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

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.

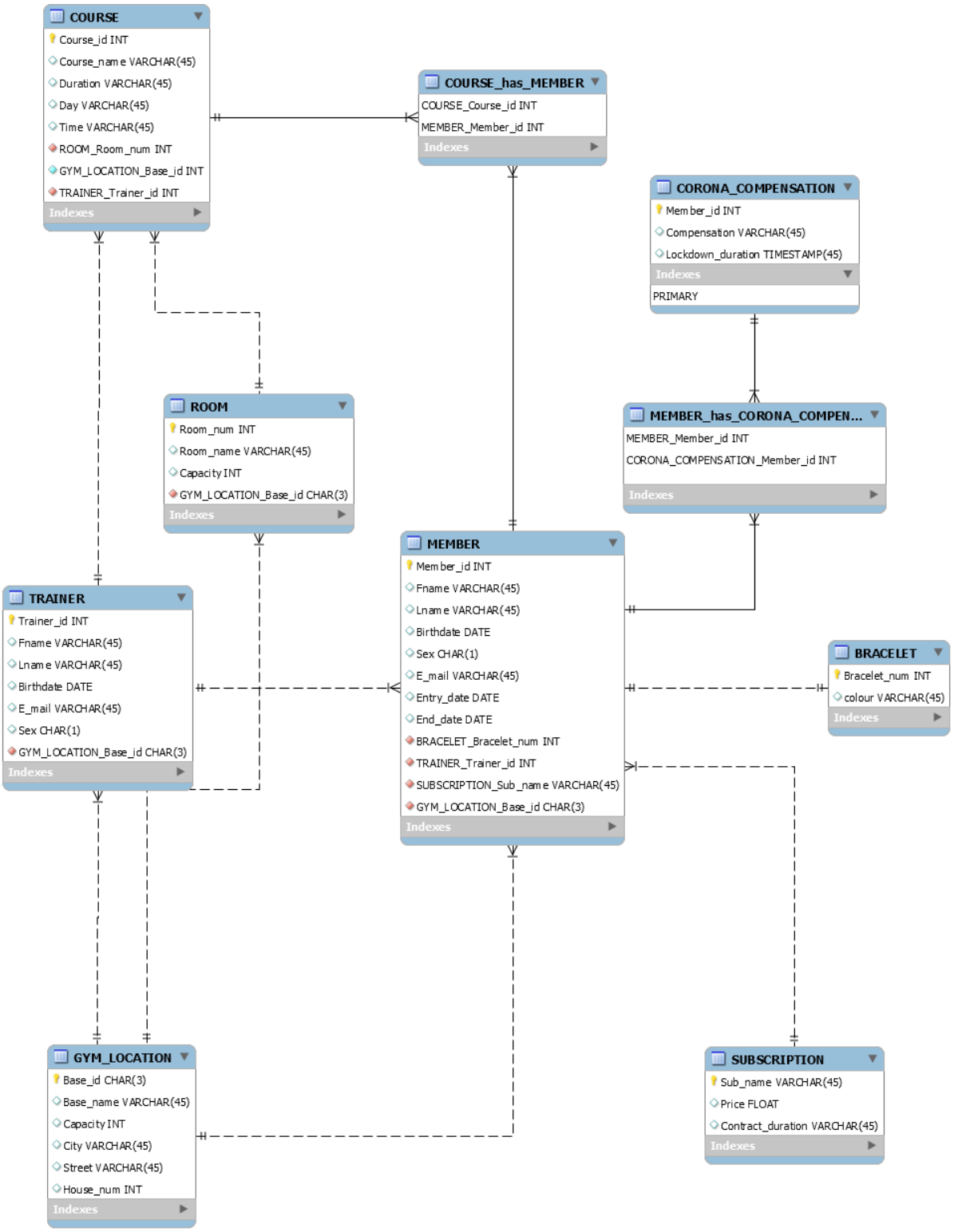
In the fitness center, the members can choose between different subscriptions. Therefore, a table keeps track of the subscription name, its monthly price as well as the duration of the contract (e.g. 1 year). Depending on the certain subscription, members have access to specific rooms in the gyms (e.g.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


COURSE
Course_id INT
Course_name VARCHAR(45)
Duration VARCHAR(45)
Day VARCHAR(45)
Time VARCHAR(45)
ROOM_Room_num INT
GYM_LOCATION_Base_id INT
TRAINER_Trainer_id INT
Indexes
COURSE_has_MEMBER
COURSE_Course_id INT
MEMBER_Member_id INT
Indexes
CORONA_COMPENSATION
Member_id INT
Compensation VARCHAR(45)
Lockdown_duration TIMESTAMP(45)
Indexes
PRIMARY
ROOM
Room_num INT
Room_name VARCHAR(45)
Capacity INT
GYM_LOCATION_Base_id CHAR(3)
Indexes
MEMBER_has_CORONA_COMPEN...
MEMBER_Member_id INT
CORONA_COMPENSATION_Member_id INT
Indexes
MEMBER
Member_id INT
Fname VARCHAR(45)
Lname VARCHAR(45)
Birthdate DATE
Sex CHAR(1)
E_mail VARCHAR(45)
Entry_date DATE
End_date DATE
BRACELET_Bracelet_num INT
TRAINER_Trainer_id INT
SUBSCRIPTION_Sub_name VARCHAR(45)
GYM_LOCATION_Base_id CHAR(3)
Indexes
TRAINER
Trainer_id INT
Fname VARCHAR(45)
Lname VARCHAR(45)
Birthdate DATE
E_mail VARCHAR(45)
Sex CHAR(1)
GYM_LOCATION_Base_id CHAR(3)
Indexes
BRACELET
Bracelet_num INT
colour VARCHAR(45)
Indexes
GYM_LOCATION
Base_id CHAR(3)
Base_name VARCHAR(45)
Capacity INT
City VARCHAR(45)
Street VARCHAR(45)
House_num INT
Indexes
SUBSCRIPTION
Sub_name VARCHAR(45)
Price FLOAT
Contract_duration VARCHAR(45)
Indexes

Create Tables (Examples) – All commands are here in Github.

```
CREATE TABLE IF NOT EXISTS MEMBER (

Member_id       INT             NOT NULL,
Fname           VARCHAR(45),
Lname           VARCHAR(45),
Birthdate       DATE,
Sex             CHAR(1),
E_mail          VARCHAR(45),
Entry_date      DATE,
End_date        DATE,
Bracelet_num    INT,
Trainer_id      INT,
Sub_name        VARCHAR(45),
Base_id         CHAR(3),
PRIMARY KEY (Member_id),

FOREIGN KEY (Bracelet_num) REFERENCES BRACELET(Bracelet_num),
FOREIGN KEY (Trainer_id) REFERENCES TRAINER(Trainer_id),
FOREIGN KEY (Sub_name) REFERENCES SUBSCRIPTION(Sub_name),
FOREIGN KEY (base_id) REFERENCES GYM_LOCATION(Base_id) );
```

```
CREATE TABLE IF NOT EXISTS TRAINER (

TRAINER_id      INT             NOT NULL,
Fname           VARCHAR(45),
Lname           VARCHAR(45),
Birthdate       DATE,
E_mail          VARCHAR(45),
Sex             CHAR(1),
Base_id         CHAR(3)         NOT NULL,
PRIMARY KEY (Trainer_id),

FOREIGN KEY (Base_id) REFERENCES GYM_LOCATION(Base_id) );
```

```
CREATE TABLE IF NOT EXISTS GYM_LOCATION (

Base_id         CHAR(3)         NOT NULL,
Base_name       VARCHAR(45),
Capacity        INT,
City            VARCHAR(45),
Street          VARCHAR(45),
House_num       INT,
PRIMARY KEY (Base_id));
```

```
CREATE TABLE IF NOT EXISTS ROOM (

Room_num        INT NOT NULL,
Room_name       VARCHAR(45),
Capacity        INT ,
Base_id         CHAR(3) NOT NULL,
PRIMARY KEY (Room_num),

FOREIGN KEY (Base_id) REFERENCES GYM_LOCATION(Base_id));
```

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”.

```
INSERT INTO GYM_LOCATION
VALUES      ('FRA','Gym Frankfurt',201,'Frankfurt','Smith-Allee',1),
            ('MHG','Gym Mannheim',173,'Mannheim','Wrigg-Allee',1),
            ('HAM','Gym Hamburg',212,'Hamburg','Becht-Allee',1),
            ('MUC','Gym Munich',179,'München','Datenbanken-Allee',1);
```

```
INSERT INTO TRAINER
VALUES          (10001,'Pascal','Smith','1999-07-19','pascalsmith@gym.de','M','FRA'),
                (10002,'Leon','Kebel','1999-07-10','lk@t-online.de','M','FRA'),
                (20001,'Tim','Gabel','1909-07-19','timmyg@gym.de','M','MHG'),
                (30001,'Peter','Smith','1996-06-16','petersmith@gym.de','M','HAM'),
                (40001,'Hansi','Mueller','1991-09-26','hansimueller@gym.de','M','MUC');
```

```
INSERT INTO ROOM
VALUES          (101,'Bodyweight-Area',64,'FRA'),
                (102,'Weightlift-Area',83,'FRA'),
                (103,'Crossfit-Area',32,'FRA'),
                (104,'Sauna',22,'FRA'),
                (105,'Pool',10,'FRA'),
                (201,'Bodyweight-Area',42,'MHG'),
                (202,'Weightlift-Area',79,'MHG'),
                (203,'Crossfit-Area',31,'MHG'),
                (204,'Sauna',21,'MHG'),
                (301,'Bodyweight-Area',71,'HAM'),
                (302,'Weightlift-Area',100,'HAM'),
                (303,'Crossfit-Area',18,'HAM'),
                (304,'Sauna',23,'HAM'),
                (401,'Bodyweight-Area',53,'MUC'),
                (402,'Weightlift-Area',77,'MUC'),
                (403,'Crossfit-Area',30,'MUC'),
                (404,'Sauna',19,'MUC');
```

SQL Queries

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

```
SELECT DISTINCT MEMBER.Fname, MEMBER.Lname, TRAINER.trainer_id
FROM MEMBER RIGHT JOIN TRAINER
ON MEMBER.Trainer_id = TRAINER.Trainer_Id;
```

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

```
SELECT *
FROM MEMBER
WHERE Trainer_id = 10001;
```

3) 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

```
SELECT DISTINCT Member_id, Fname, Lname, Base_id, Sub_name, e_mail
FROM MEMBER
WHERE (Base_id, Sub_name) IN (  SELECT Base_id, Sub_name
                FROM MEMBER
                WHERE Member_id = 1006 );
```

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

```
SELECT DISTINCT Fname, e_mail
FROM MEMBER
WHERE bracelet_num is null
ORDER BY Fname DESC;
```

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

```
SELECT *
FROM ROOM
WHERE room_name LIKE '%weight%' OR room_name LIKE '%Weight%';
```

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

```
SELECT Subscription.sub_name, COUNT(Member.sub_name) ,COUNT(Member.sub_name)*Subscription.price AS Revenue
FROM Member, Subscription
WHERE Member.Sub_name = Subscription.Sub_name
GROUP BY Subscription.sub_name, Subscription.price
```

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

```
SELECT DISTINCT MEMBER.member_id, MEMBER.Sub_name, SUBSCRIPTION.Price
FROM MEMBER LEFT JOIN SUBSCRIPTION
ON Member.sub_name = SUBSCRIPTION.sub_name;
```

View

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

```
CREATE VIEW Trainer_Overview AS
SELECT t.Fname, t.Lname, g.base_name, g.street, g.house_num, g.city
FROM TRAINER t, GYM_LOCATION g
WHERE t.Base_id = g.Base_id
```

Transaction

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

```
UPDATE MEMBER
SET e_mail='new.emailfromjan@yahoo.de'
WHERE member_id=1007
```

Stored Procedure

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

```
CREATE PROCEDURE No_bracelet
AS
SELECT DISTINCT Fname, e_mail
FROM MEMBER
WHERE bracelet_num is null
ORDER BY Fname DESC
GO;

EXECUTE PROCEDURE No_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). The attributes cannot be further devided. 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 fully 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.

In order to get the application run on all systems, we created a Docker container for the database. The file “data.sql” contains all SQL-commands which are necessary to build up the gym-database via docker-compose. Building up the container, the new database server runs on port 5433.