Групповой проект.

GymPlusPlus Database Management System.

Кремнева Майя, Макаров Илья, Рожковая Ксения, Чентырев Роман.

Описание.

GymPlusPlus DMS - это готовая система управления данными, которая поможет раскачать ваш фитнес-клуб до невероятных размеров! Наша СУБД позволяет хранить структурированные данные о посетителях, сотрудниках, услугах, оборудовании и расписании фитнес-клуба или спортивного зала, а также изменять их и управлять ими. GymPlusPlus, созданная на основе PostgreSQL, автоматизирует систему управления вашим фитнес-клубом, расширяет возможности для продуктовой и бизнес аналитики и позволяет не бояться за сохранность ценных данных. Пожмите вашу первую сотку миллионов вместе с нашей СУБД!

Пользователи.

Посетители, менеджмент, администраторы ресепшна, тренеры, системные администраторы.

Пользователи и их потребности.

1. **Клиенты:**
   a. Регистрация: возможность зарегистрироваться на сайте или в приложении.
   b. Поиск услуг: удобный интерфейс для поиска тренировок, групповых занятий и мероприятий.
   c. Бронирование тренировок: возможность забронировать занятия, тренировки и тд, выбрать тренера.
   d. Составление своего расписания и индивидуальных программ тренировок (возможность составить индивидуальный план тренировок).
   e. Оплата: возможность оплатить услугу различными способами (банковские карты, электронные кошельки).
   f. История посещений: просмотр истории своих посещений, покупок и активностей.

g. Просмотр индивидуальных метрик: в базы данных выгружаются продвинутые характеристики пользователей (по желанию) - безжировая масса тела, индекс массы тела, уровень основного обмена веществ т.п.
h. Отзывы и рейтинги: оставлять отзывы о тренерах, услугах и мероприятиях.
i. Уведомления: получение уведомлений о предстоящих занятиях, изменениях в расписании, акциях и специальных предложениях.
j. Использование шкафчиков для хранения вещей: возможность открывать и закрывать шкафчики с личными вещами (любой пользователь может воспользоваться любым свободным и не забронированным шкафчиком).
k. Бронирование шкафчиков для хранения вещей: возможность забронировать шкафчик на месяц.
l. Участие в программах лояльности: регистрация в программе лояльности позволяет получать бонусы (скидки) за участие в мероприятиях.

2. Тренеры:

a. Составление своего расписания (добавление, изменение и удаление проводимых тренировок).
b. Список клиентов: просмотр списка зарегистрированных на занятие клиентов.
c. Оценка работы: отслеживание отзывов и рейтингов от посетителей.
d. Отчеты: доступ к отчетам об активности, посещаемости и доходах.

3. Менеджмент:

a. Управление персоналом: управление списками сотрудников (тренеров), редактирование данных.
b. Финансовый учет: контроль доходов и расходов, управление финансовыми потоками.
c. Анализ статистики: Анализ посещаемости, популярности различных видов тренировок, доходности разных направлений.
d. Маркетинговая аналитика: Оценка эффективности рекламных кампаний, акций и предложений.

4. **Администраторы ресепшна:**

a. Оформление абонементов и учетных записей: регистрация клиента при первом посещении.
b. Проведение обзвонов: с целью улучшения качества обслуживания осуществляют звонки обладателям абонементов.

c. Консультирование клиентов: коммуникация с клиентами по поводу абонементов, мероприятий, акций, программ лояльности.
d. Помощь клиентам: решение проблем с абонементами и шкафчиками с вещами.

5. Системные администраторы:

a. Полный доступ ко всем данным: полная информация обо всех аспектах работы системы.
b. Настройка прав доступа: настройка уровней доступа для других ролей (посетителей, тренеров, менеджмента).
c. Мониторинг безопасности: обеспечение защиты данных, мониторинг подозрительной активности.
d. Обновление и поддержка системы: внедрение обновлений, исправление ошибок и техническая поддержка.

Функциональные требования.

1. Клиенты и их аккаунты:

a. Посетители должны иметь возможность входить в систему под своим уникальным аккаунтом.
b. Система должна позволять посетителям регистрироваться с помощью электронной почты, пароля, а также с указанием ФИО и даты рождения.
c. Должна иметься возможность изменения клиентом своих персональных данных.
d. Клиенту должны быть доступны данные о своем расписании, количестве посещенных тренировок, остатку по абонементу и тд.
e. Система должна показывать клиенту его продвинутые индивидуальные метрики в личном кабинете: безжировую массу тела, индекс массы тела, уровень основного обмена веществ т.п.
f. Клиент должен иметь возможность бронирования шкафчика для тренировки.
g. Система должна предоставлять клиентам в личном кабинете удобный поиск услуг по различным критериям: тип тренировки, время проведения, размер группы и тд.
h. Система должна давать клиентам возможность выбора тренера.
i. Система должна давать клиентам возможность записываться на тренировки, отменять и изменять записи.

j. Клиент должен иметь возможность просматривать свой баланс бонусов в рамках программы лояльности.

2. Абонементы:

a. Система должна иметь возможность продажи различных абонементов.
b. Система должна иметь функцию заморозки абонемента, активации и деактивации абонемента.
c. Система должна быть подключена к платежной системе и иметь функцию продления абонемента.

3. Тренеры:

a. Каждый тренер должен иметь возможность просмотра тренировок, которые он ведет, а также своего индивидуального расписания в личном кабинете.
b. Каждый тренер должен иметь возможность просмотра оценок, выставленных ему клиентами.
c. Тренеру должны быть доступны списки клиентов, посещающих его тренировки.

4. Тренировки и расписание:

a. Система должна хранить данные о расписании тренировок.
b. Системы должна хранить данные о том, какое оборудование требуется для каждой тренировки.
c. Система должна хранить данные о том, какие клиенты посещают каждую тренировку.
d. Система должна хранить данные о том, какой тренер ведет тренировку.
e. Система должна оповещать тренеров об изменении их расписания (появлении новых тренировок/отменах и переносах существующих).

5. Сотрудники:

a. Система должна хранить данные о всех сотрудниках, включая их должность, зарплату, личные данные, дату начала работы, графики работы (расписания) сотрудников, премии.
b. Система должна хранить расширенное количество информации о тренерах и менеджерах по сравнению с другими сотрудниками.

6. Учет оборудования:

a. система должна учитывать оборудование зала, хранить данные о его поставщике, производителе, состоянии и дате начала использования.
b. система должна хранить сроки технического обслуживания тренажеров.

7. Оплата и финансовый учет:

a. Система должна поддерживать различные способы оплаты (карты различных банков, электронные кошельки).
b. Должен быть реализован процесс подтверждения платежа и уведомления пользователя о завершении транзакции.
c. Система должна хранить историю платежей каждого клиента.
d. Система должна вести учет доходов от продажи, запланированных и незапланированных расходов.
e. Система должна производить выставление счетов клиентам по оплаченным тренировкам, абонементам и прочим услугам.
f. Система должна хранить данные о всех услугах, приобретенных клиентами.

8. Права доступа:

a. В системе должно присутствовать разграничение прав доступа в зависимости от должности сотрудника (объем доступа к базе повышается соответственно месту должности в “иерархии” сотрудников).
b. Система должна вести журнал действий пользователя.
c. Система должна давать менеджерам и администраторам возможность просмотра расписания тренировок.
d. Система должна давать администраторам доступ к изменению расписания.

9. Получение уведомлений:

a. Система должна отправлять уведомления клиентам и тренерам о предстоящих занятиях.
b. Система должна уведомлять клиента незадолго до окончания его абонемента.
c. Система должна уведомлять клиентах об акциях, проходящих в зале.

Нефункциональные требования.

Нефункциональные требования - это требования к системе, не определяемые выполнением конкретных функций непосредственно. Главным образом, наш продукт должен соответствовать таким нефункциональным требованиям, как надежность и производительность. Для повышения производительности будут созданы индексы, ускоряющие выполнение базой запросов на поиск значений в таблицах. Индексы позволяют осуществлять быстрый поиск, в частности, Index Scan или Bitmap Scan. База будет работать и возвращать нужные значения и без индексов, однако индексы значительно ускорят работу базы данных для сложных запросах. Многие запросы из

приведенных ниже при работе с большим количеством данных могут работать очень долго при последовательном сканировании. Для повышения надежности системы в базе будут реализованы транзакции.

Ограничения.

1. Клиенты:

a. Баланс бонусов по умолчанию при регистрации нового аккаунта до начала совершения покупок равен 0.
b. Каждый клиент должен обладать своим порядковым уникальным идентификатором.
c. На одну электронную почту и/или один телефон можно зарегистрировать только один аккаунт.
d. У каждого клиента должен быть счет в программе лояльности. Значение баланса по умолчанию (до начала приобретения услуг) равно 0.
e. Клиент не может забронировать шкафчик, уже занятый другим клиентом (реализуется с помощью флага).

2. Тренеры:

a. Каждый тренер должен обладать своим уникальным порядковым идентификатором.
b. Идентификатор каждого тренера должен находиться в таблице с информацией о всех сотрудниках в целом.
c. Каждый тренер подчиняется только одному менеджеру (атомарные значения).
d. На одну электронную почту и/или один телефон можно зарегистрировать только один аккаунт.

3. Менеджеры:

a. Каждый менеджер должен обладать уникальным идентификатором.
b. Идентификатор каждого тренера должен находиться в таблице с информацией о всех сотрудниках в целом.

4. Абонементы:

a. Клиент имеет возможность не приобретать абонемент: при регистрации нового аккаунта по умолчанию у нового клиента нет абонемента.

5. Тренировки:

a. Каждая тренировка должна обладать своим уникальным последовательным идентификатором.

6. Персональные расписания сотрудников:

a. У каждого персонального расписания сотрудника есть уникальный порядковый идентификатор.

7. Сотрудники:

a. Каждый менеджер обладает своим уникальным порядковым идентификатором.
b. Каждый сотрудник (включая тренеров и менеджеров) обладает уникальным порядковым идентификатором сотрудника.
c. По умолчанию (в случае, если нет акта депремирования) премия выплачивается каждому сотруднику.

8. Оборудование:

a. Каждая единица оборудования должна обладать своим уникальным порядковым идентификатором.

9. Платежи клиентов:

a. Каждый платеж обладает порядковым уникальным идентификатором.

10. Услуги клиентов:

a. Каждая услуга зала как товарная единица обладает своим уникальным идентификатором.

11. Шкафчики:

a. Каждый шкафчик обладает уникальным порядковым номером.
b. По умолчанию система считывает шкафчик как свободный, если только он не получает данных о том, что он был забронирован клиентом.

Диаграмма.

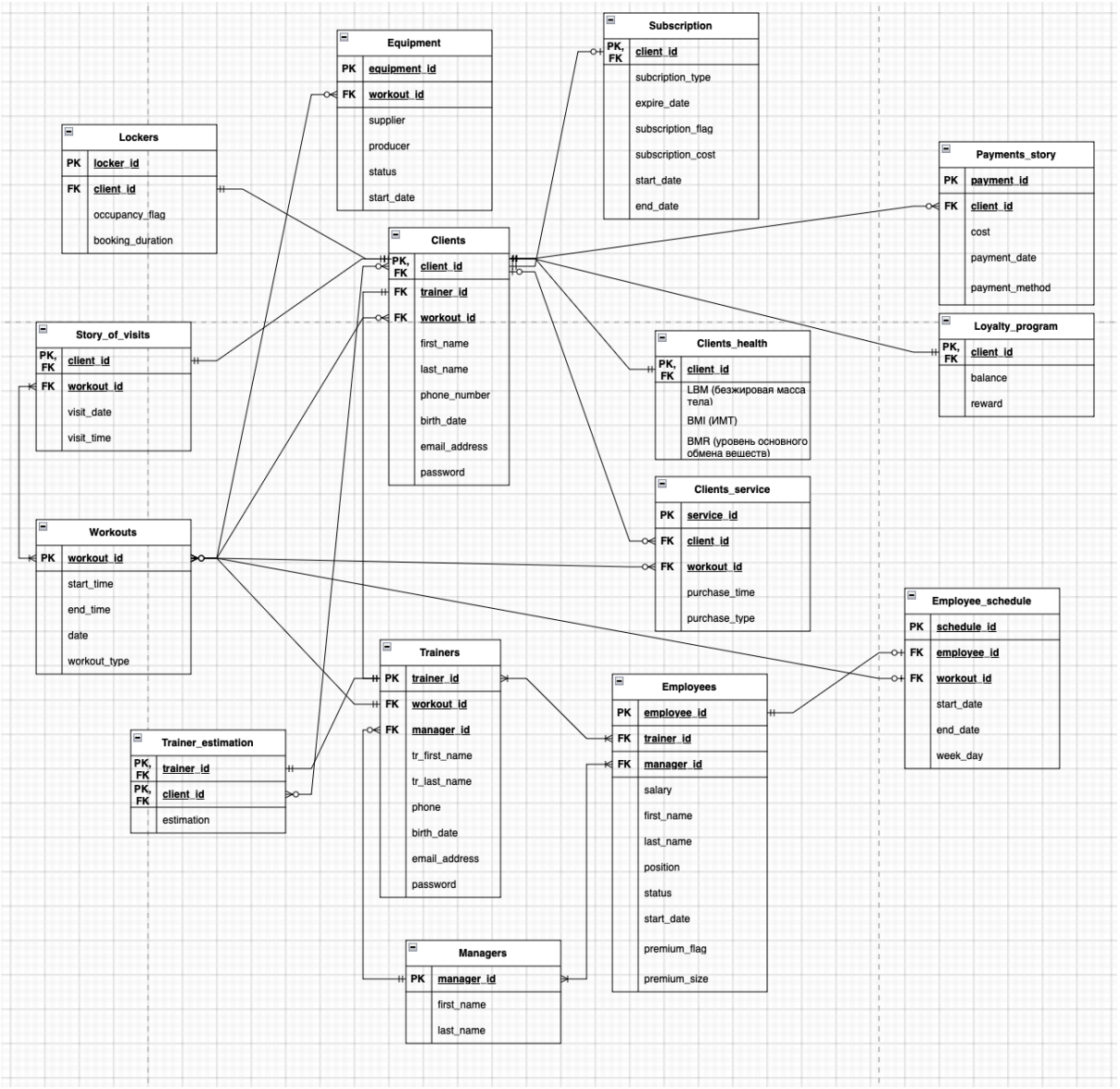

Диаграмма доступна [по ссылке на draw io.]

Функциональные зависимости (по таблицам).

1. Clients:

client_id -> {first_name, last_name, phone_number, birth_date, email_address, password}

2. Story_of_visits:

client_id -> {workout_id, visit_date, visit_time}

3. Payments_story:

payment_id -> {client_id, cost, payment_date, payment_method}

4. Subscription:

client_id -> {subscription_type, expire_date, subscription_flag, subscription_cost, start_date, end_date, last_payment_date}

5. Loyalty_program:

client_id -> {balance, reward}

6. Clients_health:

client_id -> {LBM, BMI, BMR}

7. Clients_service:

service_id -> {client_id, workout_id, purchase_time, purchase_type}

8. Lockers:

locker_id -> {client_id, occupancy_flag, booking_duration}

9. Equipment:

equipment_id -> {supplier, producer, status, start_date}

equipment_id -> {workout_id}

10. Workouts:

workout_id -> {start_time, end_time, date, workout_type}

11. Trainer_estimation:

trainer_id, client_id -> {estimation}

12. Trainers:

trainer_id -> {manager_id, tr_first_name, tr_last_name, phone, birth_date, email_address, password}

13. Managers:

manager_id -> {first_name, last_name}

14. Employees:

employee_id -> {salary, first_name, last_name, position, status, start_date, premium_flag, premium_size}

trainer_id -> {salary, first_name, last_name, position, status, start_date, premium_flag, premium_size}

manager_id -> {salary, first_name, last_name, position, status, start_date, premium_flag, premium_size}

15. Employee_schedule:

schedule_id -> {employee_id, workout_id, start_date, end_date, week_day}

employee_id -> {schedule_id}

Нормализация.

1. Первая нормальная форма:

1NF требует, чтобы у каждой таблицы был первичный ключ (PK) и чтобы все значения в столбцах были атомарными. Это требование было выполнено нами изначально при создании UML-диаграммы.

2. Вторая нормальная форма:

2NF требует, чтобы база находилась в 1NF и чтобы все неключевые атрибуты (значения в столбцах без ограничения первичного ключа) зависели от первичного ключа целиком - следовательно, в таблицах не должно быть частичных зависимостей в случае, если ключ составной. Это требование тоже выполнялось создаваемой базой изначально.

3. Третья нормальная форма:

3NF требует, чтобы база находилась в 2NF и в ее таблицах не было транзитивных зависимостей - то есть чтобы ни один неключевой атрибут не зависел от другого неключевого атрибута. В 3NF создаваемая база также находится изначально.

4. Нормальная форма Бойса-Кодда:

BCNF требует, чтобы база находилась в 3NF и при этом чтобы каждый неключевой атрибут зависел от суперключа. Последнее требование противоречит тому, как устроена таблица Employees в создаваемой в рамках данного проекта базе, так что нами было принято решение не приводить базу к BCNF.

5. Четвертая нормальная форма:

Так как одно из требований 4NF - это чтобы база находилась в BCNF, создаваемая нами база не может быть приведена к этой нормальной форме.

Таким образом, итоговая степень нормализации, в которой будет находиться наша база - это 3NF.

Аномалии с недонормализованной БД:

Третья нормальная форма предполагает отсутствие транзитивных зависимостей в таблицах. Наличие таких зависимостей приведет к сложностям при обновлениях данных. Так, если бы таблицы Payment_story и Subsription были бы едины, то атрибуты client_id зависели бы от неключевого элемента client_id (который зависел бы также от ключевого payment_id). Поэтому при обновлении данных об оплате подписки требовалось бы проверять и вносить изменения в обе таблицы, что неэффективно и может привести к потере целостности базы.

Вторая нормальная форма предполагает, что все атрибуты таблицы зависят от всех элементов составного ключа. Если это будет не так, то выполнение запросов станет неэффективным (придется вводить дополнительные условия) и может нарушиться целостность данных. Если бы в таблице Trainer_estimation атрибут estimation зависел бы только от client_id, то были бы сложности с поиском того, к какому тренеру этот отзыв относится.

Первая нормальная форма требует наличия у всех таблиц первичного ключа. Первичные ключи позволяют находить требуемые кортежи в таблице по уникальному значению. Если этого значения не будет, то нарушится реляционный принцип построения базы данных и будет невозможно строго найти требуемый кортеж. В случае отсутствия первичного ключа equipment_id в таблице Equipment, запросы могут выдавать лишний инвентарь при поиске по атрибутам.

DDL (создание базы данных):

```
--database's tables creation:

CREATE TABLE Clients
(client_id serial,
trainer_id int,
workout_id int,
first_name varchar(50),
last_name varchar(50),
phone_number integer,
birth_date date,
email_address varchar(50),
password varchar(15));

CREATE TABLE Subscription
(client_id int,
subcription_type varchar(20),
expire_date date,
subscription_flag boolean,
subscription_cost int,
```

```
start_date date,
end_date date);

CREATE TABLE Equipment
(equipment_id serial,
supplier varchar(30),
producer varchar(30),
status varchar(20),
start_date date);

CREATE TABLE Lockers
(locker_id serial,
client_id int,
occupancy_flag boolean,
booking_duration timestamp);

CREATE TABLE Story_of_visits
(client_id int,
workout_id int,
visit_date date,
visit_time time);

CREATE TABLE Clients_health
(client_id int,
LBM decimal,
BMI decimal,
BMR decimal);

CREATE TABLE Payments_story
(payment_id serial,
client_id int,
cost int,
payment_date date,
payment_method varchar(20));
```

```
CREATE TABLE Loyalty_program
(client_id int,
balance decimal,
reward varchar(20));

CREATE TABLE Clients_service
(service_id serial,
client_id int,
workout_id int,
purchase_time timestamp,
purchase_type varchar(50));

CREATE TABLE Workouts
(workout_id serial,
start_time time,
end_time time,
date date,
workout_type varchar(30));

CREATE TABLE Trainers
(trainer_id serial,
workout_id int,
manager_id int,
tr_first_name varchar(30),
tr_last_name varchar(30),
phone int,
birth_date date,
email_address varchar(30),
password varchar(20));

CREATE TABLE Trainer_estimation
(trainer_id int,
client_id int,
```

```
estimation decimal);

CREATE TABLE Managers
(manager_id serial,
first_name varchar(30),
last_name varchar(30));

CREATE TABLE Employees
(employee_id serial,
trainer_id int,
manager_id int,
salary decimal,
first_name varchar(30),
last_name varchar(30),
position varchar(30),
status varchar(30),
start_date date,
premium_flag boolean,
premium_size decimal);

CREATE TABLE Employee_schedule
(schedule_id int,
employee_id int,
workout_id int,
start_date date,
end_date date,
week_day varchar(15));
```

DML (создание ограничений):

```
--Constraints
--Workouts:
ALTER TABLE
Workouts
```

```
ADD unique(workout_id);

ALTER TABLE
Workouts
ADD PRIMARY KEY(workout_id);

--Clients:
ALTER TABLE
Clients
ADD unique(client_id);

ALTER TABLE
Clients
ADD PRIMARY KEY (client_id);

ALTER TABLE
Trainers
ADD unique(trainer_id);

ALTER TABLE
Clients
ADD constraint fk_trainer
FOREIGN KEY (trainer_id) references Trainers(trainer_id);

ALTER TABLE
Clients
ADD constraint fk_workout
FOREIGN KEY (workout_id) references Workouts(workout_id);

ALTER TABLE
Clients
ADD unique(phone_number);

ALTER TABLE
```

```
Clients
ADD unique(email_address);

--Equipment:
ALTER TABLE
Equipment
ADD unique(equipment_id);

ALTER TABLE
Equipment
ADD primary key(equipment_id);

ALTER TABLE
Equipment
ADD constraint fk_workout
FOREIGN KEY (workout_id) references Workouts(workout_id);

--Subscription:
ALTER TABLE
Subscription
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

ALTER TABLE
Subscription
ALTER COLUMN subscription_flag
SET DEFAULT false;

--Lockers:
ALTER TABLE
Lockers
ADD UNIQUE(locker_id);

ALTER TABLE
```

```
Lockers
ADD PRIMARY KEY(locker_id);

ALTER TABLE
Lockers
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

ALTER TABLE
Lockers
ALTER COLUMN occupancy_flag
SET DEFAULT false;

--Payments_story:
ALTER TABLE
Payments_story
ADD UNIQUE(payment_id);

ALTER TABLE
Payments_story
ADD PRIMARY KEY(payment_id);

ALTER TABLE
Payments_story
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

--Story_of_visits:
ALTER TABLE Story_of_visits
add unique(client_id);

ALTER TABLE Story_of_visits
add PRIMARY KEY(client_id);
```

```
ALTER TABLE
Story_of_visits
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

ALTER TABLE
Story_of_visits
ADD constraint fk_workout
FOREIGN KEY (workout_id) references Workouts(workout_id);

--loyalty_program:
ALTER TABLE loyalty_program
add unique(client_id);

ALTER TABLE loyalty_program
add PRIMARY KEY(client_id);

ALTER TABLE
loyalty_program
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

ALTER TABLE
loyalty_program
alter column balance
set DEFAULT 0;

--clients_health:
ALTER TABLE clients_health
add unique(client_id);

ALTER TABLE clients_health
add PRIMARY KEY(client_id);
```

```
ALTER TABLE
clients_health
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

--clients_service:
ALTER TABLE
Clients_service
ADD UNIQUE(service_id);

ALTER TABLE
Clients_service
ADD PRIMARY KEY(service_id);

ALTER TABLE
Clients_service
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

ALTER TABLE
Clients_service
ADD constraint fk_workout
FOREIGN KEY (workout_id) references Workouts(workout_id);

--Managers:
ALTER TABLE
Managers
ADD UNIQUE(manager_id);

ALTER TABLE
Managers
ADD PRIMARY KEY(manager_id);

--Trainers:
```

```
--UNIQUE constraint has been already made earlier.
ALTER TABLE
Trainers
ADD PRIMARY KEY(trainer_id);

ALTER TABLE
Trainers
ADD constraint fk_workout
FOREIGN KEY (workout_id) references Workouts(workout_id);

ALTER TABLE
Trainers
ADD constraint fk_manager
FOREIGN KEY (manager_id) references Managers(manager_id);

ALTER TABLE
Trainers
ADD UNIQUE(phone);

ALTER TABLE
Trainers
ADD UNIQUE(email_address);

--Trainer_estimation:
ALTER TABLE
Trainer_estimation
ADD PRIMARY KEY(trainer_id, client_id);

ALTER TABLE
Trainer_estimation
ADD constraint fk_trainer
FOREIGN KEY (trainer_id) references Trainers(trainer_id);

ALTER TABLE
```

```
Trainer_estimation
ADD constraint fk_client
FOREIGN KEY (client_id) references Clients(client_id);

--Employees:
ALTER TABLE
Employees
ADD UNIQUE(employee_id);

ALTER TABLE
Employees
ADD PRIMARY KEY(employee_id);

ALTER TABLE
Employees
ADD constraint fk_trainer
FOREIGN KEY (trainer_id) references Trainers(trainer_id);

ALTER TABLE
Employees
ADD constraint fk_manager
FOREIGN KEY (manager_id) references Managers(manager_id);

ALTER TABLE
Employees
ALTER COLUMN premium_flag
SET DEFAULT true;

--Employee_schedule
ALTER TABLE
Employee_schedule
ADD UNIQUE(schedule_id);

ALTER TABLE
```

```
Employee_schedule
ADD PRIMARY KEY(schedule_id);

ALTER TABLE
Employee_schedule
ADD constraint fk_workout
FOREIGN KEY (workout_id) references Workouts(workout_id);

ALTER TABLE
Employee_schedule
ADD constraint fk_employee
FOREIGN KEY (employee_id) references Employees(employee_id);
```

Нефункциональные требования.

Примеры транзакций.

Транзакция, гарантирующая наличие всех обязательных персональных данных у нового клиента:

```
BEGIN;
insert into
Clients(client_id, first_name, last_name,
phone_number, birth_date, email_address, password)
VALUES(1, 'Iad', 'Iadovich', 8910, '2022-12-12', 'iad@gmail.com', 'iadiadiad');
insert into Subscription
values(1, 'minimum', '2025-01-01', TRUE, 15000, '2024-12-01', '2025-01-01');
insert into Payments_story (client_id) values(1);
insert into Loyalty_program(client_id) values (1);
Insert into clients_health(client_id) values (1);
Insert into Story_of_visits(client_id) values (1);
COMMIT;
```

Транзакция, гарантирующая наличие всех обязательных персональных данных у нового тренера:

```
BEGIN;
```

```
insert into
Trainers(trainer_id, tr_first_name, tr_last_name,
phone, birth_date, email_address, password)
VALUES(1, 'Iad', 'Iadovich', 8910, '2022-12-12', 'iad@gmail.com', 'iadiadiad');
insert into Employees(trainer_id, first_name, last_name, position, start_date)
values(1, 'Iad', 'Iadovich', 'Trainer', '2022-12-14');
COMMIT;
```

Индексы.

Примеры индексов, которые будут использованы для повышения производительности базы данных:

```
CREATE INDEX b_client_id ON Clients(client_id)
CREATE INDEX b_phone_number ON Clients(phone_number)
CREATE INDEX b_employee_id ON Employees(employee_id)
```

Запросы:

Найти номера телефонов клиентов, которые записаны на тренировки, но у них заканчивается срок абонемента

```
SELECT c.phone_number FROM clients c
JOIN Story_of_visits sv ON c.client_id = sv.client_id
JOIN Subscription s ON c.client_id = s.client_id
WHERE s.end_date <= CURRENT_DATE + interval '7 days'
AND s.end_date >= CURRENT_DATE;
```

Найти email адреса клиентов, которые ходят в зал больше года и записаны на пилатес к конкретному тренеру (фамилия имя).

```
SELECT DISTINCT c.email_address
FROM clients c
JOIN Subscription s ON c.client_id = s.client_id
```

```
JOIN story_of_visits sv ON c.client_id = sv.client_id

JOIN workouts w ON sv.workout_id = w.workout_id

JOIN trainers t ON w.workout_id = t.workout_id

WHERE s.start_date <= CURRENT_DATE - INTERVAL '1 year' AND w.workout_type = 'пилатес'

AND t.tr_last_name = 'Иванов' AND t.tr_first_name = 'Иван';
```

Найти имена и фамилии сотрудников, которые получили премию больше 20000 рублей, в расписании которых больше 4 рабочих дней и которые работают на тренажерах марки BOWFLEX

```
SELECT e.first_name, e.last_name

FROM employees e

JOIN (SELECT employee_id FROM employee_schedule

GROUP BY employee_id

HAVING COUNT(DISTINCT week_day) > 4 ) sched

ON e.employee_id = sched.employee_id

JOIN employee_schedule

ON sched.employee_id = employee_schedule.employee_id

JOIN workouts ON employee_schedule.workout_id = workouts.workout_id

JOIN equipment ON equipment.workout_id  = workouts.workout_id

WHERE e.premium_size > 20000 AND equipment.producer = 'BOWFLEX';
```

Получение списка всех платежей клиентов и их общей суммы

```
SELECT clients.client_id,

first_name,

last_name,

SUM( payments.cost) AS total_payments
```

```
FROM clients

JOIN payments ON clients.client_id = payments.client_id

GROUP BY clients.client_id, first_name, last_name
```

Топ 3 тренера по количеству заработанных денег

```
SELECT t.tr_first_name,
 t.tr_last_name AS trainer_name,
SUM(e.salary) AS total_earnings
FROM trainers AS t
JOIN employees AS e ON t.trainer_id = e.trainer_id
GROUP BY trainer_name
ORDER BY total_earnings DESC
LIMIT 3
```

Найти имена, фамилии и email адреса клиентов, у которых индекс массы тела больше 35, они ходят в зал больше года, но они не посещает групповые кардио тренировки

```
SELECT c.first_name, c.last_name, c.email_address
FROM clients c
JOIN clients_health ch ON c.client_id = ch.client_id
JOIN subscription s ON s.client_id = ch.clients_id

LEFT JOIN story_of_visits sv ON c.client_id = sv.client_id

LEFT JOIN workouts w ON sv.workout_id = w.workout_id

WHERE ch.BMI > 35

AND s.start_date <= CURRENT_DATE - INTERVAL '1 year'

AND w.type != 'групповые кардио'

GROUP BY c.first_name, c.last_name, c.email

HAVING COUNT(sv.visit_id) > 0;
```

Найти среднюю оценку поставленную тренерам для каждого клиента

SELECT B.first_name, B.last_name, ROUND(AVG(A.estimation) OVER(PARTITION BY A.client_id), 2) as avg_estimation

FROM Trainer_estimation as A

JOIN Clients as B ON A.client_id = B.client_id

Найти суммарное время, проведенное 18-летними на тренировках

```
WITH q1 AS (SELECT client_id FROM Clients WHERE DATEPART(year, CURRENT_DATE()) - DATEPART (year, birth_date) = 18)
SELECT SUM(visit_time) FROM Workouts WHERE client_id IN (SELECT * FROM q1)
```

Найти тип абонемента, принесший наибольшую выгоду (количество покупок умножить на цену абонемента)

```
WITH q1 AS (SELECT COUNT(client_id) as quantity, subscription_type  FROM Subscription GROUP BY subscription_type),
q2 AS (SELECT A.subscription_type type, A.quantity * B.subscription_cost revenue FROM q1 AS A JOIN Subscription AS B ON A.subscription_type = B.subscription_type)
SELECT type FROM q2 ORDER BY revenue DESC LIMIT 1
```