*Baza danych klubu fitness*
*Jakub Steć*
*Uniwersytet Jagiellonski, 2025*

# Opis projektu

## *Cele projektu:*

Celem projektu jest stworzenie bazy danych dla klubu fitness, która umożliwi efektywne zarządzanie klientami, karnetami, zajęciami, trenerami, sprzętem oraz opiniami. Baza danych ma wspierać codzienne operacje klubu, takie jak rejestracja klientów, zarządzanie karnetami, rezerwacja zajęć, śledzenie obecności oraz utrzymanie sprzętu. Baza danych ma również umożliwiać analizę danych i generowanie raportów wspierających decyzje biznesowe.

## *Główne założenia:*

- **Struktura bazy danych** - Baza składa się z 14 tabel, które są ze sobą powiązane relacjami kluczy obcych.
- **Dziedziczenie** - Wykorzystano dziedziczenie poprzez tabele *person*, *client*, *trainer* i *employee*, gdzie *person* jest tabelą nadrzędną
- **Dynamiczne dane** - Baza przechowuje dane, które zmieniają się w czasie, takie jak status karnetów, obecność na zajęciach czy stan sprzętu.
- **Automatyzacja** - Wykorzystano procedury składowane i wyzwalacze do automatyzacji procesów, takich jak rezerwacja zajęć, anulowanie rezerwacji czy aktualizacja statusu karnetów.
- **Bezpieczeństwo danych** - Zaimplementowano dodatkowe więzy integralności danych, takie jak ograniczenia *CHECK*, *UNIQUE* i *NOT NULL*, aby zapewnić spójność danych.

## *Możliwości:*

- **Zarządzanie klientami** - Rejestracja nowych klientów, przypisywanie karnetów, śledzenie historii członkostwa.
- **Rezerwacja zajęć** - Klienci mogą rezerwować miejsca na zajęcia, a system automatycznie sprawdza dostępność miejsc.
- **Śledzenie obecności** - System rejestruje obecność klientów na zajęciach, co umożliwia analizę frekwencji.
- **Zarządzanie trenerami** - Przypisywanie trenerów do zajęć, śledzenie ich specjalizacji i dostępności.

- **Utrzymanie sprzętu** - System przechowuje informacje o stanie technicznym sprzętu oraz historię napraw.

- **Opinie klientów**- Klienci mogą wystawiać opinie o trenerach i zajęciach, co pozwala na ocenę jakości usług.

- **Analiza finansowa** - Generowanie raportów przychodów z karnetów i zajęć, co ułatwia zarządzanie finansami klubu.

*Ograniczenia:*

- **Złożoność systemu** - Duża liczba powiązanych tabel wymaga starannego projektowania zapytań oraz optymalizacji wydajności.

- **Konserwacja** - Konieczne jest regularne monitorowanie integralności danych, optymalizacja indeksów oraz tworzenie kopii zapasowych.

- **Bezpieczeństwo danych** - Ze względu na przechowywanie wrażliwych informacji (dane klientów, płatności), wymagane są odpowiednie mechanizmy ochrony, takie jak szyfrowanie i kontrola dostępu.

- **Skalowalność** - Przy dużej liczbie klientów może być konieczna optymalizacja struktury bazy lub migracja do bardziej wydajnego systemu.

*Strategia pielęgnacji bazy danych:*

**Kopie zapasowe**:
- Pełna kopia zapasowa: codziennie o 2:00.
- Kopia różnicowa: co 6 godzin (8:00, 14:00, 20:00).
- Kopia logów transakcji: co 30 minut.

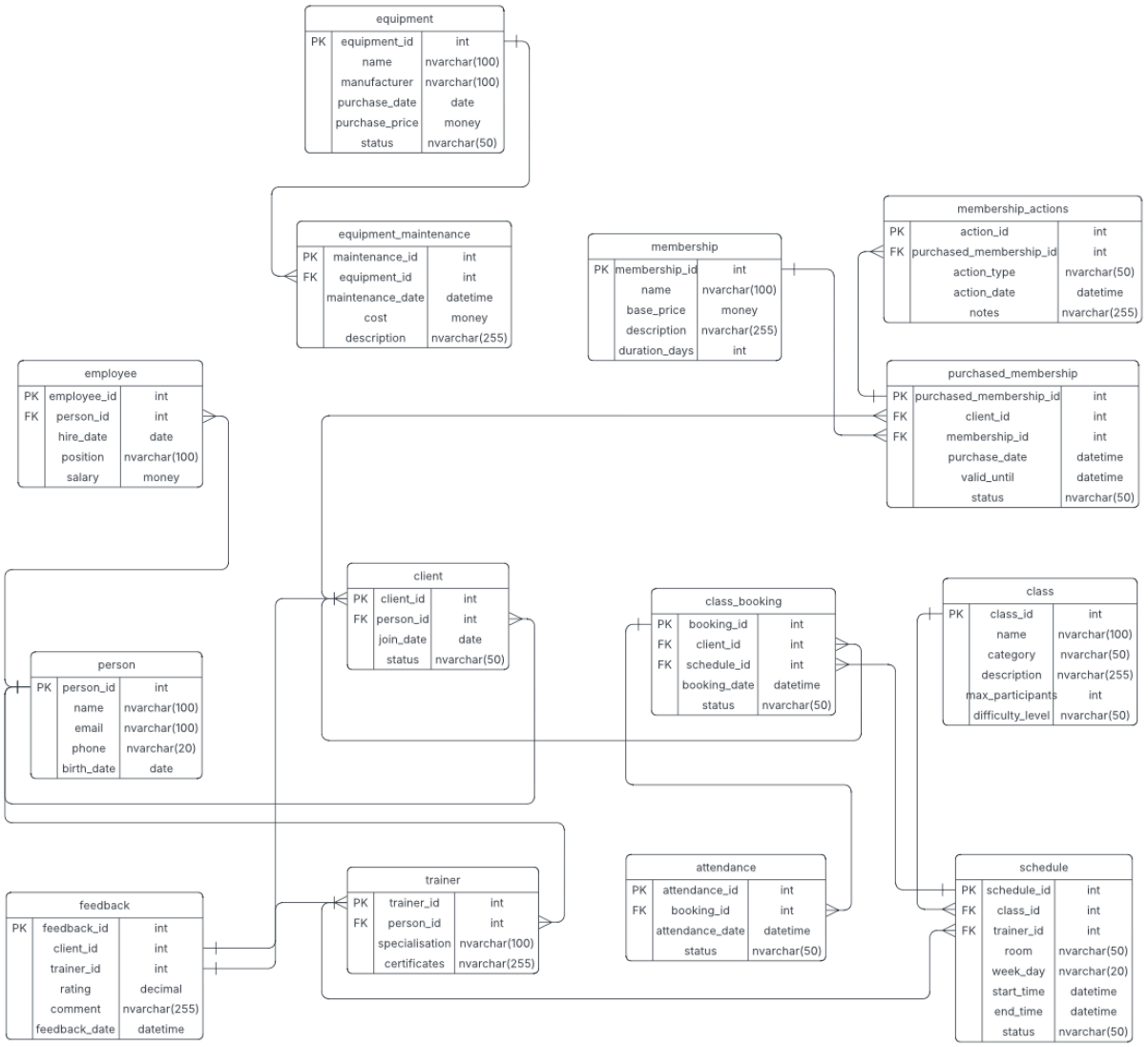
**Polityka retencji**:
- Pełne kopie zapasowe: przechowywane przez 30 dni.
- Kopie różnicowe: przechowywane przez 7 dni.
- Logi transakcji: przechowywane przez 3 dni.

**Testy odtwarzania**: Regularne testy odtwarzania danych z kopii zapasowych.

*Typowe zapytania:*

- Zapytanie o średnią ocenę każdego trenera wraz z liczbą otrzymanych recenzji
- Zapytanie o liczbę członków według typu członkostwa
- Zapytanie o najpopularniejsze zajęcia
- Zapytanie o harmonogram zajęć na dany dzień
- Zapytanie o frekwencje na zajęciach

*Diagramy Relacji:*




*Dodatkowe więzy integralności danych:*

**CHECK**:

- *purchased_membership.valid_until* musi być późniejsza niż *purchase_date*
- *feedback.rating* musi być w zakresie od 1 do 5
- *equipment.status* musi być jednym z: *active*, *requires_maintenance*, *inactive*
- *class.difficulty_level* musi być jednym z: *Beginner, Intermediate, Advanced*
- *schedule.start_time* musi być wcześniejsza niż *end_time*
- *feedback.feedback_date* nie może być przyszła
- *equipment.purchase_date* nie może być przyszła
- *purchased_membership.status* musi być jednym z: *active, inactive, expired*
- *class_booking.status* musi być jednym z: *confirmed, cancelled, pending*
- *membership_actions.action_date* nie może być przyszła.
- *equipment_maintenance.maintenance_date* nie może być przyszła

**UNIQUE**:

- *person.mail, person.phone* muszą być unikalne
- Kombinacja *schedule.room, schedule.week_day, schedule.start_time* musi być unikalna
- *class.name* musi być unikalna

**ON DELETE CASCADE**:
- *client(person_id)* - usunięcie osoby powoduje usunięcie klienta.
- *trainer(person_id)* - usunięcie osoby powoduje usunięcie trenera.
- *employee(person_id)* - usunięcie osoby powoduje usunięcie pracownika.
- *purchased_membership(client_id)* - usunięcie klienta powoduje usunięcie jego członkostwa.
- *class_booking(client_id)* - usunięcie klienta powoduje usunięcie jego rezerwacji.
- *attendance(booking_id)* - usunięcie rezerwacji powoduje usunięcie obecności.
- *equipment_maintenance(equipment_id)* - usunięcie sprzętu powoduje usunięcie jego historii konserwacji.
- *membership(purchased_membership_id)* - usunięcie członkostwa powoduje usunięcie jego historii akcji.

**NOT NULL**:
- *class(name, category, max_participants, difficulty_level)*
- *membership(name, base_price, duration_days)*
- *person(name, email, phone, birth_date)*
- *client(person_id, join_date, status)*
- *trainer(person_id, specialisation)*
- *employee(person_id, hire_date, position, salary)*
- *purchased_membership(client_id, membership_id, purchase_date, valid_until, status)*
- *schedule(class_id, trainer_id, room, week_day, start_time, end_time, status)*
- *class_booking(client_id, schedule_id, booking_date, status)*
- *equipment(name, manufacturer, purchase_date, purchase_price, status)*
- *discount(name, start_date, end_date, discount)*
- *membership_actions(purchased_membership_id, action_type, action_date)*
- *attendance(booking_id, attendance_date, status)*
- *equipment_maintenance(equipment_id, maintenance_date, cost)*
- *feedback(client_id, trainer_id, rating, feedback_date)*

## *Indeksy:*

1. *IX_Trainers_Specialization*: Przyspiesza wyszukiwanie trenerów według specjalizacji.

2. *IX_Schedule_ClubDayTime*: Optymalizuje pobieranie harmonogramów zajęć według sali, dnia tygodnia i godziny rozpoczęcia.

3. *IX_Clients_JoinDate*: Przyspiesza wyszukiwanie i sortowanie klientów według daty dołączenia.

4. *IX_Equipment_Status*: Przyspiesza wyszukiwanie sprzętu według statusu.

## *Widoki:*

1. *vw_active_clients*: Pokazuje klientów z aktywnymi karnetami.

2. *vw_get_schedule_by_day*: Zwraca harmonogram zajęć na wybrany dzień tygodnia.

3. *vw_average_attendance_by_category*: Oblicza średnią frekwencję na zajęciach w podziale na kategorie.

4. *vw_top_trainers_by_rating*: Pokazuje trenerów z najwyższą średnią ocen.

5. *vw_clients_with_expiring_memberships*: Wyświetla klientów, których karnety wygasną w ciągu 7 dni.

6. *vw_membership_revenue_by_month*: Pokazuje przychody z karnetów w podziale na miesiące.

7. *vw_equipment_needing_maintenance*: Wyświetla sprzęt wymagający naprawy.

8. *vw_TrainerPerformance*: Pokazuje statystyki trenerów, w tym liczbę prowadzonych zajęć i średnią frekwencję.

9. *vw_top_attending_clients*: Wyświetla klientów z największą liczbą odwiedzin.

10. *vw_longest_membership_clients*: Pokazuje klientów z najdłuższą historią członkostwa.

## *Procedury składowane:*

1. *sp_add_client*: Dodaje nowego klienta i przypisuje mu karnet.

2. *sp_BookClass*: Rezerwuje miejsce na zajęcia dla klienta.

3. *sp_CancelBooking*: Anuluje rezerwację zajęć.

4. *sp_MarkAttendance*: Oznacza obecność klienta na zajęciach.

5. *sp_LeaveFeedback*: Dodaje opinię klienta o trenerze.

## *Wyzwalacze:*

1. *trg_check_class_capacity*: Sprawdza, czy liczba rezerwacji nie przekracza maksymalnej liczby uczestników.

2. *trg_cancel_bookings_after_membership_expiry*: Anuluje rezerwacje, gdy karnet klienta wygaśnie.

3. *trg_auto_set_membership_validity*: Automatycznie ustawia datę ważności karnetu po zakupie.

4. *trg_prevent_class_overlap*: Sprawdza, czy nowe zajęcia nie nakładają się na istniejące.

5. *trg_update_class_status_after_booking*: Aktualizuje status zajęć na "pełne", gdy liczba rezerwacji osiągnie maksimum.

*Kod:*

```
CREATE DATABASE fitness_club_db;
GO

USE fitness_club_db;
GO

-- Tworzenie tabeli

CREATE TABLE membership (
  membership_id INT PRIMARY KEY,
  name NVARCHAR(100) NOT NULL,
  base_price MONEY NOT NULL,
  description NVARCHAR(255),
  duration_days INT NOT NULL
);

CREATE TABLE class (
  class_id INT PRIMARY KEY,
  name NVARCHAR(100) NOT NULL,
  category NVARCHAR(50) NOT NULL,
  description NVARCHAR(255),
  max_participants INT NOT NULL CHECK (max_participants > 0),
  difficulty_level NVARCHAR(50) NOT NULL
);

CREATE TABLE person (
  person_id INT PRIMARY KEY,
  name NVARCHAR(100) NOT NULL,
  email NVARCHAR(100) NOT NULL UNIQUE,
  phone NVARCHAR(20) NOT NULL UNIQUE,
  birth_date DATE NOT NULL
);

CREATE TABLE client (
  client_id INT PRIMARY KEY,
  person_id INT NOT NULL FOREIGN KEY REFERENCES person(person_id)
ON DELETE CASCADE,
  join_date DATE NOT NULL,
  status NVARCHAR(50) NOT NULL
```

```
);

CREATE TABLE trainer (
  trainer_id INT PRIMARY KEY,
  person_id INT NOT NULL FOREIGN KEY REFERENCES person(person_id)
ON DELETE CASCADE,
  specialisation NVARCHAR(100) NOT NULL,
  certificates NVARCHAR(255)
);

CREATE TABLE employee (
  employee_id INT PRIMARY KEY,
  person_id INT NOT NULL FOREIGN KEY REFERENCES person(person_id)
ON DELETE CASCADE,
  hire_date DATE NOT NULL,
  position NVARCHAR(100) NOT NULL,
  salary MONEY NOT NULL
);

CREATE TABLE purchased_membership (
  purchased_membership_id INT PRIMARY KEY,
  client_id INT NOT NULL FOREIGN KEY REFERENCES client(client_id)
ON DELETE CASCADE,
  membership_id INT NOT NULL FOREIGN KEY REFERENCES
membership(membership_id),
  purchase_date DATETIME NOT NULL,
  valid_until DATETIME NOT NULL,
  status NVARCHAR(50) NOT NULL,
  CONSTRAINT chk_valid_until CHECK (valid_until > purchase_date)
);

CREATE TABLE schedule (
  schedule_id INT PRIMARY KEY,
  class_id INT NOT NULL FOREIGN KEY REFERENCES class(class_id),
  trainer_id INT NOT NULL FOREIGN KEY REFERENCES
trainer(trainer_id),
  room NVARCHAR(50) NOT NULL,
  week_day NVARCHAR(20) NOT NULL,
  start_time DATETIME NOT NULL,
  end_time DATETIME NOT NULL,
  status NVARCHAR(50) NOT NULL
);

CREATE TABLE class_booking (
  booking_id INT PRIMARY KEY,
  client_id INT NOT NULL FOREIGN KEY REFERENCES client(client_id)
ON DELETE CASCADE,
  schedule_id INT NOT NULL FOREIGN KEY REFERENCES
schedule(schedule_id),
  booking_date DATETIME NOT NULL,
  status NVARCHAR(50) NOT NULL
);
```

```
CREATE TABLE equipment (
  equipment_id INT PRIMARY KEY,
  name NVARCHAR(100) NOT NULL,
  manufacturer NVARCHAR(100) NOT NULL,
  purchase_date DATE NOT NULL,
  purchase_price MONEY NOT NULL,
  status NVARCHAR(50) NOT NULL CHECK (status IN ('active',
'requires_maintenance', 'inactive'))
);

CREATE TABLE discount (
  discount_id INT PRIMARY KEY,
  name NVARCHAR(100) NOT NULL,
  start_date DATETIME NOT NULL,
  end_date DATETIME NOT NULL,
  discount DECIMAL(5, 2) NOT NULL CHECK (discount BETWEEN 0 AND
100),
  conditions NVARCHAR(255)
);

CREATE TABLE membership_actions (
  action_id INT PRIMARY KEY,
  purchased_membership_id INT NOT NULL FOREIGN KEY REFERENCES
purchased_membership(purchased_membership_id) ON DELETE CASCADE,
  action_type NVARCHAR(50) NOT NULL,
  action_date DATETIME NOT NULL,
  notes NVARCHAR(255)
);

CREATE TABLE attendance (
  attendance_id INT PRIMARY KEY,
  booking_id INT NOT NULL FOREIGN KEY REFERENCES
class_booking(booking_id) ON DELETE CASCADE,
  attendance_date DATETIME NOT NULL,
  status NVARCHAR(50) NOT NULL CHECK (status IN ('present',
'absent'))
);

CREATE TABLE equipment_maintenance (
  maintenance_id INT PRIMARY KEY,
  equipment_id INT NOT NULL FOREIGN KEY REFERENCES
equipment(equipment_id) ON DELETE CASCADE,
  maintenance_date DATETIME NOT NULL,
  cost MONEY NOT NULL,
  description NVARCHAR(255)
);

CREATE TABLE feedback (
    feedback_id INT PRIMARY KEY,
    client_id INT NOT NULL FOREIGN KEY REFERENCES
client(client_id) ON DELETE CASCADE,
```

```
    trainer_id INT NOT NULL FOREIGN KEY REFERENCES
trainer(trainer_id),
    rating DECIMAL(3, 2) NOT NULL CHECK (rating BETWEEN 1 AND 5),
    comment NVARCHAR(255),
    feedback_date DATETIME NOT NULL
);

-- Inserty

INSERT INTO person (person_id, name, email, phone, birth_date)
VALUES
(1, 'Jan Kowalski', 'jan.kowalski@example.com', '123-456-789',
'1985-05-15'),
(2, 'Anna Nowak', 'anna.nowak@example.com', '987-654-321',
'1990-08-22'),
(3, 'Piotr Wiśniewski', 'piotr.wisniewski@example.com',
'555-123-456', '1988-03-10'),
(4, 'Maria Zielińska', 'maria.zielinska@example.com',
'111-222-333', '1995-11-30'),
(5, 'Krzysztof Wójcik', 'krzysztof.wojcik@example.com',
'444-555-666', '1980-07-25'),
(6, 'Agnieszka Lewandowska', 'agnieszka.lewandowska@example.com',
'777-888-999', '1992-09-12'),
(7, 'Marek Dąbrowski', 'marek.dabrowski@example.com',
'222-333-444', '1987-01-20'),
(8, 'Ewa Kamińska', 'ewa.kaminska@example.com', '666-777-888',
'1993-04-05'),
(9, 'Tomasz Szymański', 'tomasz.szymanski@example.com',
'999-000-111', '1983-12-18'),
(10, 'mbappe', 'mbappe123@gmail.com', '123-123-123','2004-04-23'),
(11, 'Karolina Nowak', 'karolina.nowak@example.com',
'111-222-333', '1990-04-15'),
(12, 'Adam Kowalczyk', 'adam.kowalczyk@example.com',
'444-555-666', '1988-07-22'),
(13, 'Monika Lis', 'monika.lis@example.com', '777-888-999',
'1995-11-30'),
(14, 'Adam Nowak', 'adam.nowak@example.com', '111-222-333',
'1990-04-15'),
(15, 'Ewa Kowalska', 'ewa.kowalska@example.com', '444-555-666',
'1988-07-22');

INSERT INTO employee (employee_id, person_id, hire_date, position,
salary)
VALUES
(1, 1, '2020-01-15', 'Manager', 5000.00),
(2, 2, '2021-03-22', 'Receptionist', 3000.00),
(3, 3, '2019-07-10', 'Accountant', 4000.00);

INSERT INTO trainer (trainer_id, person_id, specialisation,
certificates)
VALUES
(1, 4, 'Yoga', 'Certified Yoga Instructor'),
```

```
(2, 5, 'CrossFit', 'CrossFit Level 1 Trainer'),
(3, 6, 'Pilates', 'Pilates Instructor Certification');

INSERT INTO client (client_id, person_id, join_date, status)
VALUES
(1, 7, '2023-01-10', 'active'),
(2, 8, '2023-02-15', 'active'),
(3, 9, '2023-03-20', 'inactive'),
(4, 10, '2023-04-25', 'active'),
(5, 11, '2025-01-01', 'active'),
(6, 12, '2025-01-10', 'active'),
(7, 13, '2025-01-15', 'active'),
(8, 14, '2023-10-01', 'active'),
(9, 15, '2023-10-05', 'active');

INSERT INTO membership (membership_id, name, base_price,
description, duration_days)
VALUES
(1, 'Monthly Pass', 100.00, 'Access to all classes and gym for 1
month', 30),
(2, '3-Month Pass', 250.00, 'Access to all classes and gym for 3
months', 90),
(3, 'Annual Pass', 800.00, 'Access to all classes and gym for 1
year', 365),
(4, 'Student Pass', 70.00, 'Discounted access for students (valid
student ID required)', 30),
(5, 'Weekend Pass', 50.00, 'Access only on weekends for 1 month',
30);

INSERT INTO purchased_membership (purchased_membership_id,
client_id, membership_id, purchase_date, valid_until, status)
VALUES
(1, 1, 1, '2023-10-01 10:00:00', '2023-10-31 23:59:59', 'active'),
(2, 2, 2, '2023-09-15 14:30:00', '2023-12-14 23:59:59', 'active'),
(3, 3, 3, '2023-01-01 09:00:00', '2023-12-31 23:59:59',
'inactive'),
(4, 4, 4, '2023-10-10 16:45:00', '2023-11-09 23:59:59', 'active'),
(5, 1, 5, '2023-10-05 11:15:00', '2023-11-04 23:59:59', 'active');

INSERT INTO membership_actions (action_id,
purchased_membership_id, action_type, action_date, notes)
VALUES
(1, 1, 'renewal', '2023-10-01 10:00:00', 'Initial purchase of
Monthly Pass'),
(2, 2, 'renewal', '2023-09-15 14:30:00', 'Initial purchase of 3-
Month Pass'),
(3, 3, 'cancellation', '2023-06-01 12:00:00', 'Membership
cancelled by client'),
(4, 4, 'renewal', '2023-10-10 16:45:00', 'Initial purchase of
Student Pass'),
(5, 5, 'renewal', '2023-10-05 11:15:00', 'Initial purchase of
Weekend Pass');
```

```sql
INSERT INTO class (class_id, name, category, description,
max_participants, difficulty_level)
VALUES
(1, 'Morning Yoga', 'Yoga', 'Relaxing yoga session to start your
day', 20, 'Beginner'),
(2, 'Power CrossFit', 'CrossFit', 'High-intensity CrossFit
workout', 15, 'Advanced'),
(3, 'Evening Pilates', 'Pilates', 'Pilates session to unwind after
work', 25, 'Intermediate'),
(4, 'Zumba Dance', 'Dance', 'Fun and energetic Zumba class', 30,
'Beginner'),
(5, 'Cycling Spin', 'Cardio', 'Indoor cycling session for cardio
fitness', 20, 'Intermediate'),
(6, 'Advanced Yoga', 'Yoga', 'Advanced yoga session for
experienced practitioners', 15, 'Advanced'),
(7, 'HIIT Workout', 'Cardio', 'High-Intensity Interval Training',
20, 'Advanced'),
(8, 'Evening Stretch', 'Flexibility', 'Relaxing stretching session
in the evening', 25, 'Beginner'),
(9, 'Friday Night Spin', 'Cardio', 'Fun indoor cycling session to
end the week', 20, 'Intermediate');

INSERT INTO schedule (schedule_id, class_id, trainer_id, room,
week_day, start_time, end_time, status)
VALUES
(1, 1, 1, 'Room A', 'Monday', '2023-10-30 08:00:00', '2023-10-30
09:00:00', 'scheduled'),
(2, 2, 2, 'Room B', 'Tuesday', '2023-10-31 18:00:00', '2023-10-31
19:00:00', 'scheduled'),
(3, 3, 3, 'Room C', 'Wednesday', '2023-11-01 19:00:00',
'2023-11-01 20:00:00', 'scheduled'),
(4, 4, 1, 'Room D', 'Thursday', '2023-11-02 17:00:00', '2023-11-02
18:00:00', 'scheduled'),
(5, 5, 2, 'Room E', 'Friday', '2023-11-03 07:00:00', '2023-11-03
08:00:00', 'scheduled'),
(6, 6, 1, 'Room A', 'Monday', '2025-02-24 18:00:00', '2025-02-24
19:00:00', 'scheduled'),
(7, 7, 2, 'Room B', 'Monday', '2025-02-24 19:00:00', '2025-02-24
20:00:00', 'scheduled'),
(8, 8, 3, 'Room C', 'Wednesday', '2025-02-26 19:00:00',
'2025-02-26 20:00:00', 'scheduled'),
(9, 9, 2, 'Room E', 'Friday', '2025-02-28 18:00:00', '2025-02-28
19:00:00', 'scheduled');

INSERT INTO class_booking (booking_id, client_id, schedule_id,
booking_date, status)
VALUES
(1, 1, 1, '2023-10-25 10:00:00', 'confirmed'),
(2, 2, 2, '2023-10-26 11:00:00', 'confirmed'),
(3, 3, 3, '2023-10-27 12:00:00', 'confirmed'),
(4, 4, 4, '2023-10-28 13:00:00', 'confirmed'),
```

```
(5, 1, 5, '2023-10-29 14:00:00', 'confirmed'),
(6, 5, 6, '2025-02-20 10:00:00', 'confirmed'),
(7, 6, 7, '2025-02-20 11:00:00', 'confirmed'),
(8, 7, 8, '2025-02-21 12:00:00', 'confirmed'),
(9, 5, 9, '2025-02-21 13:00:00', 'confirmed'),
(10, 8, 1, GETDATE(), 'confirmed'),
(11, 9, 2, GETDATE(), 'confirmed');

INSERT INTO attendance (attendance_id, booking_id,
attendance_date, status)
VALUES
(1, 1, '2023-10-30 08:00:00', 'present'),
(2, 2, '2023-10-31 18:00:00', 'present'),
(3, 3, '2023-11-01 19:00:00', 'absent'),
(4, 4, '2023-11-02 17:00:00', 'present'),
(5, 5, '2023-11-03 07:00:00', 'present'),
(6, 6, '2025-02-24 18:00:00', 'present'),
(7, 7, '2025-02-24 19:00:00', 'present'),
(8, 8, '2025-02-26 19:00:00', 'absent'),
(9, 9, '2025-02-28 18:00:00', 'present'),
(10, 10, GETDATE(), 'present'),
(11, 11, GETDATE(), 'present');

INSERT INTO equipment (equipment_id, name, manufacturer,
purchase_date, purchase_price, status)
VALUES
(1, 'Treadmill', 'Life Fitness', '2022-01-15', 2000.00, 'active'),
(2, 'Elliptical Trainer', 'Precor', '2022-03-10', 1500.00,
'active'),
(3, 'Dumbbell Set', 'Rogue Fitness', '2021-12-05', 500.00,
'active'),
(4, 'Exercise Bike', 'Peloton', '2023-05-20', 2500.00,
'requires_maintenance'),
(5, 'Rowing Machine', 'Concept2', '2023-02-28', 1800.00,
'active');

INSERT INTO equipment_maintenance (maintenance_id, equipment_id,
maintenance_date, cost, description)
VALUES
(1, 4, '2023-10-25 10:00:00', 200.00, 'Replaced worn-out pedals'),
(2, 2, '2023-09-15 14:30:00', 100.00, 'Lubricated moving parts'),
(3, 1, '2023-08-10 09:00:00', 150.00, 'Fixed belt alignment'),
(4, 5, '2023-07-05 16:45:00', 300.00, 'Replaced damaged seat'),
(5, 3, '2023-06-01 11:15:00', 50.00, 'Tightened loose screws');

INSERT INTO purchased_membership (purchased_membership_id,
client_id, membership_id, purchase_date, valid_until, status)
VALUES
(6, 5, 1, '2025-01-01 10:00:00', '2025-01-31 23:59:59', 'active'),
(7, 6, 2, '2025-01-10 14:30:00', '2025-04-10 23:59:59', 'active'),
(8, 7, 3, '2025-01-15 09:00:00', '2025-12-31 23:59:59', 'active'),
```

```
(9, 8, 1, '2023-10-01 10:00:00', DATEADD(DAY, 5, GETDATE()),
'active'),
(10, 9, 2, '2023-10-05 14:30:00', DATEADD(DAY, 3, GETDATE()),
'active');

INSERT INTO feedback (feedback_id, client_id, trainer_id, rating,
comment, feedback_date)
VALUES
(1, 1, 1, 4.5, 'Great trainer, very helpful!', '2023-10-30
10:00:00'),
(2, 2, 2, 5.0, 'Very intense workout, highly recommended!',
'2023-10-31 19:00:00'),
(3, 4, 3, 4.0, 'Calm and relaxing classes.', '2023-11-01
20:00:00'),
(4, 5, 1, 4.7, 'Professional approach to clients.', '2025-02-24
19:00:00'),
(5, 6, 2, 4.8, 'Training tailored to my needs.', '2025-02-24
20:00:00'),
(6, 7, 3, 4.2, 'Very good classes for beginners.', '2025-02-26
20:00:00'),
(7, 8, 1, 4.9, 'Amazing yoga classes!', GETDATE()),
(8, 9, 2, 4.6, 'CrossFit at the highest level.', GETDATE());

-- Dodane więzy integralności danych
ALTER TABLE class
ADD CONSTRAINT chk_difficulty_level CHECK (difficulty_level IN
('Beginner', 'Intermediate', 'Advanced'));

ALTER TABLE schedule
ADD CONSTRAINT chk_class_timing CHECK (start_time < end_time);

ALTER TABLE feedback
ADD CONSTRAINT chk_feedback_date CHECK (feedback_date <=
GETDATE());

ALTER TABLE equipment
ADD CONSTRAINT chk_purchase_date CHECK (purchase_date <=
GETDATE());

ALTER TABLE purchased_membership
ADD CONSTRAINT chk_membership_status CHECK (status IN ('active',
'inactive', 'expired'));

ALTER TABLE membership_actions
ADD CONSTRAINT chk_action_date CHECK (action_date <= GETDATE());

ALTER TABLE equipment_maintenance
ADD CONSTRAINT chk_maintenance_date CHECK (maintenance_date <=
GETDATE());

ALTER TABLE class_booking
```

```
ADD CONSTRAINT chk_booking_status CHECK (status IN ('confirmed',
'cancelled', 'pending'));

ALTER TABLE schedule
ADD CONSTRAINT uq_schedule_room_time UNIQUE (room, week_day,
start_time);

ALTER TABLE class
ADD CONSTRAINT uq_class_name UNIQUE (name);



-- Widoki

--1.
CREATE VIEW vw_active_clients AS
SELECT c.client_id, p.name, p.email, p.phone, pm.valid_until
FROM client c
JOIN person p ON c.person_id = p.person_id
JOIN purchased_membership pm ON c.client_id = pm.client_id
WHERE pm.status = 'active' AND pm.valid_until >= GETDATE();

--2.
CREATE FUNCTION vw_get_schedule_by_day (@week_day NVARCHAR(20))
RETURNS TABLE
AS
RETURN (
    SELECT s.schedule_id,
           cl.name AS class_name,
           p.name AS trainer_name,
           s.room,
           s.start_time,
           s.end_time
    FROM schedule s
    JOIN class cl ON s.class_id = cl.class_id
    JOIN trainer t ON s.trainer_id = t.trainer_id
    JOIN person p ON t.person_id = p.person_id
    WHERE s.week_day = @week_day
);

--3.
CREATE VIEW vw_average_attendance_by_category AS
SELECT cl.category,
       SUM(cl.max_participants) AS total_slots,
       COUNT(cb.booking_id) AS total_booked,
       COUNT(a.attendance_id) AS total_attended,
       ROUND((COUNT(a.attendance_id) * 100.0 /
SUM(cl.max_participants)),2) AS attendance_rate
FROM class cl
JOIN schedule s ON cl.class_id = s.class_id
JOIN class_booking cb ON s.schedule_id = cb.schedule_id
```

```sql
LEFT JOIN attendance a ON cb.booking_id = a.booking_id AND
a.status = 'present'
GROUP BY cl.category

--4.
CREATE VIEW vw_top_trainers_by_rating AS
SELECT t.trainer_id, p.name AS trainer_name, AVG(f.rating) AS
average_rating
FROM trainer t
JOIN person p ON t.person_id = p.person_id
JOIN feedback f ON t.trainer_id = f.trainer_id
GROUP BY t.trainer_id, p.name
ORDER BY average_rating DESC;

--5.
CREATE VIEW vw_clients_with_expiring_memberships AS
WITH expiring_memberships AS (
    SELECT c.client_id, p.name AS client_name, pm.valid_until
    FROM client c
    JOIN person p ON c.person_id = p.person_id
    JOIN purchased_membership pm ON c.client_id = pm.client_id
    WHERE pm.valid_until BETWEEN GETDATE() AND DATEADD(DAY, 7,
GETDATE())
)
SELECT client_id, client_name, valid_until
FROM expiring_memberships;

--6.
CREATE VIEW vw_membership_revenue_by_month AS
SELECT YEAR(pm.purchase_date) AS year, MONTH(pm.purchase_date) AS
month, SUM(m.base_price) AS total_revenue
FROM purchased_membership pm
JOIN membership m ON pm.membership_id = m.membership_id
GROUP BY YEAR(pm.purchase_date), MONTH(pm.purchase_date)

--7.
CREATE VIEW vw_equipment_needing_maintenance AS
SELECT e.equipment_id, e.name, e.manufacturer, e.purchase_date,
e.status
FROM equipment e
WHERE e.status = 'requires_maintenance';


--8.
CREATE VIEW vw_TrainerPerformance AS
SELECT
    p.name AS trainer_name,
    t.specialisation,
    COUNT(DISTINCT s.schedule_id) AS total_classes,
    COUNT(DISTINCT cb.booking_id) AS total_bookings,
    AVG(CAST(CASE WHEN a.status = 'present' THEN 1 ELSE 0 END AS
FLOAT)) AS avg_attendance_rate
```

```
FROM trainer t
JOIN person p ON t.person_id = p.person_id
JOIN schedule s ON t.trainer_id = s.trainer_id
LEFT JOIN class_booking cb ON s.schedule_id = cb.schedule_id
LEFT JOIN attendance a ON cb.booking_id = a.booking_id
GROUP BY p.name, t.specialisation;

--9.
CREATE VIEW vw_top_attending_clients AS
SELECT
    c.client_id,
    p.name AS client_name,
    COUNT(a.attendance_id) AS total_attendances
FROM client c
JOIN person p ON c.person_id = p.person_id
JOIN class_booking cb ON c.client_id = cb.client_id
JOIN attendance a ON cb.booking_id = a.booking_id
WHERE a.status = 'present'
GROUP BY c.client_id, p.name

--10.
CREATE VIEW vw_longest_membership_clients AS
SELECT
    c.client_id,
    p.name AS client_name,
    DATEDIFF(DAY, c.join_date, GETDATE()) AS
membership_duration_days
FROM client c
JOIN person p ON c.person_id = p.person_id

-- Procedury Składowane

-- 1.
CREATE PROCEDURE sp_add_client
    @name NVARCHAR(100),
    @email NVARCHAR(100),
    @phone NVARCHAR(20),
    @birth_date DATE,
    @join_date DATE,
    @status NVARCHAR(50),
    @membership_id INT
AS
BEGIN
    DECLARE @person_id INT;
    DECLARE @client_id INT;

    INSERT INTO person (name, email, phone, birth_date)
    VALUES (@name, @email, @phone, @birth_date);

    SET @person_id = SCOPE_IDENTITY();

    INSERT INTO client (person_id, join_date, status)
```

```
    VALUES (@person_id, @join_date, @status);

    SET @client_id = SCOPE_IDENTITY();

    INSERT INTO purchased_membership (client_id, membership_id,
purchase_date, valid_until, status)
    SELECT
        @client_id,
        @membership_id,
        GETDATE(),
        DATEADD(DAY, m.duration_days, GETDATE()),
        'active'
    FROM membership m
    WHERE m.membership_id = @membership_id;
END;

--2.
CREATE PROCEDURE sp_BookClass
    @client_id INT,
    @schedule_id INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        IF NOT EXISTS (
            SELECT 1
            FROM purchased_membership
            WHERE client_id = @client_id
              AND valid_until >= GETDATE()
              AND status = 'active'
        )
        BEGIN
            THROW 50001, 'Client has no active membership.', 1;
        END

        IF NOT EXISTS (
            SELECT 1
            FROM schedule
            WHERE schedule_id = @schedule_id
        )
        BEGIN
            THROW 50003, 'The specified class schedule does not
exist.', 1;
        END

        DECLARE @available_spots INT;
        SELECT @available_spots = c.max_participants -
COUNT(cb.booking_id)
        FROM schedule s
        JOIN class c ON s.class_id = c.class_id
```

```
        LEFT JOIN class_booking cb ON s.schedule_id =
cb.schedule_id AND cb.status = 'confirmed'
        WHERE s.schedule_id = @schedule_id
        GROUP BY c.max_participants;

        IF @available_spots <= 0
        BEGIN
            THROW 50002, 'No available slots for this class.', 1;
        END

        DECLARE @booking_id INT;
        INSERT INTO class_booking (client_id, schedule_id,
booking_date, status)
        VALUES (@client_id, @schedule_id, GETDATE(), 'confirmed');

        SET @booking_id = SCOPE_IDENTITY();

        INSERT INTO attendance (booking_id, attendance_date,
status)
        VALUES (@booking_id, NULL, 'absent');

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;

        THROW;
    END CATCH
END;

--3.
CREATE PROCEDURE sp_CancelBooking
    @booking_id INT
AS
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM class_booking
        WHERE booking_id = @booking_id
        AND status = 'confirmed'
    )
    BEGIN
        THROW 50003, 'Reservation does not exist or is already
cancelled.', 1;
        RETURN;
    END

    DELETE FROM attendance
    WHERE booking_id = @booking_id;

    UPDATE class_booking
    SET status = 'cancelled'
    WHERE booking_id = @booking_id;
```

```sql
END;

--4.
CREATE PROCEDURE sp_MarkAttendance
    @booking_id INT,
    @attendance_status NVARCHAR(50)
AS
BEGIN
    IF NOT EXISTS (
        SELECT 1 FROM class_booking
        WHERE booking_id = @booking_id
    )
    BEGIN
        THROW 50004, 'Reservation does not exist.', 1;
        RETURN;
    END

    UPDATE attendance
    SET status = @attendance_status,
        attendance_date = GETDATE()
    WHERE booking_id = @booking_id;
END;

--5.
CREATE PROCEDURE sp_LeaveFeedback
    @client_id INT,
    @trainer_id INT,
    @rating DECIMAL(3, 2),
    @comment NVARCHAR(255) = NULL
AS
BEGIN
    IF @rating < 1 OR @rating > 5
    BEGIN
        THROW 50008, 'Rating has to be 1 - 5.', 1;
        RETURN;
    END

    IF NOT EXISTS (
        SELECT 1 FROM purchased_membership
        WHERE client_id = @client_id
        AND valid_until >= GETDATE()
        AND status = 'active'
    )
    BEGIN
        THROW 50006, 'Client has no active membership.', 1;
        RETURN;
    END

    IF NOT EXISTS (
        SELECT 1
        FROM class_booking cb
        JOIN schedule s ON cb.schedule_id = s.schedule_id
```

```
        WHERE cb.client_id = @client_id
          AND s.trainer_id = @trainer_id
    )
    BEGIN
        THROW 50007, 'Client did not attend any classes with this
trainer.', 1;
        RETURN;
    END

    INSERT INTO feedback (client_id, trainer_id, rating, comment,
feedback_date)
    VALUES (@client_id, @trainer_id, @rating, @comment,
GETDATE());

    PRINT 'Review added successfully';
END;

-- Wyzwalacze

--1.
CREATE TRIGGER trg_check_class_capacity
ON class_booking
AFTER INSERT
AS
BEGIN
    DECLARE @max_participants INT;
    DECLARE @current_bookings INT;

    SELECT @max_participants = c.max_participants,
           @current_bookings = COUNT(cb.booking_id)
    FROM class c
    JOIN schedule s ON c.class_id = s.class_id
    JOIN class_booking cb ON s.schedule_id = cb.schedule_id
    WHERE cb.schedule_id IN (SELECT schedule_id FROM inserted)
    GROUP BY c.max_participants;

    IF @current_bookings > @max_participants
    BEGIN
     THROW 50010, 'The maximum number of participants for this
class has been exceeded.', 1;
        ROLLBACK TRANSACTION;
    END
END;

--2.
CREATE TRIGGER trg_cancel_bookings_after_membership_expiry
ON purchased_membership
AFTER UPDATE
AS
BEGIN
    IF UPDATE(valid_until)
    BEGIN
```

```
        UPDATE cb
        SET cb.status = 'cancelled'
        FROM class_booking cb
        JOIN inserted i ON cb.client_id = i.client_id
        WHERE i.valid_until < GETDATE();
    END
END;

--3.
CREATE TRIGGER trg_auto_set_membership_validity
ON purchased_membership
AFTER INSERT
AS
BEGIN
    UPDATE pm
    SET pm.valid_until = DATEADD(DAY, m.duration_days,
pm.purchase_date)
    FROM purchased_membership pm
    JOIN inserted i ON pm.purchased_membership_id =
i.purchased_membership_id
    JOIN membership m ON pm.membership_id = m.membership_id;
END;

--4.
CREATE TRIGGER trg_prevent_class_overlap
ON schedule
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN schedule s ON s.room = i.room
        WHERE s.schedule_id != i.schedule_id
        AND (
            (i.start_time BETWEEN s.start_time AND s.end_time)
            OR (i.end_time BETWEEN s.start_time AND s.end_time)
        )
    )
    BEGIN
          THROW 50020, 'Class time overlaps with existing
schedule', 1;
        ROLLBACK TRANSACTION;
    END
END;

--5.
CREATE TRIGGER trg_update_class_status_after_booking
ON class_booking
AFTER INSERT
AS
BEGIN
```

```
    DECLARE @schedule_id INT;
    DECLARE @max_participants INT;
    DECLARE @current_bookings INT;

    SELECT @schedule_id = i.schedule_id FROM inserted i;

    SELECT @max_participants = c.max_participants
    FROM schedule s
    JOIN class c ON s.class_id = c.class_id
    WHERE s.schedule_id = @schedule_id;

    SELECT @current_bookings = COUNT(*)
    FROM class_booking
    WHERE schedule_id = @schedule_id;

    IF @current_bookings >= @max_participants
    BEGIN
        UPDATE schedule
        SET status = 'full'
        WHERE schedule_id = @schedule_id;
    END
END;

-- Indeksy

CREATE INDEX IX_Trainers_Specialization
ON trainer(specialisation);

CREATE INDEX IX_Schedule_ClubDayTime
ON schedule(room, week_day, start_time);

CREATE INDEX IX_Clients_JoinDate
ON client(join_date);

CREATE INDEX IX_Equipment_Status
ON equipment(status);

-- Przykładowe zapytania

-- Wyświetl aktywnych klientów
SELECT * FROM active_clients;

-- Wyświetl harmonogram zajęć na poniedziałek
SELECT * FROM get_schedule_by_day('Monday');

-- Wyświetl trenerów z najwyższą średnią ocen
SELECT * FROM top_trainers_by_rating;

-- Wyświetl średnią ocenę każdego trenera wraz z liczbą
otrzymanych recenzji
SELECT
    t.trainer_id,
```

```sql
    p.name AS trainer_name,
    AVG(f.rating) AS average_rating,
    COUNT(f.feedback_id) AS number_of_reviews
FROM trainer t
JOIN person p ON t.person_id = p.person_id
LEFT JOIN feedback f ON t.trainer_id = f.trainer_id
GROUP BY t.trainer_id, p.name
ORDER BY average_rating DESC;

-- Wyświetl ile klientów posiada każdy typ karnetu
SELECT
    m.name AS membership_type,
    COUNT(pm.client_id) AS number_of_clients
FROM membership m
LEFT JOIN purchased_membership pm ON m.membership_id =
pm.membership_id
WHERE pm.status = 'active' AND pm.valid_until >= GETDATE()
GROUP BY m.name
ORDER BY number_of_clients DESC;

-- Wyświetl które zajęcia są najpopularniejsze (mają najwięcej
rezerwacji)
SELECT
    c.name AS class_name,
    COUNT(cb.booking_id) AS number_of_bookings
FROM class c
JOIN schedule s ON c.class_id = s.class_id
JOIN class_booking cb ON s.schedule_id = cb.schedule_id
WHERE cb.status = 'confirmed'
GROUP BY c.name
ORDER BY number_of_bookings DESC;


-- Zapytanie o frekwencję na zajęciach
SELECT
    c.name AS class_name,
    COUNT(a.attendance_id) AS attended,
    COUNT(cb.booking_id) AS booked,
    ROUND((COUNT(a.attendance_id) * 100.0 / COUNT(cb.booking_id),
2) AS attendance_rate
FROM class c
JOIN schedule s ON c.class_id = s.class_id
JOIN class_booking cb ON s.schedule_id = cb.schedule_id
LEFT JOIN attendance a ON cb.booking_id = a.booking_id AND
a.status = 'present'
GROUP BY c.name
ORDER BY attendance_rate DESC;
```