Mateusz Jędrkowiak, Karolina Kulas & Aleksander Wiśniewski

Uniwersytet Jagiellonski, 2025 # Baza danych klubu fitness - “MAK fintess”

Cele:

1. **Efektywne zarządzanie danymi** – baza danych ma na celu przechowywanie, organizowanie i przetwarzanie informacji związanych z działalnością klubu fitness, w tym faktur, płatności, członkostw, rezerwacji zajęć i opinii użytkowników.

2. **Automatyzacja procesów** – system umożliwia automatyczne rejestrowanie transakcji, monitorowanie obecności na zajęciach oraz zarządzanie harmonogramami treningów.

3. **Personalizacja usług** – dzięki przechowywaniu szczegółowych danych o użytkownikach, ich członkostwach i aktywności, można dostosowywać ofertę klubu do indywidualnych potrzeb klientów.

4. **Optymalizacja operacyjna** – baza wspiera zarządzanie personelem, sprzętem oraz zamówieniami, co usprawnia codzienne funkcjonowanie klubu fitness.

5. **Analiza i raportowanie** – umożliwia gromadzenie danych o frekwencji, dochodach, skuteczności trenerów oraz poziomie satysfakcji klientów, co pozwala na podejmowanie lepszych decyzji biznesowych.

Możliwości:

1. **Elastyczne zarządzanie członkostwami** – obsługa różnych typów członkostw (indywidualnych i firmowych) z możliwością przypisywania różnych uprawnień i korzyści.

2. **Śledzenie płatności i faktur** – rejestrowanie płatności, generowanie faktur oraz zarządzanie rabatami i kodami zniżkowymi.

3. **Rezerwacja zajęć i treningów personalnych** – użytkownicy mogą zapisywać się na zajęcia, a system może zarządzać listami oczekujących.

4. **Oceny i recenzje** – użytkownicy mogą wystawiać opinie o trenerach i zajęciach, co pozwala na ocenę jakości usług.

5. **Zarządzanie zasobami klubu** – system przechowuje dane o sprzęcie, pracownikach oraz obiektach, co pozwala na lepszą organizację pracy klubu.

6. **Rywalizacja i motywacja użytkowników** – ranking członków (Leaderboard) umożliwia prowadzenie konkursów i zwiększanie zaangażowania klientów.

7. **Obsługa sprzedaży i zamówień** – rejestrowanie sprzedaży produktów (merch), realizacja zamówień i zarządzanie stanami magazynowymi.

Ograniczenia:

1. **Złożoność systemu** – duża liczba powiązanych tabel wymaga starannego projektowania zapytań oraz optymalizacji wydajności, aby uniknąć opóźnień w przetwarzaniu danych.

2. **Potrzeba regularnej konserwacji** – konieczne jest monitorowanie integralności danych, optymalizacja indeksów oraz tworzenie kopii zapasowych, aby zapewnić niezawodność systemu.

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

4. **Skalowalność** – przy dużej liczbie użytkowników może być konieczna optymalizacja struktury bazy lub migracja do bardziej wydajnego systemu, aby uniknąć problemów z wydajnością.

5. **Integracja z innymi systemami** – możliwość wymiany danych z systemami płatności, księgowości czy aplikacjami mobilnymi może wymagać dodatkowych rozwiązań technicznych i interfejsów API.

Schemat pielęgnacji bazy danych

- **Codzienne kopie zapasowe** – każdej nocy, w czasie przestoju restauracji, należy wykonywać różnicowe backupy, aby zapewnić aktualność danych.

- **Pełne kopie zapasowe** – raz w tygodniu, w godzinach nocnych, zaleca się tworzenie pełnych kopii zapasowych, co umożliwi szybkie odtworzenie systemu w razie awarii.

- **Monitorowanie integralności danych** – regularna weryfikacja spójności i poprawności relacji między tabelami pozwoli uniknąć błędów logicznych i nieprawidłowych powiązań.

- **Optymalizacja wydajności** – okresowe odświeżanie indeksów i statystyk bazy danych w celu przyspieszenia operacji i zwiększenia efektywności zapytań.

- **Polityka retencji danych** – ustalenie jasnych zasad dotyczących archiwizacji i usuwania przestarzałych danych transakcyjnych, aby zapobiec niekontrolowanemu wzrostowi bazy danych.

- **Dodatkowe zabezpieczenia** – wdrożenie mechanizmów szyfrowania i kontroli dostępu, aby chronić wrażliwe informacje przed nieautoryzowanym dostępem.

- **Testy odtwarzania danych** – regularne przeprowadzanie próbnego przywracania systemu z backupów, aby upewnić się, że proces działa sprawnie i pozwala na szybkie odzyskanie danych w razie awarii.

Lista tabel

kod

1. **Invoices** – Przechowuje dane o fakturach, takie jak numer faktury, data wystawienia, kwota, status płatności i powiązany członek lub firma.
2. **Payments** – Zawiera informacje o dokonanych płatnościach, w tym metodę płatności, kwotę, datę oraz powiązaną fakturę.
3. **Members** – Przechowuje dane o członkach klubu, takie jak imię, nazwisko, e-mail, numer telefonu, adres i status członkostwa.
4. **Memberships** – Rejestruje informacje o członkostwach, w tym typ, czas trwania, cenę i powiązane uprawnienia.
5. **IndvidualMemberships** – Zawiera szczegółowe informacje o członkostwach indywidualnych, przypisanych do konkretnego członka.
6. **CompanyMemberships** – Przechowuje dane o członkostwach firmowych, które mogą obejmować wielu pracowników jednej firmy.
7. **MembershipActions** – Rejestruje działania związane z członkostwem, np. przedłużenia, anulowania czy zmiany pakietu.
8. **Leaderboard** – Przechowuje dane o wynikach i aktywności użytkowników w klubie, np. liczba odwiedzin, treningi, osiągnięcia.
9. **Trainers** – Zawiera informacje o trenerach, ich specjalizacjach, doświadczeniu, certyfikatach oraz dostępności.
10. **Reviews** – Ogólna tabela przechowująca recenzje różnych usług i trenerów w klubie.
11. **TrainerReviews** – Przechowuje oceny i opinie użytkowników na temat konkretnych trenerów.
12. **ClassesReviews** – Zawiera recenzje dotyczące zajęć grupowych, ich jakości, instruktora oraz poziomu trudności.
13. **Classes** – Opisuje zajęcia fitness, ich typ, poziom trudności, maksymalną liczbę uczestników i powiązanego instruktora.
14. **ClassTrainers** – Powiązanie trenerów z zajęciami, pozwala na przypisanie kilku trenerów do jednej klasy.
15. **ClassEnrollments** – Przechowuje dane o zapisach na zajęcia, w tym użytkownika, zajęcia i datę zapisania.
16. **ClassSchedules** – Zawiera harmonogramy zajęć, godziny rozpoczęcia, daty oraz

dostępność miejsc.
17. **WaitLists** – Przechowuje listy oczekujących na zajęcia, jeśli nie ma już dostępnych miejsc.
18. **PersonalTrainings** – Rejestruje indywidualne treningi, ich daty, czas trwania, trenera oraz uczestnika.
19. **Equipment** – Przechowuje informacje o sprzęcie dostępnym w klubie, jego stanie, dostępności i terminach konserwacji.
20. **FitnessClubs** – Zawiera dane o różnych lokalizacjach siłowni, ich adresach, godzinach otwarcia i oferowanych usługach.
21. **Employees** – Rejestruje pracowników klubu, ich stanowiska, grafik pracy oraz wynagrodzenia.
22. **Merch** – Przechowuje informacje o produktach sprzedawanych w klubie, np. odzież sportowa, suplementy.
23. **MerchOrders** – Zawiera dane o zamówieniach na produkty, w tym klienta, zamówione przedmioty, datę i status realizacji.
24. **DiscountCodes** – Przechowuje kody rabatowe, ich wartości, warunki użycia oraz daty ważności.
25. **Attendance** – Rejestruje obecność członków w klubie, w tym datę, godzinę wejścia i wyjścia oraz powiązane członkostwo.

Widoki

kod

1. **vwAverageTrainerRating** - **Średnia ocena trenera**
    - Ten widok oblicza średnią ocenę dla każdego trenera na podstawie ocen w recenzjach. W przypadku braku recenzji dla trenera, zwróci wartość 0 (dzięki funkcji `COALESCE`). Widok zawiera identyfikator trenera, jego nazwisko oraz średnią ocenę.
2. **vwPromoCodeTransactions** - **Transakcje z użyciem kodu promocyjnego**
    - Widok ten pokazuje, ile transakcji skorzystało z danego kodu rabatowego. Zlicza liczbę transakcji, które wykorzystały określony kod rabatowy, łącząc tabele płatności z kodami rabatowymi.
3. **vwMemberAttendance** - **Obecności członków**
    - Widok zawiera dane o obecnościach członków na zajęciach, łącząc tabele `Attendance`, `Members` i `Classes`. Zawiera identyfikator obecności, identyfikator członka, typ członkostwa, identyfikator zajęć, nazwę zajęć, datę obecności oraz status obecności (np. obecny, nieobecny).
4. **vwEnrollmentExtremes** - **Zajęcia z największą i najmniejszą liczbą zapisów**
    - Widok ten identyfikuje zajęcia z największą oraz najmniejszą liczbą zapisanych uczestników. Wykorzystuje CTE (Common Table Expressions) do zliczenia liczby zapisów na każde zajęcia oraz do znalezienia maksymalnej i minimalnej liczby zapisów. Na końcu filtruje wyniki, aby pokazać tylko te

zajęcia, które mają największą lub najmniejszą liczbę zapisanych uczestników.

5. **vwAverageClassRating** - **Średnia ocena zajęć**
    - Widok oblicza średnią ocenę dla każdego typu zajęć. Łączy tabele `Classes`, `ClassesReviews` i `Reviews`, aby obliczyć średnią ocenę dla każdego kursu, w tym także uwzględnia poziom trudności kursu.
6. **vwTop3InEachGroup - Top 3 członków w każdej grupie**
    - Ten widok pokazuje trzech najlepszych członków w tabeli wyników dla każdej grupy zajęciowej. Wybiera tylko członków z pozycją w rankingu równą lub wyższą niż 3, a następnie grupuje wynik według grup zajęciowych.
7. **vwCountMembershipType - Liczba każdego typu członkostwa**
    - Ten widok zlicza liczbę zarejestrowanych członkostw dla każdego typu członkostwa.
8. **vwActiveMembersEnrolled - Liczba aktywnych członków zapisanych na każde zaplanowane zajęcia**
    - Ten widok pokazuje, ilu aktywnych członków zapisało się na zaplanowane zajęcia. Łączy tabele `ClassSchedules` i `ClassEnrollments`, aby policzyć liczbę zapisanych członków dla każdego kursu. Dodatkowo wynik jest filtrowany, aby uwzględnić tylko aktywnych członków.
9. **vwTotalIncomePerMonth - Całkowity przychód w każdym miesiącu**
    - Ten widok pokazuje, ile klub zarobił na członkostwach w każdym miesiącu. Grupuje płatności według miesiąca i oblicza ich sumę dla każdego miesiąca.
10. **vwLocationsSortedByNeededMaintanance - Ilosc potrzebnych konserwacji w kazdej lokalizacji**
    - Ten widok pokazuje lokalizacje klubów oraz ilość sprzętu wymagającego konserwacji w każdej z nich.

Wyzwalacze

kod

1. **trAddToLeaderboard** - **Automatyczne dodanie członka do tabeli liderów po 10 treningach**
    - Ten wyzwalacz automatycznie dodaje członka do tabeli liderów, gdy po zapisaniu treningu osobistego (w tabeli `PersonalTrainings`) członek osiągnie 10 zakończonych treningów. Sprawdza, czy członek już znajduje się w tabeli liderów, a jeśli nie, dodaje go z liczbą ukończonych treningów oraz obliczoną liczbą godzin (liczba treningów pomnożona przez 1.5).
2. **trUpdateInvoiceStatus** - **Aktualizacja statusu faktury na ‘Opłacona’ po pełnej płatności**
    - Wyzwalacz ten aktualizuje status faktury na “Opłacona”, gdy po wprowadzeniu lub zaktualizowaniu płatności (tabela `Payments`) suma zapłaconej kwoty osiągnie całkowitą kwotę faktury. Sprawdza, czy suma

wszystkich płatności za daną fakturę jest większa lub równa pełnej kwocie faktury, a następnie zmienia jej status na “Opłacona”.

3. **trApplyDiscount** - **Automatyczne zastosowanie rabatu na podstawie kodu promocyjnego**
    - Ten wyzwalacz modyfikuje płatność wstawioną do tabeli `Payments` (zamiast standardowego wstawiania), aby automatycznie zastosować rabat, jeśli został użyty aktywny kod promocyjny. Oblicza nową kwotę zapłaty, uwzględniając procent rabatu związanego z kodem promocyjnym. Zastosowanie rabatu zależy od tego, czy kod promocyjny jest aktywny.
4. **trDeactivateExpiredMembership** - **Automatyczne dezaktywowanie wygasłych członkostw**
    - Wyzwalacz ten dezaktywuje członkostwo, ustawiając `MembershipID` na NULL w tabeli `Members`, gdy członek zakończy swoje członkostwo (akcja typu “Cancelation”) i data zakończenia członkostwa w tabeli `membershipactions` jest wcześniejsza lub równa bieżącej dacie. Oznacza to, że członkostwo wygasło, a członek już nie jest aktywnym użytkownikiem.
5. **trRemoveFromLeaderboard** - **Automatyczne usunięcie z tabeli liderów, gdy członkostwo wygasa**
    - Ten wyzwalacz usuwa członka z tabeli liderów, jeśli jego członkostwo wygasło (akcja typu “Cancelation” w tabeli `membershipactions`). Jeśli data zakończenia członkostwa jest wcześniejsza lub równa bieżącej dacie, członek zostaje usunięty z tabeli `Leaderboard`, co oznacza, że już nie jest częścią systemu rankingu.

Procedury składowane

kod

1. **UpdateEquipmentMaintenanceDate** - **Procedura aktualizacji daty konserwacji sprzętu**
    - Procedura ta umożliwia aktualizację daty ostatniej konserwacji sprzętu na podstawie przekazanego identyfikatora sprzętu (`@EquipmentID`) oraz nowej daty konserwacji (`@NewMaintenanceDate`). Zaktualizowana zostanie tabela `Equipment`, gdzie wartość w kolumnie `LastMaintenance` zostanie ustawiona na nową datę konserwacji.
2. **AddReview** - **Procedura dodawania recenzji**
    - Procedura ta pozwala na dodanie recenzji, która może dotyczyć trenera lub zajęć (w zależności od wartości parametru `@ReviewType`). Wstawiane są dane do tabeli `Reviews`, a następnie, w zależności od rodzaju recenzji, dodawane są wpisy do tabel `TrainerReviews` (dla recenzji trenera) lub `ClassesReviews` (dla recenzji zajęć). Jeżeli podany typ recenzji jest błędny, procedura wypisuje komunikat o błędzie.
3. **UpdateLeaderboard** - **Procedura aktualizacji tabeli liderów**

- Procedura ta aktualizuje tabelę liderów na podstawie obecności członka na zajęciach. Jeśli status obecności to “Present” (obecny), sprawdzane jest, czy dany członek i zajęcia znajdują się już w tabeli liderów. Jeśli tak, liczba treningów oraz godziny treningu są aktualizowane. Jeśli nie, członek i zajęcia są dodawane do tabeli liderów. Następnie rankingi w tabeli liderów są aktualizowane na podstawie sumy godzin treningów.

4. **CancelOverDueMembers** - **Procedura anulowania członkostwa dla członków z niezapłaconymi fakturami**
   - Procedura ta anuluje członkostwo członków, którzy mają zaległe płatności (faktury z datą wymagalności wcześniejszą niż dzisiejsza, o statusie “Unpaid” lub “Pending”). Dodatkowo, akcja anulowania członkostwa jest rejestrowana w tabeli `MembershipActions`, gdzie zapisane są szczegóły anulowania, takie jak data anulowania i powód.
5. **usp_UpdateWaitlistForClass** - **Procedura aktualizacji listy oczekujących na zajęcia**
   - Procedura ta sprawdza dostępne miejsca na zajęciach (w tabeli `ClassEnrollments`), porównując liczbę zapisanych osób z maksymalną pojemnością klasy. Jeśli są dostępne wolne miejsca, aktualizuje status osób z listy oczekujących (`Waitlists`), które mogą zostać przeniesione do listy potwierdzonych uczestników. Dodatkowo, numer kolejki na liście oczekujących jest aktualizowany, aby zachować odpowiednią kolejność.

Indeksy

kod

1. **IX_Invoices_Member_Status**
   - Ten indeks przyspiesza zapytania filtrujące faktury według identyfikatora członka (`@MemberID`) i statusu, co jest szczególnie przydatne przy wyszukiwaniu faktur o określonym statusie płatności. Dzięki łączeniu tych kolumn, baza danych szybciej odnajduje właściwe rekordy.
2. **IX_Trainers_Specialization**
   - Indeks ten zwiększa wydajność wyszukiwania trenerów według ich specjalizacji. Umożliwia to szybkie odnalezienie trenera o konkretnej specjalizacji, co jest kluczowe przy przypisywaniu ich do odpowiednich treningów.
3. **IX_ClassSchedules_ClubDayTime**
   - Ten indeks kompozytowy na kolumnach (`@FitnessClubID`), (`@Day`) oraz (`@StartTime`) optymalizuje pobieranie harmonogramów zajęć. Grupowanie tych kolumn pozwala na szybkie filtrowanie i sortowanie danych według klubu, dnia tygodnia i godziny rozpoczęcia zajęć.
4. **IX_Members_JoinDate**

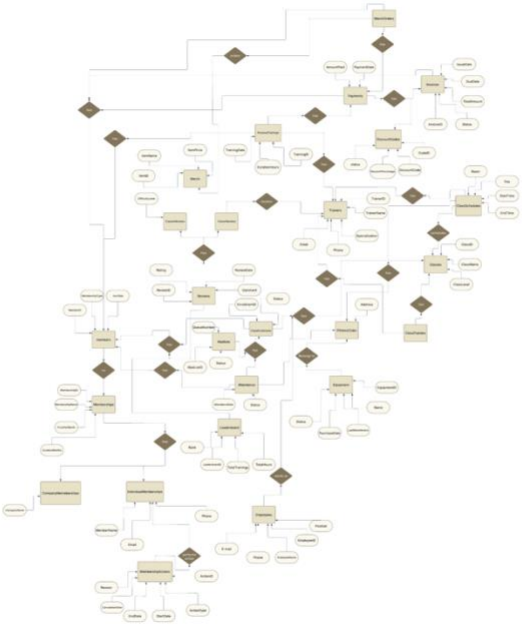
- Indeks ten przyspiesza wyszukiwanie i sortowanie rekordów w tabeli `Members` na podstawie daty dołączenia (`@JoinDate`). Dzięki temu zapytania analizujące trendy rejestracji członków działają bardziej efektywnie.

Typowe zapytania

kod

1. **Zapytanie o nieopłacone faktury**
   - To zapytanie zwraca szczegóły faktur (InvoiceID, MemberID, IssueDate, DueDate, TotalAmount), które mają status “Unpaid”.
2. **Zapytanie o aktywne kody rabatowe**
   - To zapytanie wyświetla identyfikatory, kody rabatowe i procentowe zniżki dla kodów o statusie “Active”.
3. **Zapytanie o recenzje trenerów**
   - To zapytanie łączy tabele Reviews, TrainerReviews i Trainers, aby wyświetlić recenzje wraz z nazwą trenera.
4. **Zapytanie o liczbę członków według typu członkostwa**
   - To zapytanie grupuje członków według typu członkostwa i zlicza ich ilość, wyświetlając MembershipID, MembershipName oraz liczbę członków.
5. **Zapytanie o anulowane karnety w bieżącym miesiącu**
   - To zapytanie zlicza liczbę anulowanych karnetów (ActionType = ‘Cancelation’) w bieżącym miesiącu, porównując datę anulacji z aktualną datą.

Diagramy relacji



Dodatkowe więzy integralności danych

1. **Ograniczenia CHECK**
    - Equipment -> Status: CHECK (Status IN ('Operational', 'Maintenance Required', 'Out of Service'))
    - Members -> MembershipType: CHECK (MembershipType IN ('Individual', 'Company'))
    - MembershipActions -> ActionType: CHECK (ActionType IN ('Suspension', 'Cancelation'))
    - Invoices -> Status: CHECK (Status IN ('Paid', 'Unpaid', 'Pending'))
    - DiscountCodes -> Status: CHECK (Status IN ('Active', 'Inactive'))
    - Classes -> ClassLevel: CHECK (ClassLevel BETWEEN 1 AND 5)
    - ClassEnrollments -> Status: CHECK (Status IN ('Active', 'Completed', 'Dropped'))
    - ClassSchedules -> Day: CHECK (Day IN ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'))
    - Waitlists -> Status: CHECK (Status IN ('Waiting', 'Confirmed', 'Cancelled'))
    - Attendance -> Status: CHECK (Status IN ('Present', 'Absent', 'Excused'))
    - Reviews -> Rating: CHECK (Rating BETWEEN 1 AND 5)
    - ClassesReviews -> DifficultyLevel: CHECK (DifficultyLevel BETWEEN 1 AND 5)
    - Merch -> ItemPrice: CHECK (ItemPrice >= 0)
    - MerchOrders -> Size: CHECK (Size IN ('S', 'M', 'L', 'XL'))
2. **Kaskadowe operacja usuwania** ON DELETE CASCADE zapewnia, że po usunięciu powiązanych danych usunięte zostaną również wpisy zależne (np. sprzęt po usunięciu klubu, recenzje po usunięciu członka).
    - Usunięcie klubu → usuwa sprzęt
    - Usunięcie członka → usuwa wszystkie jego powiązane dane (np. aktywności, płatności, zapisy na zajęcia, recenzje).
    - Usunięcie członkostwa → usuwa wszystkich członków, którzy je mieli.
    - Usunięcie faktury → usuwa płatności powiązane z fakturą.
    - Usunięcie zajęć → usuwa ich rejestracje, harmonogramy, oceny, listy oczekujących i obecności.

- Usunięcie rejestracji na zajęcia → usuwa z listy oczekujących i z listy obecności.
- Usunięcie recenzji → usuwa recenzje trenerów i zajęć.
- Usunięcie produktu w sklepie → usuwa zamówienia na ten produkt.
- DELETE SET NULL sprawia, że jeśli kod rabatowy zostanie usunięty, to w Payments jego wartość zmieni się na NULL.

3. **Unikalne rekordy**
   - UNIQUE zapewnia unikalność w Phone, Email, Rank i (ClassID, Rank).
4. **Ograniczenie NOT NULL** NOT NULL wymusza obecność wartości w wielu kluczowych polach
   - FitnessClubs -> Address
   - Equipment -> Status
   - Invoices -> Status
   - DiscountCodes -> Status
   - Classes -> ClassLevel
   - ClassEnrollments -> Status
   - ClassSchedules -> Day
   - Waitlists -> Status
   - Attendance -> Status
   - Leaderboard -> Rank
   - Reviews -> Rating, ReviewDate
   - TrainerReviews -> TrainerID
   - ClassesReviews -> ClassID, DifficultyLevel
   - Merch -> ItemName, ItemPrice
   - MerchOrders -> Size

SKRYPT tworzący bazę danych

```
CREATE DATABASE FitnessClub

--TABELKI--

--Club related tabels

CREATE TABLE FitnessClubs (
    FitnessClubID INT PRIMARY KEY,
    Address NVARCHAR(255) NOT NULL
```

```
);

CREATE TABLE Equipment (
    EquipmentID INT PRIMARY KEY,
    Name NVARCHAR(100),
    FitnessClubID INT,
    LastMaintenance DATE,
    PurchaseDate DATE,
    Status NVARCHAR(50) NOT NULL CHECK (Status IN ('Operational', 'Maintenance Required', 'Out of Service')),
    FOREIGN KEY (FitnessClubID) REFERENCES FitnessClubs(FitnessClubID) ON DELETE CASCADE
);

CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    EmployeeName NVARCHAR(100),
    Phone NVARCHAR(20) UNIQUE,
    Email NVARCHAR(100) UNIQUE,
    FitnessClubID INT,
    Position NVARCHAR(100),
    FOREIGN KEY (FitnessClubID) REFERENCES FitnessClubs(FitnessClubID)
);

--Member related tables

CREATE TABLE Memberships (
    MembershipID INT PRIMARY KEY,
    MembershipName NVARCHAR(100),
    PricePerMonth DECIMAL(10,2),
    DurationMonths INT
);

--Parent table for all members - both indivduals and comapnies

CREATE TABLE Members (
    MemberID INT PRIMARY KEY,
    MembershipType NVARCHAR(20) CHECK (MembershipType IN ('Individual', 'Company')),
    MembershipID INT,
```

```
    JoinDate DATE,
    FOREIGN KEY (MembershipID) REFERENCES Memberships(MembershipID) ON DELETE CASCADE
);


--subclass for indvidual members(inherits from Members)
CREATE TABLE IndividualMemberships (
    MemberID INT PRIMARY KEY,
    MemberName NVARCHAR(100),
    Email NVARCHAR(100),
    Phone NVARCHAR(20),
    MembershipID INT,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID) ON DELETE CASCADE
);
--subclass for company membersips (inherits from Members)

CREATE TABLE CompanyMemberships (
    MemberID INT PRIMARY KEY,
    CompanyName NVARCHAR(256),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID) ON DELETE CASCADE
);


CREATE TABLE MembershipActions (
    ActionID INT PRIMARY KEY,
    MemberID INT,
    ActionType NVARCHAR(20) CHECK (ActionType IN ('Suspension', 'Cancelation')),
    StartDate DATE NULL,
    EndDate DATE NULL,
    CancelationDate DATE NULL,
    Reason NVARCHAR(500),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID) ON DELETE CASCADE
);

--Payment related tables

CREATE TABLE Invoices (
```

```
    InvoiceID INT PRIMARY KEY,
    MemberID INT,
    IssueDate DATE,
    DueDate DATE,
    TotalAmount MONEY,
    Status  NVARCHAR(20) NOT NULL CHECK (Status IN ('Paid', 'Unpaid', 'Pending')),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)

);
CREATE TABLE DiscountCodes (
    CodeID INT PRIMARY KEY,
    DiscountCode NVARCHAR(20),
    DiscountPercentage INT,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN('Active', 'Inactive'))
);
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY,
    MemberID INT,
    InvoiceID INT,
    AmountPaid MONEY,
    PaymentDate DATE,
    DiscountCodeID INT,
    FOREIGN KEY (InvoiceID) REFERENCES Invoices(InvoiceID) ON DELETE CASCADE,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID),
    FOREIGN KEY (DiscountCodeID) REFERENCES DiscountCodes(CodeID) ON DELETE SET NULL
);

--Trainer related tables

CREATE TABLE Trainers (
    TrainerID INT PRIMARY KEY,
    TrainerName NVARCHAR(100),
    Specialization NVARCHAR(100),
    Phone NVARCHAR(20) UNIQUE,
    Email NVARCHAR(100) UNIQUE
);
```

```sql
CREATE TABLE PersonalTrainings (
    TrainingID INT PRIMARY KEY,
    TrainerID INT,
    MemberID INT,
    TrainingDate DATE,
    DurationHours DECIMAL(3,1),
    PaymentID INT,
    FOREIGN KEY (TrainerID) REFERENCES Trainers(TrainerID),
    FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID)
);


--Class related tables

CREATE TABLE Classes (
    ClassID INT PRIMARY KEY,
    ClassName NVARCHAR(100),
    ClassLevel INT NOT NULL CHECK (ClassLevel BETWEEN 1 AND 5)
);

CREATE TABLE ClassTrainers (
    ClassID INT,
    TrainerID INT,
    CONSTRAINT PK_ClassTypes PRIMARY KEY (ClassID, TrainerID),
    FOREIGN KEY (ClassID) REFERENCES Classes(ClassID),
    FOREIGN KEY (TrainerID) REFERENCES Trainers(TrainerID)
);

CREATE TABLE ClassEnrollments (
    EnrollmentID INT PRIMARY KEY,
    MemberID INT,
    ClassID INT,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('Active', 'Completed', 'Dropped')),
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID),
    FOREIGN KEY (ClassID) REFERENCES Classes(ClassID)
);
```

```
CREATE TABLE ClassSchedules (
    ScheduleID INT PRIMARY KEY,
    ClassID INT,
    TrainerID INT,
    Room NVARCHAR(10),
    Day NVARCHAR(10) NOT NULL CHECK (Day IN ('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday',
'Saturday', 'Sunday')),
    StartTime TIME,
    EndTime TIME,
    FitnessClubID INT,
    FOREIGN KEY (TrainerID) REFERENCES Trainers(TrainerID),
    FOREIGN KEY (ClassID) REFERENCES Classes(ClassID),
    FOREIGN KEY (FitnessClubID) REFERENCES FitnessClubs(FitnessClubID)
);

CREATE TABLE Waitlists (
    WaitListID INT PRIMARY KEY,
    QueueNumber INT,
    EnrollmentID INT,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('Waiting', 'Confirmed', 'Cancelled')),
    FOREIGN KEY (EnrollmentID) REFERENCES ClassEnrollments(EnrollmentID) ON DELETE CASCADE
);

CREATE TABLE Attendance (
    AttendanceID INT PRIMARY KEY,
    EnrollmentID INT,
    MemberID INT ,
    ClassID INT,
    AttendanceDate DATE,
    Status NVARCHAR(20) NOT NULL CHECK (Status IN ('Present', 'Absent', 'Excused')),
    FOREIGN KEY (EnrollmentID) REFERENCES ClassEnrollments(EnrollmentID) ON DELETE CASCADE,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID) ON DELETE CASCADE,
    FOREIGN KEY (ClassID) REFERENCES Classes(ClassID) ON DELETE CASCADE
);
```

```
CREATE TABLE Leaderboard (
    LeaderboardID INT IDENTITY(1,1) PRIMARY KEY,
    MemberID INT,
    ClassID INT,
    TotalTrainings INT,
    TotalHours DECIMAL(10,2),
    Rank INT NOT NULL,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID),
    FOREIGN KEY (ClassID) REFERENCES Classes(ClassID),
    UNIQUE (ClassID, Rank)

);
--Reviews - parent class for trainerReviews and ClassReviews

CREATE TABLE Reviews (
    ReviewID INT PRIMARY KEY,
    MemberID INT ,
    Rating INT NOT NULL CHECK (Rating BETWEEN 1 AND 5),
    ReviewDate DATE NOT NULL,
    Comment NVARCHAR(255) NULL,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID) ON DELETE CASCADE
);
--subclasses :

CREATE TABLE TrainerReviews (
    ReviewID INT PRIMARY KEY,
    TrainerID INT NOT NULL,
    FOREIGN KEY (ReviewID) REFERENCES Reviews(ReviewID) ON DELETE CASCADE,
    FOREIGN KEY (TrainerID) REFERENCES Trainers(TrainerID) ON DELETE CASCADE
);

CREATE TABLE ClassesReviews (
    ReviewID INT PRIMARY KEY,
    ClassID INT NOT NULL,
    DifficultyLevel INT NOT NULL CHECK (DifficultyLevel BETWEEN 1 AND 5),
    FOREIGN KEY (ReviewID) REFERENCES Reviews(ReviewID) ON DELETE CASCADE,
    FOREIGN KEY (ClassID) REFERENCES Classes(ClassID) ON DELETE CASCADE
```

```
);

--Additional tables

CREATE TABLE Merch (
    ItemID INT PRIMARY KEY,
    ItemName NVARCHAR(50) NOT NULL,
    ItemPrice DECIMAL(10,2) NOT NULL CHECK (ItemPrice >= 0)
);

CREATE TABLE MerchOrders (
    OrderID INT PRIMARY KEY,
    PaymentID INT ,
    MemberID INT,
    ItemID INT,
    Size NVARCHAR(10) NOT NULL CHECK (Size IN ('S', 'M', 'L', 'XL')),
    FOREIGN KEY (PaymentID) REFERENCES Payments(PaymentID),
    FOREIGN KEY (ItemID) REFERENCES Merch(ItemID) ON DELETE CASCADE,
    FOREIGN KEY (MemberID) REFERENCES Members(MemberID) ON DELETE CASCADE
);

--INSERT STATEMENTS ------------------------------

-------------------------------------------
-- 1. Club related INSERTS
-------------------------------------------
INSERT INTO FitnessClubs (FitnessClubID, Address) VALUES
  (1, 'ul. Główna 1, Warszawa'),
  (2, 'ul. Piękna 2, Kraków'),
  (3, 'ul. Sportowa 3, Poznań');

INSERT INTO Employees (EmployeeID, EmployeeName, Phone, Email, FitnessClubID, Position) VALUES
  (1,  'Adam Nowak',            '600-001-001', 'adam.nowak@example.com',            1, 'Manager'),
  (2,  'Ewa Kowalska',          '600-001-002', 'ewa.kowalska@example.com',          1, 'Trainer'),
  (3,  'Piotr Wiśniewski',      '600-001-003', 'piotr.wisniewski@example.com',      1, 'Receptionist'),
  (4,  'Anna Zielińska',        '600-001-004', 'anna.zielinska@example.com',        1, 'Cleaner'),
  (5,  'Krzysztof Kamiński',    '600-001-005', 'krzysztof.kaminski@example.com',    1, 'Trainer'),
```

```
(6,  'Magdalena Lewandowska',  '600-002-001', 'magdalena.lewandowska@example.com',  2, 'Manager'),
(7,  'Michał Wójcik',          '600-002-002', 'michal.wojcik@example.com',          2, 'Trainer'),
(8,  'Karolina Nowicka',       '600-002-003', 'karolina.nowicka@example.com',       2, 'Receptionist'),
(9,  'Tomasz Kamiński',        '600-002-004', 'tomasz.kaminski@example.com',        2, 'Cleaner'),
(10, 'Joanna Szymańska',       '600-002-005', 'joanna.szymanska@example.com',       2, 'Trainer'),
(11, 'Marcin Dąbrowski',       '600-003-001', 'marcin.dabrowski@example.com',       3, 'Manager'),
(12, 'Agnieszka Kwiatkowska',  '600-003-002', 'agnieszka.kwiatkowska@example.com',  3, 'Trainer'),
(13, 'Łukasz Nowakowski',      '600-003-003', 'lukasz.nowakowski@example.com',      3, 'Receptionist'),
(14, 'Monika Woźniak',         '600-003-004', 'monika.wozniak@example.com',         3, 'Cleaner'),
(15, 'Robert Jabłoński',       '600-003-005', 'robert.jablonski@example.com',       3, 'Trainer');

INSERT INTO Equipment (EquipmentID, Name, FitnessClubID, LastMaintenance, PurchaseDate, Status)
VALUES
(1, 'Treadmill',         1, '2023-12-01', '2023-01-01', 'Operational'),
(2, 'Elliptical',        2, '2023-12-05', '2023-02-01', 'Maintenance Required'),
(3, 'Stationary Bike',   3, '2023-12-10', '2023-03-01', 'Operational'),
(4, 'Rowing Machine',    1, '2023-12-15', '2023-04-01', 'Out of Service'),
(5, 'Dumbbells',         2, '2023-12-20', '2023-05-01', 'Operational');

---------------------------------------------
-- 2. Member related INSERTS
---------------------------------------------

INSERT INTO Memberships (MembershipID, MembershipName, PricePerMonth, DurationMonths) VALUES
(1, 'Basic',     29.99, 1),
(2, 'Standard',  49.99, 3),
(3, 'Premium',   69.99, 6),
(4, 'Corporate', 99.99, 12);

INSERT INTO Memberships (MembershipID, MembershipName, PricePerMonth, DurationMonths) VALUES
(5, 'Single-Entry', 10.99, 0);

INSERT INTO Members (MemberID, MembershipType, MembershipID, JoinDate) VALUES
(1,  'Individual', 1, '2023-01-10'),
(2,  'Individual', 2, '2023-01-15'),
(3,  'Individual', 3, '2023-01-20'),
(4,  'Individual', 1, '2023-01-25'),
```

```
(5,  'Individual', 2, '2023-02-01'),
(6,  'Individual', 3, '2023-02-05'),
(7,  'Individual', 1, '2023-02-10'),
(8,  'Individual', 2, '2023-02-15'),
(9,  'Individual', 3, '2023-02-20'),
(10, 'Individual', 1, '2023-02-25'),
(11, 'Individual', 2, '2023-03-01'),
(12, 'Individual', 3, '2023-03-05'),
(13, 'Individual', 1, '2023-03-10'),
(14, 'Individual', 2, '2023-03-15'),
(15, 'Individual', 3, '2023-03-20'),
(16, 'Individual', 1, '2023-03-25'),
(17, 'Individual', 2, '2023-03-30');

INSERT INTO Members (MemberID, MembershipType, MembershipID, JoinDate) VALUES
(18, 'Company', 4, '2023-01-05'),
(19, 'Company', 4, '2023-01-10'),
(20, 'Company', 4, '2023-01-15');

INSERT INTO IndividualMemberships (MemberID, MemberName, Email, Phone, MembershipID) VALUES
(1,  'Jan Kowalski',          'jan.kowalski@example.com',          '600-0001', 1),
(2,  'Anna Nowak',            'anna.nowak@example.com',            '600-0002', 2),
(3,  'Piotr Wiśniewski',      'piotr.wisniewski@example.com',      '600-0003', 3),
(4,  'Katarzyna Zielińska',   'katarzyna.zielinska@example.com',   '600-0004', 1),
(5,  'Marek Lewandowski',     'marek.lewandowski@example.com',     '600-0005', 2),
(6,  'Joanna Kamińska',       'joanna.kaminska@example.com',       '600-0006', 3),
(7,  'Andrzej Wójcik',        'andrzej.wojcik@example.com',        '600-0007', 1),
(8,  'Ewa Maj',               'ewa.maj@example.com',               '600-0008', 2),
(9,  'Tomasz Piotrowski',     'tomasz.piotrowski@example.com',     '600-0009', 3),
(10, 'Monika Szymańska',      'monika.szymanska@example.com',      '600-0010', 1),
(11, 'Michał Kamiński',       'michal.kaminski@example.com',       '600-0011', 2),
(12, 'Olga Nowicka',          'olga.nowicka@example.com',          '600-0012', 3),
(13, 'Robert Kwiatkowski',    'robert.kwiatkowski@example.com',    '600-0013', 1),
(14, 'Dorota Kowalska',       'dorota.kowalska@example.com',       '600-0014', 2),
(15, 'Paweł Lewandowski',     'pawel.lewandowski@example.com',     '600-0015', 3),
(16, 'Magdalena Nowak',       'magdalena.nowak@example.com',       '600-0016', 1),
(17, 'Grzegorz Ostrowski',    'grzegorz.ostrowski@example.com',    '600-0017', 2);
```

```
INSERT INTO CompanyMemberships (MemberID, CompanyName) VALUES
  (18, 'Alfa'),
  (19, 'Sigma'),
  (20, 'Gamma');

-------------------------------------------
-- 3. Trainer & Payment related INSERTS
-------------------------------------------
INSERT INTO Trainers (TrainerID, TrainerName, Specialization, Phone, Email) VALUES
  (1, 'Ewa Kowalska',          'Yoga',            '600-001-002', 'ewa.kowalska@example.com'),
  (2, 'Krzysztof Kamiński',    'Boxing',          '600-001-005', 'krzysztof.kaminski@example.com'),
  (3, 'Michał Wójcik',         'Pilates',         '600-002-002', 'michal.wojcik@example.com'),
  (4, 'Joanna Szymańska',      'CrossFit',        '600-002-005', 'joanna.szymanska@example.com'),
  (5, 'Agnieszka Kwiatkowska', 'Strength Training','600-003-002', 'agnieszka.kwiatkowska@example.com'),
  (6, 'Robert Jabłoński',      'HIIT',            '600-003-005', 'robert.jablonski@example.com');

INSERT INTO Invoices (InvoiceID, MemberID, IssueDate, DueDate, TotalAmount, Status)
VALUES
  (1, 1, '2023-10-01', '2023-10-15', 29.99, 'Paid'),
  (2, 2, '2023-10-02', '2023-10-16', 49.99, 'Unpaid'),
  (3, 3, '2023-10-03', '2023-10-17', 69.99, 'Pending');

INSERT INTO DiscountCodes (CodeID, DiscountCode, DiscountPercentage, Status)
VALUES
  (1, 'SAVE10', 10, 'Active'),
  (2, 'OFF20',  20, 'Active'),
  (3, 'DEAL15', 15, 'Inactive');

INSERT INTO Payments (PaymentID, MemberID, InvoiceID, AmountPaid, PaymentDate, DiscountCodeID)
VALUES
  (1, 1, 1, 29.99, '2023-10-05', 1),
  (2, 2, 2, 49.99, '2023-10-06', 2),
  (3, 3, 3, 69.99, '2023-10-07', 3);

INSERT INTO PersonalTrainings (TrainingID, TrainerID, MemberID, TrainingDate, DurationHours, PaymentID)
VALUES
```

```
  (1, 1, 1, '2023-10-20', 1.0, 1),
  (2, 2, 2, '2023-10-21', 1.5, 2),
  (3, 3, 3, '2023-10-22', 2.0, 3);

-------------------------------------------
-- 4. Class related INSERTS
-------------------------------------------
INSERT INTO Classes (ClassID, ClassName, ClassLevel) VALUES
  (1,  'Yoga Basics',         1),
  (2,  'Advanced Yoga',       2),
  (3,  'Boxing Fundamentals', 3),
  (4,  'Kickboxing',          3),
  (5,  'Pilates',             2),
  (6,  'Zumba',               1),
  (7,  'CrossFit',            4),
  (8,  'Spinning',            3),
  (9,  'HIIT',                4),
  (10, 'Strength Training',   5);

INSERT INTO ClassTrainers (ClassID, TrainerID) VALUES
  (1, 1),
  (2, 2),
  (3, 3),
  (4, 4),
  (5, 5),
  (6, 6),
  (7, 1),
  (8, 2),
  (9, 3),
  (10, 4);

INSERT INTO ClassEnrollments (EnrollmentID, MemberID, ClassID, Status) VALUES
  (1, 1, 1, 'Active'),
  (2, 2, 2, 'Active'),
  (3, 3, 3, 'Completed'),
  (4, 4, 4, 'Dropped'),
  (5, 5, 5, 'Active'),
```

```
(6, 6, 6, 'Completed'),
(7, 7, 7, 'Active'),
(8, 8, 8, 'Dropped');

INSERT INTO ClassSchedules (ScheduleID, ClassID, TrainerID, Room, Day, StartTime, EndTime, FitnessClubID)
VALUES
(1,  1, 1, 1, 'Monday',    '08:00', '09:00', 1),
(2,  2, 2, 1, 'Tuesday',   '09:15', '10:15', 1),
(3,  3, 3, 1, 'Wednesday', '10:30', '11:30', 1),
(4,  4, 4, 1, 'Thursday',  '11:45', '12:45', 1),
(5,  5, 5, 1, 'Friday',    '13:00', '14:00', 1),
(6,  6, 1, 1, 'Saturday',  '14:15', '15:15', 1),
(7,  7, 2, 1, 'Sunday',    '15:30', '16:30', 1),
(8,  8, 3, 1, 'Monday',    '16:45', '17:45', 1),
(9,  9, 4, 1, 'Tuesday',   '18:00', '19:00', 1),
(10, 10, 5, 1, 'Wednesday', '19:15', '20:15', 1);

INSERT INTO ClassSchedules (ScheduleID, ClassID, TrainerID, Room, Day, StartTime, EndTime, FitnessClubID)
VALUES
(11, 1, 1, 1, 'Monday',    '10:00', '11:00', 2),
(12, 2, 2, 1, 'Tuesday',   '11:15', '12:15', 2),
(13, 3, 3, 1, 'Wednesday', '12:30', '13:30', 2),
(14, 4, 4, 1, 'Thursday',  '13:45', '14:45', 2),
(15, 5, 5, 1, 'Friday',    '15:00', '16:00', 2),
(16, 6, 1, 1, 'Saturday',  '16:15', '17:15', 2),
(17, 7, 2, 1, 'Sunday',    '17:30', '18:30', 2),
(18, 8, 3, 1, 'Monday',    '18:45', '19:45', 2),
(19, 9, 4, 1, 'Tuesday',   '20:00', '21:00', 2),
(20, 10, 5, 1, 'Wednesday', '21:15', '22:15', 2);

INSERT INTO ClassSchedules (ScheduleID, ClassID, TrainerID, Room, Day, StartTime, EndTime, FitnessClubID)
VALUES
(21, 1, 1, 2, 'Monday',    '07:00', '08:00', 3),
(22, 2, 2, 2, 'Tuesday',   '08:15', '09:15', 3),
(23, 3, 3, 2, 'Wednesday', '09:30', '10:30', 3),
(24, 4, 4, 2, 'Thursday',  '10:45', '11:45', 3),
(25, 5, 5, 2, 'Friday',    '12:00', '13:00', 3),
```

```
(26, 6, 1, 2, 'Saturday',  '13:15', '14:15', 3),
(27, 7, 2, 3, 'Sunday',    '14:30', '15:30', 3),
(28, 8, 3, 3, 'Monday',    '15:45', '16:45', 3),
(29, 9, 4, 3, 'Tuesday',   '17:00', '18:00', 3),
(30, 10, 5, 3, 'Wednesday', '18:15', '19:15', 3);

INSERT INTO Waitlists (WaitListID, QueueNumber, EnrollmentID, Status) VALUES
(1, 1, 1, 'Waiting'),
(2, 2, 2, 'Confirmed'),
(3, 3, 3, 'Cancelled'),
(4, 4, 4, 'Waiting'),
(5, 5, 5, 'Confirmed');

INSERT INTO Attendance (AttendanceID, EnrollmentID, MemberID, ClassID, AttendanceDate, Status) VALUES
(1, 1, 1, 1, '2023-10-10', 'Present'),
(2, 2, 2, 2, '2023-10-11', 'Absent'),
(3, 3, 3, 3, '2023-10-12', 'Excused'),
(4, 4, 4, 4, '2023-10-13', 'Present'),
(5, 5, 5, 5, '2023-10-14', 'Present');

INSERT INTO Leaderboard (MemberID, ClassID, TotalTrainings, TotalHours, Rank) VALUES
(1, 1, 10, 10.00, 1),
(2, 1, 8,  8.50,  2),
(3, 2, 7,  7.00,  1),
(4, 2, 5,  5.50,  2),
(5, 3, 12, 12.00, 1);

-------------------------------------------
-- 5. Reviews related INSERTS
-------------------------------------------
INSERT INTO Reviews (ReviewID, MemberID, Rating, ReviewDate, Comment) VALUES
(1, 1, 5, '2023-09-25', 'Great trainer!'),
(2, 2, 4, '2023-09-26', 'Very professional'),
(3, 3, 5, '2023-09-27', 'Outstanding session'),
(4, 4, 3, '2023-09-28', 'Decent, but room for improvement'),
(5, 5, 4, '2023-09-29', 'Enjoyed the class'),
(6, 6, 2, '2023-09-30', 'Too difficult'),
```

```
  (7, 7, 5, '2023-10-01', 'Excellent class'),
  (8, 8, 3, '2023-10-02', 'Good, but expected more');

INSERT INTO TrainerReviews (ReviewID, TrainerID) VALUES
  (1, 1),
  (2, 2),
  (3, 3),
  (4, 4);

INSERT INTO ClassesReviews (ReviewID, ClassID, DifficultyLevel) VALUES
  (5, 1, 3),
  (6, 2, 2),
  (7, 3, 5),
  (8, 4, 4);




-------------------------------------------
-- 6. Additional INSERTS (Merch)
-------------------------------------------
INSERT INTO Merch (ItemID, ItemName, ItemPrice) VALUES
  (1, 'T-Shirt',      19.99),
  (2, 'Water Bottle',  9.99),
  (3, 'Gym Bag',      29.99),
  (4, 'Cap',          14.99),
  (5, 'Shorts',       24.99),
  (6, 'Socks',         4.99),
  (7, 'Jacket',       39.99),
  (8, 'Headband',      7.99);

INSERT INTO MerchOrders (OrderID, PaymentID, MemberID, ItemID, Size) VALUES
  (1, 1, 1, 1, 'M'),
  (2, 2, 2, 2, 'L'),
```

```
 (3, 3, 3, 3, 'S'),
 (4, 1, 4, 4, 'XL'),
 (5, 2, 5, 5, 'M');

--WIDOKI--

-- 1. View -- Average rating of each trainer

CREATE DATABASE FitnessClub
CREATE VIEW vwAverageTrainerRating AS
SELECT
   t.TrainerID,
   t.TrainerName,
   COALESCE(AVG(r.Rating * 1.0), 0) AS AverageRating ---coalesce returns average rating or when null then 0
FROM Trainers t
LEFT JOIN TrainerReviews tr ON t.TrainerID = tr.TrainerID
LEFT JOIN Reviews r ON tr.ReviewID = r.ReviewID
GROUP BY t.TrainerID, t.TrainerName

-- 2. View -- How many transactions used a given promotional code

CREATE VIEW vwPromoCodeTransactions AS
   SELECT
      dc.DiscountCode,
      COUNT(p.PaymentID) AS NumberOfTransactions
   FROM Payments p
   JOIN DiscountCodes dc ON p.DiscountCodeID = dc.CodeID
   GROUP BY dc.DiscountCode;

-- 3. View -- Member Attendance
CREATE VIEW vwMemberAttendance AS
SELECT
   a.AttendanceID,
   a.MemberID,
   m.MembershipType,
   a.ClassID,
   c.ClassName,
```

```
    a.AttendanceDate,
    a.Status
FROM Attendance a
JOIN Members m ON a.MemberID = m.MemberID
JOIN Classes c ON a.ClassID = c.ClassID;

-- 4. View -- Classes with most and least enrollments

CREATE VIEW vwEnrollmentExtremes AS

--CTE for coutning the number of enrollments for each class
WITH EnrollmentCounts AS (
    SELECT
        c.ClassID,
        c.ClassName,
        COUNT(e.EnrollmentID) AS EnrollmentCount
    FROM Classes c
    LEFT JOIN ClassEnrollments e
        ON c.ClassID = e.ClassID
    GROUP BY c.ClassID, c.ClassName
),
--CTE for maximum and minimumm enrollments
MinMax AS (
    SELECT
        MAX(EnrollmentCount) AS MaxEnrollment,
        MIN(EnrollmentCount) AS MinEnrollment
    FROM EnrollmentCounts
)
--select for viewing only the classes with min or max enrollments
SELECT
    ec.ClassID,
    ec.ClassName,
    ec.EnrollmentCount,
    CASE
        WHEN ec.EnrollmentCount = mm.MaxEnrollment THEN 'MAX'
        WHEN ec.EnrollmentCount = mm.MinEnrollment THEN 'MIN'
    END AS EnrollmentType
```

```
FROM EnrollmentCounts ec
CROSS JOIN MinMax mm
WHERE ec.EnrollmentCount = mm.MaxEnrollment
   OR ec.EnrollmentCount = mm.MinEnrollment;


-- 5. View -- Class average rating
CREATE VIEW vwAverageClassRating AS
SELECT
    c.ClassID,
    c.ClassName,
    c.ClassLevel,
    AVG(r.Rating) AS AverageRating
FROM Classes AS c
INNER JOIN ClassesReviews AS cr
    ON c.ClassID = cr.ClassID
INNER JOIN Reviews AS r
    ON cr.ReviewID = r.ReviewID
GROUP BY
    c.ClassID,
    c.ClassName,
    c.ClassLevel;

-- 6. View -- Top 3 members from each group
CREATE VIEW vwTop3InEachGroup AS
SELECT * FROM Leaderboard AS lb
WHERE lb.Rank <= 3


-- 7.View -- Count how many members have individual or company memberships
CREATE VIEW vwCountMembershipType AS
SELECT COUNT(*) as count, MembershipType FROM Members
GROUP BY MembershipType

-- 8.View -- See how many active members are enrolled in each scheduled class
CREATE VIEW vwActiveMembersEnrolled AS
select DISTINCT(classname), COUNT(*) AS ActiveMembersEnrolled from Classes C
```

```
JOIN ClassSchedules S ON S.ClassID = C.ClassID
JOIN ClassEnrollments E ON  S.ClassID = E.ClassID
WHERE status = 'Active'
GROUP BY scheduleid, classname

-- 9.View -- Total income each month
CREATE VIEW vwTotalIncomePerMonth AS
SELECT MONTH(P.PaymentDate) AS [Month],SUM(AmountPaid) AS TotalIncome FROM Payments P
GROUP BY MONTH(P.PaymentDate)

-- 10.View -- Number of maintanance needed in each location
CREATE VIEW vwLocationsSortedByNeededMaintanance AS
SELECT COUNT(*) as NeededMaintenance, e.FitnessClubID, fc.Address AS ClubID FROM Equipment e
JOIN FitnessClubs fc ON fc.FitnessClubID = e.FitnessClubID
WHERE e.Status IN ('Maintenance Required', 'Out of Service')
GROUP BY e.FitnessClubID, fc.Address

--TRIGGERY--


-- 1. Trigger -- Auto-Add Member to Leaderboard After 10 Trainings

CREATE TRIGGER trAddToLeaderboard
ON PersonalTrainings
AFTER INSERT
AS
BEGIN
   SET NOCOUNT ON;

   INSERT INTO Leaderboard (MemberID, TotalTrainings, TotalHours, Rank)
   SELECT i.MemberID, COUNT(p.TrainingID), COUNT(p.TrainingID) * 1.5, NULL
   FROM inserted i
   JOIN PersonalTrainings p ON i.MemberID = p.MemberID
   GROUP BY i.MemberID
   HAVING COUNT(p.TrainingID) >= 10
   AND NOT EXISTS (SELECT 1 FROM Leaderboard l WHERE l.MemberID = i.MemberID);
END;
```

```sql
-- 2. Trigger -- Update Invoice Status to 'Paid' After Full Payment

CREATE TRIGGER trUpdateInvoiceStatus
ON Payments
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Invoices
    SET Status = 'Paid'
    FROM Invoices i
    WHERE EXISTS (
        SELECT 1 FROM Payments p
        WHERE p.InvoiceID = i.InvoiceID
        GROUP BY p.InvoiceID
        HAVING SUM(p.AmountPaid) >= i.TotalAmount
    );
END;

-- 3. Trigger -- Apply Discount Automatically Based on Promo Code

CREATE TRIGGER trApplyDiscount
ON Payments
INSTEAD OF INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO Payments (InvoiceID, AmountPaid, DiscountCodeID, MemberID, PaymentDate)
    SELECT i.InvoiceID,
           i.AmountPaid * (1 - COALESCE(pc.DiscountPercentage, 0) / 100.0),
           i.DiscountCodeID,
           i.MemberID,
           i.PaymentDate
    FROM inserted i
```

```
    LEFT JOIN DiscountCodes pc ON i.DiscountCodeID = pc.CodeID AND pc.Status = 'Active';
END;

-- 4. Trigger -- Auto-Deactivate Expired Memberships (sets MembershipID in Members on NULL)

CREATE TRIGGER trDeactivateExpiredMembership
ON membershipactions
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE m
    SET m.MembershipID = NULL
    FROM Members m
    WHERE m.MemberID IN (
        SELECT i.MemberID
        FROM inserted i
        WHERE i.EndDate <= GETDATE()
          AND i.ActionType ='Cancelation'
    );
END;

-- 5. Trigger -- Auto-Remove from Leaderboard When Membership Expires (deletes member from leaderboard)
CREATE TRIGGER trRemoveFromLeaderboard
ON membershipactions
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE l
    FROM Leaderboard l
    WHERE l.MemberID IN (
        SELECT i.MemberID
        FROM inserted i
        WHERE i.EndDate <= GETDATE()
```

```
        AND i.ActionType = 'Cancelation'
    );
END;
--PROCEDURES--
--1)Procedure for updating maintenance date of gym equipment
CREATE PROCEDURE UpdateEquipmentMaintenanceDate
    @EquipmentID INT,
    @NewMaintenanceDate DATE
AS
BEGIN
    SET NOCOUNT ON

    UPDATE Equipment
    SET LastMaintenance = @NewMaintenanceDate
    WHERE EquipmentID = @EquipmentID
END
GO;


--2)Procedure for adding a review
CREATE PROCEDURE AddReview
    @ReviewType NVARCHAR(50),
    @ReviewID INT,
    @MemberID INT,
    @Rating INT,
    @ReviewDate DATE,
    @Comment Nvarchar(500),
    @TrainerID INT = NULL,
    @ClassID INT = NULL,
    @DifficultyLevel INT = NULL
AS
BEGIN
    SET NOCOUNT ON
    IF (@ReviewType = 'Trainer' ) OR (@ReviewType = 'Class')
        BEGIN
            INSERT INTO Reviews (ReviewID, MemberID, Rating, ReviewDate, Comment)
            VALUES (@ReviewId, @MemberID, @Rating, @ReviewDate, @Comment)
```

```
        IF(@ReviewType = 'Trainer')
            INSERT INTO TrainerReviews(ReviewID, TrainerID)
            VALUES (@ReviewID,@TrainerID)
        ELSE
            INSERT INTO ClassesReviews (ReviewID, ClassID, DifficultyLevel)
            VALUES (@ReviewID, @ClassID, @DifficultyLevel)
    END
  ELSE
    BEGIN
      Print 'Wrong ReviewType - choose from: [Trainer], [Class]'
    END

END
GO;

--3)Procedure for updating the leaderboard
CREATE PROCEDURE UpdateLeaderboard
  @AttendanceID INT,
  @DurationHours DECIMAL(10,2)
AS
BEGIN
  SET NOCOUNT ON;

  DECLARE @MemberID INT,
        @ClassID INT,
        @Status NVARCHAR(20);

  SELECT @MemberID = MemberID,
       @ClassID = ClassID,
       @Status = Status
  FROM Attendance
  WHERE AttendanceID = @AttendanceID;

  IF (@Status = 'Present')
  BEGIN
    IF EXISTS
    (
```

```
        SELECT 1  --checking if member e
        FROM Leaderboard
        WHERE MemberID = @MemberID AND ClassID = @ClassID
    )
    BEGIN
        UPDATE Leaderboard
        SET TotalTrainings = TotalTrainings + 1,
            TotalHours = TotalHours + @DurationHours
        WHERE MemberID = @MemberID AND ClassID = @ClassID;
    END
    ELSE
    BEGIN
        INSERT INTO Leaderboard (MemberID, ClassID, TotalTrainings, TotalHours, Rank)
        VALUES (@MemberID, @ClassID, 1, @DurationHours, 0);
    END;

    ;WITH Ranked AS
    (
        SELECT LeaderboardID, ROW_NUMBER() OVER (ORDER BY TotalHours DESC) AS NewRank
        FROM Leaderboard
        WHERE ClassID = @ClassID
    )
    UPDATE L
    SET L.Rank = R.NewRank
    FROM Leaderboard L
    JOIN Ranked R ON L.LeaderboardID = R.LeaderboardID;
  END
  ELSE
  BEGIN
    PRINT 'Attendance status is not Present.';
  END
END;
GO;

--4)Cancelation of the memebers that havent paid duedate
CREATE Procedure CancelOverDueMembers
AS
```

```
BEGIN
    SET NOCOUNT ON

    UPDATE M
    SET M.MembershipID = NULL
    FROM Members M
    WHERE EXISTS
    (
        SELECT 1 FROM Invoices I
        WHERE I.MemberID = M.MemberID
        AND I.DueDate < GETDATE() AND I.Status IN ('Unpaid','Pending')
    )
    --adding this action into MemberShipActions
    INSERT INTO MembershipActions(MemberID, ActionType, StartDate, EndDate, CancelationDate, Reason)
    SELECT DISTINCT M.MemberID, 'Cancelation' AS ActionType, NULL AS StartDate, NULL AS EndDate,
GETDATE() AS CancelationDate,
    'Membership canceled due to overdue payment' AS Reason
    FROM Members M
    WHERE M.MembershipID IS NULL
    AND EXISTS
    (
        SELECT 1 FROM INVOICES I
        WHERE I.MemberID = M.MemberID
        AND I.DueDate < GETDATE() AND I.Status IN ('Unpaid', 'Pending')
    )
END
GO;
--5) Updating the waitlist based on enrollments
CREATE PROCEDURE usp_UpdateWaitlistForClass
    @ClassID INT,
    @MaxCapacity INT = 10
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CurrentEnrollments INT,
            @AvailableSpots INT;
```

```
    --Counting available spots
    SELECT @CurrentEnrollments = COUNT(*)
    FROM ClassEnrollments
    WHERE ClassID = @ClassID
      AND Status = 'Active';

    SET @AvailableSpots = @MaxCapacity - @CurrentEnrollments;
    --if there are some then update waitlist
    IF @AvailableSpots > 0
    BEGIN
        ;WITH cteWaitlist AS (
            SELECT TOP (@AvailableSpots) w.WaitListID
            FROM Waitlists w
            INNER JOIN ClassEnrollments ce ON w.EnrollmentID = ce.EnrollmentID
            WHERE ce.ClassID = @ClassID
              AND w.Status = 'Waiting'
            ORDER BY w.QueueNumber ASC
        )--setting the status as confirmed
        UPDATE Waitlists
        SET Status = 'Confirmed'
        WHERE WaitListID IN (SELECT WaitListID FROM cteWaitlist);
    END
    --updating the queuenumber
    ;WITH UpdatedQueue AS (
        SELECT w.WaitListID,
               ROW_NUMBER() OVER (ORDER BY w.QueueNumber ASC) AS NewQueueNumber
        FROM Waitlists w
        INNER JOIN ClassEnrollments ce ON w.EnrollmentID = ce.EnrollmentID
        WHERE ce.ClassID = @ClassID
          AND w.Status = 'Waiting'
    )
    UPDATE w
    SET w.QueueNumber = uq.NewQueueNumber
    FROM Waitlists w
    INNER JOIN UpdatedQueue uq ON w.WaitListID = uq.WaitListID;
END;
```

```sql
--INDEKSY--
CREATE NONCLUSTERED INDEX IX_Invoices_Member_Status
ON Invoices (MemberID, Status);

CREATE NONCLUSTERED INDEX IX_Trainers_Specialization
ON Trainers (Specialization);

CREATE NONCLUSTERED INDEX IX_ClassSchedules_ClubDayTime
ON ClassSchedules (FitnessClubID, Day, StartTime);

CREATE NONCLUSTERED INDEX IX_Members_JoinDate
ON Members (JoinDate);

--Przykładowe Zapytania--

--SELECT Statment that displays which invoices are yet unpaid
SELECT InvoiceID, MemberID, IssueDate, DueDate, TotalAmount
FROM Invoices
WHERE Status = 'Unpaid';

--SELECT statement that shows which discount codes are still active
SELECT CodeID, DiscountCode, DiscountPercentage
FROM DiscountCodes
WHERE Status = 'Active';


--SELECT statement that shows reviews about Trainers
SELECT
    r.ReviewID,
    r.Rating,
    r.ReviewDate,
    r.Comment,
    t.TrainerName
FROM Reviews r
JOIN TrainerReviews tr ON r.ReviewID = tr.ReviewID
JOIN Trainers t ON tr.TrainerID = t.TrainerID;

--SELECT statement that displays how many members of each membershipType
```

```sql
SELECT
    ms.MembershipID,
    ms.MembershipName,
    COUNT(*) AS MembershipCount
FROM Members m
JOIN Memberships ms ON m.MembershipID = ms.MembershipID
GROUP BY ms.MembershipID, ms.MembershipName;


--SELECT statement that displays how many canceled memberships occured this month
SELECT COUNT(*) AS CancelledMembershipCount
FROM MembershipActions
WHERE ActionType = 'Cancelation'
  AND MONTH(CancelationDate) = MONTH(GETDATE())
  AND YEAR(CancelationDate) = YEAR(GETDATE());
```

Mateusz Jędrkowiak, Karolina Kulas & Aleksander Wiśniewski

Uniwersytet Jagiellonski, 2025