Podstawy Bazy Danych

Dokumentacja projektu:

System zarządzania konferencjami

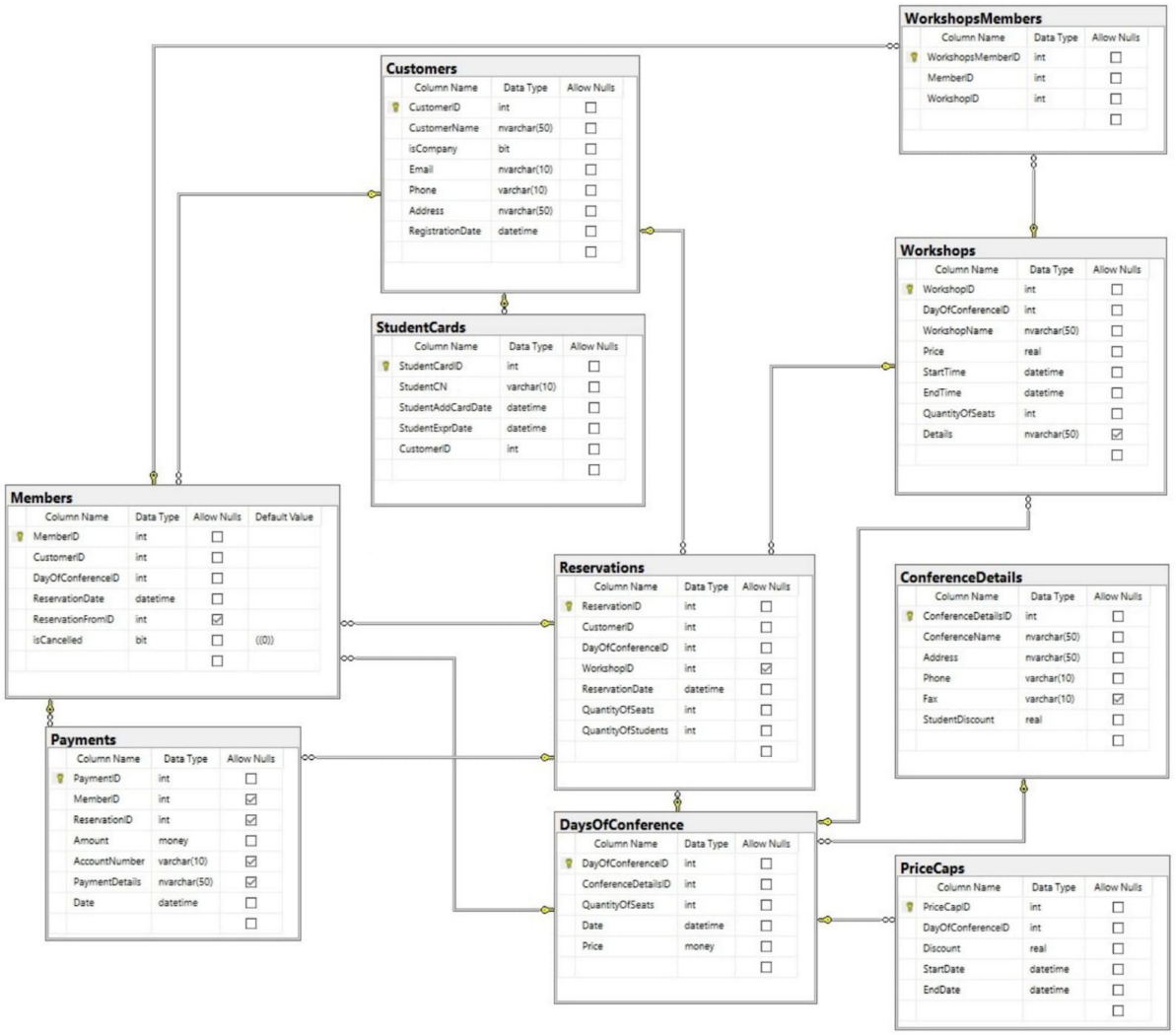
Jakub Buziewicz
Rafał Krajewski
2018

Spis treści:

1. Opis Systemu

Firma organizuje jedno- lub kilkudniowe konferencje. Klienci, zarówno osoby indywidualne oraz firmy, rejestrują się na dany dzień konferencji i na dane warsztaty. Firma nie musi podawać od razu danych osobowych uczestników, natomiast na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić. Jeżeli nie uzupełni powiadamiani są pracownicy tworzący konferencję. Dany uczestnik warsztatów nie może być zapisany na więcej niż 1 warsztat w tym samym czasie. System uwzględnia również ograniczoną ilość miejsc na dany dzień konferencji oraz na dany warsztat. Opłata za konferencję zależna jest od daty rezerwacji oraz uwzględniana jest zniżka studencka dla uczestników z ważną legitymacją. Koszt warsztatów jest stały lub zerowy (wejście darmowe). Na zapłatę klienci mają tydzień od rezerwacji na konferencję - jeśli do tego czasu nie pojawi się opłata, rezerwacja jest anulowana. Dla organizatora istnieje możliwość generowania raportów z każdego dnia konferencji oraz z każdego warsztatu z informacją o płatnościach oraz listą osobową uczestników.

2. Schemat


WorkshopsMembers
Column Name Data Type Allow Nulls
WorkshopsMemberID int
MemberID int
WorkshopID int
Customers
Column Name Data Type Allow Nulls
CustomerID int
CustomerName nvarchar(50)
isCompany bit
Email nvarchar(10)
Phone varchar(10)
Address nvarchar(50)
RegistrationDate datetime
Workshops
Column Name Data Type Allow Nulls
WorkshopID int
DayOfConferenceID int
WorkshopName nvarchar(50)
Price real
StartTime datetime
EndTime datetime
QuantityOfSeats int
Details nvarchar(50)
StudentCards
Column Name Data Type Allow Nulls
StudentCardID int
StudentCN varchar(10)
StudentAddCardDate datetime
StudentExprDate datetime
CustomerID int
Members
Column Name Data Type Allow Nulls Default Value
MemberID int
CustomerID int
DayOfConferenceID int
ReservationDate datetime
ReservationFromID int
isCancelled bit ((0))
Reservations
Column Name Data Type Allow Nulls
ReservationID int
CustomerID int
DayOfConferenceID int
WorkshopID int
ReservationDate datetime
QuantityOfSeats int
QuantityOfStudents int
ConferenceDetails
Column Name Data Type Allow Nulls
ConferenceDetailsID int
ConferenceName nvarchar(50)
Address nvarchar(50)
Phone varchar(10)
Fax varchar(10)
StudentDiscount real
Payments
Column Name Data Type Allow Nulls
PaymentID int
MemberID int
ReservationID int
Amount money
AccountNumber varchar(10)
PaymentDetails nvarchar(50)
Date datetime
DaysOfConference
Column Name Data Type Allow Nulls
DayOfConferenceID int
ConferenceDetailsID int
QuantityOfSeats int
Date datetime
Price money
PriceCaps
Column Name Data Type Allow Nulls
PriceCapID int
DayOfConferenceID int
Discount real
StartDate datetime
EndDate datetime

3. Opis Tabel

3.1 **Customers** - tabela reprezentująca dane klientów indywidualnych i firmowych (za rozróżnienie odpowiada pole “isCompany”). Przechowywane są tu takie dane jak “CustomerName” (imię i nazwisko klienta lub nazwa firmy), “Email” (adres email), “Phone” (numer telefonu), “Adress” (adres do korespondecji), “RegistrationDate” (data rejestracji). Dodatkowo mamy warunki integralnościowe, które gwarantują nam, że data rejestracji będzie przeszła oraz, że email oraz telefon nie będą danymi niepoprawnymi według standardów.

```
CREATE TABLE Customers (
   CustomerID int NOT NULL PRIMARY KEY,
   CustomerName nvarchar(50) NOT NULL,
   isCompany bit NOT NULL DEFAULT 0,
   Email nvarchar(50) NOT NULL,
   Phone varchar(10) NOT NULL,
   Address nvarchar(50) NOT NULL,
   RegistrationDate datetime NOT NULL,
   check (RegistrationDate<=CURRENT_TIMESTAMP),
   check (Email LIKE '%_@__%.__%'),
   check (ISNUMERIC(Phone)=1)
);
```

3.2 **StudentCards** - tabela do której dodawane są informację o legitymacjach studenckich uczestników. Mamy tu informację o numerze legitymacji(“StudentCN”) oraz dacie dodania i dacie ważności karty (“StudentAddCardDate”, “StudentExprDate”) oraz przypisanie karty do uczestnika (“CustomerID”) - jest to klucz zewnętrzny do tabeli “Customers”. Jedynym warunkiem niezbędny tutaj jest sprawdzanie, czy dodawana legitymacja w ogóle jest ważna w trakcie dodania.

```
CREATE TABLE StudentCards (
   StudentCardID int NOT NULL PRIMARY KEY,
   StudentCN varchar(10) NOT NULL,
   StudentAddCardDate datetime NOT NULL,
   StudentExprDate datetime NOT NULL,
   CustomerID int NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
   check (StudentAddCardDate<=StudentExprDate)
);
```

3.3 **ConferenceDetails** - tabela w której przechowywane są szczegóły konferencji. Nazwa (“ConferenceName”), adres (“Address”), numer telefonu (“Phone”) oraz ewentualnie numer fax (“Fax”). Dodatkowo mamy tutaj informację jaka jest wartość zniżki studenckiej na konferencję(“StudentDiscount”). Warunkami integralnościowymi jest tutaj poprawność numeru telefonu oraz czy zniżka studencka jest w zakresie 0-100%.

```
CREATE TABLE ConferenceDetails (
   ConferenceDetailsID int NOT NULL PRIMARY KEY,
   ConferenceName nvarchar(50) NOT NULL,
   Address nvarchar(50) NOT NULL,
   Phone varchar(10) NOT NULL,
   Fax varchar(10),
   StudentDiscount real NOT NULL,
   check (StudentDiscount BETWEEN 0 AND 1),
   check (ISNUMERIC(Phone)=1)
);
```

3.4 **DaysOfConference** - tabela reprezentująca dany dzień konferencji. Przechowywany jest tu przede wszystkim klucz zewnętrzny do tabeli “ConferenceDetails”, gdzie przechowujemy szczegółowe informację o konferencji której to jest dzień. Znajduje się tutaj również informacja o ilości miejsc na danym dniu (“QuantityOfSeats”), dacie konferencji (“Date”) oraz cenie (“Price”). Warunki integranlościowe gwarantują nam, że cena oraz ilość miejsc będą dodatnie.

```
CREATE TABLE DaysOfConference (
   DayOfConferenceID int NOT NULL PRIMARY KEY,
   ConferenceDetailsID int NOT NULL FOREIGN KEY REFERENCES ConferenceDetails(ConferenceDetailsID),
   QuantityOfSeats int NOT NULL,
   Date datetime NOT NULL,
   Price money NOT NULL,
   check (SIGN(QuantityOfSeats)>=0),
   check (SIGN(Price)>=0),
);
```

3.5 **Workshops** - tabela reprezentująca konkretny warsztat. Każdy warsztat jest dopisany do danego dnia konferencji za pomocą klucza zewnętrznego (“DayOfConferenceID”). Są tu również informację o nazwie warsztatu (“WorkshopName”), cenie (“Price”), dacie z godziną rozpoczęcia oraz zakończenia (“StartTime”, “EndTime”), ilości miejsc (“QuantityOfSeats”) oraz tekst dla opisu danego warsztatu (“Details”). Warunki integranlościowe gwarantują nam, że data rozpoczęcia nie będzie późniejsza niż data zakończenia oraz, że cena oraz ilość miejsc będą dodatnie.

```
CREATE TABLE Workshops (
   WorkshopID int NOT NULL PRIMARY KEY,
   DayOfConferenceID int NOT NULL FOREIGN KEY REFERENCES DaysOfConference(DayOfConferenceID),
   WorkshopName nvarchar(50) NOT NULL,
   Price real NOT NULL,
   StartTime datetime NOT NULL,
   EndTime datetime NOT NULL,
   QuantityOfSeats int NOT NULL,
   Details nvarchar(50),
   check (SIGN(Price)>=0),
   check (StartTime<=EndTime),
```

```
    check (SIGN(QuantityOfSeats)>=0),
);
```

3.6 **Reservations** - tabela reprezentująca rezerwację na konkretny warsztat lub tylko dzień konferencji. Każda rezerwacja posiada klucz do klienta, który ją stworzył (“CustomerID”) oraz klucz którego dnia konferencji to dotyczy (“DayOfConferenceID”). Dodatkowo, jeżeli rezerwacja dotyczy danego warsztatu, dopisywany jest klucz do warsztatu w polu “WorkshopID”. Sama rezerwacja posiada dane takie jak data rezerwacji (“ReservationDate”) potrzbną do obliczania zniżki oraz ilośc rezerwowaną ilość miejsc oraz ile z nich będzie studentami (“QuantityOfSeats”, “QuantityOfStudents”). Warunki integralnościowe gwarantują nam, że rezerwacje nie będą rezerwowane w przyszłość oraz, że ilości rezerwowanych miejsc i studentów będą dodatnie.

```
CREATE TABLE Reservations (
    ReservationID int NOT NULL PRIMARY KEY,
    CustomerID int NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
    DayOfConferenceID int NOT NULL FOREIGN KEY REFERENCES DaysOfConference(DayOfConferenceID),
    WorkshopID int FOREIGN KEY REFERENCES Workshops(WorkshopID),
    ReservationDate datetime NOT NULL,
    QuantityOfSeats int NOT NULL,
    QuantityOfStudents int NOT NULL,
    check (ReservationDate<=CURRENT_TIMESTAMP),
    check (SIGN(QuantityOfSeats)>=0),
    check (SIGN(QuantityOfStudents)>=0),
);
```

3.7 **Members** - tabela reprezentująca uczestników danego dnia konferencji. Każdy uczestnik posiada klucz do tabeli “Customers” gdzie są szczegółowe informację o kliencie (“CustomerID”) oraz klucz którego dnia konferencji to dotyczy (“DayOfConferenceID”). Dodatkowo, jeżeli uczestnik pochodzi z rezerwacji, to mamy również klucz do rezerwacji w polu “ReservationFromID”. Mamy również informację o dacie stworzenia uczestnika (“ReservationDate”) oraz informację czy uczestnik nie odwołał swojej obecności (“isCancelled”). Warunek integralnościowy gwarantuje nam, że data rezerwacji będzie przeszła.

```
CREATE TABLE Members (
    MemberID int NOT NULL PRIMARY KEY,
    CustomerID int NOT NULL FOREIGN KEY REFERENCES Customers(CustomerID),
    DayOfConferenceID int NOT NULL FOREIGN KEY REFERENCES DaysOfConference(DayOfConferenceID),
    ReservationDate datetime NOT NULL,
    ReservationFromID int FOREIGN KEY REFERENCES Reservations(ReservationFromID),
    isCancelled bit NOT NULL DEFAULT 0,
    check (ReservationDate<=CURRENT_TIMESTAMP),
);
```

3.8 **WorkshopsMembers** - tabela pośrednia reprezentująca uczestnictwo danego uczestnika na danym warsztacie. Jej celem jest stworzenie relacji wiele-do-wielu

między tabelami “Members” oraz “Workshop” za pomocą kluczy zewnętrznych (“MembersID”, “WorkshopID”).

```
CREATE TABLE WorkshopsMembers (
   WorkshopsMemberID int NOT NULL PRIMARY KEY,
   MemberID int NOT NULL FOREIGN KEY REFERENCES Members(MemberID),
   WorkshopID int NOT NULL FOREIGN KEY REFERENCES Workshops(WorkshopID),
);
```

3.9 **PriceCaps** - tabela przechowująca informację o zniżkach w zależności od daty rezerwacji. Klucz zewnętrzny “DayOfConferenceID” posiada informację jakiego dnia dotyczy dana zniżka, w polu “Discount” przechowywana jest wartość zniżki, a w polach “StartDate” oraz “EndDate” jest zakres dat których dotyczy dana zniżka. Warunki integralnościowe gwarantują nam, że zniżka będzie z zakresu 0-100% oraz, że data początku nie będzie późniejsza niż data końca.

```
CREATE TABLE PriceCaps (
   PriceCapID int NOT NULL PRIMARY KEY,
   DayOfConferenceID int NOT NULL FOREIGN KEY REFERENCES DaysOfConference(DayOfConferenceID),
   Discount real NOT NULL,
   StartDate datetime NOT NULL,
   EndDate datetime NOT NULL,
   check (Discount BETWEEN 0 AND 1),
   check (StartDate<=EndDate),
);
```

3.10 **Payments** - tabela przechowująca informację o płatnościach. W zależności czy płatność jest za rezerwacje czy za uczestnika dodawana jest relacja do pola “MemberID” lub “ReservationID”. Tabela posiada również informację o kwocie wpłaty (“Amount”), numerze rachunku (“AccountNumber”) oraz reszcie informacji z płatności (“PaymentDetails”). Dodatkowo przechowywana jest data zaksięgowania (“Date”). Warunki integralnościowe gwarantują nam, że kwota przelewu będzie dodatnia oraz, że płatność nie będzie nigdzie dopisana.

```
CREATE TABLE Payments (
   PaymentID int NOT NULL PRIMARY KEY,
   MemberID int FOREIGN KEY REFERENCES Members(MemberID),
   ReservationID int FOREIGN KEY REFERENCES Reservations(ReservationID),
   Amount money NOT NULL,
   AccountNumber varchar(10),
   PaymentDetails nvarchar(50),
   Date datetime NOT NULL,
   check (SIGN(Amount)>0),
   check ((MemberID IS NULL) AND (ReservationID IS NULL))
);
```

4. Spis procedur

```
CREATE PROCEDURE AddConference -- Procedura dodania konferencji
  @ConferenceName nvarchar(50), -- Nazwa konferencji
  @Address nvarchar(50), -- Adress konferencji
  @Phone varchar(10), -- Numer kontaktowy konferencji
  @Fax varchar(10) = null, -- Ewentualny fax dla konferencji
  @StudentDiscount real -- wartość zniżki od 0 do 1
  AS
  BEGIN
  SET nocount ON
  DECLARE @idConf INT

  INSERT INTO ConferenceDetails(
    ConferenceName,
    Address,
    Phone,
    Fax,
    StudentDiscount
  ) VALUES (
    @ConferenceName,
    @Address,
    @Phone,
    @Fax,
    @StudentDiscount
  )
  SET @idConf = @@IDENTITY
  RETURN @idConf
  END

CREATE PROCEDURE AddConferenceDay --Procedura dodania danego dnia konferencji
  @ConferenceName int, -- Nazwa konferencji której dotyczy dzień
  @QuantityOfSeats int, -- Ilośc miejsc na danym dniu konferencji
  @Date datetime, -- Dokładna data danego dnia konferencji
  @Price money -- Cena wstępu na dany dzień konferencji
  AS
  BEGIN
    SET nocount ON
    DECLARE @idConfDay INT

    DECLARE @ConferenceDetailsID int = (SELECT ConferenceDetailsID FROM ConferenceDetails WHERE ConferenceName=@ConferenceName)
    INSERT INTO DaysOfConference (
      ConferenceDetailsID,
      QuantityOfSeats,
      Date,
      Price
    ) VALUES (
      @ConferenceDetailsID,
      @QuantityOfSeats,
      @Date,
      @Price
    )
   SET @idConfDay = @@IDENTITY
   RETURN @idConfDay
  END


CREATE PROCEDURE AddCustomer -- Procedura dodania klienta
  @CustomerName nvarchar(50), -- Nazwa firmy lub imie i nazwisko klienta
  @isCompany bit, -- 1 jeżeli firma, 0 jeżeli klient indywidualny
  @Email nvarchar(50), -- email klienta
  @Phone varchar(10), -- telefon klienta
  @Address nvarchar(50), -- adres klienta
  @Registrationdate datetime -- data rejestracji klienta
  AS
  BEGIN
    SET nocount ON
    DECLARE @idCustomer INT

    INSERT INTO Customers (
    CustomerName,
    isCompany,
    Email,
    Phone,
    Address,
    Registrationdate
    ) VALUES (
      @CustomerName,
      @isCompany,
```

```
    @Email,
    @Phone,
    @Address,
    @Registrationdate
  )
SET @idCustomer = @@IDENTITY
RETURN @idCustomer
  END


CREATE PROCEDURE AddWorkshop -- Procedura dodawania warsztatu
  @DayOfConferenceID int, -- ID dnia konferencji podczas którego będzie dany warsztat
  @WorkshopName nvarchar(50), -- nazwa warsztatu
  @Price real, -- cena za uczestnictwo w warsztacie
  @StartTime datetime, -- dokładna data z godziną rozpoczęcia warsztatu
  @EndTime datetime, -- dokładna data z godziną zakończenia warsztatu
  @QuantityOfSeats int, -- ilość miejsc na warsztacie
  @Details nvarchar(50) = null
  AS
  BEGIN
    SET nocount on
    DECLARE @idWorkshop INT

    INSERT INTO Workshops
    (
      DayOfConferenceID,
      WorkshopName,
      Price,
      StartTime,
      EndTime,
      QuantityOfSeats,
      Details
    ) VALUES (
      @DayOfConferenceID,
      @WorkshopName,
      @Price,
      @StartTime,
      @EndTime,
      @QuantityOfSeats,
      @Details
    )
  SET @idWorkshop = @@IDENTITY
  RETURN @idWorkshop
  END

CREATE PROCEDURE AddMember -- Procedura dodania uczestnika
  @CustomerID int, -- ID klienta z pełnymi danymi uczestnika
  @DayOfConferenceID int, -- ID dnia konferencji na którą uczestnik się wybiera
  @ReservationFromID int = null -- ewentualne ID, jeżeli była rezerwacja wcześniej
  AS
  BEGIN
    SET nocount on
   DECLARE @idMember INT

    INSERT INTO Members (
      CustomerID,
      DayOfConferenceID,
      ReservationDate,
      ReservationFromID,
      isCancelled
    ) VALUES (
      @CustomerID,
      @DayOfConferenceID,
      CURRENT_TIMESTAMP,
      @ReservationFromID,
      0
    )
  SET @idMember = @@IDENTITY
 RETURN @idMember
  END

CREATE PROCEDURE AddPayment -- Procedura dodawania informacji o płatności
  @MemberID int, -- ID uczestnika którego płatność dotyczy
  @ReservationID int, -- ID rezerwacji której płatność dotyczy
  @Amount money, -- kwota przelana w danej płatności
  @AccountNumber varchar(10) = null, -- ewentualny numer z którego przyszła płatność
  @PaymentDetails nvarchar(50) = null -- ewentualne dodatkowe informacje o płatności
  AS
  BEGIN
    SET nocount ON
    DECLARE @idPayment INT
    INSERT INTO Payments
    (
      MemberID,
```

```
    ReservationID,
    Amount,
    AccountNumber,
    PaymentDetails,
    Date
  ) VALUES (
    @MemberID,
    @ReservationID,
    @Amount,
    @AccountNumber,
    @PaymentDetails,
    CURRENT_TIMESTAMP
  )
SET @idPayment = @@IDENTITY
RETURN @idPayment
END

CREATE PROCEDURE AddReservation -- Precedura dodawania rezerwacji
  @CustomerID int, -- ID klienta tworzącego rezerwacje
  @DayOfConferenceID int, -- ID dnia konferencji którego dotyczy rezerwacja
  @WorkshopID int=null, -- ewentualne ID jeżeli rezerwacja dotyczy danego warsztatu
  @QuantityOfSeats int, -- ilość rezerwowanych miejsc
  @QuantityOfStudents int -- ilość osób z legitymacją studencką
  AS
  BEGIN
    SET nocount ON
    DECLARE @idReservation INT
    INSERT INTO Reservations (
      CustomerID,
      DayOfConferenceID,
      WorkshopID,
      ReservationDate,
      QuantityOfSeats,
      QuantityOfStudents
    ) VALUES (
      @CustomerID,
      @DayOfConferenceID,
      @WorkshopID,
      CURRENT_TIMESTAMP,
      @QuantityOfSeats,
      @QuantityOfStudents
    )
  SET @idReservation = @@IDENTITY
  RETURN @idReservation
  END



CREATE PROCEDURE AddWorkshopMember -- Procedura dodania uczestnika do danego warsztatu
  @MemberID int, -- ID uczestnika
  @WorkshopID int -- ID warsztatu
  AS
  BEGIN
    SET nocount ON
    DECLARE @idWMember INT

    INSERT INTO WorkshopsMembers(
      MemberID,
      WorkshopID
    ) VALUES (
      @MemberID,
      @WorkshopID
    )
  SET @idWMember = @@IDENTITY
  RETURN @idWMember
  END


CREATE PROCEDURE AddPriceCap -- Procedura dodania zniżki w zależności od zarezerwowanej daty
  @DayOfConferenceID int, -- ID dnia konferencji której dotyczy zniżka
  @Discount real, -- wartość zniżki z zakresu 0 do 1
  @StartDate datetime, -- data rozpoczęcia okresu ze zniżką
  @EndDate datetime -- data zakończeniaokresu ze zniżką
  AS
  BEGIN
    SET nocount ON
    DECLARE @idPriceC INT

    INSERT INTO PriceCaps(
      DayOfConferenceID,
      Discount,
      StartDate,
      EndDate
```

```
  ) VALUES (
    @DayOfConferenceID,
    @Discount,
    @StartDate,
    @EndDate
  )
SET @idPriceC = @@IDENTITY
RETURN @idPrice
END


CREATE PROCEDURE CancelMember -- Procedura anulowania uczestnictwa danego uczestnika
  @MemberID int -- ID uczestnika
  AS
  IF EXISTS (SELECT * FROM Members WHERE MemberID = @MemberID)
  BEGIN
    SET nocount ON
    UPDATE Members
    Set isCancelled = 1
    WHERE MemberID = @MemberID
  END
  ELSE
     BEGIN
       RAISERROR ('Nie ma takiej osoby', -1, -1)
   END


CREATE PROCEDURE ChangeConferenceQuantityOfSeats  -- Procedura zmiany ilości miejsc na danym dniu konferencji
  @DayOfConferenceID int, -- ID dnia konferencji którego dotyczy operacja
  @NewLimit int -- nowa ilość miejsc
  AS
  BEGIN
    SET nocount ON
    DECLARE @CurrentlyOccupied AS int
    SET @CurrentlyOccupied = (SELECT SUM(quantityOfSeats)
                                        FROM DaysOfConference  c JOIN  Members m
                                       ON c.DayOfConferenceID = m.DayOfConferenceID
                                       WHERE DayOfConferenceID=@DayOfConferenceID AND m.isCancelled = 0)
    IF @CurrentlyOccupied<=@NewLimit
    BEGIN
      UPDATE DaysOfConference
      SET QuantityOfSeats = @NewLimit
      WHERE DayOfConferenceID=@DayOfConferenceID
    END
    ELSE
    BEGIN
      RAISERROR ('Nie można zmniejszyć limitu!', -1,-1)
    END
  END


CREATE PROCEDURE ChangeWorkshopQuantityOfSeats -- Procedura zmiany ilości miejsc na warsztacie
  @WorkshopID int, -- ID warsztatu którego dotyczy operacja
  @NewLimit int -- nowa ilość miejsc
  AS
  BEGIN
    SET nocount ON
    DECLARE @CurrentlyOccupied AS int
    SET @CurrentlyOccupied = (SELECT SUM(quantityOfSeats) FROM Workshops WHERE WorkshopID=@WorkshopID)
    IF @CurrentlyOccupied<=@NewLimit
    BEGIN
      UPDATE Workshops
      SET QuantityOfSeats = @NewLimit
      WHERE WorkshopID=@WorkshopID
    END
    else
    BEGIN
      RAISERROR ('Nie można zmniejszyć limitu!', -1,-1)
    END
  END


CREATE PROCEDURE ChangeQuantityOfSeatsInReservation -- Procedura zmiany ilość miejsc w rezerwacji
  @Reservationid int, -- ID rezerwacji której dotyczy zmiana
  @NewQuantityOfSeats int -- nowa ilość miejsc
  AS
  BEGIN
    SET nocount ON
    Update Reservations
    SET QuantityOfSeats = @NewQuantityOfSeats
    WHERE Reservationid = @Reservationid
  END
```

```
CREATE PROCEDURE HowManyFreeConferenceSeats -- Procedura zwracająca ilość wolnych miejsc na konferencji
  @DayOfConferenceID int -- ID dnia konferencji
  AS
  BEGIN
  SET nocount on
    DECLARE @Quantity AS int
    SET @Quantity = (SELECT QuantityOfSeats FROM DaysOfConference WHERE DayOfConferenceID=@DayOfConferenceID)

    DECLARE @Occupied AS int
    SET @Occupied = (SELECT SUM(QuantityOfSeats) FROM reservations WHERE DayOfConferenceID=@DayOfConferenceID)

    DECLARE @Result AS int
    SET @Result = (@Quantity-@Occupied)

    RETURN @Result
  END


CREATE PROCEDURE HowManyFreeWorkshopSeats -- Procedura zwracająca ilość wolnych miejsc na warsztacie
  @WorkshopID int  -- ID warsztatu
  AS
  BEGIN
    SET nocount on
    DECLARE @Quantity AS int
    SET @Quantity = (SELECT QuantityOfSeats FROM Workshops WHERE @WorkshopID=@WorkshopID)

    DECLARE @Occupied AS int
    SET @Occupied = (SELECT SUM(QuantityOfSeats) FROM reservations WHERE WorkshopID=@WorkshopID)

    DECLARE @Result AS int
    SET @Result = (@Quantity-@Occupied)

    RETURN @Result
  END




CREATE PROCEDURE ToPayForMember -- Procedura zwracająca wartość jaką należy jeszcze zapłacić za danego uczestnika
  @MemberID int -- ID uczestnika
  AS
  BEGIN
    SET nocount OFF
    DECLARE @FeeForDay AS int
    DECLARE @FeeForWorkshops AS int
    DECLARE @AlreadyPayed AS int
    SET @FeeForDay = (SELECT SUM(
              (CASE WHEN pc.Discount IS NULL THEN 1 ELSE (1-pc.Discount) END)*
              (CASE WHEN sc.StudentCN IS NULL THEN 1 ELSE (1-cd.StudentDiscount) END)*
              d.price)
            FROM Members AS m
            INNER JOIN Customers AS c
            ON c.CustomerID=m.CustomerID
            INNER JOIN DaysOfConference AS d
            ON m.DayOfConferenceID=d.DayOfConferenceID
            LEFT OUTER JOIN PriceCaps AS pc
            ON m.DayOfConferenceID=pc.DayOfConferenceID AND m.ReservationDate BETWEEN pc.StartDate AND pc.EndDate
            LEFT OUTER JOIN StudentCards AS sc
            ON c.CustomerID=sc.CustomerID AND d.Date BETWEEN sc.StudentAddCardDate AND sc.StudentExprDate
            INNER JOIN ConferenceDetails AS cd
            ON d.ConferenceDetailsID=cd.ConferenceDetailsID
            WHERE m.isCancelled=0
            GROUP BY m.MemberID)
    SET @FeeForWorkshops = (SELECT SUM(w.Price)
                FROM Members AS m
                INNER JOIN WorkshopsMembers AS wm
                ON m.MemberID=wm.MemberID
                INNER JOIN Workshops AS w
                ON w.WorkshopID=wm.WorkshopID
                GROUP BY m.MemberID)
    SET @AlreadyPayed = (SELECT SUM(Amount) FROM Payments WHERE MemberID=@MemberID GROUP BY MemberID)
    SET @AlreadyCanceled = (SELECT isCancelled FROM Members WHERE MemberID=@MemberID)
    IF @AlreadyCanceled > 0
    BEGIN
      RETURN 0
```

```
    END
    ELSE
    BEGIN
      RETURN @FeeForDay+@FeeForWorkshops-@AlreadyPayed
    END
  END



CREATE PROCEDURE ToPayForReservation -- Procedura zwracająca wartość jaką należy jeszcze zapłacić za konkretną rezerwacje
  @ReservationID int -- ID rezerwacji
  AS
  BEGIN
    SET nocount OFF
    DECLARE @ActualID AS int=-1
    DECLARE @MembersQuantity AS int=0
    DECLARE @StudentQuantity AS int=0
    DECLARE @MembersSum AS int=0
    DECLARE @WorkshopID AS int
    DECLARE @WorkshopFee AS int=0
    DECLARE @DayID AS int
    DECLARE @DayFee AS int
    DECLARE @StudentDiscount AS int
    DECLARE @DateDiscount AS int
    DECLARE @AlreadyPayed AS int
    WHILE (1 = 1)
    BEGIN  --komentarz dlaczego zrobilismy tu petle
      SELECT TOP 1 @ActualID=MemberID
      FROM Members
      WHERE @ActualID<MemberID AND ReservationFromID=@ReservationID
      ORDER BY MemberID
      IF @@ROWCOUNT = 0 BREAK

      SELECT c.CustomerID
      FROM Customers AS c
      INNER JOIN Members AS m
      ON m.CustomerID = c.CustomerID
      INNER JOIN DaysOfConference AS d
      ON m.DayOfConferenceID = d.DayOfConferenceID
      INNER JOIN StudentCards AS sc
      ON c.CustomerID=sc.CustomerID AND d.date BETWEEN sc.StudentAddCardDate AND sc.StudentExprDate
      IF @@ROWCOUNT > 0
      BEGIN
        SET @StudentQuantity=@StudentQuantity+1
      END
      SET @MembersQuantity=@MembersQuantity+1
      DECLARE @MemberFee int
      EXECUTE @MemberFee = ToPayForMember @ActualID
      SET @MembersSum=@MembersSum+@MemberFee
    END

    SET @AlreadyPayed = (SELECT SUM(Amount) FROM Payments WHERE MemberID IS NULL AND ReservationID=@ReservationID GROUP BY ReservationID)

    SELECT @WorkshopID=WorkshopID
    FROM Reservations
    WHERE WorkshopID IS NOT NULL AND ReservationID=@ReservationID
    IF @@ROWCOUNT > 0
    BEGIN
      SET @WorkshopFee=(SELECT Price FROM Workshops WHERE WorkshopID=@WorkshopID)
    END
    SET @DayID = (SELECT DayOfConferenceID FROM Reservations WHERE ReservationID=@ReservationID)
    SET @DateDiscount = (SELECT Discount FROM PriceCaps
                         WHERE DayOfConferenceID=@DayID AND CURRENT_TIMESTAMP BETWEEN StartDate AND EndDate)

    SELECT @DayFee=d.Price, @StudentDiscount=cd.StudentDiscount
    FROM DaysOfConference AS d
    INNER JOIN ConferenceDetails AS cd
    ON d.ConferenceDetailsID=cd.ConferenceDetailsID
    WHERE d.DayOfConferenceID=@DayID

    RETURN
@MembersSum+@DayFee*(1-@DateDiscount)*((1-@StudentDiscount)*@StudentQuantity+(@MembersQuantity-@StudentQuantity))+@WorkshopFee*@MembersQua
ntity
  END

CREATE PROCEDURE listOfAttendeeWorkshop -- procedura wyświetlająca listę uczestników danego warsztatu
  @WorkshopID int -- ID warsztatu
  AS
     BEGIN
       SELECT c.CustomerName FROM WorkshopsMembers w JOIN Members m
         ON w.MemberID = m.MemberID JOIN Customers c
               ON m.CustomerID = c.CustomerID WHERE w.WorkshopID = @WorkshopID AND m.isCancelled = 0
```

```
 end


CREATE PROCEDURE listOfAttendeeDayOfConference -- procedura wyświetlająca listę uczestników danego dnia konferencji
  @DayOfConferenceID int -- ID dnia konferencji
  AS
      BEGIN
       SELECT c.CustomerName FROM  Members m JOIN Customers c
       ON m.CustomerID = c.CustomerID WHERE m.DayOfConferenceID = @DayOfConferenceID AND m.isCancelled = 0

 end
```

5. Spis funkcji

```
CREATE FUNCTION conference_day_free_seats -- Sprawdza ile wolnych miejsc na danej konferencji
  (
  @DayOfConferenceID int -- ID dnia konferencji
  )
  RETURNS int
  AS
  BEGIN
    DECLARE @allSeats int = (SELECT QuantityOfSeats FROM DaysOfConference WHERE DayOfConferenceID = @DayOfConferenceID)
    DECLARE @occupiedSeats int = (SELECT sum(QuantityOfSeats) FROM Reservations WHERE DayOfConferenceID = @DayOfConferenceID)
    DECLARE @isCancelled int = (SELECT COUNT(DayOfConferenceID) FROM DaysOfConference
                                WHERE DayOfConferenceID = @DayOfConferenceID AND isCancelled = 1)
    DECLARE @freeSeats int = @allSeats - @occupiedSeats + @isCancelled
    IF @allSeats IS NULL
      SELECT @freeSeats = 0
   RETURN @freeSeats
  end

CREATE FUNCTION workshop_free_seats -- Sprawdza ile wolnych miejsc na danych warsztatach
  (
  @WorkshopID int -- ID warsztatu
  )
  RETURNS int
  AS
  BEGIN
    DECLARE @allSeats int = (SELECT QuantityOfSeats FROM Workshops WHERE WorkshopID = @WorkshopID)
    DECLARE @occupiedSeats int = (SELECT sum(QuantityOfSeats) FROM Reservations WHERE WorkshopID = @WorkshopID)
    DECLARE @isCancelled int = (SELECT COUNT(WorkshopID) FROM Workshops WHERE WorkshopID = @WorkshopID AND isCancelled = 1)
    DECLARE @freeSeats int = @allSeats - @occupiedSeats + @isCancelled

    IF @allSeats IS NULL
      SELECT @freeSeats = 0
   RETURN @freeSeats
  end

CREATE FUNCTION conference_day_occupied_seats -- Sprawdza ile zajetych miejsc na danej konferencji
  (
  @DayOfConferenceID int -- ID dnia konferencji
  )
  RETURNS int
  AS
  BEGIN
    DECLARE @occupiedSeats int = (SELECT ISNULL(sum(QuantityOfSeats),0) FROM Reservations WHERE DayOfConferenceID = @DayOfConferenceID)
    DECLARE @isCancelled int = (SELECT COUNT(DayOfConferenceID) FROM DaysOfConference
                                WHERE DayOfConferenceID = @DayOfConferenceID AND isCancelled = 1)
   RETURN @occupiedSeats - @isCancelled
  end


CREATE FUNCTION workshop_occupied_seats -- Sprawdza ile miejsc zajętych na danych warsztatach
  (
  @WorkshopID int -- ID dnia warsztatu
  )
  RETURNS int
  AS
  BEGIN
    DECLARE @occupiedSeats int = (SELECT ISNULL(sum(QuantityOfSeats),0) FROM Reservations WHERE WorkshopID = @WorkshopID)
    DECLARE @isCancelled int = (SELECT COUNT(WorkshopID) FROM Workshops WHERE WorkshopID = @WorkshopID AND isCancelled = 1)
   RETURN @occupiedSeats - @isCancelled
  end

CREATE FUNCTION day_price_on_date -- Sprawdza ile trzeba zapłacić za daną konferencje w danym dniu
  (
  @DayOfConferenceID int, -- ID dnia konferencji
  @Date date -- data dnia, którego sprawdzana jest cena
  )
  RETURNS int
  AS
  BEGIN
    DECLARE @discount real = ( SELECT TOP 1 Discount FROM PriceCaps
            WHERE DayOfConferenceID = @DayOfConferenceID AND @Date BETWEEN StartDate AND EndDate ORDER BY EndDate)
    DECLARE @price money = ( SELECT Price FROM DaysOfConference WHERE DayOfConferenceID = @DayOfConferenceID)
    RETURN @price * (1 - @discount)
  end

CREATE FUNCTION how_many_cancelled -- Sprawdza ile uczestników usuniętych z dnia konferencji
  (
  @DayOfConferenceID int
```

```
 )
 RETURNS int
 AS
  BEGIN
    DECLARE @isCancelled int = (SELECT count(*) FROM Members WHERE isCancelled = 1 AND DayOFConferenceID = @DayOfConferenceID)
    RETURN @isCancelled
  end



CREATE FUNCTION Is_paid -- Sprawdza czy opłacona rezerwacja
 (
   @ReservationID int -- ID rezerwacji
 )
 RETURNS BIT
 AS
 BEGIN
   DECLARE @payments_to_pay money

   SELECT @payments_to_pay = SUM(p.Amount) FROM Payments p WHERE p.ReservationID = @ReservationID

   IF (dbo.ToPayForReservation (@ReservationID) <= @payments_to_pay)
     RETURN 1

   RETURN 0
 END
```

6. Spis triggerów

```
CREATE TRIGGER reservationMaxCount -- trigger sprawdzający, czy nie przekroczono ilości miejsc w danym dniu konferencji
 ON Reservations
 AFTER INSERT, UPDATE
 AS
 IF exists(select *
        from inserted i
        where dbo.conference_day_occupied_seats (i.DayOfConferenceID) >
        (select QuantityOfSeats from DaysOfConference d where d.DayOfConferenceID = i.DayOfConferenceID))
    BEGIN
      raiserror('Za mało wolnych miejsc', -1, 1)
      rollback transaction
    END

CREATE TRIGGER workshopReservationMaxCount -- trigger sprawdzający, czy nie przekroczono ilości miejsc na warsztacie
 ON Reservations
 AFTER INSERT, UPDATE
 AS
 IF exists(select *
        from inserted i
        where dbo.workshop_occupied_seats (i.WorkshopID) >
        (select w.QuantityOfSeats from Workshops w where w.WorkshopID = i.WorkshopID))
    BEGIN
      raiserror('Za mało wolnych miejsc', -1, 1)
      rollback transaction
    END

CREATE TRIGGER worshopBadDay -- trigger sprawdzający czy warsztat dodawany jest z poprawnymi datami
 ON Workshops
 AFTER INSERT
 AS
 IF exists(select *
        from inserted i join DaysOfConference dc
        on i.DayOfconferenceID = dc.DayOfConferenceID
        where (CONVERT(VARCHAR(10), dc.Date, 104) = CONVERT(VARCHAR(10), i.StartTime, 104)
                AND CONVERT(VARCHAR(10), dc.Date, 104) = CONVERT(VARCHAR(10), i.EndTime, 104)
                AND i.DayOfConferenceID = dc.DayOfConferenceID)
    BEGIN
      raiserror ('Zła data warsztatów', -1, 1)
      rollback transaction
    end

CREATE TRIGGER PriceCapGoodDatesCheck -- trigger sprawdzający czy zniżki w zależności od daty się nie zazębiają
 ON PriceCaps
 AFTER INSERT, UPDATE
 AS
 DECLARE @startDate AS datetime
 SET @startDate = (SELECT StartDate FROM inserted)
 DECLARE @endDate AS datetime
 set @endDate = (SELECT EndDate FROM inserted)
 IF exists(SELECT *
    FROM PriceCaps AS pc
    WHERE (@startDate BETWEEN pc.StartDate AND pc.EndDate) OR (@endDate BETWEEN pc.StartDate AND pc.EndDate)
            )
    BEGIN
      raiserror ('Zła data zniżki - wykryto zazębienie', -1, 1)
      rollback transaction
    end

CREATE TRIGGER CancelMemberNotCanceled -- trigger sprawdzający czy nie próbujemy anulować anulowanego uczestnika
 ON Members
 AFTER UPDATE
 AS
 DECLARE @memberID AS int
 SET @memberID = (SELECT MemberID FROM inserted)
 DECLARE @wantCancel AS bit
 SET @wantCancel = (SELECT isCancelled FROM inserted)
 DECLARE @isCanceled AS bit
 SET @isCanceled = (SELECT isCancelled FROM Members WHERE MemberID=@memberID)
 IF (@isCanceled=1 AND @wantCancel=1)
    BEGIN
      raiserror ('Wykryto próbę anulowania uczestnika już wcześniej anulowanego', -1, 1)
      rollback transaction
    end
```

```
CREATE TRIGGER OneWorkshopAtTheSameTime -- trigger sprawdzający czy uczestnik nie rezerwuje drugiego warsztatu w tym samym czasie
  ON WorkshopsMembers
  AFTER INSERT, UPDATE
  AS
  DECLARE @startTime AS datetime
  SET @startTime = (SELECT w.StartTime FROM Workshops w JOIN inserted i On w.WorkshopID=i.WorkShopID )
  DECLARE @endTime AS datetime
  SET @endTime = (SELECT w.EndTime FROM Workshops w JOIN inserted i On w.WorkshopID=i.WorkshopID )
  IF exists(SELECT *
     FROM Workshops AS w
     WHERE (@startTime BETWEEN w.StartTime AND w.EndTime) OR (@endTime BETWEEN w.StartTime AND w.EndTime)
     )
     BEGIN
       raiserror ('Nie mozna zarezerwowac 2 warsztatow w tym samym czasie', -1, 1)
       rollback transaction
     end
```

7. Spis widoków

```
CREATE VIEW cancelled_members -- widok na wszystkich uczestników, którzy anulowali rezerwacje
  AS SELECT MemberID FROM Members WHERE isCancelled = 1


CREATE VIEW upcoming_workshops -- widok na wszystkie przyszłe warsztaty
  AS SELECT WorkShopID, WorkshopName FROM Workshops WHERE StartTime > GETDATE()


CREATE VIEW upcoming_and_ongoing_conferences -- widok na wszystkie trwające i przyszłe konferencje
  AS SELECT c.ConferenceDetailsID, c.ConferenceName FROM ConferenceDetails c JOIN DaysOfConference dc
                                                    ON c.ConferenceDetailsID = dc.ConferenceDetailsID AND dc.Date >= GETDATE()
                                                   GROUP BY c.ConferenceDetailsID, c.ConferenceName

CREATE VIEW most_popular_conferences -- widok wyświetlający listę 5 najpopularniejszych konferencji względem ilości rezerwacji
  AS SELECT TOP 5 c.ConferenceDetailsID, c.ConferenceName FROM ConferenceDetails c JOIN DaysOfConference dc
            ON c.ConferenceDetailsID = dc.ConferenceDetailsID JOIN Reservations r
            ON dc.DayOfConferenceId = r.DayOfConferenceID
    GROUP BY c.ConferenceDetailsID, c.ConferenceName
     ORDER BY count(r.ReservationID) DESC


CREATE VIEW most_popular_workshops -- widok wyświetlający listę 5 najpopularniejszych warsztatów względem ilości rezerwacji
  AS SELECT TOP 5 w.WorkShopID, w.WorkshopName FROM Workshops w JOIN Reservations r
            ON w.WorkshopID = r.WorkshopID
   GROUP BY w.WorkshopID, w.WorkshopName
   ORDER BY count(r.ReservationID) DESC


CREATE VIEW show_only_companies -- widok wyświetlający tylko klientów firmowych
  AS SELECT CustomerID, CustomerName, Email, Phone, Address FROM Customers WHERE isCompany = 1


CREATE VIEW show_only_private_clients -- widok wyświetlający tylko klientów infywidualnych
  AS SELECT CustomerID, CustomerName, Email, Phone, Address FROM Customers WHERE isCompany = 0


CREATE VIEW show_only_students -- widok wyświetlający tylko klientów z ważną legitymacją studencką
  AS SELECT c.CustomerID, CustomerName, Email, Phone, Address FROM Customers c JOIN StudentCards sc
                                          ON c.CustomerID = sc.CustomerID WHERE GETDATE() BETWEEN sc. StudentAddCardDate AND sc.StudentExprDate


CREATE VIEW show_conference_days_with_free_seats -- widok wyświetlający dni konferencji z wolnymi miejscami
  AS SELECT dc.DayOfConferenceID, dc.QuantityOfSeats - sum(r.QuantityOfSeats) AS wolne_miejsca FROM DaysOfConference dc JOIN Reservations r
            ON dc.DayOfConferenceID = r.DayOfConferenceID
    GROUP BY dc.DayOfConferenceID, dc.QuantityOfSeats
    HAVING sum(r.QuantityOfSeats) < dc.QuantityOfSeats
```

8. Spis indeksów

Ze względu na liczne korelację między tabelami, utworzone zostaną indeksy dla wszystkich kluczy zewnętrznych oraz indeksy sortujące według dat i alfabetycznie, ze względu na częstotliwość zapytań tego typu. Powinno zdecydowanie poprawić to wydajność. Dodatkowe indeksy powinny być wprowadzone podczas szczegółowej analizy zachowań użytkowników w trakcie działania bazy danych.

```
CREATE NONCLUSTERED INDEX Customers_Alph_Index ON Customers(CustomerName)
WITH (PAD_INDEX=ON, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX StudentCards_FK_withDates_Index ON StudentCards(CustomerID,StudentAddCardDate,StudentExprDate)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX ConferenceDetails_Alph_Index ON ConferenceDetails (ConferenceName)
WITH (PAD_INDEX=ON, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX DaysOfConference_FK_withDate_Index ON DaysOfConference(ConferenceDetailsID, Date)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX Workshops_Alph_Index ON Workshops(WorkshopName)
WITH (PAD_INDEX=ON, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX Workshops_FK_withDates_Index ON Workshops(DayOfConferenceID, StartTime, EndTime)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX Reservations_FK_withDate_Index ON Reservations(CustomerID, DayOfConferenceID, WorkshopID, ReservationDate)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX Members_FK_withDate_Index ON Members(CustomerID, DayOfConferenceID, ReservationFromID, ReservationDate)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX WorkshopsMembers_FK_Index ON WorkshopsMembers(MemberID, WorkshopID)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)


CREATE NONCLUSTERED INDEX PriceCaps_FK_wIthDates_Index ON PriceCaps(DayOfConferenceID, StartDate, EndDate)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)

CREATE NONCLUSTERED INDEX Payments_FK_wIthDate_Index ON Payments(MemberID, ReservationID, Date)
WITH (PAD_INDEX=OFF, SORT_IN_TEMPDB=ON)
```

9. Generator danych

Generator danych został napisany w PHP, w celu wygenerowania danych najlepiej skorzystać z curl ze strony “http://wyniki-online.pl/gensql.php” (gdzie aktualnie zmniejszono ilość danych ze względu na długi czas obliczania).

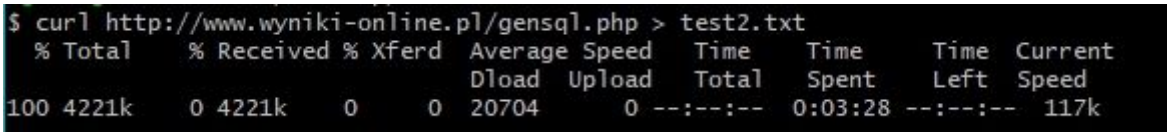
```
set_time_limit(0);
$Companies = array("Openlane", "Yearin", "Goodsilron", "Condax", ..., "Doncon"); //Tablice przykładowych danych zostały skasowane, aby nie przedłużać niepotrzebnie.
$Names = array("Claretta Ridinger", "Hertha Tabon", "Mandy Sheperd", "Ida Sedillo", ..., "Jeffry Elridge");
$Adresses = array ("83 Shore Street Roselle, IL 60172", "747 Pawnee Ave. Staunton, VA 24401", … , "620 Trusel Court Kernersville, NC 27284", "613 Mayflower St. Macomb, MI 48042", "567 Church St. Midlothian, VA 23112", "93 Trout Drive New York, NY 10002");
$Lorem="Lorem ipsum … laborum.";
function getRandArray(&$array){
  return $array[rand(0,count($array)-1)];
}
function getEmail($string){
  return str_replace(' ', '.', $string)."@ex.com";
}
function getPhone(){
  return rand(1,9).rand(0,9).rand(0,9).rand(0,9).rand(0,9).rand(0,9).rand(0,9).rand(0,9).rand(0,9).rand(0,9);
}
function checkSize(&$array, $max){
  for($i=0;$i<count($array);$i++)
  if(strlen($array[$i])>=$max)
    echo $array[$i]."\r\n";
}
function getRandDate($year, $month, $startdate=null){
  if($startdate==null)
    $startdate = $year."-".$month."-01";
  return date('Y-m-d', strtotime($startdate. ' + '.rand(0,cal_days_in_month(CAL_GREGORIAN,$month, $year)-4).' day')) ;
}
function getRandStr($length) {
   $chars = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
   $charsLength = strlen($chars);
   $ret = '';
   for ($i = 0; $i < $length; $i++) {
      $ret .= $chars[rand(0, $charsLength - 1)];
   }
   return $ret;
}
function getLorem($length=11){
  global $Lorem;
  return substr($Lorem,0,$length);
}
function percToDec($percent){
  return number_format((float)($percent/100), 2, '.', '');
}
function getFax(){
  if(rand(0,1)==1)
    return getPhone();
  return "NULL";
}
function getDatePrev($date){
  $checkdate=getRandDate(rand(2016,2018),rand(1,12));
  while(strtotime($checkdate)>=strtotime($date))
    $checkdate=getRandDate(rand(2016,2018),rand(1,12));
  return $checkdate;
}
//checksize($Adresses,50);
$qCustomers=3*12*2*300;//300
$CustomerIsCompany=array();
for($i=0;$i<$qCustomers;$i++){
  if($i%1000==0)
    echo "INSERT INTO Customers (CustomerID, CustomerName, isCompany, Email, Phone, Address, RegistrationDate) VALUES"."\r\n";
  $isCompany=rand(0,3);
  array_push($CustomerIsCompany,$isCompany);
  if($isCompany==0){
    $name=getRandArray($Names);
    echo "  (".$i.", '".$name."', ".$isCompany.", '".getEmail($name)."', '".getPhone()."', '".getRandArray($Adresses)."', '".getRandDate(2015,rand(1,12))."')";
  } else {
    $name=getRandArray($Companies);
    echo "  (".$i.", '".$name."', ".$isCompany.", '".getEmail($name)."', '".getPhone()."', '".getRandArray($Adresses)."', '".getRandDate(2015,rand(1,12))."')";
  }
  if($i+1<$qCustomers AND $i%1000!=999)
    echo ",";
  echo "\r\n";
}
```

```
echo "\r\n";
$qStudents=$qCustomers/2;
for($i=0;$i<$qStudents;$i++){
  if($i%1000==0)
    echo "INSERT INTO StudentCards (StudentCardID, StudentCN, StudentAddCardDate, StudentExprDate, CustomerID) VALUES"."\r\n";
  $date=getRandDate(rand(2016,2018),rand(1,12));
  echo "  (".$i.", '".getRandStr(8)."', '".$date."', '".date('Y-m-d', strtotime($date. ' + 1 year'))."', ".rand(0,$qCustomers-1).")";
  if($i+1<$qStudents AND $i%1000!=999)
    echo ",";
  echo "\r\n";
}

echo "\r\n";
$qConferences=2*12*3;
for($i=0;$i<$qConferences;$i++){
  if($i%1000==0)
    echo "INSERT INTO ConferenceDetails (ConferenceDetailsID, ConferenceName, Address, Phone, Fax, StudentDiscount) VALUES"."\r\n";
  echo "  (".$i.", '".getRandArray($Companies)." meet', '".getRandArray($Adresses)."', '".getPhone()."', '".getFax()."', ".percToDec(rand(0,100)).")";
  if($i+1<$qConferences AND $i%1000!=999)
    echo ",";
  echo "\r\n";
}

echo "\r\n";
$DaysOfConference=array();
$idDay=0;
$idConf=0;
$DaysSeats=array();
for($y=2016;$y<=2018;$y++){
  for($m=1;$m<=12;$m++){
    for($c=0;$c<2;$c++){
      if($idDay%1000==0)
        echo "INSERT INTO DaysOfConference (DayOfConferenceID, ConferenceDetailsID, QuantityOfSeats, Date, Price) VALUES"."\r\n";
      $days=rand(2,3);
      $date=getRandDate($y,$m);
      for($i=0;$i<$days;$i++){
        $quantitySeats=rand(100,1500);
        array_push($DaysOfConference,array($idConf,date('Y-m-d', strtotime($date. ' + '.$i.' day'))));
        array_push($DaysSeats,$quantitySeats);
        echo "  (".$idDay.", ".$idConf.", ".rand(100,300).", '".date('Y-m-d', strtotime($date. ' + '.$i.' day'))."', ".$quantitySeats.")";
        if(($idConf+1<3*12*2 OR $i+1<$days) AND $idDay%1000!=999)
          echo ",";
        echo "\r\n";
        $idDay++;
      }
      $idConf++;
    }
  }
}
$WorkshopsInDays=array();
echo "\r\n";
$idWork=0;
$WorkshopSeats=array();
for($idDay=0;$idDay<count($DaysOfConference);$idDay++){
  $qWorks=rand(2,6);
  $temp=array($DaysOfConference[$idDay][1]);
  for($i=0;$i<$qWorks;$i++){
    if($idWork%1000==0)
      echo "INSERT INTO Workshops (WorkshopID, DayOfConferenceID, WorkshopName, Price, StartTime, EndTime, QuantityOfSeats) VALUES"."\r\n";
    array_push($temp,$idWork);
    $quantitySeats=rand(10,60);
    array_push($WorkshopSeats,$quantitySeats);
    $startdate=date('Y-m-d h:i:s', strtotime($DaysOfConference[$idDay][1]. ' + '.rand(5,18).' hour'));
    echo "  (".$idWork.", ".$idDay.", '".getLorem(rand(5,40))."', ".rand(0,100).", '".$startdate."', '".date('Y-m-d h:i:s', strtotime($startdate. ' + '.rand(1,6).' hour'))."',
".$quantitySeats.")";
    if(($idDay+1<count($DaysOfConference) OR $i+1<$qWorks) AND $idWork%1000!=999)
      echo ",";
    echo "\r\n";
    $idWork++;
  }
  array_push($WorkshopsInDays, $temp);
}
echo "\r\n";
$qReservations=200;
$ReservationSeats=array();
$ReservationInDay=array();
for($i=0;$i<$qReservations;$i++){
  if($i%1000==0)
    echo "INSERT INTO Reservations (ReservationID, CustomerID, DayOfConferenceID, WorkshopID, ReservationDate, QuantityOfSeats, QuantityOfStudents)
VALUES"."\r\n";
  $dayOfConferenceID=rand(0,$idDay-1);
  $workshop='NULL';
  if(rand(0,1)==1)
```

```
    $workshop=$WorkshopsInDays[$dayOfConferenceID][rand(1,count($WorkshopsInDays[$dayOfConferenceID])-1)];

  $seats=rand(1,13);
  while($seats>$DaysSeats[$dayOfConferenceID] OR ($workshop!='NULL' AND $seats>$WorkshopSeats[$workshop])){
    $seats=rand(1,13);
  }
  array_push($ReservationInDay,array($dayOfConferenceID,$workshop));
  $DaysSeats[$dayOfConferenceID]-=$seats;
  if($workshop!='NULL')
    $WorkshopSeats[$workshop]-=$seats;
  array_push($ReservationSeats,$seats);
  echo "   (".$i.", ".rand(0,$qCustomers-1).", ".$dayOfConferenceID.", ".$workshop.", '".getDatePrev($WorkshopsInDays[$dayOfConferenceID][0])."', ".$seats.",
".rand(0,$seats).")";
  if($i+1<$qReservations AND $i%1000!=999)
    echo ",";
  echo "\r\n";
}
echo "\r\n";
$qMembers=3*12*2*200;//200
$MembersInDays=array();
for($i=0;$i<$qMembers;$i++){
  if($i%1000==0)
    echo "INSERT INTO Members (MemberID, CustomerID, DayOfConferenceID, ReservationDate, isCancelled) VALUES"."\r\n";

  $reservation='NULL';
  $dayOfConferenceID=rand(0,$idDay-1);
  if(rand(0,9)==0){ //z rezerwacji
    $reservation=rand(0,$qReservations-1);
    $dayOfConferenceID=$ReservationInDay[$reservation][0];
  } else { //bez rezerwacji
    while($DaysSeats[$dayOfConferenceID]==0)
      $dayOfConferenceID=rand(0,$idDay-1);
    $DaysSeats[$dayOfConferenceID]--;
  }
  $customer=rand(0,$qCustomers-1);
  while($CustomerIsCompany[$customer]==1)
    $customer=rand(0,$qCustomers-1);

  array_push($MembersInDays,$dayOfConferenceID);
  echo "   (".$i.", ".rand(0,$qCustomers-1).", ".$dayOfConferenceID.", '".getDatePrev($DaysOfConference[0][1])."', ".floor(rand(0,10)/10).")";
  if($i+1<$qMembers AND $i%1000!=999)
    echo ",";
  echo "\r\n";
}

echo "\r\n";
$qWorkshopMembers=3*12*2*200;//200
$WorkshopAttendens=array();
for($i=0;$i<$idWork;$i++)
  array_push($WorkshopAttendens,array());
for($i=0;$i<$qWorkshopMembers;$i++){
  if($i%1000==0)
    echo "INSERT INTO WorkshopsMembers (WorkshopsMemberID, MemberID, WorkshopID) VALUES"."\r\n";
  $memberID=rand(0,$qMembers-1);
  $dayOfConferenceID=$MembersInDays[$memberID];
  $workshopID=$WorkshopsInDays[$dayOfConferenceID][rand(1,count($WorkshopsInDays[$dayOfConferenceID])-1)];
  while($WorkshopSeats[$workshopID]==0 || in_array($memberID,$WorkshopAttendens[$workshopID])){
    $memberID=rand(0,$qMembers-1);
    $dayOfConferenceID=$MembersInDays[$memberID];
    $workshopID=$WorkshopsInDays[$dayOfConferenceID][rand(1,count($WorkshopsInDays[$dayOfConferenceID])-1)];
  }
  $WorkshopSeats[$workshopID]--;
  array_push($WorkshopAttendens[$workshopID],$memberID);
  echo "   (".$i.", ".$memberID.", ".$workshopID.")";
  if($i+1<$qWorkshopMembers AND $i%1000!=999)
    echo ",";
  echo "\r\n";
}
echo "\r\n";


$qPriceCaps=floor($idDay/3);
$DaysWithCap=array();
$idPriceCap=0;
for($i=0;$i<$qPriceCaps;$i++){

  $dayOfConferenceID=rand(0,$idDay-1);
  while(in_array($dayOfConferenceID, $DaysWithCap))
    $dayOfConferenceID=rand(0,$idDay-1);
  array_push($DaysWithCap,$dayOfConferenceID);
  $quantityCaps=rand(1,3);
  $enddate=$DaysOfConference[$dayOfConferenceID][1];
  for($j=0;$j<$quantityCaps;$j++){
    if($idPriceCap%1000==0)
```

```
      echo "INSERT INTO PriceCaps (PriceCapID, DayOfConferenceID, Discount, StartDate, EndDate) VALUES"."\r\n";
    $enddate=date('Y-m-d', strtotime($enddate. ' - '.rand(1,3).' day'));
    $startdate=date('Y-m-d', strtotime($enddate. ' - '.rand(1,5).' day'));

    echo "   (".$idPriceCap.", ".$dayOfConferenceID.", ".percToDec(rand(0,100)).", '".$startdate."', '".$enddate."')";

    $enddate=$startdate;
    if(($i73+1<$qPriceCaps OR $j+1<$quantityCaps) AND $idPriceCap%1000!=999)
      echo ",";
    echo "\r\n";
    $idPriceCap++;
  }
}
echo "\r\n";

$qPayments=floor($qMembers/10);
$PayedMembers=array();
for($i=0;$i<$qPayments;$i++){
  if($i%1000==0)
    echo "INSERT INTO Payments (PaymentID, MemberID, ReservationID, Amount, AccountNumber, PaymentDetails, Date) VALUES"."\r\n";

  $memberID=rand(0,$qMembers-1);
  while(in_array($memberID,$PayedMembers))
    $memberID=rand(0,$qMembers-1);
  array_push($PayedMembers,$memberID);
  $reservationID='NULL';

  echo "   (".$i.", ".$memberID.", ".$reservationID.", ".rand(0,100).", '".rand(0,9).rand(0,9).rand(0,9).rand(0,9)."', '".getLorem(rand(11,45))."',
'".getRandDate(rand(2016,2018),rand(1,12))."')";

  if($i+1<$qPayments AND $i%1000!=999)
    echo ",";
  echo "\r\n";
}
echo "\r\n";
```

Generator uwzględnia wszystkie założenia, m.in pilnując, aby ilość miejsc zajętych nie przekraczała ilości miejsc ogólnie, co mocno wydłuża jego działanie. Wygenerowanie 63 tysięcy rekordów zajęło ok 3 i pół minuty, co odpowiada 3 letniej działalności firmy:



10. Role

Obsługa konferencji będzie przez stronę WWW. Aplikacja będzie stosowana przez “Pracownika firmowego”, “Organizatora konferencji” oraz “Uczestnika konferencji”. Sama aplikacja będzie wyświetlała i umożliwia edycję danych w bazie danych. System WWW będzie korzystał tylko z wcześniej zadeklarowanych w bazie procedur, funkcji oraz widoków.

1. **Administrator** - Osoba ze znajomością języka SQL. Posiada pełny dostęp za pomocą programów zarządzających bazą danych w przypadku nieprzewidzianych sytuacji losowych.
2. **Pracownik firmowy** - Pierwsza osoba z którą kontaktuje się organizator konferencji w przypadku problemów. Za pomocą aplikacji WWW może przełączyć się na widoki innych ról i przeprowadzać ich akcję (oczywiście nie Administratora).
3. **Organizator konferencji** - Osoba która za pomocą naszej bazy danych chce stworzyć konferencję.
   - 3.1. Dostęp do procedur:
     - 3.1.1. AddConference
     - 3.1.2. AddConferenceDay
     - 3.1.3. AddCustomer
     - 3.1.4. AddWorkshop
     - 3.1.5. AddMember
     - 3.1.6. AddPayment
     - 3.1.7. AddReservation
     - 3.1.8. AddWorkshopMember
     - 3.1.9. AddPriceCap
     - 3.1.10. CancelMember
     - 3.1.11. ChangeConferenceQuantityOfSeats
     - 3.1.12. ChangeWorkshopQuantityOfSeats
     - 3.1.13. ChangeQuantityOfSeatsInReservation
     - 3.1.14. HowManyFreeConferenceSeats
     - 3.1.15. HowManyFreeWorkshopSeats
     - 3.1.16. ToPayForMember
     - 3.1.17. ToPayForReservation
     - 3.1.18. listOfAttendeeWorkshop
     - 3.1.19. listOfAttendeeDayOfConference
   - 3.2. Dostęp do funkcji:
     - 3.2.1. conference_day_free_seats
     - 3.2.2. workshop_free_seats
     - 3.2.3. conference_day_occupied_seats
     - 3.2.4. workshop_occupied_seats
     - 3.2.5. day_price_on_date
     - 3.2.6. how_many_cancelled
     - 3.2.7. Is_paid
   - 3.3. Dostęp do widoków:
     - 3.3.1. cancelled_members
     - 3.3.2. upcoming_workshops
     - 3.3.3. upcoming_and_ongoing_conferences
     - 3.3.4. most_popular_conferences
     - 3.3.5. most_popular_workshops
     - 3.3.6. show_only_companies
     - 3.3.7. show_only_private_clients
     - 3.3.8. show_only_students
     - 3.3.9. show_conference_days_with_free_seats

4. **Uczestnik konferencji** - Osoba rejestrująca się na stronie WWW na konkretne konferencje.
    - 4.1. Dostęp do procedur:
        - 4.1.1. AddMember
        - 4.1.2. AddReservation
        - 4.1.3. AddWorkshopMember
        - 4.1.4. CancelMember
        - 4.1.5. HowManyFreeConferenceSeats
        - 4.1.6. HowManyFreeWorkshopSeats
        - 4.1.7. ToPayForMember
        - 4.1.8. ToPayForReservation
    - 4.2. Dostęp do funkcji:
        - 4.2.1. conference_day_free_seats
        - 4.2.2. workshop_free_seats
        - 4.2.3. conference_day_occupied_seats
        - 4.2.4. workshop_occupied_seats
        - 4.2.5. day_price_on_date
        - 4.2.6. Is_paid
    - 4.3. Dostęp do widoków:
        - 4.3.1. upcoming_workshops
        - 4.3.2. upcoming_and_ongoing_conferences
        - 4.3.3. most_popular_conferences
        - 4.3.4. most_popular_workshops
        - 4.3.5. show_conference_days_with_free_seats