A.C.O
Alan’s Club Organization

By Simoni Handoo

Database Management Project

Professor Alan Labouseur

A.C.O
Alan's
Club
Organization

TABLE OF CONTENTS

Executive Summary

Alan's Club Organization is a start-up organization located right across Marist College. It was recently founded by Alan and it consists of clubs that were founded by Alan, members in each club, and instructors for each club. It stores all the clubs along with their accompanying members, instructors, names of the club, start and end date, and so on. It keeps track of which instructor instructs which club in which season, and the members who are enrolled in which season. The database also consists of tables for storing the different meetings and events that are held by different clubs. The database also keeps track of the equipments needed for each club. A.C.O database is helpful since we can use this to track which members are in which club, or if they are in multiple clubs. It also keeps track of all the meetings and events that were held within each club and who attended those meetings/events to keep track of members who are loyal. This database has a separate table for confidential information for both members and instructors which includes how much they pay/earn, when their expiration/contract expires and what type of payment method they use.

Alan, who is the creator of this database, is also an instructor at this organization. Hence, a role called 'admin' has been created for Alan in the database since he is the one who can access all information. In this database, a member can only be a member, he/she cannot be an instructor if they are enrolled as a member.

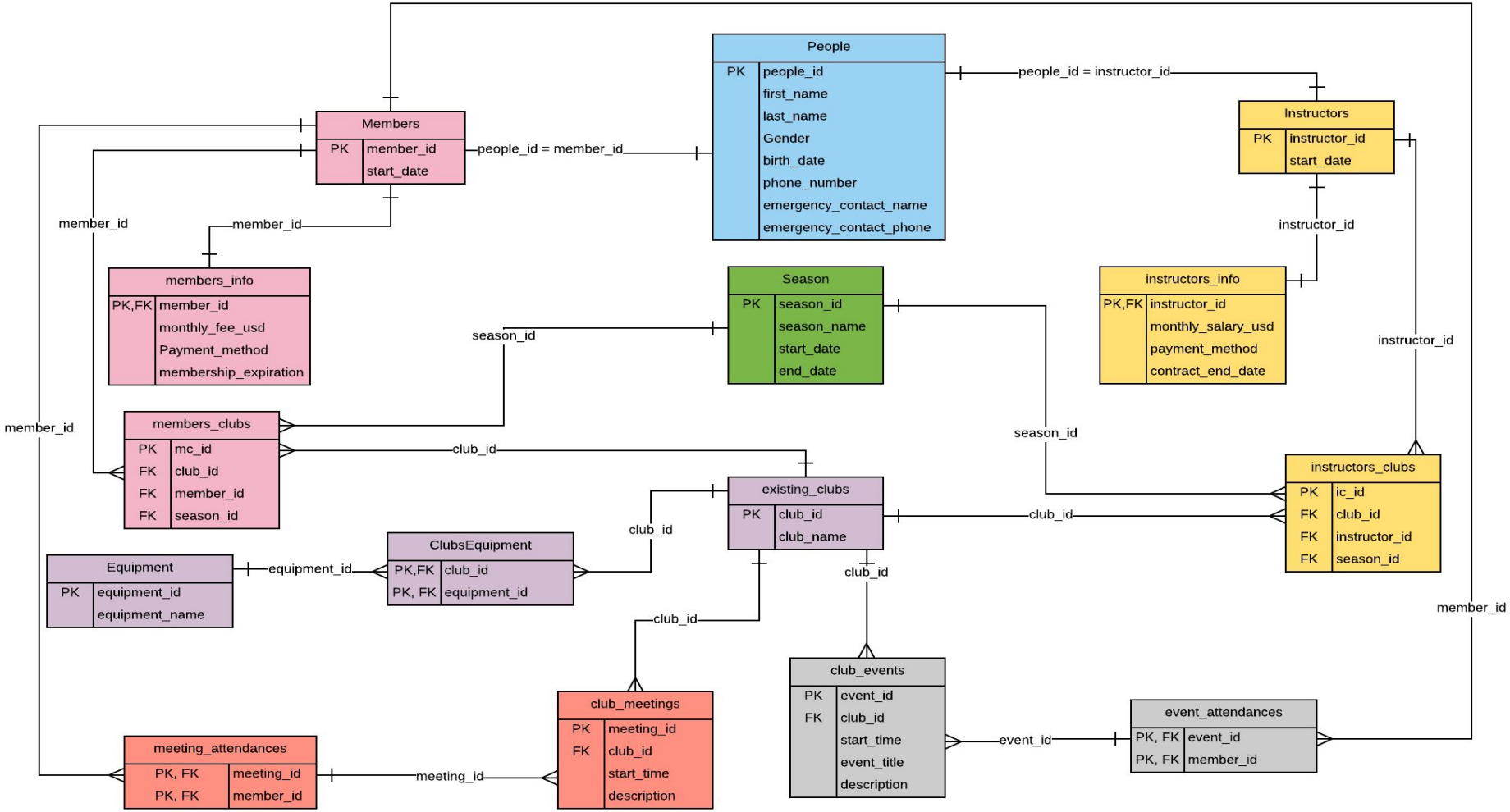
People
PK people_id
first_name
last_name
Gender
birth_date
phone_number
emergency_contact_name
emergency_contact_phone
people_id = instructor_id
Members
PK member_id
start_date
people_id = member_id
Instructors
PK instructor_id
start_date
member_id
member_id
instructor_id
members_info
PK,FK member_id
monthly_fee_usd
Payment_method
membership_expiration
Season
PK season_id
season_name
start_date
end_date
instructors_info
PK,FK instructor_id
monthly_salary_usd
payment_method
contract_end_date
season_id
instructor_id
member_id
members_clubs
PK mc_id
FK club_id
FK member_id
FK season_id
club_id
season_id
existing_clubs
PK club_id
club_name
instructors_clubs
PK ic_id
FK club_id
FK instructor_id
FK season_id
club_id
club_id
Equipment
PK equipment_id
equipment_name
equipment_id
ClubsEquipment
PK,FK club_id
PK, FK equipment_id
club_id
club_id
club_id
member_id
club_meetings
PK meeting_id
FK club_id
start_time
description
club_events
PK event_id
FK club_id
start_time
event_title
description
event_id
event_attendances
PK, FK event_id
PK, FK member_id
meeting_attendances
PK, FK meeting_id
PK, FK member_id
meeting_id

Table 1: People

```
CREATE TYPE gen AS ENUM ('f', 'm');
CREATE TABLE IF NOT EXISTS People (
people_id SERIAL                    NOT NULL PRIMARY KEY,
first_name VARCHAR(100)             NOT NULL,
last_name VARCHAR(100)              NOT NULL,
gender                              gen,
birth_date                          NOT NULL,
phone_number                        char(15) NOT NULL,
Emergency_contact_name              char(100) NOT NULL,
emergency_contact_phone             char(15) NOT NULL
);
```

Functional Dependencies:

people_id → first_name, last_name, gender, phone_number, emergency_contact_name, emergency_contact_phone

people_id integer	first_name character varying(100)	last_name character varying(100)	gender gen	birth_date date	phone_number character(15)	emergency_contact_name character varying(100)	emergency_contact_phone character(15)
1	Alan	Labouseur	m	1987-01-23	8456728902	Tien	8796783672
2	Bob	Mendez	m	2001-03-27	7682678290	Alessia	9167882678
3	Grace	Helbig	f	1997-01-24	2278993678	Kiana	2987890825
4	Emily	Farell	f	2000-01-24	7256789256	Sammy	7892456728
5	Josiah	Conway	m	1994-02-04	4567892678	Manny	8926789035
6	Matt	Germano	m	1992-04-15	6789997829	Molly	8678897826
7	Samantha	Payne	f	2002-03-16	3890786278	Zack	5627890267
8	David	Borak	m	1989-10-20	3678929036	Robert	6708902836
9	Nathaniel	Vanderbilt	m	1994-05-30	4567892568	Darcy	7835678920
10	Blaire	Wardolf	f	2001-11-19	89034567829	Nat	4785619036
11	Dan	Humphrey	m	1997-06-18	3787828378	Aaron	9278902847
12	Karen	Fillipeli	f	1995-09-14	5603892728	Tara	8934567836
13	Pam	Beasley	f	2002-08-12	3678926738	Elsa	5693782256
14	Jim	Halpert	m	1998-10-26	7356784937	Charlie	3490896678
15	Dwight	Schrute	m	1991-07-21	2126785567	Bryant	9125678889
16	Jeff	Damon	m	1995-12-18	8452406178	Gary	7467778926
17	Rupert	Fernandez	m	1999-07-05	8452873345	Elsa	7467824490
18	Hannah	Marin	f	1996-09-01	9087895678	Ashley	6458890345
19	Wassila	George	f	1997-01-24	7689906234	Elliot	8785563345

Table 2: Members

```
CREATE TABLE IF NOT EXISTS members (
member_id          INT references people(people_id) PRIMARY KEY,
start_date         date NOT NULL
);
```

Functional Dependencies:

member_id → start_date

member_id integer	start_date date
2	2014-08-12
3	2017-01-01
5	2015-04-09
6	2014-08-09
8	2015-10-05
9	2014-07-17
10	2015-02-09
12	2017-01-23
13	2016-06-05
14	2014-11-04
15	2014-10-05
16	2016-03-19
17	2015-10-05

Table 3: Members Information

```
CREATE TYPE payment AS ENUM ('credit', 'debit', 'cash');
CREATE TABLE IF NOT EXISTS members_info (
member_id              INT references members(member_id),
monthly_fee_usd        money NOT NULL,
payment_method         payment,
membership_expiration date NOT NULL,
CHECK (monthly_fee_usd >= '20')
);
```

Functional Dependencies:

member_id → monthly_fee_usd, payment_method, membership_expiration

member_id integer	monthly_fee_usd money	payment_method payment	membership_expiration date
2	$35.00	credit	2015-08-12
3	$30.00	credit	2018-01-01
5	$40.00	cash	2016-04-09
6	$40.00	debit	2015-08-09
8	$35.00	cash	2016-10-05
9	$30.00	credit	2015-07-17
10	$35.00	debit	2016-02-09
12	$45.00	cash	2018-01-23
13	$45.00	credit	2017-06-05
14	$30.00	debit	2015-11-04
15	$30.00	credit	2015-10-05
16	$40.00	credit	2017-03-19
17	$35.00	cash	2016-10-05

Table 4: Instructors

```
CREATE TABLE IF NOT EXISTS instructors (
instructor_id       INT references people(people_id) PRIMARY KEY,
start_date          date NOT NULL
);
```

Functional Dependencies:

instructor_id → start_date

instructor_id integer	start_date date
1	2014-11-12
4	2016-08-06
7	2015-04-07
11	2014-10-18
18	2015-05-16
19	2014-09-05
20	2015-01-01

Table 5: Instructor Information

```
CREATE TYPE payment AS ENUM ('credit', 'debit', 'cash');
CREATE TABLE IF NOT EXISTS instructors_info (
instructor_id          INT references instructors(instructor_id),
monthly_salary_usd     INT NOT NULL,
payment_method         payment,
contract_end_date      date NOT NULL,
CHECK (monthly_salary_usd >= '250')
);
```

Functional Dependencies:

instructor_id → monthly_salary_usd, payment_method, contract_end_date

instructor_id integer	monthly_salary_usd money	payment_method payment	contract_end_date date
1	$300.00	credit	2015-11-12
4	$350.00	debit	2017-08-06
7	$350.00	credit	2016-04-07
11	$300.00	cash	2015-10-18
18	$350.00	debit	2016-05-16
19	$300.00	debit	2015-09-05
20	$350.00	credit	2016-01-01

Table 9: Members_Clubs

Usefulness: This table keeps track of which which member is part of which club(s)

```
CREATE TABLE IF NOT EXISTS members_clubs (
mc_id           serial NOT NULL PRIMARY KEY,
member_id       INT REFERENCES members(member_id) NOT NULL,
club_id         INT REFERENCES existing_clubs(club_id) NOT NULL,
season_id       INT REFERENCES Season(season_id) NOT NULL
);
```

mc_id integer	member_id integer	club_id integer	season_id integer
1	2	3	4
2	3	1	1
3	5	4	6
4	6	2	6
5	8	3	5
6	9	6	5
7	10	5	9
8	12	8	10
9	13	7	7
10	14	1	5
11	15	2	9
12	16	8	10
13	17	5	9
14	17	3	4
15	14	7	5

Functional Dependencies:

mc_id → member_id, club_id, season_id

Table 7: Instuctors_Clubs

Usefulness: This table keeps track of which instructor(s) teach which club(s)

```
CREATE TABLE IF NOT EXISTS instructors_clubs (
ic_id                serial NOT NULL PRIMARY KEY,
instructor_id        INT REFERENCES instructors(instructor_id) NOT NULL,
club_id              INT REFERENCES existing_clubs(club_id) NOT NULL,
season_id            INT REFERENCES Season(season_id) NOT NULL
);
```

ic_id integer	instructor_id integer	club_id integer	season_id integer
1	1	3	1
2	4	1	4
3	7	4	2
4	11	8	5
5	18	7	3
6	19	2	9
7	20	8	9

Functional Dependencies:

ic_id → member_id, club_id, season_id

Table 6: Season

```
CREATE TABLE IF NOT EXISTS Season (
season_id           SERIAL NOT NULL PRIMARY KEY,
season_name         varchar(100) NOT NULL,
start_date          date NOT NULL,
end_date            date NOT NULL
);
```

Functional Dependencies:

Season_id → season_name, start_date, end_date

season_id integer	season_name character varying(100)	start_date date	end_date date
1	Spring2014	2014-01-01	2015-05-01
2	Summer2014	2014-05-01	2014-08-25
3	Fall2014	2015-01-09	2014-12-23
4	Spring2015	2015-01-01	2015-05-01
5	Summer2015	2015-05-10	2016-08-25
6	Fall2015	2015-09-01	2015-12-23
7	Spring2016	2016-01-01	2016-05-01
8	Summer2016	2016-05-10	2016-08-25
9	Fall2016	2016-09-01	2016-12-22
10	Spring2017	2017-01-01	2017-05-01

Table 8: Clubs

```
CREATE TABLE IF NOT EXISTS existing_clubs (
club_id         SERIAL NOT NULL PRIMARY KEY,
club_name       VARCHAR(255) NOT NULL
);
```

Functional Dependencies:

club_id → club_name

club_id integer	club_name character varying(255)
1	Swimming
2	Basketball
3	Soccer
4	Ballet
5	Table Tennis
6	Badminton
7	Tennis
8	Hip Hop Dance

Table 10: club_meetings

Usefulness: Keeps track of which club is holding what kind of meeting

```
CREATE TABLE IF NOT EXISTS club_meetings(
meeting_id          SERIAL NOT NULL PRIMARY KEY,
club_id             INT REFERENCES existing_clubs(club_id) NOT NULL,
start_time          timestamp with time zone NOT NULL,
description         VARCHAR(255)
);
```

Functional Dependencies:

meeting_id → club_id, start_time, description

meeting_id integer	club_id integer	start_time timestamp with time zone	description character varying(255)
1	2	2014-09-01 17:00:00-04	Competition Details
2	4	2015-06-01 11:00:00-04	Pratice before Finale
3	5	2016-11-02 00:00:00-04	Choosing Partners
4	1	2017-01-10 00:00:00-05	Under 15 Race details
5	7	2016-12-01 00:00:00-05	Upcoming Tournament Details
6	8	2015-03-24 00:00:00-04	Routine Practice Deatils

Table 11: club_events

Usefulness: Keeps track of which club is holding which event

```
CREATE TABLE IF NOT EXISTS club_events (
event_id            SERIAL NOT NULL PRIMARY KEY,
club_id             INT REFERENCES existing_clubs(club_id) NOT NULL,
start_time          TIMESTAMP with time zone NOT NULL,
event_title         VARCHAR(255),
description         VARCHAR(255)
);
```

Functional Dependencies:

event_id → club_id, start_time,

event_id integer	club_id integer	start_time timestamp with time zone	event_title character varying(255)	description character varying(255)
1	8	2016-11-23 19:00:00-05	Ball 2016	A formal ball for all the members of the Hip Hop Dance Group
2	4	2017-01-23 17:00:00-05	Appreciation Day	Appreciating your peers
3	1	2015-08-18 15:30:00-04	FreeStyle 100m	Race
4	2	2014-10-03 09:00:00-04	Boys Tournament	In-house Tournament Boys
5	6	2016-05-17 16:00:00-04	Meet and Greet	<NULL>

Table 12: meeting_attendances

Usefulness: Keeps track of which member(s) attended a particular club meeting (Associative entity)

```
CREATE TABLE IF NOT EXISTS meeting_attendances (
meeting_id        INT REFERENCES club_meetings(meeting_id) NOT NULL,
member_id         INT REFERENCES members(member_id) NOT NULL,
PRIMARY KEY       (meeting_id, member_id)
);
```

Functional Dependencies:

meeting_id, member_id → ____

There are no non-key attributes, therefore there are no dependencies for this table.

meeting_id integer	member_id integer
1	6
1	15
2	5
3	10
3	17
4	3
4	14
5	13
5	14
6	12
5	16

Table 13: event_attendances

Usefulness: Keeps track of which member(s) attended a specific club event (Associative entity)

```
CREATE TABLE IF NOT EXISTS event_attendances(
event_id        INT REFERENCES club_events(event_id) NOT NULL,
member_id       INT REFERENCES members(member_id) NOT NULL,
primary key     (event_id, member_id)
);
```

Functional Dependencies:

event_id, member_id → ____

There are no non-key attributes, therefore there are no dependencies for this table.

event_id integer	member_id integer
1	12
1	16
2	5
3	3
3	14
4	6
4	15
5	9

Table 14: equipment

```
CREATE TABLE IF NOT EXISTS equipment (
equipment_id            INT NOT NULL UNIQUE PRIMARY KEY,
equipment_name          VARCHAR(255) NOT NULL
);
```

Functional Dependencies:

equipment_id → equipment_name

equipment_id integer	equipment_name character varying(255)
1	swimming goggles
2	swimming cap
3	basketball
4	soccer ball
5	shin pads
6	Speakers
7	table tennis racket
8	ping pong ball
9	badminton racket
10	shuttlecock
11	tennis racket
12	tennis ball green

Table 15: ClubsEquipment

Usefulness: Keeps track of which club needs which equipment (Associative entity)

```
CREATE TABLE IF NOT EXISTS ClubsEquipment (
club_id             INT REFERENCES existing_clubs(club_id) NOT NULL,
equipment_id        INT REFERENCES equipment(equipment_id) NOT NULL,
PRIMARY KEY        (club_id, equipment_id)

);
```

Functional Dependencies:

club_id, equipment_id → ____

There are no non-key attributes, therefore there are no dependencies for this table.

club_id integer	equipment_id integer
1	1
1	2
2	3
3	4
3	5
5	7
5	8
6	9
6	10
7	11
7	12
8	6

REPORTS

To check which instructors are paid more than $300

```
SELECT instructors_info.monthly_salary_usd,
People.first_name, People.last_name
FROM instructors_info
INNER JOIN Instructors
ON Instructors.instructor_id =
instructors_info.instructor_id
INNER JOIN People
ON People.people_id = Instructors.instructor_id
GROUP BY People.first_name,
instructors_info.monthly_salary_usd,
People.last_name
HAVING monthly_salary_usd > '300';
```

monthly_salary_usd money	first_name character varying(100)	last_name character varying(100)
$350.00	Emily	Farell
$350.00	Hannah	Marin
$350.00	Katie	Zulets
$350.00	Samantha	Payne

This query returns the percentage of people under 21

```
SELECT TRUNC(
              CAST(
                   (SELECT COUNT(people_id) as count
                    FROM People
                    WHERE date_part('year', age(People.birth_date)) < 21
                    ) as decimal(5,2)
                      ) / (SELECT COUNT(people_id) as total
                           FROM People
                    ) * 100
              ) as Underage;
```

underage numeric
60

60% of the people in this database are under 21 years of age

This query returns the percentage of male and female people in the database.

```
SELECT Gender, (Count(Gender)* 100 / (Select Count(*)
                                      FROM People))
as Percentage
FROM People
GROUP BY Gender;
```

gender gen	percentage bigint
m	55
f	45

INNER AND OUTER JOINS

To find out which instructor instructed which Season

```
SELECT first_name, season_name
FROM People
INNER JOIN Instructors
ON Instructors.instructor_id =
People.people_id
INNER JOIN instructors_clubs
ON Instructors.instructor_id =
instructors_clubs.instructor_id
INNER JOIN Season
ON instructors_clubs.season_id =
Season.season_id;
```

first_name character varying(100)	season_name character varying(100)
Alan	Summer2014
Emily	Summer2015
Samantha	Fall2014
Dan	Fall2015
Hannah	Spring2015
Wassila	Spring2017
Katie	Spring2017

Which member pays how much, along with their start date and expiration

```
SELECT first_name, start_date,
membership_expiration, monthly_fee_usd
FROM People
INNER JOIN Members
ON People.people_id = Members.member_id
INNER JOIN members_info
ON Members.member_id =
members_info.member_id
ORDER BY monthly_fee_usd;
```

first_name character varying(100)	start_date date	membership_expiration date	monthly_fee_usd money
Bob	2014-08-12	2015-08-12	$35.00
Grace	2017-01-01	2018-01-01	$30.00
Josiah	2015-04-09	2016-04-09	$40.00
Matt	2014-08-09	2015-08-09	$40.00
David	2015-10-05	2016-10-05	$35.00
Nathaniel	2014-07-17	2015-07-17	$30.00
Blaire	2015-02-09	2016-02-09	$35.00
Karen	2017-01-23	2018-01-23	$45.00
Pam	2016-06-05	2017-06-05	$45.00
Jim	2014-11-04	2015-11-04	$30.00
Dwight	2014-10-05	2015-10-05	$30.00
Jeff	2016-03-19	2017-03-19	$40.00
Rupert	2015-10-05	2016-10-05	$35.00

To determine which club needs what equipment, or if there is a club that does not require equipment

```
SELECT equipment_name, club_name
FROM existing_clubs
FULL OUTER JOIN ClubsEquipment
ON existing_clubs.club_id =
ClubsEquipment.club_id
FULL OUTER JOIN Equipment
ON Equipment.equipment_id =
ClubsEquipment.equipment_id
ORDER BY club_name;
```

club_name character varying(255)	equipment_name character varying(255)
Swimming	swimming goggles
Swimming	swimming cap
Basketball	basketball
Soccer	soccer ball
Soccer	shin pads
Table Tennis	table tennis racket
Table Tennis	ping pong ball
Badminton	badminton racket
Badminton	shuttlecock
Tennis	tennis racket
Tennis	tennis ball green
Hip Hop Dance	Speakers
Ballet	<NULL>

To determine who are the members in the database and who are not, along with the club(s) they are registered into

```
SELECT first_name, club_name
FROM People
LEFT OUTER JOIN Members
ON People.people_id = Members.member_id
FULL OUTER JOIN meeting_attendances
ON Members.member_id =
meeting_attendances.member_id
FULL OUTER JOIN club_meetings
ON meeting_attendances.meeting_id =
club_meetings.meeting_id
FULL OUTER JOIN existing_clubs
ON club_meetings.club_id =
existing_clubs.club_id;
```

first_name character varying(100)	club_name character varying(255)
Matt	Basketball
Dwight	Basketball
Josiah	Ballet
Blaire	Table Tennis
Rupert	Table Tennis
Grace	Swimming
Jim	Swimming
Pam	Tennis
Jim	Tennis
Karen	Hip Hop Dance
Jeff	Tennis
Samantha	<NULL>
Alan	<NULL>
Emily	<NULL>
Wassila	<NULL>
Hannah	<NULL>
Dan	<NULL>
Katie	<NULL>
Bob	<NULL>
David	<NULL>
Nathaniel	<NULL>

The NULL values mean that the person is NOT a member

To show if people are both members and instructors. This query is important since we want to make sure that a person is either a member of an instructor, he/she cannot be both as per current rules of the organization. Therefore, this query helps us keep track of this.

```
SELECT first_name, last_name, instructor_id,
member_id
FROM Instructors
LEFT OUTER JOIN People
ON Instructors.instructor_id =
People.people_id
LEFT OUTER JOIN Members
ON Members.member_id = People.people_id
ORDER BY member_id;
```

first_name character varying(100)	last_name character varying(100)	instructor_id integer	member_id integer
Alan	Labouseur	1	<NULL>
Emily	Farell	4	<NULL>
Samantha	Payne	7	<NULL>
Dan	Humphrey	11	<NULL>
Hannah	Marin	18	<NULL>
Wassila	George	19	<NULL>
Katie	Zulets	20	<NULL>

SUBQUERIES

This subquery returns first and last name of the member who attended the club meeting “Choosing Partners” (with meeting_id =3). This query can be used to find out which member has attended which meetings in order to keep track of the meetings that take place.

```
SELECT first_name, last_name
FROM People
WHERE people_id IN (SELECT member_id
                    FROM Members
                    WHERE member_id IN (SELECT meeting_id
                                        FROM meeting_attendances
                                        WHERE meeting_id = 3)
                         );
```

first_name character varying(100)	last_name character varying(100)
Grace	Helbig

This subquery returns the first and last name of the instructors who are currently an instructor for the ongoing season (Spring2017). Using this query, we can find out which member was part of which Season if we ever want to do a look up.

```
SELECT first_name, last_name
FROM People
WHERE people_id IN (SELECT instructor_id
                    FROM Instructors
                    WHERE instructor_id IN (SELECT instructor_id
                                            FROM instructors_clubs
                                            WHERE season_id = 9 )
                   );
```

first_name character varying(100)	last_name character varying(100)
Wassila	George
Katie	Zulets

VIEWS

Members information View

```
CREATE OR REPLACE VIEW
membersInformation AS
SELECT     p.first_name,
           p.last_name,
           p.gender,
           p.phone_number,
           p.emergency_contact_name,
           p.emergency_contact_phone
FROM People p, Members m
WHERE p.people_id = m.member_id
ORDER BY p.last_name DESC;
```

SELECT * FROM membersInformation;

first_name character varying(100)	last_name character varying(100)	gender gen	birth_date date	phone_number character(15)	emergency_contact_name character varying(100)	emergency_contact_phone character(15)
Blaire	Wardolf	f	2001-11-19	89034567829	Nat	4785619036
Nathaniel	Vanderbilt	m	1994-05-30	4567892568	Darcy	7835678920
Dwight	Schrute	m	1991-07-21	2126785567	Bryant	9125678889
Bob	Mendez	m	2001-03-27	7682678290	Alessia	9167882678
Grace	Helbig	f	1997-01-24	2278993678	Kiana	2987890825
Jim	Halpert	m	1998-10-26	7356784937	Charlie	3490896678
Matt	Germano	m	1992-04-15	6789997829	Molly	8678897826
Karen	Fillipeli	f	1995-09-14	5603892728	Tara	8934567836
Rupert	Fernandez	m	1999-07-05	8452873345	Elsa	7467824490
Jeff	Damon	m	1995-12-18	8452406178	Gary	7467778926
Josiah	Conway	m	1994-02-04	4567892678	Manny	8926789035
David	Borak	m	1989-10-20	3678929036	Robert	6708902836
Pam	Beasley	f	2002-08-12	3678926738	Elsa	5693782256

Instructor's information View

```
CREATE OR REPLACE VIEW instructorsInformation AS
SELECT      p.first_name,
            p.last_name,
            p.gender,
            p.birth_date,
            p.phone_number,
            p.emergency_contact_name,
            p.emergency_contact_phone
FROM People p, Instructors i
WHERE p.people_id = i.instructor_id
ORDER BY p.last_name DESC
```

SELECT * FROM instructorsInformation;

first_name character varying(100)	last_name character varying(100)	gender gen	birth_date date	phone_number character(15)	emergency_contact_name character varying(100)	emergency_contact_phone character(15)
Katie	Zulets	f	1998-01-29	9236678892	Diana	7724456666
Samantha	Payne	f	2002-03-16	3890786278	Zack	5627890267
Hannah	Marin	f	1996-09-01	9087895678	Ashley	6458890345
Alan	Labouseur	m	1987-01-23	8456728902	Tien	8796783672
Dan	Humphrey	m	1997-06-18	3787828378	Aaron	9278902847
Wassila	George	f	1997-01-24	7689906234	Elliot	8785563345
Emily	Farell	f	2000-01-24	7256789256	Sammy	7892456728

This View displays which clubs are ongoing right now (present time)

```
DROP VIEW IF EXISTS ClubsThisSeason;
CREATE OR REPLACE VIEW ClubsThisSeason AS
SELECT distinct club_name
FROM existing_clubs, members_clubs, Season
WHERE existing_clubs.club_id = members_clubs.club_id
AND members_clubs.season_id = season.season_id
AND NOW() > season.start_date
AND NOW() < season.end_date
```

club_name character varying(255)
Basketball
Table Tennis

SELECT * FROM ClubsThisSeason;

This view is useful as it only displays information about the Equipments that are needed for each club

```
DROP VIEW IF EXISTS ClubEquipments;
CREATE OR REPLACE VIEW ClubEquipments AS
SELECT club_name, equipment_name
FROM existing_clubs
INNER JOIN ClubsEquipment
ON existing_clubs.club_id = ClubsEquipment.club_id
INNER JOIN Equipment
ON Equipment.equipment_id = ClubsEquipment.equipment_id;
```

SELECT * FROM ClubEquipments;

club_name character varying(255)	equipment_name character varying(255)
Swimming	swimming goggles
Swimming	swimming cap
Basketball	basketball
Soccer	soccer ball
Soccer	shin pads
Table Tennis	table tennis racket
Table Tennis	ping pong ball
Badminton	badminton racket
Badminton	shuttlecock
Tennis	tennis racket
Tennis	tennis ball green

This view displays which member pays with a credit card. This query can be altered to see who pays with a debit card or who pays with cash.

```
DROP VIEW IF EXISTS ptyp;
CREATE OR REPLACE VIEW ptype AS
SELECT first_name, payment_method
FROM People
INNER JOIN Members
ON people.people_id = Members.member_id
INNER JOIN members_info
ON Members.member_id = members_info.member_id
WHERE payment_method = 'credit';
```

SELECT * FROM ptype;

first_name character varying(100)	payment_method payment
Bob	credit
Grace	credit
Nathaniel	credit
Pam	credit
Dwight	credit
Jeff	credit

This view displays which member(s) was/were a part of a club in Fall2015. This query view can be used to determine which member was part of which club at a certain point

```
CREATE OR REPLACE VIEW membersFall2015 AS
SELECT first_name, last_name, season_name
FROM People
INNER JOIN Members
ON People.people_id = Members.people_id
INNER JOIN members_clubs
ON Members.member_id = members_clubs.member_id
INNER JOIN Season
ON Season.season_id = members_clubs.season_id
WHERE season_name = 'Fall2015';
```

SELECT * FROM membersFall2015;

first_name character varying(100)	last_name character varying(100)	season_name character varying(100)
David	Borak	Fall2015
Nathaniel	Vanderbilt	Fall2015
Jim	Halpert	Fall2015

;

STORED PROCEDURES

This stored procedure returns the number of meeting(s) a member attended for a particular club in a particular semester.

```
DROP FUNCTION IF EXISTS memberForMeetings(int, int, int);
CREATE FUNCTION memberForMeetings (club_id INT, season_id INT, member_id INT)
RETURNS BIGINT as $$
SELECT count(cm.meeting_id) as Attendances
FROM club_meetings cm, meeting_attendances ma, Season
WHERE cm.club_id = $1
AND cm.start_time >= Season.start_date
AND cm.start_time <= Season.end_date
AND season.season_id = $2
AND ma.member_id = $3
AND cm.meeting_id = ma.meeting_id;
$$ language 'sql';
```

	totalmeetingsbyseason bigint
1	0

SELECT * FROM MemberForEvents(3,1,4);

This Stored Procedure returns the number of event(s) a member attended for a particular club in a particular season.

```
DROP FUNCTION IF EXISTS MemberForEvents(int, int, int);
CREATE FUNCTION MemberForEvents (event_id int, season_id int, member_id
int)
RETURNS BIGINT as $$
SELECT count(ce.event_id) as Attendances
FROM club_events ce, event_attendances ea, Season
WHERE ce.start_time >= season.start_date
AND ce.start_time <= season.end_date
AND season.season_id = $2
AND ea.member_id = $3
AND ce.event_id = ea.event_id
$$ language 'sql';
```

```
SELECT * FROM memberForEvents(3,2,4);
```

	memberforevents bigint
1	0

This Stored Procedure basically awards points to a member that has been a loyal Client at the A.C.O. If a member has been part of a lot of club meetings for different clubs, this stored procedure will return a boolean value. If it returns true, this member will gain points, which are basically future benefits and discounts (such as a free 1 month membership).. If a member gains enough points, they will be rewarded.

```
DROP FUNCTION IF EXISTS rewardPoints(int, int, int);
CREATE FUNCTION rewardPoints(season_id int, club_id int, member_id int)
RETURNS BOOLEAN as $$
SELECT (100 * memberForMeetings(members_clubs.club_id, members_clubs.season_id,
members_clubs.member_id) / totalMeetingsBySeason(members_clubs.club_id,
members_clubs.season_id) >= 50
    AND memberForMeetings(members_clubs.club_id, members_clubs.season_id,
members_clubs.member_id)/ totalMeetingsBySeason(members_clubs.club_id,
members_clubs.season_id) >= 50)
FROM members_clubs, season
WHERE members_clubs.season_id = $1
AND members_clubs.club_id = $1
AND members_clubs.member_id = $2;
$$ language 'sql';
SELECT * FROM rewardPoints(1,1,2);
```

This stored procedure returns the total number of records that are in the People's table

```
CREATE OR REPLACE FUNCTION totalRecords()
RETURNS integer AS $total$
declare
    total integer;
BEGIN
   SELECT count(*) into total FROM People;
   RETURN total;
END;
$total$ LANGUAGE plpgsql;

select totalrecords();
```

totalrecords integer
20

This function returns the cost to Alan's Club Organization for Instructors

```
CREATE OR REPLACE FUNCTION totalSalary()
RETURNS money AS $salary$
declare
     salary money;
BEGIN
   SELECT SUM(monthly_salary_usd) into salary FROM instructors_info;
   RETURN salary;
END;
$salary$ LANGUAGE plpgsql;

SELECT totalSalary();
```

totalsalary money
$2,300.00

This stored procedure returns the amount of profit that A.C.O is making through membership

```
CREATE OR REPLACE FUNCTION totalFees()
RETURNS money AS $Fee$
declare
     Fee money;
BEGIN
   SELECT SUM(monthly_fee_usd) into Fee FROM members_info;
   RETURN Fee;
END;
$Fee$ LANGUAGE plpgsql;


select totalFees() - totalSalary() AS Profit;
```

profit money
$227.00

TRIGGERS

New member who decided to join a club must be managed by this database separately using this trigger. This example triggers on a new member entry being created.

```
CREATE OR REPLACE FUNCTION new_member ()
RETURNS trigger AS $$
BEGIN
     IF NEW.is_member = true THEN
     INSERT INTO Members VALUES (NEW.member_id, NEW.start_date);
     END IF;
     RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER add_member
AFTER INSERT OR UPDATE ON People
FOR EACH ROW
EXECUTE PROCEDURE new_member();
```

New instructor(s) who have been hired by Alan to instruct a club must be managed by this database separately using this trigger. This example triggers on a new instructor entry being created.

```
CREATE OR REPLACE FUNCTION new_instructor ()
RETURNS trigger AS $$
BEGIN
     IF NEW.is_instructor = true THEN
     INSERT INTO Instructors VALUES (NEW.instructor_id, NEW.start_date);
     END IF;
     RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```
CREATE TRIGGER add_member
AFTER INSERT OR UPDATE ON People
FOR EACH ROW
EXECUTE PROCEDURE new_instructor();
```

This trigger checks that the end date comes after the start date when inserting into or updating the Season table. If the end date is entered before the start date, it throws an exception.

```
CREATE OR REPLACE FUNCTION date_order() RETURNS trigger
AS $date_order$
BEGIN
     IF NEW.start_date > NEW.end_date THEN
          RAISE EXCEPTION 'end date % should come after
          start_date %.', NEW.start_date, NEW.end_date;
     END IF;
     RETURN NEW;
END;
$date_order$ LANGUAGE plpgsql;

    CREATE TRIGGER check_season_dates
     BEFORE INSERT or UPDATE
     ON season
     FOR EACH ROW EXECUTE PROCEDURE date_order();
```

SECURITY IMPLEMENTATION

GRANT FOR ADMINS (main admin, equipment admin and events admin)

```
CREATE role
databaseadmin;
GRANT ALL ON ALL TABLES
IN SCHEMA PUBLIC
TO databaseadmin;
```

```
CREATE role EquipmentAdmin;
GRANT SELECT, INSERT, UPDATE,
DELETE ON Equipment,
ClubsEquipment, existing_clubs
TO EquipmentAdmin;
```

```
CREATE role eventsadmin;
GRANT SELECT, INSERT, UPDATE,
DELETE ON club_meetings,
club_events,
meeting_attendances,
event_attendances
TO eventsadmin;
```

GRANT FOR Instructors and Members

```
CREATE ROLE instructors;
GRANT SELECT, INSERT, UPDATE ON
Members, members_clubs, club_events,
club_meetings, ClubsEquipment,
Season, meeting_attendances,
event_attendances, existing_clubs
TO instructors;
```

```
CREATE ROLE members;
GRANT SELECT ON members, members_clubs,
club_events, Season, club_meetings,
meeting_attendances, event_attendances,
ClubsEquipment, existing_clubs
TO members;
```

REVOKE (no one apart from the admin should have access to the following columns)

```
REVOKE ALL PRIVILEGES ON People FROM Members;

REVOKE ALL PRIVILEGES ON members_info FROM Members;

REVOKE ALL PRIVILEGES ON Instructors FROM Members;

REVOKE ALL PRIVILEGES ON members_info FROM Instructors;

REVOKE ALL PRIVILEGES ON instructors_clubs FROM Members;

REVOKE ALL PRIVILEGES ON instructors_info FROM Members;

REVOKE ALL PRIVILEGES ON event_attendances FROM Members;

REVOKE ALL PRIVILEGES ON meeting_attendances  FROM Members;
```

Implementation Notes and Known Problems

Implementation Notes:

- The purpose of this database is to allow Alan's Club Organization to run smoothly. As the club organization becomes more known/popular, people are going to enroll themselves in different clubs. We need to keep track of who is enrolled in which club, how much they are paying, when their membership expires, etc.
- This database not only keeps track of assigning a club to a member, but it also keeps track of members' and instructors' personal information (payment method, how much they pay, etc.)
- Another purpose of the database is to overlook the equipments each club needs. The organization needs to be informed about getting any other types of equipment that might be needed by a certain club.
- **TEST DATA:** The test data that was added in this database is dummy data (randomly generated).

Known Problems:

- You cannot create a view of members and instructors simultaneously from the People's table.
- There is no relationship between Season and Members or Season and Instructors, which means we cannot run a ***FAST*** query that would return which Member was enrolled in which season.
- To find out which member pays how much for a club, we have to go through 4 tables (members_info → members → meeting_attendances → club_meetings → existing_clubs). This is not very efficient since we shouldn't have to go through club_meetings table if we are just trying to find out which member pays how much for which club, there should be a direct relationship (link) between the two entities.

Future Enhancements

- I would probably make a few changes in my database. First of all, I would create a table called “Benefits”. I would add attributes such as “reward_points” and “num_of_clubs” which would contain how many clubs a member is enrolled in. If a member is enrolled in a lot of clubs, he or she will gain benefits such as discounts, a free 1 month membership etc. This would attract more potential clients. This would also serve as a great marketing/advertising scheme.
- Secondly, I would create a table for guests. The “Guests” table would contain attributes such as “guest_name” and “hourly_fee”. The Guests table is for people who would like to attend a club on the spot. A guest could be a family member or a friend of one of the enrolled members. Guests can use the club services but they would not have access to club benefits. For example, the Guests table would have no relationship with the “reward_points” table since it is not applicable to them.
- I would add a table for “payment_transactions” so that it would be easier to keep track of
- Thirdly, to fix the known problems that exist in the database, I will try to reduce the number of tables it takes to return a query for better results. For example, to get how much a member pays for a certain club, I would like to reduce the number of queries the database has to go through and try to create an associative entity which would link the 2 tables together.

THE END