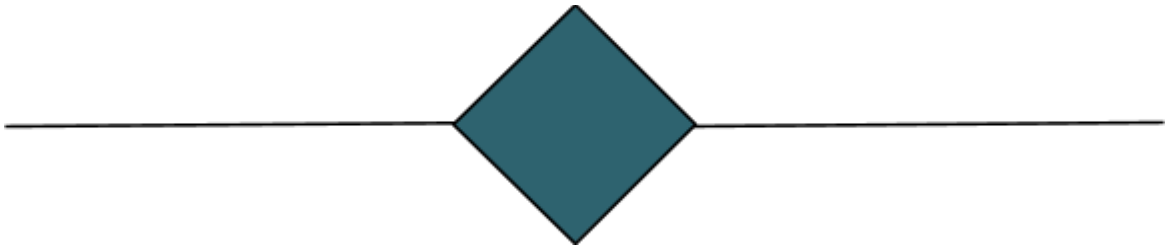
Final Project Databases



PREPARED BY

Dertje [redacted]

Lina [redacted]

Jaylee [redacted]

Lize [redacted]

Introduction

Our amateur football club has been up and running for a few years now and we have had the amazing opportunity to scale up the club and allow more members to join and train with us. We have been using shared excel files and an e-mailing system for members to reserve a football pitch to train at. In order to be able to scale up, we would like to replace this method with a database.

In our football club each trainer has the possibility to reserve pitches in order to train a team. A trainer of a team can choose the pitch for a certain date and time based on the type of grass the pitch has, keeping in mind that some pitches are indoor and some outdoor. Each trainer can make multiple reservations for different time slots. Once a reservation has been made it cannot be changed, only deleted. A reservation is for one hour and if a trainer wants to have a longer training session, he needs to make another reservation.

Every team consists of multiple members and each trainer trains multiple teams. If a trainer makes a reservation to train, we need to register the date, starting time of the training session, which team they're training and on which pitch. One pitch can have several reservations in a day at different time slots. Each trainer trains multiple teams and can train at different pitches and a team only has one trainer. Thus, when a pitch is reserved by a trainer, the trainer specifies which team he/she will be training, so the team and corresponding trainer are assigned to that pitch automatically. Members of the club can only be in one team and a team needs at least 11 members.

For our database to work as is due, we need to keep track of our current football pitches and their characteristics including type (artificial or natural grass) and place (indoor or outdoor). Next to that we keep track of our members and trainers contact information including name, surname, email and home address. Additionally, for each member we note which team they belong to.

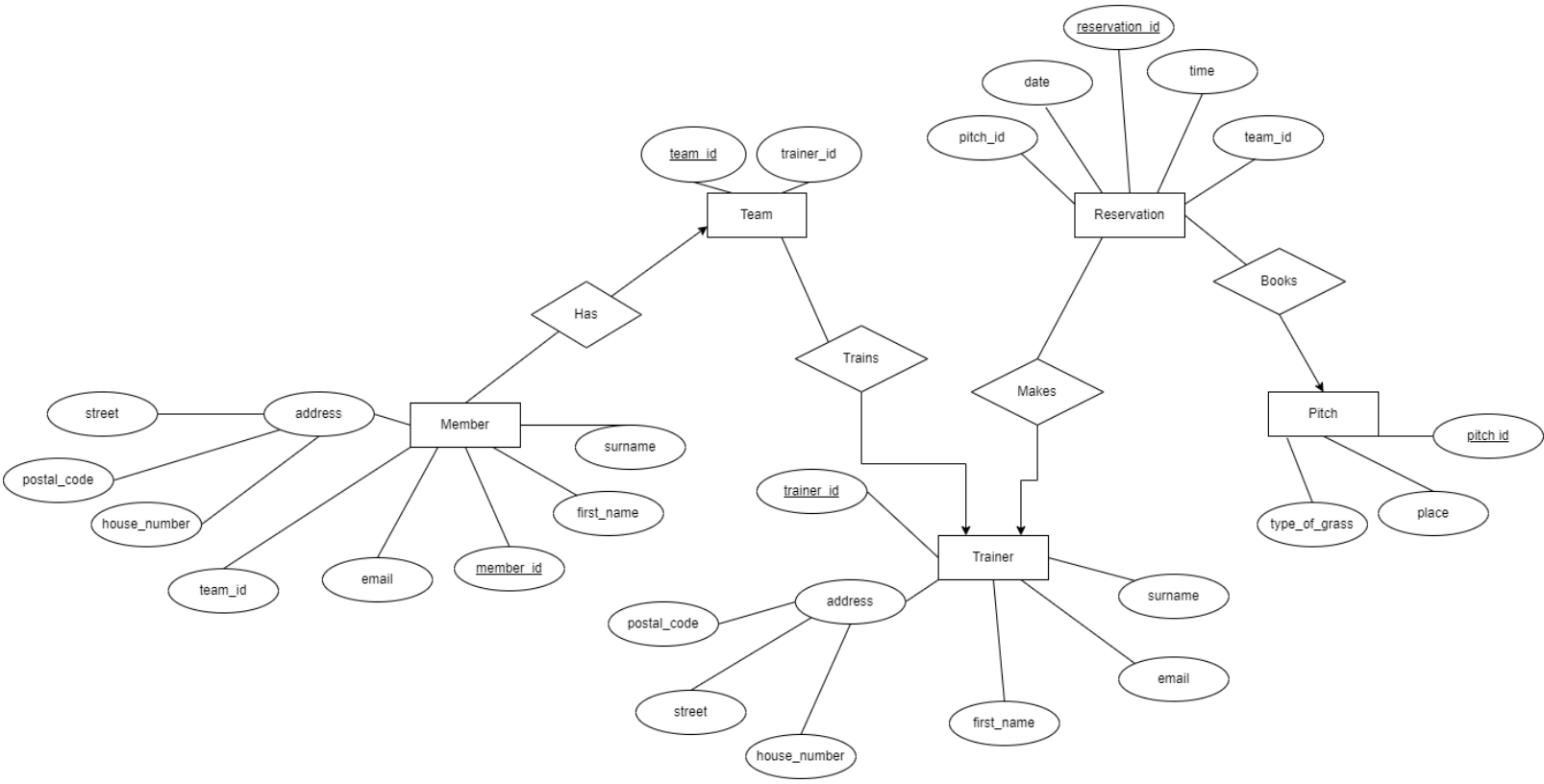
Information Requirements

To solve our problem we can use multiple sources of information. We're going to use the data from our own group members such as our names, addresses, email addresses and dates of birth as well as additional made-up information in order to test out our SQL queries. We're also going to be using information from the website of an existing football club to get an idea of what kind of information is given about trainers and the available pitches. Additionally, we will check with an existing reservation system with a sportclub that is similar to a football club to see what kind of information is needed to make a reservation.

List Of Entities And Attributes

Trainer	Member	Reservation	Pitch	Team
trainer_id	member_id	reservation_id	pitch_id	team_id
name	name	date	place	trainer_id
surname	surname	time	type_of_grass	
address (street, house_number, postal_code)	address (street, house_number, postal_code)	team_id		
email	email			
	team_id			

ER Diagram


reservation_id
date
time
pitch_id
team_id
team_id
trainer_id
Team
Reservation
Books
Has
Trains
Makes
street
address
Member
surname
Pitch
pitch id
postal_code
house_number
first_name
trainer_id
team_id
email
member_id
Trainer
surname
address
postal_code
email
street
first_name
house_number
type_of_grass
place

Relational Model

Entities:

- Member (member_id, first_name, surname, email, team_id, street, postal_code, house_number)
    - FK team_id refers to team_id in Team
- Team (team_id, trainer_id)
    - FK trainer_id refers to trainer_id in Trainer
- Trainer(trainer_id, first_name, surname, email, postal_code, street, house_number)
- Reservation(reservation_id, time, date, team_id, pitch_id)
    - FK pitch_id refers to pitch_id in Pitch
    - FK team_id refers to team_id in Team
- Pitch(pitch_id, place, type_of_grass)

Relationships:

- Has(member_id, team_id)
    - FK member_id refers to member_id in Member
    - FK team_id refers to team_id in Team
- Trains(team_id, trainer_id)
    - FK team_id refers to team_id in Team
    - FK team_id refers to trainer_id in Trainer
- Makes(reservation_id, trainer_id)
    - FK trainer_id refers to trainer_id in Trainer
    - FK reservation_id refers to reservation_id in Reservation
- Books(reservation_id, pitch_id)
    - FK pitch_id refers to pitch_id in Pitch
    - FK reservation_id refers to reservation_id in Reservation

Normalization

For our project proposal we decided to normalize our relations to the 3rd normal form. By normalizing the relations to 3NF we remove anomalies but we preserve all functional dependencies. Normalization beyond 3NF may cause a loss of functional dependencies and could cause the database to slow down. A highly normalized database with many tables and joins between tables is slower than a database that is less normalized. Therefore for each relation in our database, we highlight the functional dependencies and then find the canonical cover. Based on the canonical cover we can see whether the relation is in 3NF or not which allows us to further decompose the relation into new relations. All relations are already compliant with 1NF and 2NF.

1. **Member(member id, first_name, surname, email, team_id, street, postal_code, house_number) - FK team_id refers to team_id in Team**

Functional dependencies:

- member_id → first_name, surname, email, team_id, street, postal_code, house_number
    - A → B, C, D, E, F, G, H
- postal_code, house_number→ street
    - GH → F

Canonical cover:

a) Normal form

A → B
A → C
A → D
A → E
A → F
A → G

A → H
GH → F

b) Closure of FDs in normal form

{A}+ = {B, A, C, D, E, F, G, H}
{A}+ = {C, A, B, D, E, F, G, H}
{A}+ = {D, A, B, C, E, F, G, H}
{A}+ = {E, F, G, H, A, B, C, D}

{A}+ = {F, G, H, A, B, C, D, E}
{A}+ = {G, H, A, B, C, D, E, F}
{A}+ = {H, A, B, C, D, E, F, G}
{GH}+ = {F}

c) Apply left reduction

Check for {GH} if {G} is redundant:
(cannot use closure {GH}+ = {F})

{G}+ = {G} → not the same closure

Check for {EG} if {G} is redundant:
(cannot use closure {GH}+ = {F})

{H}+ = {H} → not the same closure

Not redundant so the FD stays the same

d) Remove redundant FDs

If we remove A → B, closure of {A}+ is {A, C, D, E, F, G, H}, not redundant
If we remove A → C, closure of {A}+ is {A, B, D, E, F, G, H}, not redundant
If we remove A → D, closure of {A}+ is {A, B, C, E, F, G, H}, not redundant
If we remove A → E, closure of {A}+ is {A, B, C, D, F, G, H}, not redundant
If we remove A → F, closure of {A}+ is {A, B, C, D, F, G, H, F}, this FD is redundant
If we remove A → G, closure of {A}+ is {A, B, C, D, E, F, H}, not redundant
If we remove A → H, closure of {A}+ is {A, B, C, D, E, F, G}, not redundant
If we remove EG → F, closure of {GH}+ is {GH}, so it's not redundant

Final FDs in canonical cover:
A → BCDEGH
GH → F

Candidate key:
{AGH}

None of the FDs are trivial; {A} is part of a candidate key and {F} is not a part of the CK; {GH} is not a superkey. So the table is not in 3NF.

Decomposition:
R1(A, B, C, D, E, G, H)
→ Member (member_id, first_name, surname, email, team_id, postal_code, house_number)

R2(F, G, H)
→ Address (postal_code, house_number, street)

2. **Team(<u>team_id</u>, trainer_id) - FK trainer_id refers to trainer_id in Trainer**

Functional dependencies:

- team_id → trainer_id
  - A → B

Canonical form:

a) Normal form

A → B

b) Closure of FD's in normal form

{A+} = {A, B}

c) Apply left reduction

Not applicable

d) Remove redundant FD's

If we remove A -> B, closure of {A+} is {A}, so it's not redundant

Candidate key:
{A}

The FD is not trivial, {A} is a candidate key and {B} is not a part of the CK. So the table is in 3NF.

3. **Trainer(<u>trainer_id</u>, first_name, surname, email, postal_code, street, house_number)**

Functional dependencies:

- trainer_id → first_name, surname, email, postal_code, street, house_number
  - A → B, C, D, E, F, G
- postal_code, house_number→ street
  - EG → F

Canonical cover:

a) Normal form

A → B
A → C
A → D
A → E
A → F
A → G
EG → F

b) Closure of FDs in normal form

{A}+ = {B, A, C, D, E, F, G}
{A}+ = {C, A, B, D, E, F, G}
{A}+ = {D, A, B, C, E, F, G}
{A}+ = {E, F, G, A, B, C, D}
{A}+ = {F, G, A, B, C, D, E}
{A}+ = {G, A, B, C, D, E, F}
{EG}+ = {F}

c) Apply left reduction

Check for {EG} if {E} is redundant:
(cannot use closure {EG}+ = {F})

{G}+ = {G} → not the same closure

Check for {EG} if {G} is redundant:
(cannot use closure {EG}+ = {F})

{E}+ = {E} → not the same closure

Not redundant so the FD stays the same

d) Remove redundant FDs

If we remove A → B, closure of {A}+ is {A, C, D, E, F, G}, not redundant
If we remove A → C, closure of {A}+ is {A, B, D, E, F, G}, not redundant
If we remove A → D, closure of {A}+ is {A, B, C, E, F, G}, not redundant
If we remove A → E, closure of {A}+ is {A, B, C, D, F, G}, not redundant
If we remove A → F, closure of {A}+ is {A, B, C, D, F, G}, this FD is redundant
If we remove A → G, closure of {A}+ is {A, B, C, D, E, F}, not redundant
If we remove EG → F, closure of {EG}+ is {EG}, so it's not redundant

Final FDs in canonical cover:
A → BCDEG
EG → F

None of the FDs are trivial, {A} is a candidate key and {F} is not a part of the CK and {EG} is not a superkey. So the table is not in 3NF.

Decomposition:
R1(A, B, C, D, E, G)
→ Trainer (trainer_id, first_name, surname, email, postal_code, house_number)
R2(E, G, F)
→ Address (postal_code, house_number, street)

4. Reservation(reservation_id, time, date, team_id, pitch_id) - FK pitch_id refers to pitch_id in Pitch, FK team_id refers to team_id in Team

Functional dependencies:

- reservation_id → time, date, team_id, pitch_id
    - A → B, C, D, E

Canonical cover:

a) Normal form

A → B
A → C
A → D
A → E

b) Closure of FDs, in normal form

{A}+ = {A, B, C, D, E}
{A}+ = {A, B, C, D, E}
{A}+ = {A, B, C, D, E}
{A}+ = {A, B, C, D, E}

c) Apply left reduction

Not applicable

d) Remove redundant FDs

If we remove A → B, the closure of {A}+ is not the same so it's not redundant
If we remove A → C, the closure of {A}+ is not the same so it's not redundant
If we remove A → D, the closure of {A}+ is not the same so it's not redundant
If we remove A → E, the closure of {A}+ is not the same so it's not redundant

Final FDs:
A → B, C, D, E

Candidate key:
{A}

The FD is not trivial and {A} is a candidate key, therefore the table is in 3NF.

5. Pitch(pitch_id, place, type_of_grass)

Functional dependencies:

- pitch_id → place, type_of_grass
  - A → B, C
- place -> type_of_grass
  - B → C

Canonical cover:

a) Normal form

A → B
A → C
B → C

b) Closure of FDs in normal form

{A}+ = {A, B, C}
{A}+ = {A, B, C}
{B}+ = {B, C}

c) Apply left reduction

Not applicable

d) Remove redundant FDs

If we remove A → B, closure of {A}+ is {A, C}, not redundant
If we remove A → C, closure of {A}+ is {A, B, C}, redundant
If we remove B → C, closure of {B}+ is {B}, not redundant

Final FDs:
A → B
B → C

Candidate key:
{A}

None of the FDs are trivial, {A} is a candidate key, {C} is not part of the CK and {B} is not a superkey. So the table is not in 3NF.

Decomposition:
R1(A, B)
→ Pitch (pitch_id, place)
R2(B, C)
→ Typepitch(place, type_of_grass)

Database Implementation

Below are the SQL DDL and SQL DML for creating the tables and inserting the data into the tables. Additionally, there is one statement to alter a table, because the data type for team_id had to be changed from integer to varchar (4).

Creating tables for entities and relationships:

```
CREATE TABLE Member (
member_id int NOT NULL,
first_name varchar(50) NOT NULL,
surname varchar(50) NOT NULL,
email varchar(50) NOT NULL,
team_id int NOT NULL,
postal_code char(6) NOT NULL,
house_number varchar(10) NOT NULL,
PRIMARY KEY(member_id));

CREATE TABLE Trainer (
trainer_id int NOT NULL,
first_name varchar(50) NOT NULL,
surname varchar(50) NOT NULL,
email varchar(50) NOT NULL,
postal_code char(6) NOT NULL,
house_number varchar(10),
PRIMARY KEY(trainer_id));

CREATE TABLE Team (
team_id varchar(4) NOT NULL,
trainer_id int NOT NULL,
PRIMARY KEY(team_id),
FOREIGN KEY(trainer_id) REFERENCES Trainer(trainer_id));

CREATE TABLE Address (
postal_code char(6) NOT NULL,
house_number varchar(10),
street varchar(50) NOT NULL,
PRIMARY KEY(postal_code,house_number));
```

```
CREATE TABLE Reservation (
reservation_id int NOT NULL,
date_time datetime NOT NULL,
team_id varchar(4) NOT NULL,
pitch_id int NOT NULL,
PRIMARY KEY (reservation_id),
FOREIGN KEY (team_id) REFERENCES Team(team_id)
FOREIGN KEY (pitch_id) REFERENCES Pitch(pitch_id));

CREATE TABLE Pitch (
pitch_id int NOT NULL,
place varchar(50) NOT NULL,
PRIMARY KEY (pitch_id));

CREATE TABLE Typepitch (
place varchar(50) NOT NULL,
type_of_grass varchar(50) NOT NULL,
PRIMARY KEY (place));

CREATE TABLE Has (
member_id int NOT NULL,
team_id varchar(4) NOT NULL,
PRIMARY KEY (member_id),
FOREIGN KEY (member_id) REFERENCES Member(member_id),
FOREIGN KEY (team_id) REFERENCES Team(team_id));

CREATE TABLE Trains (
team_id varchar(4) NOT NULL,
trainer_id int NOT NULL,
PRIMARY KEY (team_id),
FOREIGN KEY (trainer_id) REFERENCES Trainer(trainer_id),
FOREIGN KEY (team_id) REFERENCES Team(team_id));

CREATE TABLE Makes (
reservation_id int NOT NULL,
trainer_id int NOT NULL,
PRIMARY KEY (reservation_id),
```

```
FOREIGN KEY (trainer_id) REFERENCES Trainer(trainer_id),
FOREIGN KEY (reservation_id) REFERENCES Reservation(reservation_id));

CREATE TABLE Books (
pitch_id int NOT NULL,
reservation_id int NOT NULL,
PRIMARY KEY (reservation_id),
FOREIGN KEY (pitch_id) REFERENCES Pitch(pitch_id),
FOREIGN KEY (reservation_id) REFERENCES Reservation(reservation_id));
```

Inserting data into tables:

```
INSERT INTO Member (member_id, first_name, surname,
email, team_id, postal_code, house_number)
VALUES ('1', 'Dertje', 'Roggeveen', 'd.j.roggeveen@student.rug.nl', 'A1',
'9479PC', '10'),
('2', 'Carlijn', 'Barkema', 'c.barkema@student.rug.nl', 'A1', '9718AS', '17'),
('3', 'Lize', 'Dekker', 'lize.dekker14@gmail.com', 'A1', '9501AD', '29'),
('4', 'Jaylee', 'van Ligten', 'jayleevanligten@gmail.com', 'A1', '9745DS', '6'),
('5', 'Lina', 'Al Najar', 'linaalnajar1@gmail.com', 'A1', '9741NP', '107-9'),
('6', 'Anna', 'van Linden', 'annavanlinden@gmail.com', 'A1', '8465LN', '12'),
('7', 'Emma', 'de Jong', 'emmadejong@gmail.com', 'A1', '6325FO', '96'),
('8', 'Gerda', 'Zeilstra', 'gerdazeilstra@gmail.com', 'A1', '9621XZ', '4'),
('9', 'Carlijn', 'Kamminga', 'carlijnkamminga@gmail.com', 'A1', '8942LM', '16'),
('10', 'Amal', 'Clooney', 'amalclooney@gmail.com', 'A1', '1234AB', '5'),
('11', 'Jildau', 'Sluis', 'jildausluis@gmail.com', 'A1', '3214HW', '13'),
('12', 'Kaylee', 'Huls', 'kayleehuls@gmail.com', 'A2', '6294JK', '67'),
('13', 'Suze', 'Neinders', 'suzeneinders@gmail.com', 'A2', '9502FI', '23'),
('14', 'Lotte', 'Bosma', 'lottebosma@gmail.com', 'A2', '9234IN', '89'),
('15', 'Samantha', 'Visser', 'samanthavisser@gmail.com', 'A2', '9450HJ', '3'),
('16', 'Mathilde', 'Bos', 'mathildebos@gmail.com', 'A2', '9405FH', '34'),
('17', 'Louise', 'Venema', 'louisevenema@gmail.com', 'A2', '9574EN', '98'),
('18', 'Joosje', 'Bakker', 'joosjebakker@gmail.com', 'A2', '9345HF', '45'),
('19', 'Veerle', 'Rozema', 'veerlerozema@gmail.com', 'A2', '2348EH', '7'),
('20', 'Marleine', 'Trenning', 'marleinetrenning@gmail.com', 'A2', '9073SN', '9'),
('21', 'Rosanne', 'Kasse', 'rosannekasse@gmail.com', 'A2', '9450EJ', '2'),
('22', 'Ilse', 'Groeneveld', 'ilsegroeneveld@gmail.com', 'A2', '9530ET', '6'),
;
```

```
INSERT INTO Address(postal_code, house_number, street)
VALUES ('9741NP', '107-9', 'Duindoornstraat'),
('9479PC', '10', 'Lageweg'),
('9718AS', '17', 'Westerkade'),
('9501AD', '29', 'Heusdenlaan'),
('9745DS', '6', 'Hanslodeizenstraat'),
('7843DS', '4', 'Abeelstraat'),
('7851XK', '5', 'Meerstraat'),
('5223LH', '6', 'Paleiskwartier'),
('1234AB', '5', 'Mariahoeve'),
('8465LN', '12', 'Harmoniestraat'),
('6325FO', '96', 'Academiestraat'),
('9621XZ', '4', 'Harroldlaan'),
('8942LM', '16', 'Albertheijnlaan'),
('3214HW', '13', 'Jumbostraat'),
('6294JK', '67', 'Dikkedirkstraat'),
('9502FI', '23', 'koffiestraat'),
('9234IN', '89', 'knopstraat'),
('9450HJ', '3', 'lampstraat'),
('9405FH', '34', 'jasweg'),
('9574EN', '98', 'laderstraat'),
('9345HF', '45', 'trapstraat'),
('2348EH', '7', 'tafelstraat'),
('9073SN', '9', 'stoelstraat'),
('9450EJ', '2', 'flesweg'),
('9530ET', '6', 'zeestraat')
;

INSERT INTO Trainer (trainer_id, first_name, surname,
email, postal_code, house_number)
VALUES ('1', 'Zendaya', 'Coleman', 'zendayacoleman@gmail.com', '7843DS',
'4'),
('2', 'Tom', 'Holland', 'tomholland@gmail.com', '7851XK', '5')
;

INSERT INTO Team (team_id, trainer_id)
VALUES ('A1', '1'),
('A2', '2')
```

```
;


INSERT INTO Reservation (reservation_id, date_time, team_id, pitch_id)
VALUES ('1', '2022-01-22 09-00-00', 'A1', '1'),
('2', '2022-01-22 10-00-00', 'A2', '5'),
('3', '2022-02-11 14-00-00', 'A2', '3'),
('4', '2022-02-27 15-00-00', 'A1', '3')
;

INSERT INTO Pitch (pitch_id, place)
VALUES ('1', 'indoor'), ('2', 'outdoor'),
('3', 'outdoor',),
('4', 'indoor'),
('5', 'outdoor')
;

INSERT INTO Typepitch(place, type_of_grass)
VALUES('indoor', 'artificial'),
('outdoor', 'natural')
;
```

Altering a table:

```
ALTER TABLE Member MODIFY COLUMN team_id varchar(4)
```

SQL Queries To Retrieve Data

Below are the SQL queries we made to retrieve data from the database. In some cases we provide two options for finding the data. Questions 1, 2 and 8 contain advanced queries.

1. **Check which members are in team A1.**

Option 1

```
SELECT m.member_id, m.first_name, m.surname
FROM Member m, Team t
WHERE t.team_id = m.team_id
AND t.team_id = 'A1'
```

Option 2

```
SELECT m.member_id, m.first_name, m.surname
FROM Member m
WHERE EXISTS
(SELECT t.team_id
FROM Team t
WHERE m.team_id = t.team_id
AND t.team_id = 'A1');
```

2. **Check which trainer trains team A1.**

Option 1 (without surname)

```
SELECT t.trainer_id
FROM Team t
WHERE t.team_id IN(
SELECT t.team_id
FROM Team t
WHERE t.team_id = 'A1');
```

Option 2 (with surname)

```
SELECT tr.trainer_id, tr.surname
FROM Trainer tr
WHERE EXISTS(
SELECT t.team_id
FROM Team t
WHERE t.team_id = 'A1'
AND t.trainer_id = tr.trainer_id);
```

3. **Get the complete address of each member.**

```
SELECT m.member_id, m.surname, a.house_number, a.street, a.postal_code
FROM Address a, Member m
WHERE m.postal_code = a.postal_code
AND m.house_number = a.house_number;
```

4. **Get the complete address of each trainer.**

```
SELECT tr.trainer_id, tr.surname, a.house_number, a.street, a.postal_code
FROM Address a, Trainer tr
WHERE tr.postal_code = a.postal_code
AND tr.house_number = a.house_number;
```

5. **Check how many members are in each team.**

```
SELECT t.team_id, COUNT(t.team_id)
FROM Member m, Team t
WHERE t.team_id = m.team_id
GROUP BY t.team_id
```

6. **Check which teams and trainers have reservations on a specific date.**

Option 1

```
SELECT r.reservation_id, t.team_id, tr.trainer_id, tr.surname
FROM Reservation r, Pitch p, team t, Trainer tr
WHERE r.date_time LIKE '2022-01-22%'
AND r.pitch_id = p.pitch_id
AND r.team_id = t.team_id
AND tr.trainer_id = t.trainer_id
```

Option 2

```
SELECT t.team_id, tr.trainer_id, tr.surname
FROM Reservation r
INNER JOIN team t ON r.team_id = t.team_id
INNER JOIN trainer tr ON t.trainer_id = tr.trainer_id
WHERE r.date_time LIKE '2022-01-22%'
```

7. **How many times have trainers made reservations in a specific month?**

```
SELECT COUNT(r.reservation_id) AS reservations_amount, tr.trainer_id, tr.surname
FROM Reservation r, Trainer tr, Team t
WHERE tr.trainer_id = t.trainer_id
```

```
AND t.team_id = r.team_id
AND r.date_time LIKE '2022-01%'
GROUP BY r.reservation_id
```

8. **Which reservations are outdoor?**

```
SELECT r.reservation_id
FROM Reservation r
WHERE EXISTS
(SELECT p.pitch_id
FROM Pitch p
WHERE r.pitch_id = p.pitch_id
AND p.place = 'outdoor')
ORDER BY r.reservation_id;
```

9. **How many reservations does each pitch have on a specific date?**

```
SELECT COUNT(r.reservation_id) AS reservations_amount, p.pitch_id, p.place
FROM Reservation r, Pitch p
WHERE r.pitch_id = p.pitch_id
AND r.date_time LIKE '2022-01-022%'
GROUP BY p.pitch_id;
```