COMP 3005 A

Project

Project Report

Report Recipient: Professor Abdelghny Orogat

Author: Michael De Santis

CUID: 101213450

e-mail: michaeldesantis@cmail.carleton.ca

Date Submitted: December 10th, 2023

Executive Summary

The document herein represents the design of a Health and Fitness Club Management System, from the initial collection of requirements to the final design of the relational database schema that was used for implementation in a PostgreSQL database.

Acknowledgements

Thank you very much to the kind Professor Abdelghny Orogat for all the knowledge I have acquired in his COMP 3005 course, of which I hope some is apparent in this project report.

Table of Contents

1.0 Problem Statement

The problem statement upon which the design in this document is based. This problem statement was distributed at the outset of the project, and is reproduced here.

1.1 Problem Statement

Design and implement an application for a “Health and Fitness Club Management System”. This system will serve as a comprehensive platform catering to the diverse needs of club members, trainers, and administrative staff.

Members should be able to register and manage their profiles, setting personal fitness goals and inputting health metrics. Once registered, they will gain access to a personalized dashboard that tracks their exercise routines, fitness achievements, and health statistics. The platform will also enable members to schedule, reschedule, or cancel personal training sessions with certified trainers. Furthermore, members can register for group fitness classes, workshops, and other events, ensuring they always stay updated with their schedules and receive timely reminders for their sessions.

On the other side, the system should empower trainers with tools to manage their schedules, view member profiles, and input progress notes after each training session. Administrative staff, the backbone of the club’s operations, should have features that allow them to oversee club resources effectively. This includes managing room bookings, monitoring fitness equipment maintenance, and updating class schedules. Additionally, they should have a robust system to oversee billing, process payments for membership fees, personal training sessions, and other services, and monitor club activities for quality assurance. The club’s unique selling point is its

loyalty program; every transaction earns members loyalty points, which can be redeemed for future services.

2.0 Relational Database Design

The following section details the various design stages and processes in the implementation of the Health and Fitness Club Management System (HFCMS) for a Health and Fitness Club (HFC).

2.1 Conceptual Design

Beginning from the problem statement above, requirements were collected and analyzed. As this project was completed by a single individual, only certain requirements were selected from those present in the problem statement to ensure adequate scope and feasibility for implementation. These selected requirements are briefly summarized below. Where applicable, assumptions regarding entities and their participations in relations are also documented.

First, human entities were considered for inclusion in the database schema. Minimally, these entities were deemed to be Members, Trainers, and Administrators. Members would be the customers of the HFC using its facilities and services; Trainers would be employees of the HFC, responsible for facilitating both individual training Sessions and group Workshops; Administrators would be the staff that administrate and manage the HFC, handling member accounts, subscriptions, service requests, and equipment maintenance, among other duties. With these strong entities determined, additional strong entities were selected to represent the HFC's Equipment, group Workshops, and individual training Sessions.

Once a Member registers for the HFC's service, they have their basic information tracked in the Members relation, which includes single entries for most information, but permits multiple entries for Phones and Emails for customer convenience. To use the HFC's services, Members must purchase a Subscription, which is managed through the HFCMS. Subscriptions are available

in three Tiers, each corresponding to various monthly costs and certain club benefits. A user must have a single Subscription to use the HFC facilities and services, and each Subscription must have only a single Member. A Subscription, once created, must be managed by an Administrator, who is responsible for, amongst other things, collecting credit card payment for the Subscription from the Member according to the cost of the active Tier. Each Subscription must be managed by an Administrator, and an Administrator may manage any number of Subscriptions, if any. As Subscriptions are determined by the Member with which they are associated, they are considered weak entities.

Once registered and subscribed, Members are able to freely use the facilities in alignment with their Subscription Tier; additionally, they are also able to register for individual training Sessions and group Workshops, all led by HFC Trainers. Sessions are conducted once, and are scheduled for a specific date and time. Each session must have a single Member and a single Trainer. Each session also has an explicit focus (eg. cardio, weights, etc.) requested by the Member. A Member may attend any number of Sessions, and a Trainer may conduct any number of sessions, if any (as long as there is no schedule conflict).

Group Workshops are also offered by the HFC. Each Workshop has a descriptive name which details its content (eg. Power Pilates), has a maximum Member capacity, and occurs once per week on a set day over a range of weeks (eg. every Wednesday for six weeks). Each Workshop must have a single Trainer lead it, although any number of Members (up to its capacity) may participate in any Workshop. A Trainer may lead any number of sessions, if any (as long as there is no schedule conflict).

Administrators, in addition to managing Member Subscriptions, are also in charge of maintenance of the HFC's Equipment. Each piece of Equipment may have any number of Service

Requests filed against it, if any. Members may file a Service Request against a single specific piece of Equipment, which must contain the date of filing as well as the details of the request (ie. a comment explaining what service is necessary). A Member may file any number of Service Requests, but each Service Request must be filed by a single Member. When a Service Request is filed, it must also be assigned to a single Administrator who is responsible for monitoring the Service Request and coordinating its resolution. An Administrator may monitor any number of Service Requests, if any. Once a Service Request has been fulfilled (ie. the Equipment has been correctively serviced), the Administrator will resolve the Service Request by assigning it a date of resolution.

While the design above only fulfills selected requirements of the problem statement, it allows for a basic, but functional, operation of the HFC through the use of the HFCMS. This design was translated into an Entity-Relationship Diagram, as in Fig. 1 below.

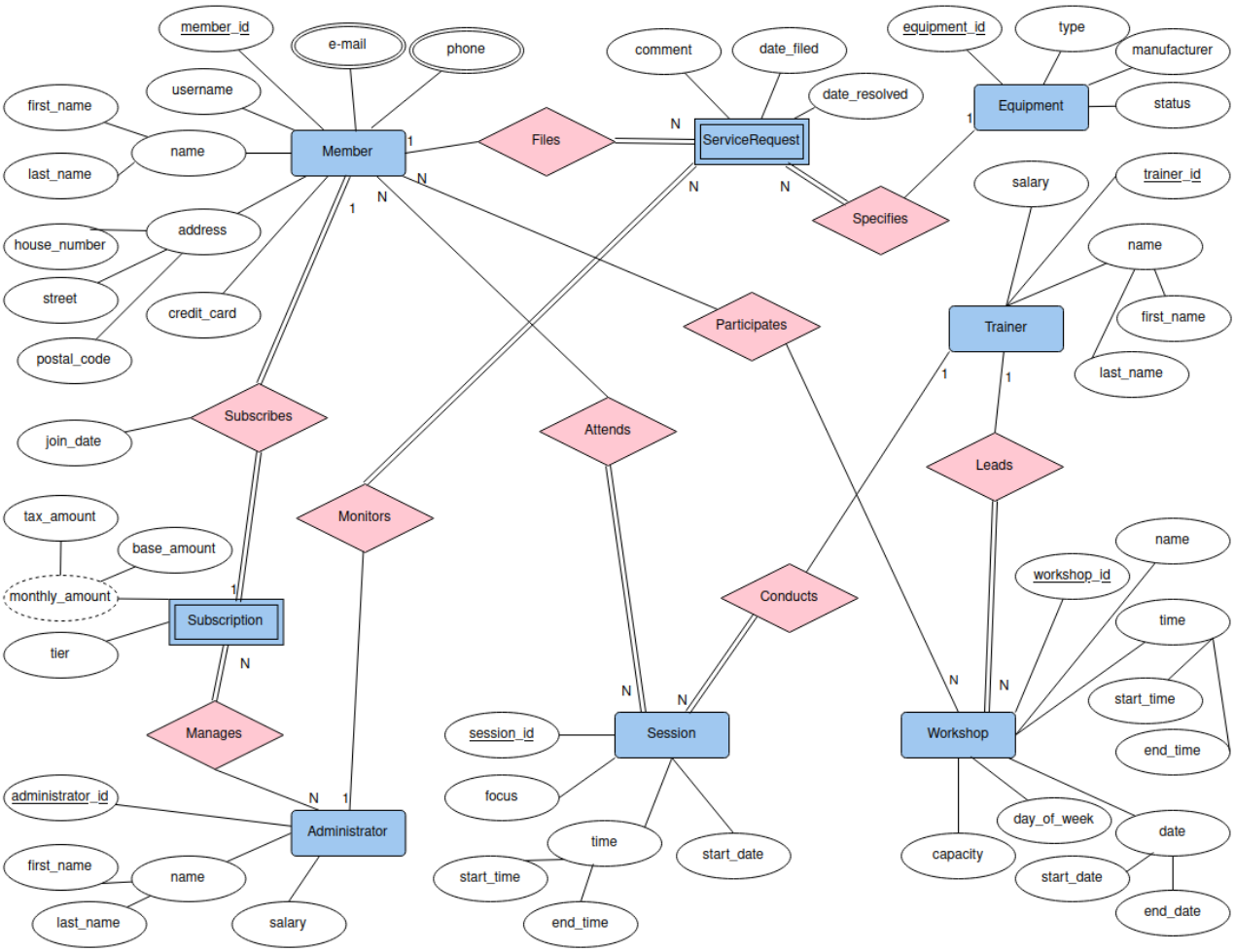

Figure 1: Entity-Relationship Diagram

2.2 Reduction to Relation Schemas

Relation schemas were devised to model all entities and relationships captured in the above Entity-Relationship diagram of Fig. 1. These relation schemas were mapped into the initial Relational Database Schema diagram in Fig. 2 below.

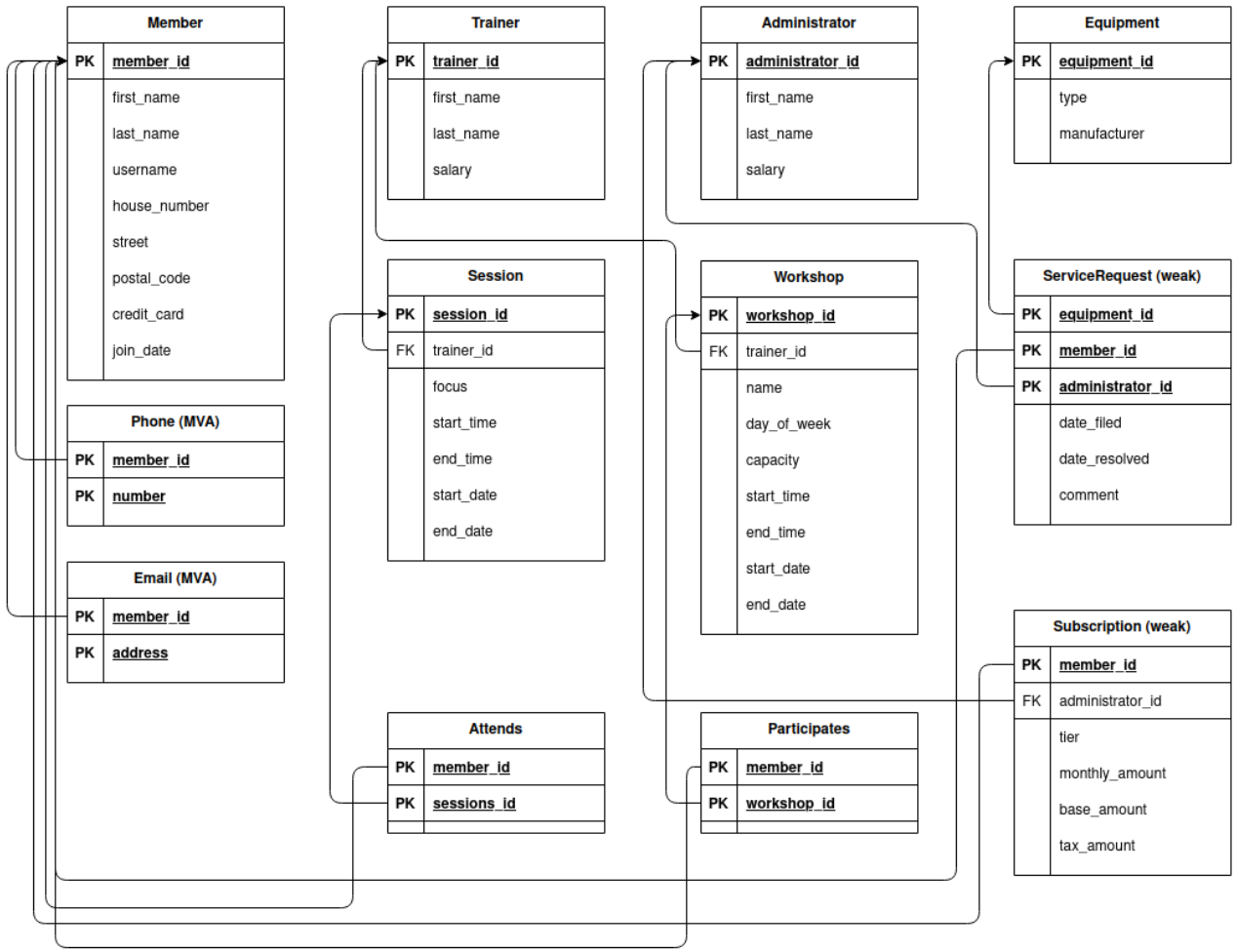

Figure 2: Initial Relational Database Schema

2.3 Normalization of Relation Schemas

Each of the schema in Fig 2. above were evaluated for First, Second, and Third Normal Form (ie. 1NF, 2NF, and 3NF) compliance.

The **Members** relation was determined to be in 1NF, 2NF, and 3NF form, and did not require modification. As there is only a single primary key (*member_id*) for each Member, all functional dependencies were determined to be full functional dependencies, as the primary key is atomic and does not permit partial keys nor partial functional dependencies, therefore becoming 2NF compliant. Additionally, this relation was determined to be 3NF compliant, as none of its non-

prime attributes may be used to identify any other of its non-prime attributes, therefore precluding any transitive dependencies.

The **Trainers** relation was determined to be in 1NF, 2NF, and 3NF form, and did not require modification. As there is only a single primary key (*trainer_id*) for each Trainer, all functional dependencies were determined to be full functional dependencies, as the primary key is atomic and does not permit partial keys nor partial functional dependencies, therefore becoming 2NF compliant. Additionally, this relation was determined to be 3NF compliant, as none of its non-prime attributes may be used to identify any other of its non-prime attributes, therefore precluding any transitive dependencies.

The **Administrators** relation was determined to be in 1NF, 2NF, and 3NF form, and did not require modification. As there is only a single primary key (*administrator_id*) for each Administrator, all functional dependencies were determined to be full functional dependencies, as the primary key is atomic and does not permit partial keys nor partial functional dependencies, therefore becoming 2NF compliant. Additionally, this relation was determined to be 3NF compliant, as none of its non-prime attributes may be used to identify any other of its non-prime attributes, therefore precluding any transitive dependencies.

The **Equipment** relation was determined to be in 1NF, 2NF, and 3NF form, and did not require modification. As there is only a single primary key (*equipment_id*) for each Equipment, all functional dependencies were determined to be full functional dependencies, as the primary key is atomic and does not permit partial keys nor partial functional dependencies, therefore becoming 2NF compliant. Additionally, this relation was determined to be 3NF compliant, as none of its non-prime attributes may be used to identify any other of its non-prime attributes, therefore precluding any transitive dependencies.

The **Sessions** relation was determined to be in 1NF, 2NF, and 3NF form, and did not require modification. As there is only a single primary key (*session_id*) for each Session, all functional dependencies were determined to be full functional dependencies, as the primary key is atomic and does not permit partial keys nor partial functional dependencies, therefore becoming 2NF compliant. Additionally, this relation was determined to be 3NF compliant, as none of its non-prime attributes may be used to identify any other of its non-prime attributes, therefore precluding any transitive dependencies.

The **Workshops** relation was determined to be in 1NF, 2NF, and 3NF form, and did not require modification. As there is only a single primary key (*workshop_id*) for each Workshop, all functional dependencies were determined to be full functional dependencies, as the primary key is atomic and does not permit partial keys nor partial functional dependencies, therefore becoming 2NF compliant. Additionally, this relation was determined to be 3NF compliant, as none of its non-prime attributes may be used to identify any other of its non-prime attributes, therefore precluding any transitive dependencies.

The **Phones** and **Emails** relations were determined to be 1NF, 2NF, and 3NF compliant, as they are simple relations derived from Multi-Valued Attributes (MVAs). As such, they only contain the partial primary key of the Member they are associated with, in addition to the partial primary key of the multi-valued attribute they were designed to satisfy (ie. Phone number, Email address). As such, there are no partial nor transitive functional dependencies.

The **Attends** and **Participates** relations were determined to be 1NF, 2NF, and 3NF compliant, as they are simple cross-reference, bridge relations used to simplify the associations between Members registered for Sessions and Workshops. As such, they only contain the partial primary key of the Member they are associated with, in addition to the partial primary key of the

corresponding Session or Workshop. As such, there are no partial nor transitive functional dependencies.

The **ServiceRequests** relation was determined to be in 1NF, 2NF, and 3NF form, and did not require modification. Although there are three partial primary keys (*equipment_id, member_id, administrator_id*) for each ServiceRequest, all functional dependencies were determined to be full functional dependencies, as no subset of these partial keys can be used to uniquely identify any of the relation's attributes. Additionally, this relation was determined to be 3NF compliant, as none of its non-prime attributes may be used to identify any other of its non-prime attributes, therefore precluding any transitive dependencies.

The **Subscriptions** relation was determined to be in 1NF, 2NF, but **not** in 3NF, and therefore did require **modification**. As there is only a single primary key (*member_id*) for each Subscription, all functional dependencies were determined to be full functional dependencies, as the primary key is atomic and does not permit partial keys nor partial functional dependencies, therefore becoming 2NF compliant. However, this relation was determined **not** to be 3NF compliant, as one of its non-prime attributes (*tier_level*) could be used to identify its other non-prime attributes (*monthly_amount, base_amount, tax_amount*), therefore establishing transitive dependencies. To normalize this, the Subscriptions relation was **decomposed** into two relations: **Subscriptions** and **Tiers**. The modified Subscriptions relation retains the *tier_level* attribute as a foreign key pointing to the new Tiers relation, which contains the attributes relating to the cost associated with each tier. Each of these new relations adheres to 1NF, 2NF, and 3NF form, as expected by the solution of decomposition. These new normalized relations may be seen in Fig. 3 below.

2.4 Database Schema Diagram

With the results of the normalization analysis of the previous section, a revised Relational Database Schema diagram was produced to incorporate the revisions required by the normalization process. The final Relational Database Schema is in Fig. 3 below. This schema represents the schema used for the implementation of the database.

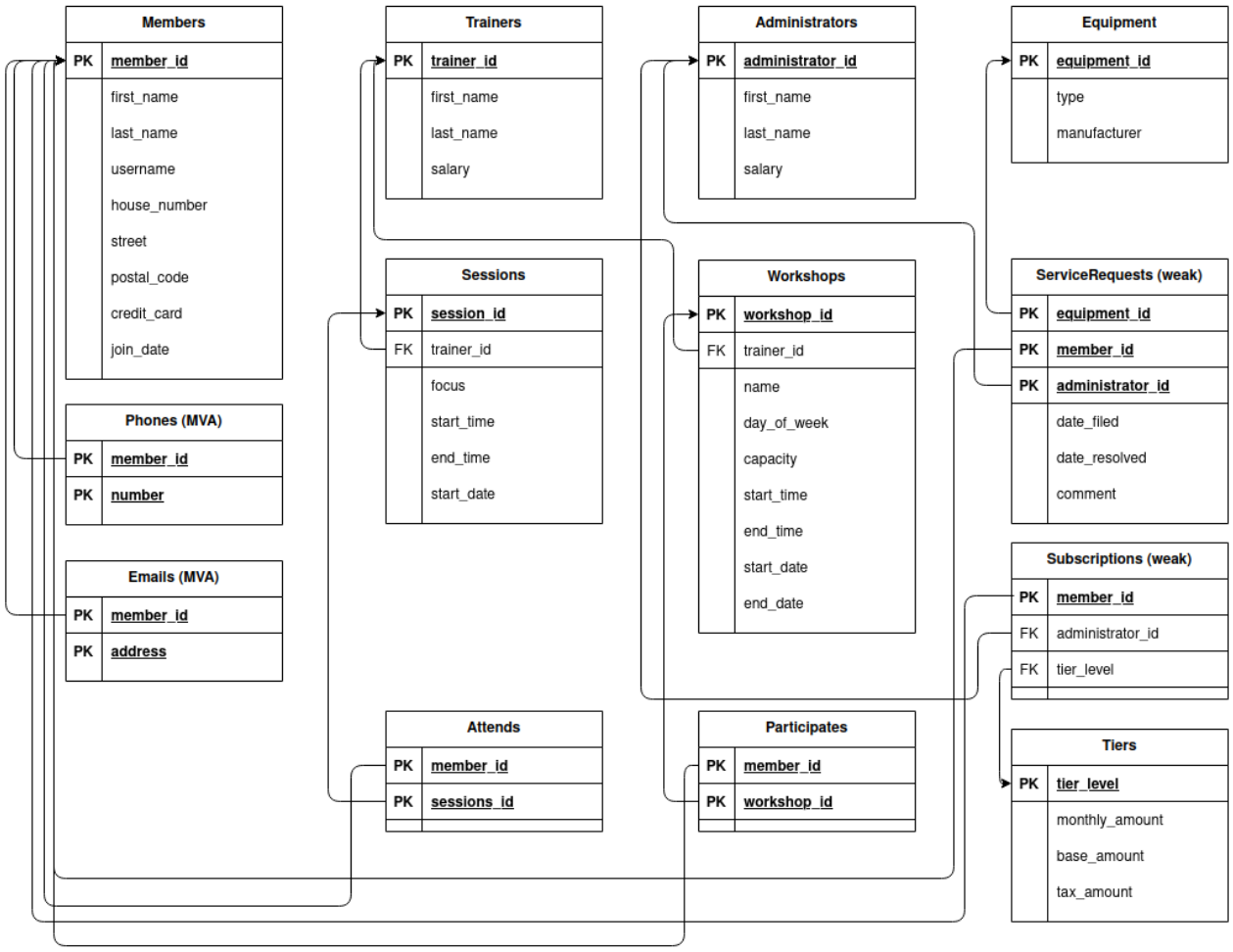

Figure 3: Revised Relational Database Schema

2.5 Implementation

The revised Relational Database Schema from Fig.3 above was implemented in PostgreSQL, using pgAdmin. The commands used to generate this schema may be found in the Data Definition Language file, *ddl.sql*, appended at the end of this report. Once instantiated, the schema may be populated with sample data by using the Data Manipulation Language file, *dml.sql*, appended below. Some sample queries and additional operations that may be used against this sample instance are contained in the file *queries.sql,* appended below.

2.6 GitHub Repository

This report, and all other project components, may be found at the following public GitHub repository: https://github.com/mpdesantis/comp3005-project.git

3.0 Appendix

Appended documents referenced by this report. These documents are also available as plain text in the project's public GitHub repository, linked above.

3.1 *ddl.sql*

```
-- ddl.sql
-- COMP 3005: Project
-- Michael De Santis
-- CUID: 101213450

-- Tables --

-- Table: Members
create table Members (
    member_id serial unique not null,
    first_name varchar(255) not null,
    last_name varchar(255) not null,
    username varchar(255) not null,
    house_number int,
    street varchar(255) not null,
    postal_code varchar(255) not null,
    credit_card int,
    join_date date default current_date,

    primary key (member_id)
);


-- Table: Trainers
create table Trainers (
    trainer_id serial unique not null,
    first_name varchar(255) not null,
    last_name varchar(255) not null,
    salary int,

    primary key (trainer_id)
);


-- Table: Administrators
create table Administrators (
    administrator_id serial unique not null,
    first_name varchar(255) not null,
    last_name varchar(255) not null,
    salary int,

    primary key (administrator_id)
);
```

```
-- Table: Equipment
create table Equipment (
    equipment_id serial unique not null,
    type varchar(255) not null,
    manufacturer varchar(255) not null,

    primary key (equipment_id)
);


-- Table: Tiers
create table Tiers (
    tier_level serial unique not null,
    monthly_amount int,
    base_amount int,
    tax_amount int,

    primary key (tier_level)
);


-- Table: Sessions
create table Sessions (
    session_id serial unique not null,
    trainer_id int,
    focus varchar(255) not null,
    start_time time without time zone,
    end_time time without time zone,
    start_date date,

    primary key (session_id),
    foreign key (trainer_id)
        references Trainers
);


-- Table: Workshops
create table Workshops (
    workshop_id serial unique not null,
    trainer_id int,
    name varchar(255) not null,
    day_of_week varchar(255) not null,
    capacity int,
    start_time time without time zone,
    end_time time without time zone,
    start_date date,
    end_date date,

    primary key (workshop_id),
    foreign key (trainer_id)
        references Trainers
);
```

```
-- Table: ServiceRequests
create table ServiceRequests (
    equipment_id int,
    member_id int,
    administrator_id int,
    date_filed date,
    date_resolved date,
    comment varchar(255) not null,

    primary key (equipment_id, member_id, administrator_id),

    foreign key (equipment_id)
        references Equipment,
    foreign key (member_id)
        references Members,
    foreign key (administrator_id)
        references Administrators
);


-- Table: Subscriptions
create table Subscriptions (
    member_id int,
    administrator_id int,
    tier_level int,

    primary key (member_id),

    foreign key (administrator_id)
        references Administrators,
    foreign key (tier_level)
        references Tiers
);


-- Table: Participates
create table Participates (
    member_id int,
    workshop_id int,

    primary key (member_id, workshop_id),

    foreign key (member_id)
        references Members,
    foreign key (workshop_id)
        references Workshops
);


-- Table: Attends
create table Attends (
    member_id int,
    session_id int,

    primary key (member_id, session_id),
```

```
    foreign key (member_id)
        references Members,
    foreign key (session_id)
        references Sessions
);


-- Table: Phones
create table Phones (
    member_id int,
    number int,

    primary key (member_id, number),

    foreign key (member_id)
        references Members
);


-- Table: Emails
create table Emails (
    member_id int,
    address varchar(255) not null,

    primary key (member_id, address),

    foreign key (member_id)
        references Members
);
```

3.2 *dml.sql*

```
-- dml.sql
-- COMP 3005: Project
-- Michael De Santis
-- CUID: 101213450


-- Populate Members Table
INSERT INTO Members (
    first_name,
    last_name,
    username,
    house_number,
    street,
    postal_code,
    credit_card,
    join_date
)
VALUES
(
    'Reggie',
    'Mustache',
    'rmustache',
    10,
    'Mustache Boulevard',
    'A1A B2B',
    12345678,
    '2023-09-09'
),
(
    'Gary',
    'Lasagna',
    'glasagna',
    22,
    'Penne Lane',
    'P4S 7AS',
    12345678,
    '2022-12-09'
),
(
    'Rebecca',
    'Pumpkin',
    'rpumpkin',
    66,
    'Pie Way',
    'B3B B3B',
    44448888,
    '2019-10-11'
),
(
    'Doris',
    'Doorbell',
```

```
    'ddoorbell',
    123,
    'Ding Dong Parkway',
    'M3M PAP',
    12345678,
    '2012-07-04'
);


-- Populate Trainers Table
INSERT INTO Trainers (
    first_name,
    last_name,
    salary
)
VALUES
(
    'Bob',
    'Bicep',
    66000
),
(
    'Lesley',
    'Legs',
    76000
),
(
    'Tim',
    'Trapezoid',
    68000
),
(
    'Harry',
    'Hamstring',
    59000
);


-- Populate Administrators Table
INSERT INTO Administrators (
    first_name,
    last_name,
    salary
)
VALUES
(
    'Annie',
    'Admin',
    56000
),
(
    'Sally',
    'Spreadsheet',
    67000
),
```

```
(
    'Rodney',
    'Rolodex',
    48000
),
(
    'Peter',
    'Pencils',
    58000
);



-- Populate Equipment Table
INSERT INTO Equipment (
type,
manufacturer
)
VALUES
(
    'Ab Machine',
    'Abme Fitness'
),
(
    'Chest Press',
    'Fancy Fitness'
),
(
    'Leg Pusher',
    'Fancy Fitness'
),
(
    'Squat Thing',
    'Fancy Fitness'
),
(
    'Rowing Machine',
    'Fresh Fitness'
),
(
    'Stationary Bike',
    'Forward Fitness'
);


-- Populate Tiers Table
INSERT INTO Tiers (
    monthly_amount,
    base_amount,
    tax_amount
)
VALUES
(
    105,
    100,
```

```
    5
),
(
    210,
    200,
    10
),
(
    330,
    300,
    15
);


-- Populate Sessions Table
INSERT INTO Sessions
(
    trainer_id,
    focus,
    start_time,
    end_time,
    start_date
)
VALUES
(
    1,
    'cardio',
    '11:00:00',
    '12:00:00',
    '2023-12-19'
),
(
    2,
    'weights',
    '12:00:00',
    '13:00:00',
    '2023-12-20'
),
(
    2,
    'rolling',
    '15:00:00',
    '16:00:00',
    '2023-12-22'
),
(
    2,
    'jumping',
    '15:00:00',
    '16:00:00',
    '2023-12-22'
),
(
    3,
    'flexibility',
```

```
    '14:00:00',
    '15:00:00',
    '2023-12-21'
);


-- Populate Workshops Table
INSERT INTO Workshops
(
    trainer_id,
    name,
    day_of_week,
    capacity,
    start_time,
    end_time,
    start_date,
    end_date
)
VALUES
(
    1,
    'Power Pilates',
    'Friday',
    20,
    '15:00:00',
    '16:00:00',
    '2023-12-22',
    '2024-03-22'
),
(
    1,
    'Super Stretches',
    'Wednesday',
    30,
    '19:00:00',
    '20:00:00',
    '2023-12-22',
    '2024-03-22'
),
(
    1,
    'Wild Weights',
    'Wednesday',
    10,
    '08:00:00',
    '09:00:00',
    '2023-12-22',
    '2024-03-22'
),
(
    3,
    'Burly Bending',
    'Tuesday',
    15,
    '09:00:00',
```

```
    '10:00:00',
    '2023-12-22',
    '2024-03-22'
),
(
    2,
    'Lethal Legs',
    'Monday',
    25,
    '10:00:00',
    '11:00:00',
    '2023-12-22',
    '2024-03-22'
);


-- Populate ServiceRequests Table
INSERT INTO ServiceRequests
(
    equipment_id,
    member_id,
    administrator_id,
    date_filed,
    date_resolved,
    comment
)
VALUES
(
    1,
    3,
    1,
    '2023-12-14',
    '2023-12-16',
    'Machine feels funny.'
),
(
    2,
    2,
    1,
    '2023-09-09',
    '2023-10-10',
    'Machine is leaking stuff.'
),
(
    3,
    2,
    3,
    '2023-12-12',
    NULL,
    'Machine has a big hole in it.'
),
(
    4,
    2,
    3,
```

```
    '2023-12-12',
    NULL,
    'Machine is missing a thing it needs.'
);

-- Populate Subscriptions Table
INSERT INTO Subscriptions
(
    member_id,
    administrator_id,
    tier_level
)
VALUES
(
    1,
    1,
    1
),
(
    2,
    1,
    3
),
(
    3,
    2,
    3
),
(
    4,
    3,
    2
);


-- Populate Participates Table
INSERT INTO Participates
(
    member_id,
    workshop_id
)
VALUES
(
    1,
    1
),
(
    1,
    2
),
(
    3,
    2
),
(
```

```
    4,
    2
),
(
    3,
    1
);


-- Populate Attends Table
INSERT INTO Attends
(
    member_id,
    session_id
)
VALUES
(
    1,
    2
),
(
    2,
    1
),
(
    3,
    4
),
(
    4,
    3
);


-- Populate Phones Table
INSERT INTO Phones
(
    member_id,
    number
)
VALUES
(
    1,
    5551234
),
(
    1,
    5557070
),
(
    2,
    5559876
),
(
    3,
```

```
    5554466
),
(
    4,
    5557321
);


-- Populate Emails Table
INSERT INTO Emails
(
    member_id,
    address
)
VALUES
(
    1,
    'rmustache@mustache.com'
),
(
    1,
    'reggie@supergymguy.com'
),
(
    2,
    'glasagna@pasta.com'
),
(
    3,
    'rpumpkin@pie.com'
),
(
    4,
    'ddoorbell@dingdong.com'
);
```

3.3 *queries.sql*

```
-- queries.sql
-- COMP 3005: Project
-- Michael De Santis
-- CUID: 101213450

-- Queries --

-- Query 1. Query the first and last names of all members.
select first_name, last_name
from Members;

-- Query 2. Query the address of all members who have joined after 2022-06-06
select house_number, street, postal_code
from Members
where join_date > to_date('2022-06-06', 'YYYY-MM-DD');

-- Query 3. Query the total number of workshops being faciliated by each
trainer.
select first_name, last_name, count(trainer_id) as workshops_facilitated
from Trainers natural join Workshops
group by first_name, last_name;

-- Query 4. Query the machine type, manufacturer, and complaints of all gym
machines with unresolved service requests.
select type, manufacturer, comment
from Equipment natural join
(
    select equipment_id, comment
    from ServiceRequests
    where date_resolved is null
);

-- Query 5. Query all Administrators who earn over $55000.
select first_name, last_name
from Administrators
where salary > 55000;

-- Query 6. Register Gary Lasagna for the Power Pilates workshop.
insert into Participates
(
    member_id,
    workshop_id
)
values
(
    (
        select member_id
        from Members
        where first_name='Gary' and last_name='Lasagna'
    ),
```

```
    (
        select workshop_id
        from Workshops
        where name='Power Pilates'
    )
);

-- Query 7. Give Annie Admin a raise to $62000.
update Administrators
set salary=62000
where first_name='Annie' and last_name='Admin';

-- Query 8. Fire the Administrator named Peter Pencils.
delete from Administrators
where first_name='Peter' and last_name='Pencils';

--Query 9. Determine the Trainer's name who is running the Burly Bending
workshop.
select first_name, last_name
from Workshops natural join Trainers
where name='Burly Bending';

-- Query 10. Determine the name and salary of the lowest paid Trainer.
select first_name, last_name, salary
from Trainers
order by salary asc
limit 1;
```