Podstawy Baz Danych

Projekt: System zarządzania konferencjami

Dokumentacja

Tomasz Szewczyk, Mateusz Szwed

Spis treści

- Procedura *AddSeminarReservation*
- Procedura *AddAttendant*
- Procedura *AddStudent*
- Procedura *ChangeSeatsConferenceDay*
- Procedura *ChangeSeatsSeminar*
- Procedura *CancelConference*
- Procedura *CancelConferenceDay*
- Procedura *CancelSeminar*
- Procedura *CancelReservation*
- Procedura *CancelSeminarReservation*

System zarządzania konferecjami

Opis problemu

Projekt wspiera działanie firmy organizującej konferencje.

Firma organizuje konferencje, które mogą być jedno lub kilkudniowe. Ponadto w konferencji zawierają się warsztaty na które należy rejestrować się oddzielnie. Klientami mogą być zarówno indywidualne osoby jak i firmy, natomiast uczestnikami konferencji są osoby. Firma nie musi podawać od razu przy rejestracji listy uczestników. Może zarezerwować odpowiednią ilość miejsc na określone dni oraz na warsztaty, natomiast na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić. Dla konferencji kilkudniowych, uczestnicy mogą rejestrować się na dowolne z tych dni, dowolną liczbę osób.

System obsługiwany jest przez serwis www, który powinien korzystać z procedur i funkcji udostępnianych przez bazę. Baza dostarcza danych do wyświetlenia na stronie, natomiast poprzez serwis użytkownik może obsługiwać bazę bez bezpośredniego dostępu do niej.

Użytkownicy

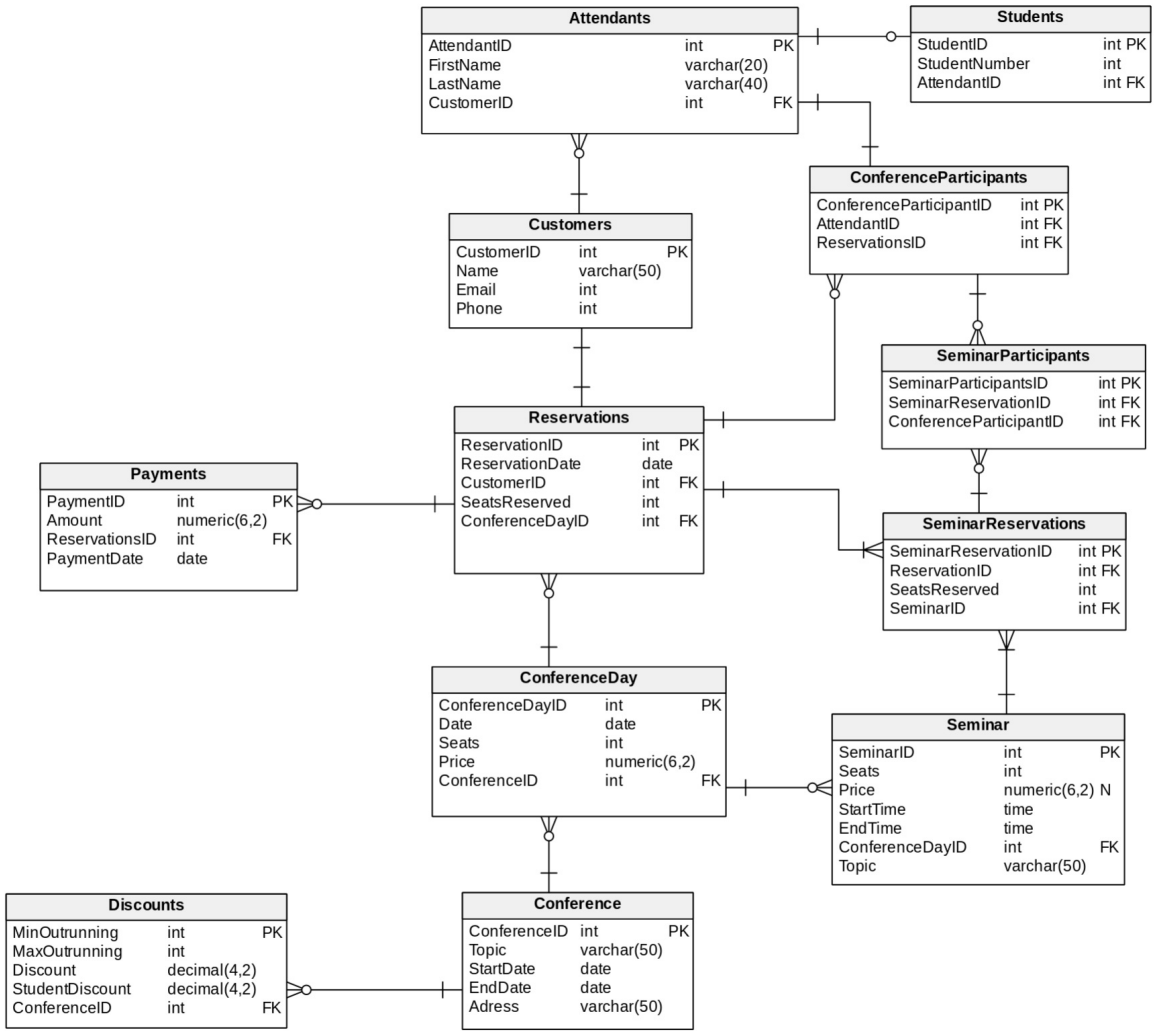
1. Klient - rejestracja oraz dodawanie rezerwacji.
2. Pracownik - księgowanie płatności, generowanie podsumowań.
3. Zarząd - generowanie podsumowań dotyczących najaktywniejszych klientów oraz najpopularniejszych produktów.

Wygenerowane dane

Na potrzeby testów została wynerowana przykładowa baza zawierająca:

- 500 klientów,
- 100 konferencji,
- 100 zniżek,
- 300 dni konferencji,
- 1000 warsztatów,
- 2000 rezerwacji,
- 2000 rezerwacji warsztatowych,
- 1000 zaksięgowanych płatności,
- 1000 uczestników.

Schemat bazy danych



Opis tabel

Tabela *Attendants*

Tabela opisująca uczestników wydarzenia. Każdy wiersz opisuje jedną fizyczną osobę znajdującą się na konferencji.

Kolumna	Typ	Opis
AttendantID	int NOT NULL	Unikalny identyfikator uczestnika
FirstName	varchar(20) NOT NULL	Imię uczestnika
LastName	varchar(40) NOT NULL	Nazwisko uczestnika
CustomerID	int NOT NULL	Unikalny identyfikator klienta, do którego przypisany jest uczestnik

```
CREATE TABLE Attendants
(
  AttendantID int         NOT NULL PRIMARY KEY IDENTITY (1,1),
  FirstName   varchar(20) NOT NULL,
  LastName    varchar(40) NOT NULL,
  CustomerID  int         NOT NULL,
);
```

Tabela *Conference*

Tabela opisująca konferecje.

Kolumna	Typ	Opis
ConferenceID	int NOT NULL	Unikalny identyfikator konferecji
Topic	varchar(50) NOT NULL	Temat lub nazwa konferencji
StartDate	date NOT NULL	Data rozpoczęcia konferecji
EndDate	date NOT NULL	Data zakończenia konferecji
Adress	varchar(50) NOT NULL	Miejsce wydarzenia
IsCanceled	int DEFAULT 0	Flaga implikująca anulowanie konferecji

```
CREATE TABLE Conference
(
  ConferenceID int         NOT NULL,
  Topic        varchar(50) NOT NULL,
  StartDate    date        NOT NULL,
  EndDate      date        NOT NULL,
  Adress       varchar(50) NOT NULL,
);
```

Tabela *ConferenceDay*

Tabela opisująca poszczególne dnie konferecji.

Kolumna	Typ	Opis
ConferenceDayID	int NOT NULL	Unikalny indentyfikator dnia konferencji
Date	date NOT NULL	Data wskazująca na dzień konferecji
Seats	int NOT NULL	Suma ogółu miejsc podczas dnia konferencji
Price	numeric(6,2) NOT NULL	Cena danego dnia konferecji
ConferenceID	int NOT NULL	Unikalny identyfikator konferecji, do której należy dzień konferencji

```
CREATE TABLE ConferenceDay
(
  ConferenceDayID int           NOT NULL PRIMARY KEY IDENTITY (1,1),
  Date            date          NOT NULL,
  Seats           int           NOT NULL,
  Price           numeric(6, 2) NOT NULL,
  ConferenceID    int           NOT NULL,
  IsCanceled      int DEFAULT 0
);
```

Tabela *ConferenceParticipants*

Tabela reprezentująca uczestników konferencji. Służy do skojarzenia *Attendants* z *Reservations*.

Kolumna	Typ	Opis
ConferenceParticipantID	int NOT NULL	Unikalny indentyfikator uczestnika konferencji
AttendantID	int NOT NULL	Unikalny indentyfikator uczestnika
ReservationsID	int NOT NULL	Unikalny indentyfikator rezerwacji

```
CREATE TABLE ConferenceParticipants
(
  ConferenceParticipantID int NOT NULL PRIMARY KEY IDENTITY (1,1),
  AttendantID             int,
  ReservationsID          int NOT NULL,
  IsCanceled              int DEFAULT 0
);
```

Tabela *Customers*

Tabela reprezentująca klientów dokonujących rezerwacje. Mogą to być firmy lub osoby prywatne.

Kolumna	Typ	Opis
CustomerID	int NOT NULL	Unikalny indentyfikator klienta
Name	varchar(50) NOT NULL	Nazwa klienta
Email	int NOT NULL	E-Mail klienta
Phone	int NOT NULL	Telefon do klienta

```
CREATE TABLE Customers
(
  CustomerID int         NOT NULL PRIMARY KEY IDENTITY (1,1),
  Name       varchar(50) NOT NULL,
  Email      varchar(50) NOT NULL,
  Phone      varchar(9)  NOT NULL,
);
```

Tabela *Discounts*

Tabela reprezentująca dostępne zniżki.

Kolumna	Typ	Opis
DiscountID	int NOT NULL	Unikalny indentyfikator zniżki
MinOutrunning	int NOT NULL	Minimalne wyprzedzenie
MaxOutrunning	int NOT NULL	Maksymalne wyprzedzenie
Discount	decimal(4,2) NOT NULL	Wielkość zniżki
StudentDiscount	decimal(4,2) NOT NULL	Wielkość zniżki dla studentów
ConferenceID	int NOT NULL	Unikalny indentyfikator konferencji, dla której obowiązują zniżki

```
CREATE TABLE Discounts
(
  DiscountID      int           NOT NULL PRIMARY KEY IDENTITY (1,1),
  MinOutrunning   int           NOT NULL,
  MaxOutrunning   int           NOT NULL,
  Discount        decimal(4, 2) NOT NULL,
  StudentDiscount decimal(4, 2) NOT NULL,
  ConferenceID    int           NOT NULL,
);
```

Tabela *Payments*

Tabela przechowująca zaksięgowane płatności.

Kolumna	Typ	Opis
PaymentID	int NOT NULL	Unikalny indentyfikator zaksięgowanej płatności
Amount	numeric(6,2) NOT NULL	Kwota zaksięgowanej płatności
ReservationsID	int NOT NULL	Unikalny indentyfikator rezerwacji, dla której dokonano płatności

```
CREATE TABLE Payments
(
  PaymentID      int           NOT NULL PRIMARY KEY IDENTITY (1,1),
  Amount         numeric(6, 2) NOT NULL,
  ReservationsID int           NOT NULL,
  PaymentDate    date          NOT NULL,
);
```

Tabela *Reservations*

Tabela przechowująca wykonanae rezerwacje.

Kolumna	Typ	Opis
ReservationID	int NOT NULL	Unikalny indentyfikator rezerwacji
ReservationDate	date NOT NULL	Data wykonania rezerwacji
PaymentDate	date NULL	Ostateczna data płatności
CustomerID	int NOT NULL	Unikalny indentyfikator klienta
SeatsReserved	int NOT NULL	Ilość zarezerwowanych miejsc
ConferenceDayID	int NOT NULL	Unikalny indentyfikator dnia, na który wykonano rezerwację

```
CREATE TABLE Reservations
(
  ReservationID   int  NOT NULL PRIMARY KEY IDENTITY (1,1),
  ReservationDate date NOT NULL,
  PaymentDate     date NULL,
  CustomerID      int  NOT NULL,
  SeatsReserved   int  NOT NULL,
  ConferenceDayID int  NOT NULL,
  IsCanceled      int DEFAULT 0
);
```

Tabela *Seminar*

Tabela przechowująca warsztaty

Kolumna	Typ	Opis
SeminarID	int NOT NULL	Unikalny indentyfikator warsztatu
Seats	int NOT NULL	Ilość miejsc dostępnych na warsztacie
Price	numeric(6,2) NULL	Cena warsztatu
StartTime	time NOT NULL	Godzina rozpoczęcia warsztatu
EndTime	time NOT NULL	Godzina końca warsztatu

ConferenceDayID	int NOT NULL	Unikalny indentyfikator dnia, w którym odbywa się warsztat

```
CREATE TABLE Seminar
(
  SeminarID       int           NOT NULL PRIMARY KEY IDENTITY (1,1),
  Seats           int           NOT NULL,
  Price           numeric(6, 2) NULL,
  StartTime       time          NOT NULL,
  EndTime         time          NOT NULL,
  ConferenceDayID int           NOT NULL,
  IsCanceled      int DEFAULT 0
);
```

Tabela *SeminarParticipants*

Tabela reprezentująca uczestników warsztatu. Służy do skojarzenia *SeminarAttendants* z *ConferenceParticipantID*.

Kolumna	Typ	Opis
SeminarParticipantsID	int NOT NULL	Unikalny indentyfikator uczestnika warsztatu
SeminarReservationID	int NOT NULL	Unikalny indentyfikator rezerwacji warsztatu
ConferenceParticipantID	int NOT NULL	Unikalny indentyfikator uczestnika konferencji

```
CREATE TABLE SeminarParticipants
(
  SeminarParticipantsID   int NOT NULL PRIMARY KEY IDENTITY (1,1),
  SeminarReservationID    int NOT NULL,
  ConferenceParticipantID int NOT NULL,
  IsCanceled              int DEFAULT 0
);
```

Tabela *SeminarReservations*

Tabela przechowująca wykonanae rezerwacje na warsztaty.

Kolumna	Typ	Opis
SeminarReservationID	int NOT NULL	Unikalny indentyfikator rezerwacji na warsztat
ReservationID	int NOT NULL	Unikalny indentyfikator rezerwacji na dzień konferencji
SeatsReserved	int NOT NULL	Ilość zarezerwowanych miejsc
SeminarID	int NOT NULL	Unikalny indentyfikator warsztatu

```
CREATE TABLE SeminarReservations
(
  SeminarReservationID int NOT NULL PRIMARY KEY IDENTITY (1,1),
  ReservationID        int NOT NULL,
  SeatsReserved        int NOT NULL,
  SeminarID            int NOT NULL,
  IsCanceled           int DEFAULT 0
);
```

Tabela *Students*

Tabela opisująca studentów.

Kolumna	Typ	Opis
StudentID	int NOT NULL	Unikalny indentyfikator studenta
StudentNumber	int NOT NULL	Numer legitymacji studenckiej
AttendantID	int NOT NULL	Unikalny indentyfikator uczestnika, który jest studentem

```
CREATE TABLE Students
(
  StudentID     int NOT NULL PRIMARY KEY IDENTITY (1,1),
  StudentNumber int NOT NULL,
  AttendantID   int NOT NULL,
);
```

Warunki integralności

Warunek *Attendants_ConferenceParticipants*

Warunek referencji łączący tabele *Attendants* oraz *ConferenceParticipants*

```
ALTER TABLE ConferenceParticipants
  ADD CONSTRAINT Attendants_ConferenceParticipants
  FOREIGN KEY (AttendantID) REFERENCES Attendants (AttendantID);
```

Warunek *Attendants_Customers*

Warunek referencji łączący tabele *Attendants* oraz *Customers*

```
ALTER TABLE Attendants
  ADD CONSTRAINT Attendants_Customers
  FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID);
```

Warunek *ConferenceDay_Conference*

Warunek referencji łączący tabele *ConferenceDay* oraz *Conference*

```
ALTER TABLE ConferenceDay
  ADD CONSTRAINT ConferenceDay_Conference
  FOREIGN KEY (ConferenceID) REFERENCES Conference (ConferenceID);
```

Warunek *ConferenceParticipants_Reservations*

Warunek referencji łączący tabele *ConferenceParticipants* oraz *Reservations*

```
ALTER TABLE ConferenceParticipants
  ADD CONSTRAINT ConferenceParticipants_Reservations
  FOREIGN KEY (ReservationsID) REFERENCES Reservations (ReservationID);
```

Warunek *Conference_Discounts*

Warunek referencji łączący tabele *Conference* oraz *Discounts*

```
ALTER TABLE Discounts
  ADD CONSTRAINT Conference_Discounts
  FOREIGN KEY (ConferenceID) REFERENCES Conference (ConferenceID);
```

Warunek *Payments_Reservations*

Warunek referencji łączący tabele *Payments* oraz *Reservations*

```
ALTER TABLE Payments
  ADD CONSTRAINT Payments_Reservations
  FOREIGN KEY (ReservationsID) REFERENCES Reservations (ReservationID);
```

Warunek *Reservations_ConferenceDay*

Warunek referencji łączący tabele *Reservations* oraz *ConferenceDay*

```
ALTER TABLE Reservations
  ADD CONSTRAINT Reservations_ConferenceDay
  FOREIGN KEY (ConferenceDayID) REFERENCES ConferenceDay (ConferenceDayID);
```

Warunek *Reservations_Customers*

Warunek referencji łączący tabele *Reservations* oraz *Customers*

```
ALTER TABLE Reservations
  ADD CONSTRAINT Reservations_Customers
  FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID);
```

Warunek *Reservations_SeminarReservations*

Warunek referencji łączący tabele *Reservations* oraz *SeminarReservations*

```
ALTER TABLE SeminarReservations
  ADD CONSTRAINT Reservations_SeminarReservations
  FOREIGN KEY (ReservationID) REFERENCES Reservations (ReservationID);
```

Warunek *SeminarParticipants_ConferenceParticipants*

Warunek referencji łączący tabele *SeminarParticipants* oraz *ConferenceParticipants*

```
ALTER TABLE SeminarParticipants
  ADD CONSTRAINT SeminarParticipants_ConferenceParticipants
  FOREIGN KEY (ConferenceParticipantID)
  REFERENCES ConferenceParticipants (ConferenceParticipantID);
```

Warunek *SeminarParticipants_SeminarReservations*

Warunek referencji łączący tabele *SeminarParticipants* oraz *SeminarReservations*

```
ALTER TABLE SeminarParticipants
  ADD CONSTRAINT SeminarParticipants_SeminarReservations
  FOREIGN KEY (SeminarReservationID)
  REFERENCES SeminarReservations (SeminarReservationID);
```

Warunek *SeminarReservations_Seminar*

Warunek referencji łączący tabele *SeminarReservations* oraz *Seminar*

```
ALTER TABLE SeminarReservations
  ADD CONSTRAINT SeminarReservations_Seminar
  FOREIGN KEY (SeminarID)
  REFERENCES Seminar (SeminarID);
```

Warunek *Seminar_ConferenceDay*

Warunek referencji łączący tabele *Seminar* oraz *ConferenceDay*

```
ALTER TABLE Seminar
  ADD CONSTRAINT Seminar_ConferenceDay
  FOREIGN KEY (ConferenceDayID)
  REFERENCES ConferenceDay (ConferenceDayID);
```

Warunek *Students_Attendants*

Warunek referencji łączący tabele *Students* oraz *Attendants*

```
ALTER TABLE Students
  ADD CONSTRAINT Students_Attendants
  FOREIGN KEY (AttendantID)
  REFERENCES Attendants (AttendantID);
```

Warunek *Conference_DateCheck*

Warunek walidujący daty startu i końca konferencji

```
ALTER TABLE Conference
  ADD CONSTRAINT Conference_DateCheck
  CHECK (Conference.StartDate <= Conference.EndDate);
```

Warunek *ConferenceDay_SeatsCheck*

Warunek walidujący sensowność ilości miejsc

```
ALTER TABLE ConferenceDay
  ADD CONSTRAINT ConferenceDay_SeatsCheck
  CHECK (ConferenceDay.Seats >= 0);
```

Warunek *Seminar_SeatsCheck*

Warunek walidujący sensowność ilości miejsc

```
ALTER TABLE Seminar
  ADD CONSTRAINT Seminar_SeatsCheck
  CHECK (Seminar.Seats >= 0);
```

Warunek *Discounts_OutrunningCheck*

Warunek walidujący wyprzedzenie w tabeli ze zniżkami

```
ALTER TABLE Discounts
  ADD CONSTRAINT Discounts_OutrunningCheck
  CHECK (Discounts.MinOutrunning <= Discounts.MaxOutrunning);
```

Warunek *Reservation_SeatsResercedCheck*

Warunek walidujący sensowność ilości zarezerwowanych miejsc

```
ALTER TABLE Reservations
  ADD CONSTRAINT Reservation_SeatsResercedCheck
  CHECK (Reservations.SeatsReserved >= 0);
```

Warunek *SeminarReservations_SeatsReservedCheck*

Warunek walidujący sensowność ilości zarezerwowanych miejsc

```
ALTER TABLE SeminarReservations
  ADD CONSTRAINT SeminarReservations_SeatsReservedCheck
  CHECK (SeminarReservations.SeatsReserved >= 0);
```

Widoki

Widok *AllConferencesView*

Pokaż wszystkie konferencje w systemie.

```
CREATE OR ALTER VIEW AllConferencesView
  AS (SELECT *from Conference)
```

Widok *UpcomingConferencesView*

Pokaż nadchodzące konferencje.

```
CREATE OR ALTER VIEW UpcomingConferencesView AS
  SELECT *
  FROM Conference
  WHERE DATEDIFF(month, GETDATE(), StartDate) BETWEEN 0 AND 3
```

Widok *NotPaitReservationView*

Pokaż niezapłacone rezerwacje.

```
CREATE OR ALTER VIEW NotPaitReservationView AS
  SELECT
    SUM(Amount) AS Paid,
    dbo.GetConferenceReservationCost(ReservationsID) AS Cost,
    ReservationsID AS ReservationID
  FROM Payments
  GROUP BY ReservationsID
  HAVING (SUM(Amount) < dbo.GetConferenceReservationCost(ReservationsID))
```

Widok *ReservationsWithoutAttendants*

Pokaż rezerwacje dla których nie wypełniono daych uczestnika.

```
CREATE OR ALTER VIEW ReservationsWithoutAttendants AS
SELECT Conference.Topic, Customers.*
FROM ConferenceParticipants
       LEFT JOIN Reservations
        ON Reservations.ReservationID = ConferenceParticipants.ReservationsID
       LEFT JOIN ConferenceDay
        ON ConferenceDay.ConferenceDayID = Reservations.ConferenceDayID
       LEFT JOIN Conference
        ON Conference.ConferenceID = ConferenceDay.ConferenceID
       LEFT JOIN Customers
        ON Customers.CustomerID = Reservations.CustomerID
WHERE ConferenceParticipants.AttendantID IS NULL
```

Widok *CustomersReservationCount*

Pokaż ilość rezerwacji dla każdego klienta.

```
CREATE OR ALTER VIEW CustomersReservationCount AS
SELECT Customers.CustomerID, COUNT(*) AS NumOfReservations
FROM Customers
       JOIN Reservations ON Reservations.CustomerID = Customers.CustomerID
GROUP BY Customers.CustomerID
```

Widok *BestCustomersView*

Pokaż najlepszych klientów.

```
CREATE OR ALTER VIEW BestCustomersView AS
SELECT Customers.*, customer_count.NumOfReservations
FROM Customers
       JOIN CustomersReservationCount customer_count
       ON customer_count.CustomerID = Customers.CustomerID
```

Widok *ConferenceDayView*

Pokaż dni konferencji.

```
CREATE OR ALTER VIEW ConferenceDayView AS
  SELECT *
  FROM ConferenceDay
```

Widok (funkcja) *DaysWithinConferenceView*

Pokaż dni w danej konferencji.

```
CREATE OR ALTER FUNCTION DaysWithinConferenceView(@ConferenceID INT) RETURNS TABLE
  AS RETURN SELECT *
            FROM ConferenceDay
            WHERE ConferenceDay.ConferenceID = @ConferenceID
```

Widok *DiscountsView*

Pokaż zniżki.

```
CREATE OR ALTER VIEW DiscountsView AS
  SELECT *
  FROM Discounts
```

Widok (funkcja) *DiscountsWithinConferenceView*

Pokaż zniżki dla danej konferencji.

```
CREATE OR ALTER FUNCTION DiscountsWithinConferenceView(@ConferenceID INT) RETURNS TABLE
  AS RETURN SELECT *
            FROM Discounts
            WHERE Discounts.ConferenceID = @ConferenceID
```

Widok *SeminarView*

Pokaż warsztaty

```
CREATE OR ALTER VIEW SeminarView AS
SELECT *
FROM Seminar
```

Widok (funkcja) *SeminarWithinConferenceDayView*

Pokaż warsztaty w danym dniu konferencji.

```
CREATE OR ALTER FUNCTION SeminarWithinConferenceDayView(@ConferenceDayID INT) RETURNS TABLE
  AS RETURN SELECT *
            FROM Seminar
            WHERE Seminar.ConferenceDayID = @ConferenceDayID
```

Widok (funkcja) *SeminarWithinConferenceView*

Pokaż warsztaty w danej konferencji.

```
CREATE OR ALTER FUNCTION SeminarWithinConferenceView(@Conference INT) RETURNS TABLE
  AS RETURN
  SELECT Seminar.*
  FROM Seminar
         LEFT JOIN ConferenceDay
          ON Seminar.ConferenceDayID = ConferenceDay.ConferenceDayID
  WHERE ConferenceDay.ConferenceID = @Conference
```

Widok *ReservationsView*

Pokaż rezerwacje.

```
CREATE OR ALTER VIEW ReservationsView AS
SELECT *,
       dbo.GetConferenceReservationCost(Reservations.ReservationID) AS Total,
       dbo.GetReservationPaid(Reservations.ReservationID)           AS Paid
FROM Reservations
```

Widok (funkcja) *ReservationWithinConferenceDayView*

Pokaż rezerwacje na dany dzień.

```
CREATE OR ALTER FUNCTION ReservationWithinConferenceDayView(@ConferenceDayID INT)
RETURNS TABLE
  AS RETURN SELECT *,
                   dbo.GetConferenceReservationCost(Reservations.ReservationID) AS Total,
                   dbo.GetReservationPaid(Reservations.ReservationID)           AS Paid
            FROM Reservations
            WHERE Reservations.ConferenceDayID = @ConferenceDayID
```

Widok *DuePaidReservationView*

Pokaż niezapłacone rezerwacje.

```
CREATE OR ALTER VIEW DuePaidReservationView
AS
  SELECT *,
         dbo.GetConferenceReservationCost(Reservations.ReservationID) AS Total,
         dbo.GetReservationPaid(Reservations.ReservationID)           AS Paid
  FROM Reservations
  WHERE dbo.GetConferenceReservationCost(Reservations.ReservationID) >
        dbo.GetReservationPaid(Reservations.ReservationID)
```

Widok *ReservationsView*

Pokaż rezerwacje.

```
CREATE OR ALTER VIEW ReservationsView AS
  SELECT *
  FROM SeminarReservations
```

Widok (funkcja) *ReservationWithinConferenceDayView*

Pokaż rezerwacje na dany dzień.

```
CREATE OR ALTER FUNCTION ReservationWithinConferenceDayView(@ConferenceDayReservationID INT)
RETURNS TABLE
  AS RETURN SELECT *
            FROM SeminarReservations
            WHERE SeminarReservations.ReservationID = @ConferenceDayReservationID
```

Widok (funkcja) *ConferenceDayListView*

Pokaż podsumowanie dnia.

```
CREATE OR ALTER function ConferenceDayListView(@DayID int)
  RETURNS TABLE
    AS RETURN
    SELECT DISTINCT FirstName, LastName
    from Attendants
           JOIN ConferenceParticipants ON ConferenceParticipants.AttendantID =
                                          Attendants.AttendantID
           JOIN Reservations ON ConferenceParticipants.ReservationsID =
                                Reservations.ReservationID
    WHERE @DayID = Reservations.ConferenceDayID
```

Widok (funkcja) *SeminarDayListView*

Pokaż podsumowanie warsztatu.

```
CREATE OR ALTER function SeminarDayListView(@SeminarID int)
  RETURNS TABLE
    AS RETURN
    SELECT DISTINCT FirstName, LastName
    from Attendants
           JOIN ConferenceParticipants ON ConferenceParticipants.AttendantID =
                                          Attendants.AttendantID
           JOIN SeminarParticipants
                ON ConferenceParticipants.ConferenceParticipantID =
                   SeminarParticipants.ConferenceParticipantID
           JOIN SeminarReservations SR ON SeminarParticipants.SeminarReservationID =
                                          SR.SeminarReservationID
    WHERE @SeminarID = SeminarID
```

Funkcje

Funkcja *GetConferenceReservationCost*

Pobierz całkowity koszt rezerwacji.

```
CREATE OR ALTER function GetConferenceReservationCost(@ReservationID INT)
  RETURNS NUMERIC(6, 2)
AS
BEGIN
  DECLARE @result AS NUMERIC(6, 2)
  SET @result = (SELECT SeatsReserved *
                        (SELECT Price
                         FROM ConferenceDay
                         WHERE ConferenceDay.ConferenceDayID = Reservations.ConferenceDayID)
                 FROM Reservations
                 WHERE ReservationID = @ReservationID);
  RETURN @result
END
```

Funkcja *GetReservationPaid*

Pobierz zapłaconą kwotę dla danej rezerwacji.

```
CREATE OR ALTER FUNCTION GetReservationPaid(@ReservationID INT) RETURNS NUMERIC(6, 2)
AS
BEGIN
  DECLARE @result AS NUMERIC(6, 2)
  SET @result = (SELECT SUM(Amount)
                 FROM Payments
                 WHERE ReservationsID = @ReservationID
                 GROUP BY ReservationsID)
  IF @result IS NOT NULL
    BEGIN
      RETURN @result
    END

  RETURN 0
END
```

Funkcja *GetConferenceStartDate*

Pobierz datę startu konferencji.

```
CREATE OR ALTER FUNCTION GetConferenceStartDate(@ConferenceID INT)
  RETURNS DATE
AS
BEGIN
  DECLARE @result as DATE
  SET @result = (SELECT StartDate
                 FROM Conference
                 WHERE Conference.ConferenceID = @ConferenceID)
  RETURN @result
END
```

Funkcja *GetConferenceEndDate*

Pobierz datę końca konferencji.

```
CREATE OR ALTER FUNCTION GetConferenceEndDate(@ConferenceID INT)
  RETURNS DATE
AS
BEGIN
  DECLARE @result as DATE
  SET @result = (SELECT EndDate FROM Conference
                 WHERE Conference.ConferenceID = @ConferenceID)
  RETURN @result
END
```

Funkcja *GetFreeSeatsByConferenceDayID*

Pobierz ilość wolnych miejsc w dniu.

```
CREATE OR ALTER FUNCTION GetFreeSeatsByConferenceDayID(@ConferenceDayID INT)
  RETURNS INT
AS
BEGIN

  DECLARE @all_seats_at_conference AS INT
  DECLARE @already_taken_seats AS INT
  DECLARE @result AS INT
  SET @all_seats_at_conference =
      (SELECT Seats FROM ConferenceDay
        WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID)
  SET @already_taken_seats =
      (SELECT SUM(SeatsReserved)
       FROM Reservations
       WHERE Reservations.ConferenceDayID = @ConferenceDayID
       GROUP BY Reservations.ConferenceDayID)
  SET @result = @all_seats_at_conference - @already_taken_seats
  RETURN @result
END
```

Funkcja *GetFreeSeatsBySeminarID*

Pobierz ilość wolnych miejsc w warsztacie.

```
CREATE OR ALTER FUNCTION GetFreeSeatsBySeminarID(@SeminarID INT)
  RETURNS INT
AS
BEGIN
  DECLARE @all_seats_at_seminar AS INT
  DECLARE @already_taken_seats AS INT
  DECLARE @result AS INT
  SET @all_seats_at_seminar = (SELECT Seats FROM Seminar
                               WHERE Seminar.SeminarID = @SeminarID)
  SET @already_taken_seats =
      (SELECT SUM(SeatsReserved)
       FROM SeminarReservations
       WHERE SeminarReservations.SeminarID = @SeminarID
       GROUP BY SeminarReservations.SeminarID)
  SET @result = @all_seats_at_seminar - @already_taken_seats
  RETURN @result
END
```

Funkcja *IsAStudent*

Sprawdź czy klient jest studentem.

```
CREATE OR ALTER FUNCTION IsAStudent(@CustomerID int)
  returns bit
AS
BEGIN
  IF ((SELECT COUNT(AttendantID)
       FROM Attendants
       GROUP BY CustomerID
       HAVING CustomerID = @CustomerID) > 1)
    BEGIN
      return 0
    end
  else
    DECLARE @AttendantID INT = (SELECT MAX(AttendantID)
                                FROM Attendants
                                Group BY CustomerID
                                HAVING CustomerID = @CustomerID)
  if EXISTS(SELECT * FROM Students WHERE AttendantID = @AttendantID)
    BEGIN
      RETURN 1
    END

  RETURN 0;
end
```

Triggery

Trigger *CancelConferenceDayOnConferenceCancelation*

Wyzwawacz zapewniający, że dzień konferencji należący do konferencji, która jest anulowana, również zostanie anulowany.

```
CREATE OR ALTER TRIGGER CancelConferenceDayOnConferenceCancelation
  ON Conference
  AFTER UPDATE
  AS
  UPDATE ConferenceDay
  SET ConferenceDay.IsCanceled = (ConferenceDay.IsCanceled | Conference.IsCanceled)
  FROM ConferenceDay
         LEFT JOIN Conference ON ConferenceDay.ConferenceID = Conference.ConferenceID
```

Trigger *CancelReservationOnConferenceDayCancelation*

Wyzwawacz zapewniający, że rezerwacja należąca do dnia konferencji, który jest anulowany, również zostanie anulowana.

```
CREATE OR ALTER TRIGGER CancelReservationOnConferenceDayCancelation
  ON ConferenceDay
  AFTER UPDATE
  AS
  UPDATE Reservations
  SET Reservations.IsCanceled = (Reservations.IsCanceled | ConferenceDay.IsCanceled)
  FROM Reservations
         LEFT JOIN ConferenceDay ON Reservations.ConferenceDayID =
                                    ConferenceDay.ConferenceDayID
```

Trigger *CancelSeminarOnConferenceDayCancelation*

Wyzwawacz zapewniający, że warsztat należący do dnia konferencji, który jest anulowany, różwnież zostanie anulowany.

```
CREATE OR ALTER TRIGGER CancelSeminarOnConferenceDayCancelation
  ON ConferenceDay
  AFTER UPDATE
  AS
  UPDATE Seminar
  SET Seminar.IsCanceled = (Seminar.IsCanceled | ConferenceDay.IsCanceled)
  FROM Seminar
         LEFT JOIN ConferenceDay ON Seminar.ConferenceDayID = ConferenceDay.ConferenceDayID
```

Trigger *CancelSeminarReservationOnSeminarCancelation*

Wyzwawacz zapewniający, że rezerwacja na warsztat, który został anulowany, również zostanie anulowana.

```
CREATE OR ALTER TRIGGER CancelSeminarReservationOnSeminarCancelation
  ON Seminar
  AFTER UPDATE
  AS
  UPDATE SeminarReservation
  SET SeminarReservation.IsCanceled = (SeminarReservation.IsCanceled | Seminar.IsCanceled)
  FROM SeminarReservation
         LEFT JOIN Seminar ON SeminarReservation.SeminarID = Seminar.SeminarID
```

Trigger *CancelSeminarParticipantOnSeminarReservationCancelation*

Wyzwawacz zapewniający, że uczestnicy warsztatu, który został anulowany, również zostaną anulowani.

```
CREATE OR ALTER TRIGGER CancelSeminarParticipantOnSeminarReservationCancelation
  ON SeminarReservations
  AFTER UPDATE
  AS
  UPDATE SeminarParticipant
  SET SeminarParticipant.IsCanceled =
    (SeminarParticipant.IsCanceled | SeminarReservations.IsCanceled)
  FROM SeminarParticipant
         LEFT JOIN SeminarReservations
                   ON SeminarParticipant.SeminarReservationID =
                      SeminarReservations.SeminarReservationID
```

Trigger *CancelParticipantOnReservationCancelation*

Wyzwawacz zapewniający, że uczestnicy dnia konferencji, który został anulowany, również zostaną anulowani.

```
CREATE OR ALTER TRIGGER CancelParticipantOnReservationCancelation
  ON Reservations
  AFTER UPDATE
  AS
  UPDATE Participant
  SET Participant.IsCanceled = (Participant.IsCanceled | Reservations.IsCanceled)
  FROM Participant
         LEFT JOIN Reservations ON Participant.ReservationID = Reservations.ReservationID
```

Procedury

Procedura *AddConferenceWithEndDate*

Procedura dodająca nową konferecję z datą końcową oraz tworząca dni konferencji na postawie daty.

```
CREATE OR ALTER PROCEDURE AddConferenceWithEndDate @Topic TEXT,
                                                   @StartDate DATE,
                                                   @EndDate DATE,
                                                   @Address TEXT,
                                                   @DefaultPrice NUMERIC(6, 2),
                                                   @DefaultSeats INT
AS
BEGIN
  IF (@StartDate > @EndDate)
    BEGIN
      RAISERROR ('Start date cannot be later than end date', 0, 0)

      RETURN
    END

  INSERT INTO Conference(Topic, StartDate, EndDate, Address)
  VALUES (@Topic, @StartDate, @EndDate, @Address);

  DECLARE @conference_day AS DATE
  DECLARE @last_conference_id AS INT
  SET @conference_day = @StartDate
  SET @last_conference_id = @@IDENTITY
  WHILE @conference_day <= @EndDate
  BEGIN
    EXEC AddConferenceDay
         @Date = @conference_day,
         @Seats = @DefaultSeats,
         @Price = @DefaultPrice,
         @ConferenceID = @last_conference_id;
    SET @conference_day = DATEADD(day, 1, @conference_day)
  END
END
```

Procedura *AddConference*

Procedura dodająca nową konferecję, przyjmując za datę końcową datę startu.

```
CREATE OR ALTER PROCEDURE AddConference @Topic TEXT,
                                       @StartDate DATE,
                                       @Address TEXT,
                                       @DefaultPrice NUMERIC(6, 2),
                                       @DefaultSeats INT
AS
EXEC AddConferenceWithEndDate
     @Topic = @Topic,
     @StartDate = @StartDate,
     @EndDate = @StartDate,
     @Address = @Address,
     @DefaultPrice = @DefaultPrice,
     @DefaultSeats = @DefaultSeats
```

Procedura *AddConferenceDay*

Procedura dodająca dzień konferencji.

```
CREATE OR ALTER PROCEDURE AddConferenceDay @Date DATE,
                                           @Seats INT,
                                           @Price NUMERIC(6, 2),
                                           @ConferenceID INT
AS
BEGIN
  IF @Seats < 0
    BEGIN
      RAISERROR ('Number of seats must be a positive number', 0, 0)

      RETURN
    END

  IF NOT @Date BETWEEN dbo.GetConferenceStartDate(@ConferenceID) AND
                       dbo.GetConferenceEndDate(@ConferenceID)
    BEGIN
      RAISERROR ('Conference day date must be within conference', 0, 0)

      RETURN
    END

  INSERT INTO ConferenceDay(Date, Seats, Price, ConferenceID)
  VALUES (@Date, @Seats, @Price, @ConferenceID)
END
```

Procedura *AddDiscount*

Procedura dodająca zniżkę.

```
CREATE OR ALTER PROCEDURE AddDiscount @MinOutrunning INT,
                                      @MaxOutrunning INT,
                                      @Discount NUMERIC(4, 2),
                                      @StudentDiscount NUMERIC(4, 2),
                                      @ConferenceID INT
AS
BEGIN
  IF @MinOutrunning > @MaxOutrunning
    BEGIN
      RAISERROR ('Min outrunning cannot be bigger than max outrunning', 0, 0)

      RETURN
    END

  IF NOT EXISTS(SELECT * FROM Conference WHERE Conference.ConferenceID = @ConferenceID)
    BEGIN
      RAISERROR ('Conference ID does not exist', 0, 0)

      RETURN
    END

  INSERT INTO Discounts(MinOutrunning,
                        MaxOutrunning,
                        Discount,
                        StudentDiscount,
                        ConferenceID)
  VALUES (@MinOutrunning,
          @MaxOutrunning,
          @Discount,
          @StudentDiscount,
          @ConferenceID)
END
```

Procedura *AddSeminar*

Procedura dodająca warsztat.

```
CREATE OR ALTER PROCEDURE AddSeminar @Seats INT,
                                     @Price NUMERIC(6, 2),
                                     @StartTime TIME,
                                     @EndTime TIME,
                                     @ConferenceDayID INT
AS
BEGIN
  IF @StartTime > @EndTime
    BEGIN
      RAISERROR ('Seminar start time cannot be later than end time', 0, 0)

      RETURN
    end

  IF NOT EXISTS(SELECT * FROM ConferenceDay
                WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID)
    BEGIN
      RAISERROR ('Conference day ID does not exist', 0, 0)

      RETURN
    END

  IF @Seats < 0
    BEGIN
      RAISERROR ('Number of seats must be a positive number', 0, 0)

      RETURN
    END

  INSERT INTO Seminar(Seats, Price, StartTime, EndTime, ConferenceDayID)
  VALUES (@Seats, @Price, @StartTime, @EndTime, @ConferenceDayID)
END
```

Procedura *AddReservation*

Procedura dodająca rezerwację oraz dodająca uczestników.

```
CREATE OR ALTER PROCEDURE AddReservation @CustomerID INT,
                                         @SeatsReserved INT,
                                         @ConferenceDayID INT
AS
BEGIN
  IF @SeatsReserved < 0
    BEGIN
      RAISERROR ('Reservation for zero seats does not make sense', 0, 0)

      RETURN
    END

  IF @SeatsReserved > dbo.GetFreeSeatsByConferenceDayID(@ConferenceDayID)
    BEGIN
      RAISERROR ('Not enough free seats to make a reservation', 0, 0)

      RETURN
    END

  IF NOT EXISTS(SELECT * FROM Customers WHERE Customers.CustomerID = @CustomerID)
    BEGIN
      RAISERROR ('Conference day ID does not exist', 0, 0)

      RETURN
    END

  IF NOT EXISTS(SELECT * FROM ConferenceDay
                WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID)
    BEGIN
      RAISERROR ('Conference day ID does not exist', 0, 0)

      RETURN
    END

  INSERT INTO Reservations(ReservationDate,
                           PaymentDate,
                           CustomerID,
                           SeatsReserved,
                           ConferenceDayID)
  VALUES (GETDATE(),
          null,
          @CustomerID,
          @SeatsReserved,
          @ConferenceDayID)

  DECLARE @last_added_reservation_id AS INT
  SET @last_added_reservation_id = @@identity
  DECLARE @index AS INT
  SET @index = 0
  WHILE @index < @SeatsReserved
  BEGIN
    EXEC AddConferenceParticipant @ReservationsID = @last_added_reservation_id,
                                  @AttendantID = null
    SET @index = @index + 1
  end
END
```

Procedura *AddCustomer*

Procedura dodająca klienta.

```
CREATE OR ALTER PROCEDURE AddCustomer @Name VARCHAR(50),
                                      @Email VARCHAR(50),
                                      @Phone VARCHAR(9)
AS
BEGIN
  IF @Name = ''
    BEGIN
      RAISERROR ('Name cannot be empty string', 0, 0)

      RETURN
    end

  IF @Email = ''
    BEGIN
      RAISERROR ('Email cannot be empty string', 0, 0)

      RETURN
    end

  IF @Phone = ''
    BEGIN
      RAISERROR ('Phone cannot be empty string', 0, 0)

      RETURN
    end

  INSERT INTO Customers(Name, Email, Phone) VALUES (@Name, @Email, @Phone)
end
```

Procedura *AddPayment*

Procedura dodająca nową wpłatę.

```
CREATE OR ALTER PROCEDURE AddPayment @Amount NUMERIC(6, 2), @ReservationsID INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM Reservations
                WHERE Reservations.ReservationID = @ReservationsID)
    BEGIN
      RAISERROR ('Reservation ID does not exist', 0, 0)

      RETURN
    END

  INSERT INTO Payments(Amount, ReservationsID, PaymentDate)
    VALUES (@Amount, @ReservationsID, GETDATE())
end
```

Procedura *AddConferenceParticipant*

Procedura dodająca uczestnika konferencji.

```
CREATE OR ALTER PROCEDURE AddConferenceParticipant @ReservationsID INT, @AttendantID INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM Reservations
                WHERE Reservations.ReservationID = @ReservationsID)
    BEGIN
      RAISERROR ('Reservation ID does not exist', 0, 0)

      RETURN
    END

  INSERT INTO ConferenceParticipants (AttendantID, ReservationsID)
    VALUES (@AttendantID, @ReservationsID)
end
```

Procedura *AddSeminarParticipant*

Procedura dodająca uczestnika warsztatu.

```
CREATE OR ALTER PROCEDURE AddSeminarParticipant @SeminarReservationID INT,
                                                @ConferenceParticipantID INT
AS
BEGIN

  IF NOT EXISTS(
      SELECT * FROM SeminarReservations
      WHERE SeminarReservations.SeminarReservationID = @SeminarReservationID)
    BEGIN
      RAISERROR ('Seminar reservation ID does not exist', 0, 0)

      RETURN
    END

  IF NOT EXISTS(SELECT *
                FROM ConferenceParticipants
                WHERE ConferenceParticipants.ConferenceParticipantID =
                  @ConferenceParticipantID)
    BEGIN
      RAISERROR ('Conference participant ID does not exist', 0, 0)

      RETURN
    END
end
```

Procedura *AddSeminarReservation*

Procedura dodająca rezerwację na warsztat oraz tworząca uczestników warsztatu.

```
CREATE OR ALTER PROCEDURE AddSeminarReservation @ReservationID INT,
                                                @SeatsReserved INT,
                                                @SeminarID INT
AS
BEGIN
  IF @SeatsReserved < 0
    BEGIN
      RAISERROR ('Reservation for zero seats does not make sense', 0, 0)

      RETURN
    END

  IF NOT EXISTS(SELECT * FROM Seminar WHERE Seminar.SeminarID = @SeminarID)
    BEGIN
      RAISERROR ('Seminar ID does not exist', 0, 0)

      RETURN
    END

  IF @SeatsReserved > dbo.GetFreeSeatsBySeminarID(@SeminarID)
    BEGIN
      RAISERROR ('Not enough free seats to make a reservation', 0, 0)

      RETURN
    END

  IF NOT EXISTS(SELECT * FROM Reservations
                WHERE Reservations.ReservationID = @ReservationID)
    BEGIN
      RAISERROR ('Conference reservation ID does not exist', 0, 0)

      RETURN
    END

  INSERT INTO SeminarReservations(ReservationID,
                                  SeatsReserved,
                                  SeminarID)
  VALUES (@ReservationID, @SeatsReserved, @SeminarID)

  DECLARE @last_added_reservation_id AS INT
  SET @last_added_reservation_id = @@identity
  DECLARE @index AS INT
  SET @index = 0
  WHILE @index < @SeatsReserved
  BEGIN
    EXEC AddSeminarParticipant @SeminarReservationID = @last_added_reservation_id,
                               @ConferenceParticipantID = null
    SET @index = @index + 1
  end
END
```

Procedura *AddAttendant*

Procedura dodająca uczestnika wydarzenia.

```
CREATE OR ALTER PROCEDURE AddAttendant @FirstName VARCHAR(20),
                                       @LastName VARCHAR(40),
                                       @CustomerID INT
AS
BEGIN
  IF @FirstName = ''
    BEGIN
      RAISERROR ('First name cannot be empty string', 0, 0)

      RETURN
    end

  IF @LastName = ''
    BEGIN
      RAISERROR ('Last name cannot be empty string', 0, 0)

      RETURN
    end

  IF NOT EXISTS(SELECT * FROM Customers WHERE Customers.CustomerID = @CustomerID)
    BEGIN
      RAISERROR ('Customer ID does not exist', 0, 0)

      RETURN
    END

  INSERT INTO Attendants(FirstName, LastName, CustomerID)
    VALUES (@FirstName, @LastName, @CustomerID)
end
```

Procedura *AddStudent*

Procedura dodająca informacje o legitymacji studenckiej.

```
CREATE OR ALTER PROCEDURE AddStudent @AttendantID INT, @StudendNumber INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM Attendats WHERE Attendants.AttendantID = @AttendantID)
    BEGIN
      RAISERROR ('Attendant ID does not exist', 0, 0)

      RETURN
    END

  INSERT INTO Students(AttendantID, StudentNumber) VALUES (@AttendantID, @StudendNumber)
END
```

Procedura *ChangeSeatsConferenceDay*

Procedura zmieniająca ilość dostepnych miejsc w dniu konferencji.

```
CREATE OR ALTER PROCEDURE ChangeSeatsConferenceDay @ConferenceDayID INT, @NewSeats INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM ConferenceDay
                WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID)
    BEGIN
      RAISERROR ('Conference day ID does not exist', 0, 0)
      RETURN
    END

  IF @NewSeats < 0
    BEGIN
      RAISERROR ('Negative number of seats does not make sense', 0, 0)
      RETURN
    END

  DECLARE @current_seats AS INT
  SET @current_seats = (SELECT Seats FROM ConfrenceDay
    WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID)
  DECLARE @diff AS INT
  SET @diff = @NewSeats - @current_seats

  IF dbo.GetFreeSeatsByConferenceDayID(@ConferenceDayID) < (@diff * -1)
    BEGIN
      RAISERROR ('Cannot remove already reserved seats', 0, 0)
      RETURN
    END

  UPDATE ConferenceDay SET Seats = @NewSeats WHERE ConferenceDayID = @ConferenceDayID
end
```

Procedura *ChangeSeatsSeminar*

Procedura zmieniająca ilość miejsc na warsztacie.

```
CREATE OR ALTER PROCEDURE ChangeSeatsSeminar @SeminarID INT, @NewSeats INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM Seminar WHERE Seminar.SeminarID = @SeminarID)
    BEGIN
      RAISERROR ('Seminar ID does not exist', 0, 0)
      RETURN
    END

  IF @NewSeats < 0
    BEGIN
      RAISERROR ('Negative number of seats does not make sense', 0, 0)
      RETURN
    END

  DECLARE @current_seats AS INT
  SET @current_seats = (SELECT Seats FROM Seminar WHERE Seminar.SeminarID = @SeminarID)
  DECLARE @diff AS INT
  SET @diff = @NewSeats - @current_seats

  IF dbo.GetFreeSeatsBySeminarID(@SeminarID) < (@diff * -1)
    BEGIN
      RAISERROR ('Cannot remove already reserved seats', 0, 0)
      RETURN
    END

  UPDATE Seminar SET Seats = @NewSeats WHERE SeminarID = @SeminarID
end
```

Procedura *CancelConference*

Procedura anulująca konferencję.

```
CREATE OR ALTER PROCEDURE CancelConference @ConferenceID INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM Conference WHERE Conference.ConferenceID = @ConferenceID)
    BEGIN
      RAISERROR ('Conference ID does not exist', 0, 0)
      RETURN
    END

  UPDATE Conference SET IsCanceled = 1 WHERE ConferenceID = @ConferenceID
end
```

Procedura *CancelConferenceDay*

Procedura anulująca dzień konferencji.

```
CREATE OR ALTER PROCEDURE CancelConferenceDay @ConferenceDayID INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM ConferenceDay
                WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID)
    BEGIN
      RAISERROR ('Conference day ID does not exist', 0, 0)
      RETURN
    END

  UPDATE ConferenceDay SET IsCanceled = 1 WHERE ConferenceDayID = @ConferenceDayID
end
```

Procedura *CancelSeminar*

Procedura anulująca warsztat.

```
CREATE OR ALTER PROCEDURE CancelSeminar @SeminarID INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM Seminar WHERE Seminar.SeminarID = @SeminarID)
    BEGIN
      RAISERROR ('Seminar ID does not exist', 0, 0)
      RETURN
    END

  UPDATE Seminar SET IsCanceled = 1 WHERE SeminarID = @SeminarID
end
```

Procedura *CancelReservation*

Procedura anulująca rezerwację.

```
CREATE OR ALTER PROCEDURE CancelReservation @ReservationID INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM Reservation WHERE Reservation.ReservationID = @ReservationID)
    BEGIN
      RAISERROR ('Reservation ID does not exist', 0, 0)
      RETURN
    END

  UPDATE Reservation SET IsCanceled = 1 WHERE ReservationID = @ReservationID
end
```

Procedura *CancelSeminarReservation*

Procedura anulująca na warsztat.

```
CREATE OR ALTER PROCEDURE CancelSeminarReservation @SeminarReservationID INT
AS
BEGIN

  IF NOT EXISTS(SELECT * FROM SeminarReservation
                WHERE SeminarReservation.SeminarReservationID = @SeminarReservationID)
    BEGIN
      RAISERROR ('Seminar Reservation ID does not exist', 0, 0)
      RETURN
    END

  UPDATE SeminarReservation SET IsCanceled = 1
    WHERE SeminarReservationID = @SeminarReservationID
end
```