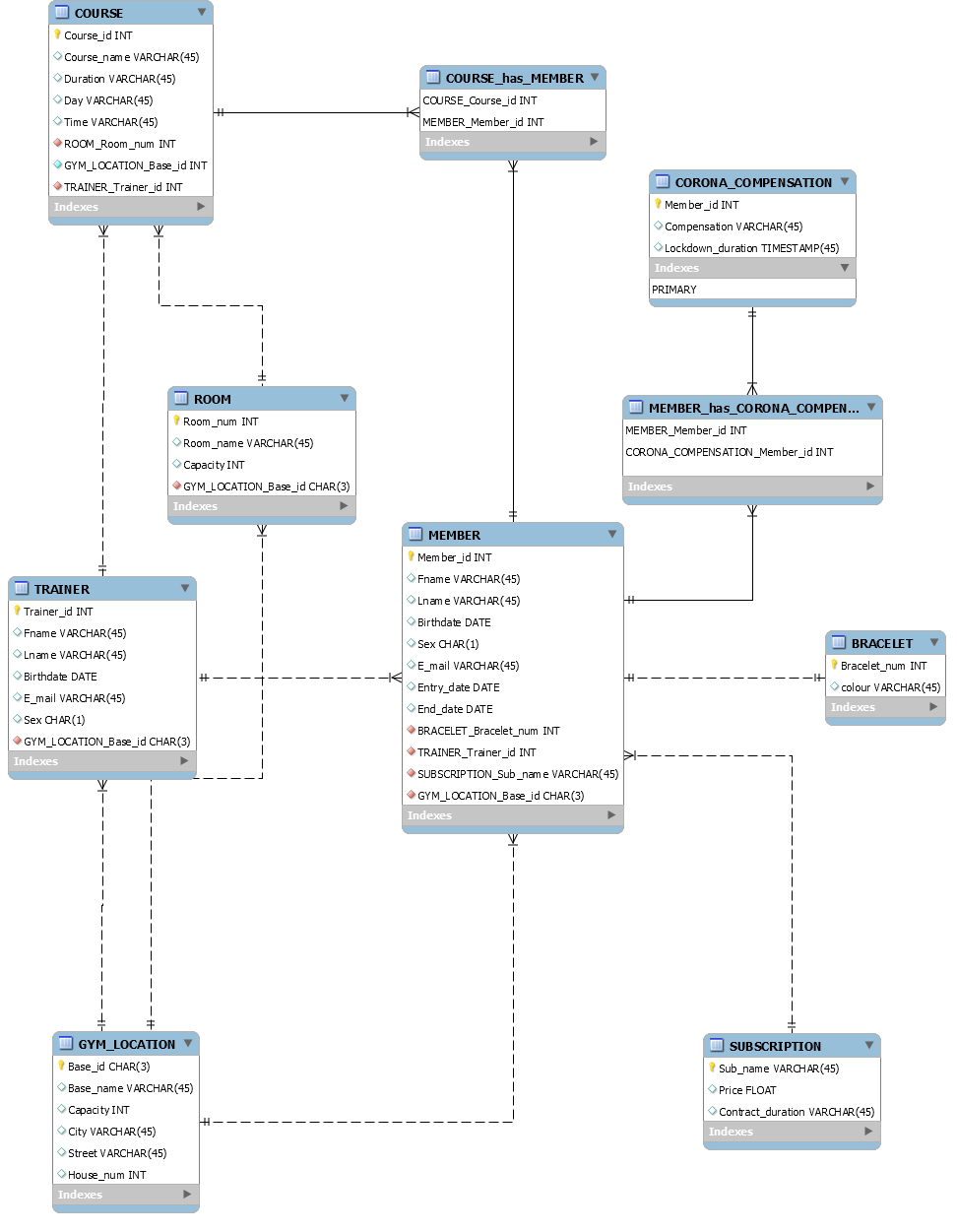
In the fitness center, the members can choose between different subscriptions. Therefore, a table keeps track of the subscription name, its monthly price as well as the duration of the contract (e.g. 1 year). Depending on the certain subscription, members have access to specific rooms in the gyms (e.g. spa). Therefore, the assigned Room\_ids are stored in the subscription table.

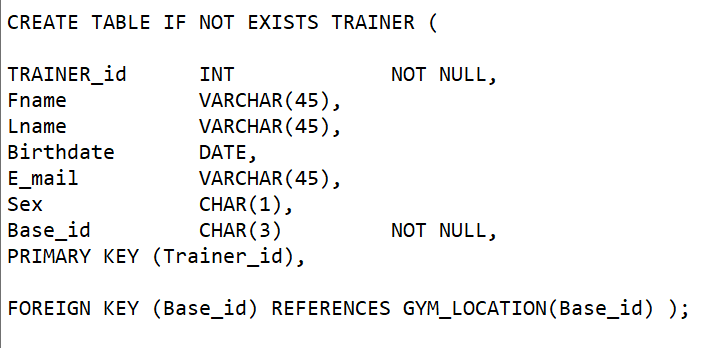
As already mentioned, each room has a unique Room\_id as well as a room name. In the gyms are different rooms, some rooms are used for courses, others for e.g. spa or sauna. Every room has an indication for its capacity, e.g. a maximum amount of 10 people.

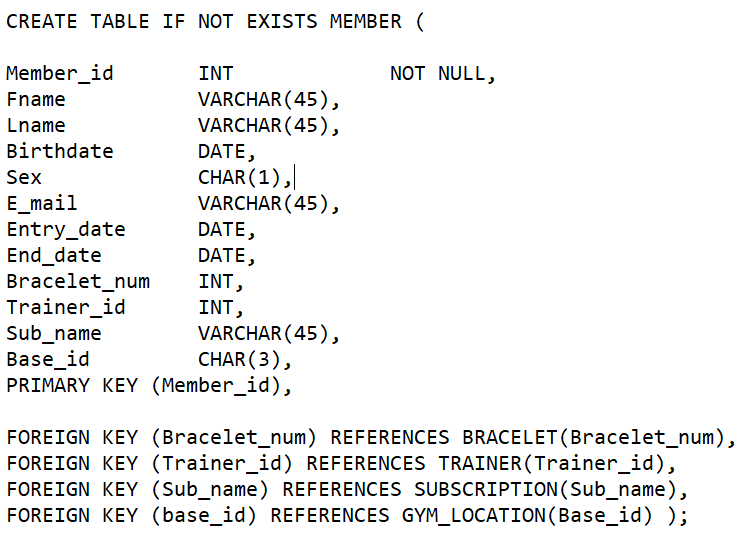
Every gym of the company chain is stored in a location-table, where the Base\_id, the gym’s name, its address and also its capacity are represented.

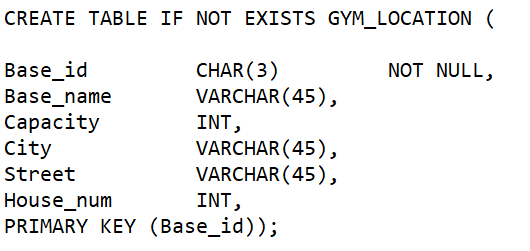
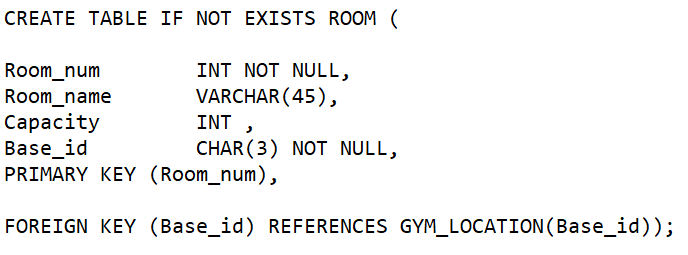
Due to the Corona Crisis, all the gyms are currently closed. That’s the reason why the gym is offering compensations to the members. Therefore, the compensation (e.g. a protein shake flatrate or a guest card) are stored in the compensation-table. For those members who have chosen a compensation, the Member\_id is linked to the compensation. Moreover, the table contains the period of time for which the compensation is chosen.

# ER- Diagram



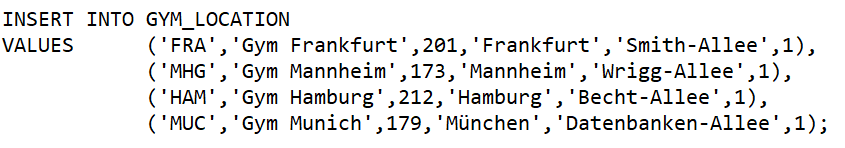
Create Tables (Examples) **– All commands are attached to this documentation.**

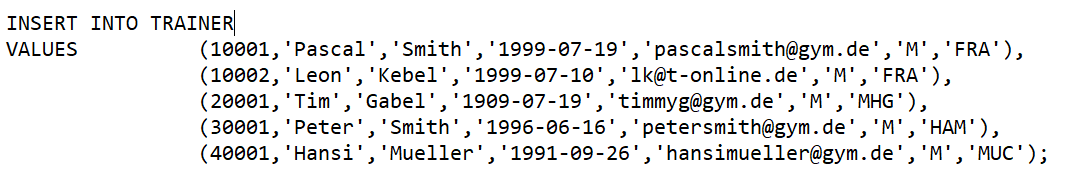


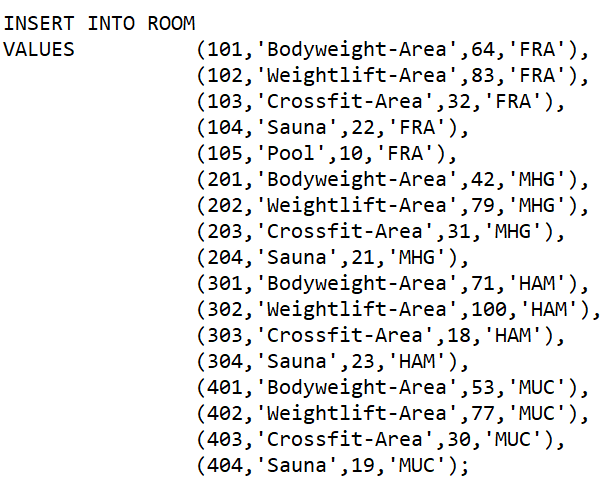


Insert Values (Examples) **– All commands are attached to this documentation.**

As we have created the database only once, we only had to insert the values at one stage. This is the reason why we didn´t write “INSERT IF NOT EXISTS”.





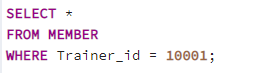


# SQL Queries

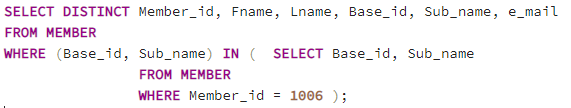
1. Retrieve the first and last name of those members, who have a trainer



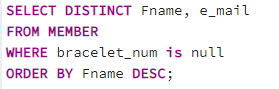
1. Retrieve all information of the members, who have the same trainer (identified by ID: 10001)



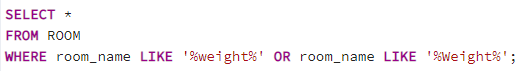
1. Retrieve the Member-ID, first & last name, base-ID, subscription type and e-mail address of those members, who work out at the same location and who have the same subscription as Member 1006



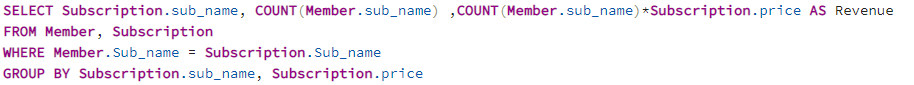
1. Retrieve the first name and the e-mail address of those customers who do not have a bracelet yet and order them descending.



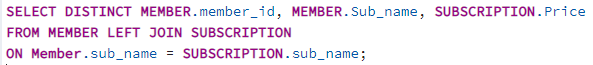
1. Retrieve the room number, the room name, the courses´ capacity and the base, in which the courses take place that contain the word “weight”.



1. Provide an overview of the gyms´ revenues for each subscription type.

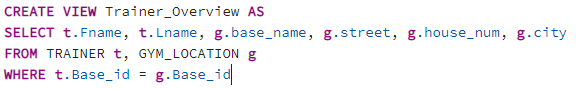


1. Retrieve the Member-ID, their subscription and the monthly fare of each customer.



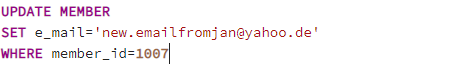
**View**

Create a view, to get an overview of all the trainers and their location. Provide some detailed information of their base.



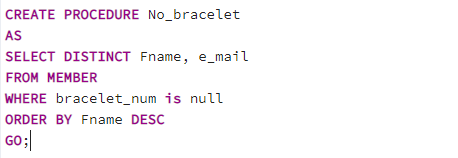
# Transaction

Update the e-mail address of the member (with member\_id: 1007) to “new.emailfromjan@yahoo.de”.



# Stored Procedure

Procedure, that extracts all Members, who don´t have a bracelet.





# Database Normalization

The database relations should satisfy at least 3NF.

Our database meets the requirements of 1NF, as all attributes are atomic (e.g. The addresses of the gym locations are subdivided in city, street, housenumber). What is more, there are no relations within other relations in our database.

Moreover, our relation schema is in the second normal form, due to the fact that it is in 1NF and every non-key attribute is full functionally dependent on any candidate key of our relation. Therefore, we created separate tables (e.g. for Trainer and Course).

Lastly, our database is also in 3NF because it is in 2NF and furthermore there is no non-key attribute of our relation schema which is transitively dependent on a candidate key. (e.g. We created a Bracelet relation, so that the bracelet\_num (which is a key attribute) and the bracelet colour (which is a non-key attribute) are not part of the Member relation).

# Application

In order to access our database, we have created an application in Python. This application enables the trainers of the gyms to manipulate the data in the database, to create and delete a member and to retrieve some specific information of a member. The application can, as a consequence, be used as an administrative programme for the company.

The structure of the coding is quiet intuitive.

In the first step, we are connecting our application to our database (PostreSQL) with the database adapter “psycopg2”. Therefore, we need to pass our credentials like the database name, the user name and password as well as the host and port where the database is located, in order to establish a connection.

Next, we are configuring our interface, which is based on the graphical user interface toolkit “Tkinter”. For structural purposes, we decided to go for a class-based concept.

As a result, we are first creating a class “gui\_settings”, where some general configurations are set. Once our application is being executed, the “start\_page” will be visible for the user. In this view, the user can choose from four use cases, which cover the basic *create, read, update & delete* functions.

In our case, the user can decide whether he wants to…

* create a new member,
* view some information of a customer,
* change the base of a member or,
* delete a member that has quit the contract.

By checking one of the buttons from the start page, the user will be redirected to the next view respectively.

Even though each view conducts different operations, the general structure of those views does not differ too much. First, we have configured the page itself. Second, we adjusted the design of the columns and buttons.

Technical-wise, we decided to use embedded SQL statements in our application. All the information which needs to be transferred individually into the SQL statements, will be parsed by variables. This is the reason why we used f-Strings. These f-Strings are then executed and immediately applied on our database.

The user will be asked to enter some mandatory information in the Entry-Columns, to execute the desired functionality. In the case of creating a new member for example, the user will be asked to fill in information about person like the member-id, the first and last name, but also the sex, the gym´s location and the e-mail address. In contrast, the user only needs the member-id, in order to retrieve some the customer´s information in the “List Member” entity. Once all mandatory fields are filled, the user can check the confirmation button at the bottom to execute the task.

Additionally, the user has the possibility to navigate through the application by checking the “Back” button.