Dokumentacja projektu systemu do zarządzania konferencjami

Natalia Brzozowska
Kamil Burkiewicz

Spis Treści

I. Zadanie projektowe:

A. Klienci indywidualni i nieindywidualni

Celem projektu jest utworzenie systemu bazodanowego dla firmy zajmującej się organizacją konferencji. Na jedno- lub kilkudniowe konferencje, klienci indywidualni jak i nieindywidualni mogą rejestrować się przez system www.

Klienci indywidualni są jednocześnie uczestnikami konferencji (swoimi reprezentantami), ale mogą zapisać na konferencję ze swojego profilu kilka osób (w tym np. swoje dzieci).

Klienci nieidywidualni (np. firmy) rejestrują na konferencję uczestników (np. swoich pracowników) przez reprezentanta firmy, z którym będzie odbywał się dalszy kontakt odnośnie uzupełnienia danych o uczestnikach i opłata udziału.

Klienci indywidualni jak i nieindywidualni mogą rejestrować siebie i uczestników na poszczególne dni konferencji i poszczególne warsztaty.

Klienci muszą podać dane uczestników do dwóch tygodni.

Aby uczestnik mógł wziąć udział danego dnia w warsztatach musi być zarejestrowany na konferencję na dany dzień.

B. Opłaty

Za udział w konferencji klientów nieindywidualnych płacą reprezentanci firm. Za udział w konferencji osób indywidualnych płaci reprezentant osób indywidualnych.

Dodatkowo cena udziału w konferencji zmienia się w zależności od tego jak dużo czasu pozostało do wydarzenia. Zapisy rozpoczynają się na dwa miesiące przed datą rozpoczęcia konferencji i każdego tygodnia cena rośnie o pewną ustaloną wartość dla każdej konferencji.

Istnieje możliwość skorzystania ze zniżki studenckiej, po okazaniu legitymacji i podaniu numeru indeksu. Zniżka jest określona dla każdej konferencji osobno.

Zniżki udzielane są od całościowej ceny konferencji, zawierającej w sobie cenę za warsztaty. Na zapłatę klienci mają tydzień od rejestracji na konferencję.

II. Aktorzy i funkcje systemu:

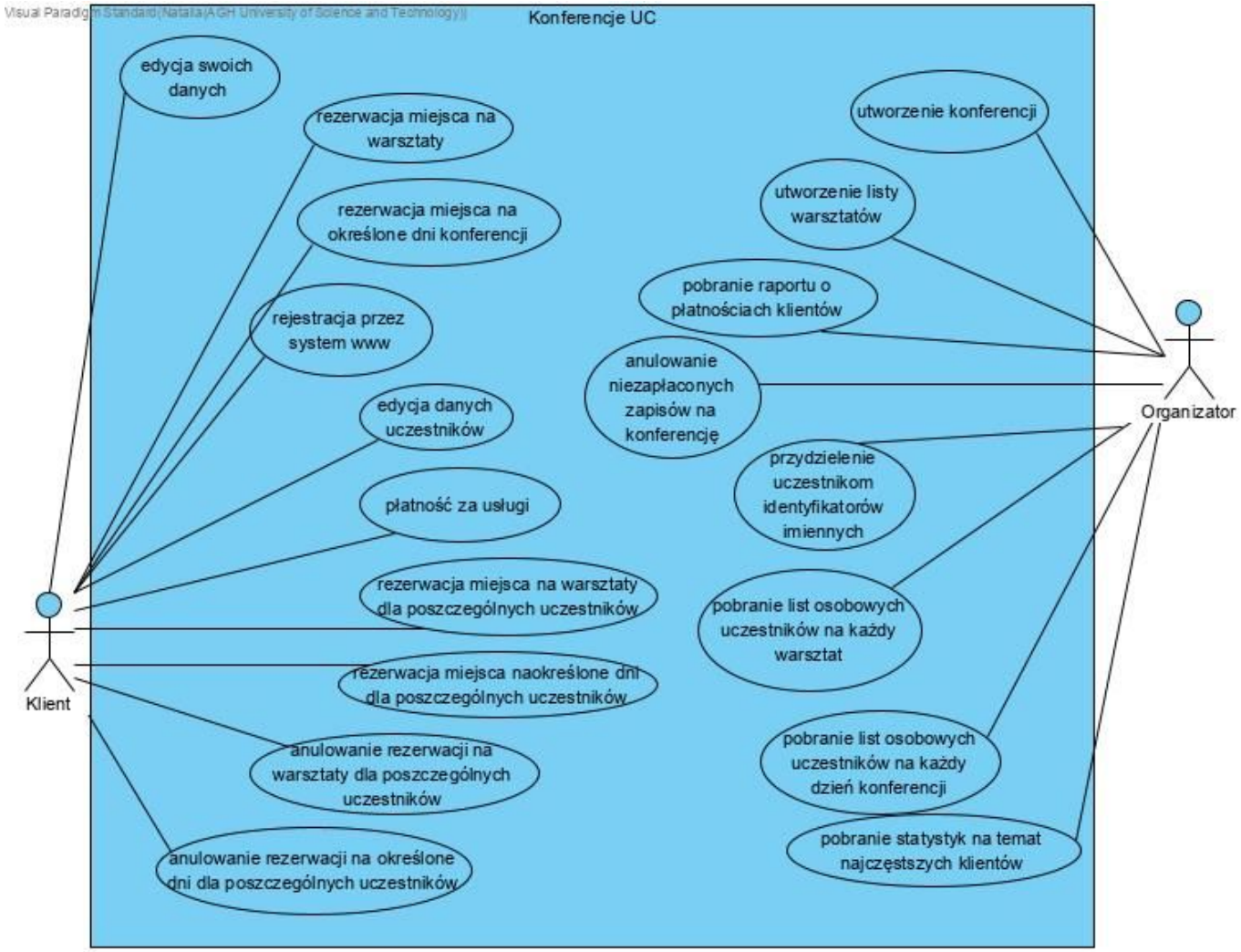
A. Aktorzy:

a. Klient indywidualny
b. Klient nieidywidualny
c. Organizator

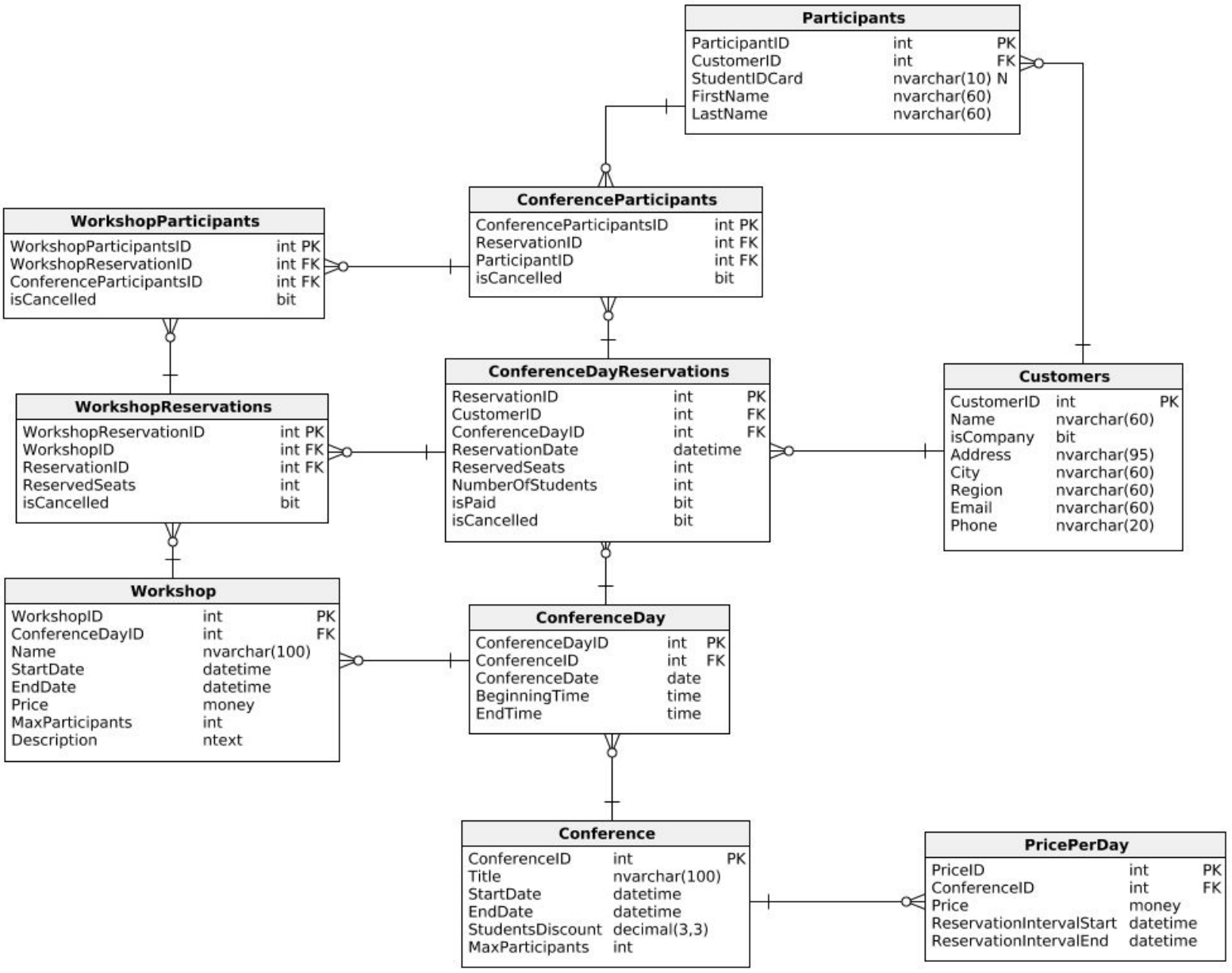
B. Funkcje systemu:

a. Klient indywidualny i nieindywidualny
    1. rejestracja poprzez system www
    2. edycja swoich danych
    3. edycja danych uczestników
    4. rezerwacja miejsca na warsztaty
    5. rezerwacja miejsca na warsztaty dla poszczególnych uczestników
    6. rezerwacja miejsca na określone dni konferencji
    7. anulowanie rezerwacji na warsztaty
    8. anulowanie rezerwacji na warsztaty dla poszczególnych uczestników
    9. anulowanie rezerwacji na konferencje
    10. płatność za usługi
b. Organizator
    1. utworzenie konferencji
    2. utworzenie warsztatów
    3. pobranie list osobowych uczestników na każdy dzień konferencji
    4. pobranie list osobowych uczestników na każdy warsztat
    5. pobranie raportu o płatnościach klientów
    6. pobranie statystyk na temat najczęstszych klientów
    7. przydzielenie uczestnikom identyfikatorów imiennych
    8. anulowanie niezapłaconych zapisów na konferencji

III. Diagram UC



IV. Schemat bazy danych:



Opisy tabeli:

W tabeli **Participants** znajdują się uczestnicy konferencji, mający swoje ID, imię, nazwisko wraz z identyfikatorem osoby, która ich zarejestrowała (CustomerID), oraz numerem legitymacji studenckiej (StudentIDCard), która może być NULL-em jeśli uczestnik nie jest studentem.

```
CREATE TABLE Participants
(
  ParticipantID int       NOT NULL PRIMARY KEY IDENTITY (1,1),
  CustomerID    int       NOT NULL,
  StudentIDCard nvarchar(10) NULL,
  FirstName     nvarchar(60) NOT NULL,
  LastName      nvarchar(60) NOT NULL,
);

ALTER TABLE Participants
  ADD CONSTRAINT Participants_Customers
     FOREIGN KEY (CustomerID)
       REFERENCES Customers (CustomerID);

ALTER TABLE Participants
  ADD CONSTRAINT StudentCardCheck
     CHECK (StudentIDCard LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]%')
```

W tabeli **ConferenceParticipants** znajdują się identyfikatory uczestników, ID dnia konkretnej konferencji, na którą zostali zapisani (ReservationID) oraz ID łączące te dwie informacje.

```
CREATE TABLE ConferenceParticipants
(
  ConferenceParticipantsID int NOT NULL PRIMARY KEY IDENTITY (1,1),
  ReservationID         int NOT NULL,
  ParticipantID         int NOT NULL,
isCancelled bit  NOT NULL DEFAULT(0),

);

ALTER TABLE ConferenceParticipants
  ADD CONSTRAINT ConferenceParticipants_ConferenceDayReservations
     FOREIGN KEY (ReservationID)
       REFERENCES ConferenceDayReservations (ReservationID);

ALTER TABLE ConferenceParticipants
  ADD CONSTRAINT ConferenceParticipants_Participants
```

```
    FOREIGN KEY (ParticipantID)
      REFERENCES Participants (ParticipantID);
```

W tabeli **WorkshopParticipants** znajdują się identyfikatory uczestników oraz ID dnia, w którym odbywa się konkretny warsztat, na który zostali zapisani (WorkshopReservationID).

```
CREATE TABLE WorkshopParticipants (
  WorkshopParticipantsID int  NOT NULL PRIMARY KEY ,
  WorkshopReservationID int  NOT NULL,
  ConferenceParticipantsID int  NOT NULL,
isCancelled bit  NOT NULL DEFAULT(0),
);

ALTER TABLE WorkshopParticipants
  ADD CONSTRAINT WorkshopParticipants_ConferenceParticipants
    FOREIGN KEY (ConferenceParticipantsID)
      REFERENCES ConferenceParticipants (ConferenceParticipantsID);

ALTER TABLE WorkshopParticipants
  ADD CONSTRAINT WorkshopParticipants_WorkshopReservations
    FOREIGN KEY (WorkshopReservationID)
      REFERENCES WorkshopReservations (WorkshopReservationID);
```

W tabeli **ConferenceDayReservations** znajdują się ID dokonanej rezerwacji, ID klienta, ID dnia konferencji, data dokonania rezerwacji, ilość zarezerwowanych miejsc, ilość miejsc zarezerwowanych dla studentów oraz wartość określająca czy klient zapłacił już całościową cenę za konferencję wraz z warsztatami.

```
CREATE TABLE ConferenceDayReservations
(
  ReservationID   int     NOT NULL PRIMARY KEY IDENTITY (1,1),
  CustomerID      int     NOT NULL,
  ConferenceDayID  int     NOT NULL,
  ReservationDate  datetime NOT NULL,
  ReservedSeats   int     NOT NULL CHECK (ReservedSeats > 0),
  NumberOfStudents int     NOT NULL CHECK (NumberOfStudents >= 0),
  isPaid          bit     NOT NULL DEFAULT(0),
isCancelled bit  NOT NULL DEFAULT(0),
);

ALTER TABLE ConferenceDayReservations
```

```
  ADD CONSTRAINT ConferenceDayReservations_ConferenceDay
    FOREIGN KEY (ConferenceDayID)
      REFERENCES ConferenceDay (ConferenceDayID);

ALTER TABLE ConferenceDayReservations
  ADD CONSTRAINT ConferenceDayReservations_Customers
    FOREIGN KEY (CustomerID)
      REFERENCES Customers (CustomerID);
```

W tabeli **Customers** znajdują się zebrane informacje o klientach- ich identyfikatory, nazwa firmy lub klienta indywidualnego (Name), wartość określająca czy klient jest firmą czy klientem indywidualnym (isCompany) oraz dane kontaktowe klienta.

```
CREATE TABLE Customers
(
  CustomerID int       NOT NULL PRIMARY KEY IDENTITY (1,1),
  Name     nvarchar(60) NOT NULL,
  isCompany  bit       NOT NULL,
  Address   nvarchar(95) NOT NULL,
  City     nvarchar(60) NOT NULL,
  Region    nvarchar(60) NOT NULL,
  Email     nvarchar(60) NOT NULL,
  Phone     nvarchar(20) NOT NULL,
);

ALTER TABLE Customers
  ADD CONSTRAINT EmailCheck
    CHECK (Email like '%_@___%.___%')
```

W tabeli **WorkshopReservations** znajdują się ID rezerwacji na warsztaty, ID warsztatu, ID rezerwacji na konferencję oraz ilość zarezerwowanych miejsc na warsztat.

```
CREATE TABLE WorkshopReservations
(
  WorkshopReservationID int NOT NULL PRIMARY KEY IDENTITY (1,1),
  WorkshopID         int NOT NULL,
  ReservationID       int NOT NULL,
  ReservedSeats      int NOT NULL CHECK (ReservedSeats >= 0),
isCancelled bit  NOT NULL DEFAULT(0),
);

ALTER TABLE WorkshopReservations
```

```
  ADD CONSTRAINT WorkshopReservations_ConferenceDayReservations
    FOREIGN KEY (ReservationID)
      REFERENCES ConferenceDayReservations (ReservationID);

ALTER TABLE WorkshopReservations
  ADD CONSTRAINT WorkshopReservations_Workshop
    FOREIGN KEY (WorkshopID)
      REFERENCES Workshop (WorkshopID);
```

W tabeli **ConferenceDay** znajdują się dni, w których trwa konferencja, identyfikator konferencji, która się w tym czasie odbywa, godzina rozpoczęcia oraz zakończenia konferencji w danym dniu. Każdy dzień konferencji ma przypisane ID (ConferenceDayID).

```
CREATE TABLE ConferenceDay
(
  ConferenceDayID int  NOT NULL PRIMARY KEY IDENTITY (1,1),
  ConferenceID   int  NOT NULL,
  ConferenceDate  date NOT NULL,
  BeginningTime   time NOT NULL,
  EndTime       time NOT NULL,
  CONSTRAINT ConferenceDay_time CHECK (EndTime > BeginningTime),
CHECK(DATEDIFF(hour, EndTime, BeginningTime)<20)
);

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

W tabeli **Workshop** znajdują się numer identyfikacyjny warsztatu, identyfikator dnia konferencji, w którym będzie odbywał się warsztat, nazwa warsztatu, godzina rozpoczęcia i zakończenia warsztatu, cena warsztatu, maksymalna ilość uczestników oraz opis warsztatu.

```
CREATE TABLE Workshop
(
  WorkshopID     int         NOT NULL PRIMARY KEY IDENTITY (1,1),
  ConferenceDayID int         NOT NULL,
  Name         nvarchar(100) NOT NULL,
  StartDate      datetime     NOT NULL,
  EndDate        datetime     NOT NULL,
  Price        money       NOT NULL,
  MaxParticipants int         NOT NULL CHECK (MaxParticipants > 0),
  Description    ntext        NOT NULL,
```

```
  CONSTRAINT Workshop_date CHECK (DATEDIFF(day, EndDate, StartDate) = 0),
CHECK(DATEDIFF(hour, EndDate, StartDate)<15)
);

ALTER TABLE Workshop
  ADD CONSTRAINT Workshop_ConferenceDay
    FOREIGN KEY (ConferenceDayID)
      REFERENCES ConferenceDay (ConferenceDayID);
```

W tabeli **PricePerDay** znajduje się ID konferencji, cena udziału w konferencji, jeżeli klient zarejestruje się w konkretnym przedziale czasowym określonym przez ReservetionIntervalStart oraz ReservationIntervalEnd. Każda tak określona cena posiada swoje ID (PriceID).

```
CREATE TABLE PricePerDay
(
  PriceID             int     NOT NULL PRIMARY KEY IDENTITY (1,1),
  ConferenceID        int     NOT NULL,
  Price               money   NOT NULL CHECK (Price > 0),
  ReservationIntervalStart datetime NOT NULL,
  ReservationIntervalEnd   datetime NOT NULL,
  CONSTRAINT PricePerDay_date CHECK (DATEDIFF(day, ReservationIntervalEnd,
ReservationIntervalStart) = 14)
);

ALTER TABLE PricePerDay
  ADD CONSTRAINT PricePerDay_Conference
    FOREIGN KEY (ConferenceID)
      REFERENCES Conference (ConferenceID);
```

W tabeli **Conference** znajdują się identyfikator konferencji, nazwa konferencji, data rozpoczęcia konferencji, data zakończenia konferencji,zniżka studencka dla danej konferencji oraz maksymalna liczba uczestników.

```
CREATE TABLE Conference
(
  ConferenceID    int         NOT NULL PRIMARY KEY IDENTITY (1,1),
  Title         nvarchar(100) NOT NULL,
  StartDate       datetime    NOT NULL,
  EndDate         datetime    NOT NULL,
  StudentsDiscount decimal(3, 3) NOT NULL CHECK (StudentsDiscount <= 1 and
StudentsDiscount >= 0),
  MaxParticipants  int        NOT NULL CHECK (MaxParticipants > 0 and
MaxParticipants < 200),
```

```
  CONSTRAINT Conference_date CHECK (EndDate >= StartDate),
CHECK(DATEDIFF(day, EndDate, StartDate)<7)
);
```

V. Implementacja

A. Widoki

a) **v_ConferenceDayParticipants** -przedstawia widok zawierający zestawienie uczestników konferencji na każdy dzień konferencji i każdą konferencję

```
create view v_ConferenceDayParticipants as
  select p.FirstName as 'First Name', p.LastName as 'Last Name', c.Name
'Company Name', conf.Title as 'Conference Title', cd.ConferenceDate as
'Conference Date'
  from Participants as p
  inner join conferenceParticipants as cp on
cp.ParticipantID=p.ParticipantID
  inner join ConferenceDayReservations as cdr on
cdr.ReservationID=cp.ReservationID
  inner join Customers as c on c.CustomerID = cdr.CustomerID
  inner join ConferenceDay as cd on
cd.ConferenceDayID=cdr.ConferenceDayID
  inner join Conference as conf on conf.ConferenceID=cd.ConferenceID
```

b) **v_WorkshopParticipants** - przedstawia widok zawierający zestawienie uczestników warsztatów, każdej konferencji

```
create view v_WorkshopParticipants as
  select  p.FirstName as 'First Name', p.LastName as 'Last Name', w.Name as
'Workshop Name', c.Title as 'Conference Title', cd.ConferenceDate as
'Conference Date'
  from Participants as p
  inner join WorkshopParticipants as wp on
wp.ConferenceParticipantsID=p.ParticipantID
  inner join WorkshopReservations as wr on
wr.WorkshopReservationID=wp.WorkshopReservationID
inner join Workshop w on w.WorkshopID=wr.WorkshopID
inner join ConferenceDay cd on w.ConferenceDayID = cd.ConferenceDayID
inner join Conference c on cd.ConferenceID = c.ConferenceID
```

c) **v_BestCustomers** - zawiera nazwy 10 klientów najczęściej korzystających z usług

```
create view v_BestCustomers as
select
top 10
[Customer Name]
,
count(cdr.ReservationID) as 'Number of Conference Reservations'
,
sum([Charge]) as 'Sum of Payments'
from v_Payments
      inner join ConferenceDayReservations as cdr on cdr.CustomerID =
v_Payments.CustomerID
group by [Customer Name];
```

d) **v_Payments** - przedstawia widok zestawiający dane klientów wraz ze sumowanymi należnościami i informacją czy płatność została już dokonana

```
create view v_Payments as
select c.CustomerID,
    c.Name                as 'Customer Name',
    conf.Title            as 'Conference Title',
    [dbo].[sumCosts](cdr.ReservationID) as 'Charge',
    cdr.isPaid            as 'Paid?'
from Customers as c
      inner join ConferenceDayReservations cdr on c.CustomerID =
cdr.CustomerID
      inner join ConferenceDay CD on cdr.ConferenceDayID =
CD.ConferenceDayID
      inner join Conference conf on CD.ConferenceID = conf.ConferenceID
group by conf.Title, c.Name, cdr.ReservationID, cdr.isPaid, c.CustomerID
```

e) **v_ConferenceSummary** - widok przedstawiający podsumowanie konferencji (zawiera nazwę konferencji, liczbę zarezerwowanych miejsc, liczbę wolnych miejsc, ilość zapisanych studentów, ilość osób niebędących studentami, ilość rezerwacji na konferencję, ilość dostępnych warsztatów, ilość rezerwacji na warsztaty oraz ilość zapisanych osób na warsztaty)

```
create view v_ConferenceSummary as
select conf.Title                    as 'Conference Title',
    sum(cdr.ReservedSeats)               'Reserved Seats',
    [dbo].[availableConferenceSeats](conf.ConferenceID) as 'Available Seats',
    sum(cdr.NumberOfStudents)            as 'Number of Students',
```

```
    (sum(cdr.ReservedSeats) - sum(cdr.NumberOfStudents))  as 'Number of
not Student People',
    count(cdr.ReservationID)              as 'Number of Conference
reservations',
    count(w.WorkShopID)                  as 'Number of Available Workshops',
    count(wr.WorkshopReservationID)         as 'Number of Reservations for
Workshops',
    sum(wr.ReservedSeats)                as 'Reserved Seats for Workshops',
     [dbo].[availableWorkshopSeats](WorkshopReservationID) as 'Available
Workshop Seats'
from Conference as conf
      inner join ConferenceDay cd on conf.ConferenceID = cd.ConferenceID
      inner join ConferenceDayReservations cdr on cd.ConferenceDayID =
cdr.ConferenceDayID
      inner join WorkshopReservations wr on cdr.ReservationID =
wr.ReservationID
      inner join Workshop w on w.WorkshopID = wr.WorkshopReservationID
group by conf.Title, conf.ConferenceID, WorkshopReservationID
```

f) v_WorkshopSummary - przedstawia widok podsumowujący warsztaty (zawiera nazwę konferencji, do której "należy" warsztat, nazwę warsztatu, ilość zapisanych osób oraz ilość dostępnych miejsc na dany warsztat)

```
create view v_WorkshopSummary as
select conf.Title          as 'Conference Title',
    w.Name              as 'Workshop Name',
    count(wr.ReservedSeats) as 'Seats Reserved for Workshop',
    [dbo].[availableWorkshopSeats](wr.WorkshopID) as 'Available Seats for
Workshop'

from Workshop as w
      inner join WorkshopReservations wr on w.WorkshopID =
wr.WorkshopID
      inner join ConferenceDay as cd on w.ConferenceDayID =
cd.ConferenceDayID
      inner join Conference conf on CD.ConferenceID = conf.ConferenceID
group by w.Name, wr.WorkshopID, conf.Title
```

B. Funkcje

a) **getPrice** - oblicza cenę za zapis na konferencję w danym okresie (im bliżej rozpoczęcia konferencji, tym cena za udział w niej jest wyższa)

```
create function getPrice
 (
 @ConferenceID integer,
 @resDate date
 )
 returns money
 as
 begin
 declare @result money=(select price from PricePerDay
       where  @ConferenceID = PricePerDay.ConferenceID and
 @resDate>= ReservationIntervalStart and @resDate <=ReservationIntervalEnd)
 return @result;
end
```

b) **sumWorkshopCosts** - służy do obliczania należności za zarezerwowane warsztaty

```
create function sumWorkshopCosts(
  @ReservationID integer
)
  returns money
as
begin
  declare @Workshops table
                (
                  ID          int,
                  ReservedSeats int,
                  Price       money
                )
  insert @Workshops
  select WorkshopReservationID, ReservedSeats, Workshop.Price
  from WorkshopReservations
        inner join Workshop on Workshop.WorkshopID =
WorkshopReservations.WorkshopReservationID
  where isCancelled = 0

  declare @rowsCount integer = (select count(ReservedSeats) from
WorkshopReservations)
  declare @i integer =1
  declare @workshopCosts money=0

  while(@i < @rowsCount)
    begin
      declare @ID integer = (select ID from @Workshops where ID = @i)

      if (@ID = @ReservationID)
        begin
          declare @seats integer = (select ReservedSeats from @Workshops where
ID = @i)
          declare @price integer = (select Price from @Workshops where ID = @i)
          set @workshopCosts = @workshopCosts + (@seats * @price)
        end
    end
  return @workshopCosts;
end
```

c) **sumCosts** - podlicza koszty, za konferencje i warsztaty dla danej rezerwacji klienta

```
create function sumCosts(
  @ReservationID integer
)
  returns money
as
begin
  declare @reservationDate datetime = (select ReservationDate
                        from ConferenceDayReservations
                        where @ReservationID =
ConferenceDayReservations.ReservationID)

  declare @pricePerDay money = [dbo].[getPrice]((select ConferenceID
                          from ConferenceDay
                              inner join ConferenceDayReservations
                                  on ConferenceDayReservations.ConferenceDayID
=
                                      ConferenceDay.ConferenceDayID
                          where @ReservationID =
ConferenceDayReservations.ReservationID),
                          @reservationDate)

  declare @studentsDiscount decimal(3, 3) = (select StudentsDiscount
                        from Conference
                            right outer join  ConferenceDay on
ConferenceDay.ConferenceID = Conference.ConferenceID
                            inner join ConferenceDayReservations
                                on ConferenceDayReservations.ConferenceDayID =
                                  ConferenceDay.ConferenceDayID
                        where @ReservationID =
ConferenceDayReservations.ReservationID)

  declare @noOfStudents integer=(select NumberOfStudents
                    from ConferenceDayReservations
                    where @ReservationID =
ConferenceDayReservations.ReservationID)
  declare @reservedSeats integer=(select ReservedSeats
                    from ConferenceDayReservations
                    where @ReservationID =
ConferenceDayReservations.ReservationID)


  return ([dbo].[sumWorkshopCosts](@ReservationID) +
      @noOfStudents * (@pricePerDay - @pricePerDay * @studentsDiscount) +
```

```
    (@reservedSeats - @noOfStudents) * @pricePerDay)

end
```

d) **availableConferenceSeats** - zwraca ilość dostępnych miejsc na dany dzień konkretnej konferencji

```
create function availableConferenceSeats(
  @ConferenceDayID int
)
  returns int
as
begin
  declare @maxSeats integer = (select MaxParticipants
                  from Conference
                      right outer join ConferenceDay on
ConferenceDay.ConferenceID = Conference.ConferenceID
                  where ConferenceDay.ConferenceDayID =
@ConferenceDayID)

  declare @reservedSeats integer = (select sum(ReservedSeats)
                    from ConferenceDayReservations
                    where ConferenceDayID = @ConferenceDayID)

  return (@maxSeats - @reservedSeats)

end
```

e) **availableWorkshopSeats** - zwraca ilość dostępnych miejsc na konkretny warsztat

```
create function availableWorkshopSeats(
  @WorkshopID int
)
  returns int
as
begin
  declare @maxSeats integer = (select MaxParticipants
                  from Workshop
                  where Workshop.WorkshopID = @WorkshopID)

  declare @reservedSeats integer = (select sum(ReservedSeats)
                    from WorkshopReservations
                    where WorkshopID = @WorkshopID)
```

```
  return (@maxSeats - @reservedSeats)

end
```

f) **confDayParticipants** - zwraca tabelę zawierającą dane uczestników konkretnego dnia, konkretnej konferencji

```
create function confDayParticipants(@ConferenceID int,
                    @Date datetime)
  returns @table table
          (
            @ParticipantID int,
            @FirstName    nvarchar(60),
            @LastName     nvarchar(60),
            @StudentIDCard nvarchar(10)
          )
as
begin
  insert @table
  select p.ParticipantID,
      p.FirstName    as 'First Name',
      p.LastName     as 'Last Name',
      p.StudentIDCard as 'StudentIDCard Number'
  from Participants as p
       inner join ConferenceParticipants CP on p.ParticipantID =
CP.ParticipantID
       inner join ConferenceDayReservations CDR on CP.ReservationID =
CDR.ReservationID
       inner join ConferenceDay cd on CDR.ConferenceDayID =
cd.ConferenceDayID
  where cd.ConferenceID = @ConferenceID
   and cd.ConferenceDate = @Date

  return
end
```

g) **confDayWorkshopParticipants** - zwraca tabelę zawierającą dane uczestników konkretnego warsztatu, odbywającego się konkretnego dnia, konkretnej konferencji

```
create function confDayWorkshopParticipants(@ConferenceID int,
                            @Date datetime,
                            @WorkshopID int)
  returns @table table
            (
              @ParticipantID int,
              @FirstName    nvarchar(60),
              @LastName     nvarchar(60),
              @StudentIDCard nvarchar(10)
            )
as
begin
  insert @table
  select p.ParticipantID,
       p.FirstName    as 'First Name',
       p.LastName     as 'Last Name',
       p.StudentIDCard as 'StudentIDCard Number'
  from Participants as p
        inner join WorkshopParticipants wp on p.ParticipantID =
wp.ConferenceParticipantsID
        inner join WorkshopReservations wr on wp.WorkshopReservationID =
wr.WorkshopReservationID
        inner join ConferenceDayReservations cdr on cdr.ReservationID =
wr.ReservationID
        inner join ConferenceDay cd on CDR.ConferenceDayID =
cd.ConferenceDayID
  where cd.ConferenceID = @ConferenceID
    and cd.ConferenceDate = @Date
    and @WorkshopID = wr.WorkshopID

  return
end
```

D. Triggery

a) **CheckWorkshopParticipants** - sprawdza czy dana osoba nie jest już zapisana na inny warsztat w tym samym czasie

```
create trigger CheckWorkshopParticipants
  on WorkshopParticipants
  after insert, update
  as begin
     declare @personID int = (select ConferenceParticipantsID from inserted)

     declare @noOfWorkshops int = (select count(*) from
WorkshopParticipants
                        inner join  WorkshopReservations WR on
WorkshopParticipants.WorkshopReservationID =
WR.WorkshopReservationID
                        inner join Workshop W on WR.WorkshopID =
W.WorkshopID
                        inner join  Workshop w2 on
w2.ConferenceDayID=w.ConferenceDayID
                        where @personID=ConferenceParticipantsID and
                wr.isCancelled=0 and WorkshopParticipants.isCancelled=0 and
                           ((w.StartDate<w2.StartDate and
w2.StartDate<w.EndDate) or
                            (w2.StartDate<w.StartDate and
w.StartDate<w2.EndDate)))
     if @noOfWorkshops>0
     begin
        raiserror ('this person is already enrolled in the workshop in this time
', 16, 1);
        rollback transaction
     end
end
```

b) **CheckReservedSeats_ConferenceDayReservations** - sprawdza czy nie zarejestrowano więcej osób niż zadeklarowano na konferencję

```
create trigger CheckReservedSeats_ConferenceDayReservations
  on ConferenceDayReservations
  after insert, update
  as
begin
  declare @reservationID int

  set @reservationID = (select ReservationID from inserted)

  declare @participantsAdded int = (select count(*)
                    from ConferenceParticipants
                    where ReservationID = @reservationID and isCancelled=0)
  declare @participantsDeclared int = (select ReservedSeats
                      from ConferenceDayReservations
                      where ReservationID = @reservationID)

  if (@participantsAdded > @participantsDeclared)
    begin
      RAISERROR ('Number of participants added is greater than declared number
of participants', -1, -1);
      ROLLBACK TRANSACTION
    end
end
```

c) **CheckReservedSeats_WorkshopReservations** -sprawdza czy nie zarejestrowało się więcej osób niż było zadeklarowanych na warsztat

```
create trigger CheckReservedSeats_WorkshopReservations
  on WorkshopReservations
  after insert, update
  as
begin
  declare @wReservationID int

  set @wReservationID = (select WorkshopReservationID from inserted)

  declare @participantsAdded int = (select count(*)
                    from WorkshopParticipants
                    where WorkshopReservationID = @wReservationID and
isCancelled=0)

  declare @participantsDeclared int = (select ReservedSeats
                     from WorkshopReservations
                     where WorkshopReservationID = @wReservationID)

  if (@participantsAdded > @participantsDeclared)
    begin
      raiserror ('Number of participants added to workshop is greater than
declared number of participants', -1, -1);
      rollback transaction
    end
end
```

d) **NewPastConference** - sprawdza czy nowo dodana konferencja nie jest w przeszłości

```
create trigger NewPastConference on Conference
  after insert
  as begin

    declare @startDate datetime = (select StartDate from inserted)

    if (@startDate<GETDATE())
    begin
      raiserror ('Conference cannot happen in teh past ', 16, 1);
      rollback transaction
    end
end
```

e) **UpdateNumberOfStudents** - aktualizuje studentów po dodaniu nowej rezerwacji

```
create trigger UpdateNumberOfStudents
  on ConferenceDayReservations
  after insert, update
  as
begin
  declare @reservationID int = (select ReservationID from inserted)

  declare @noOfStudents int = (select count(*) from ConferenceParticipants
                    inner join Participants P on
ConferenceParticipants.ParticipantID = P.ParticipantID
                    where p.StudentIDCard is not null and isCancelled=0 and
@reservationID=ReservationID)

  if (@noOfStudents is not null)
  begin
    update ConferenceDayReservations
    set NumberOfStudents=@noOfStudents
    from inserted where
inserted.ReservationID=ConferenceDayReservations.ReservationID
  end
end
```

E. Procedury

Procedury dodające dane

a) AddConferenceDay - procedura dodająca nowy dzień

```
CREATE PROCEDURE AddConferenceDay
 @ConferenceID   int,
 @ConferenceDate  date,
 @BeginningTime   time,
 @EndTime       time
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION
   IF NOT EXISTS
     (
       SELECT * FROM Conference
       WHERE ConferenceID = @ConferenceID
     )
   BEGIN
     ;THROW 51000, 'There is no conference with such ID.', 1
   END
   IF EXISTS
     (
       SELECT * FROM ConferenceDay
       WHERE ConferenceDate = @ConferenceDate
     )
   BEGIN
     ;THROW 51000, 'This day is already occupied.', 1
   END
   INSERT INTO ConferenceDay (ConferenceID, ConferenceDate,
BeginningTime, EndTime)
   VALUES (@ConferenceID, @ConferenceDate, @BeginningTime,
@EndTime)

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
       = 'Cannot add Conference Day. Error message: ' + ERROR_MESSAGE();
   ;THROW 51000, @errorMsg, 1
```

```
 END CATCH
END
GO
```

b) **AddConference** - procedura dodająca konferencję

```
CREATE PROCEDURE AddConference
 @Title nvarchar(100),
 @StartDate datetime,
 @EndDate datetime,
 @StudentsDiscount decimal(3,3),
 @MaxParticipants int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

   INSERT INTO   Conference (Title, StartDate, EndDate, StudentsDiscount,
MaxParticipants)
   VALUES (@Title, @StartDate, @EndDate, @StudentsDiscount,
@MaxParticipants)

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
       = 'Cannot add Conference. Error message: ' + ERROR_MESSAGE();
   ;THROW 52000, @errorMsg, 1
 END CATCH
END
GO
```

c) **AddParticipant** - procedura dodająca uczestnika konferencji

```
CREATE PROCEDURE AddParticipant
 @CustomerID   int,
 @StudentIDCard nvarchar(10),
 @FirstName    nvarchar(60),
 @LastName     nvarchar(60)
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

   IF NOT EXISTS
     (
       SELECT * FROM Customers
       WHERE CustomerID = @CustomerID
     )
   BEGIN
     ;THROW 51000, 'There is no customer with such ID', 1
   END

   INSERT INTO Participants (CustomerID, StudentIDCard, FirstName,
LastName)
   VALUES (@CustomerID, @StudentIDCard, @FirstName, @LastName)

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
       = 'Cannot add Participant. Error message: ' + ERROR_MESSAGE();
   ;THROW 51000, @errorMsg, 1
 END CATCH
END
GO
```

d) AddPricePerDay - procedura dodawania ceny za przedział czasu

```
CREATE PROCEDURE AddPricePerDay
 @ConferenceID          int,
 @Price                 money,
 @ReservationIntervalStart datetime,
 @ReservationIntervalEnd   datetime
AS
BEGIN
 SET NOCOUNT ON;

  BEGIN TRY
 BEGIN TRANSACTION

   IF NOT EXISTS
     (
       SELECT * FROM Conference
       WHERE ConferenceID = @ConferenceID
     )
   BEGIN
     ;THROW 51000, 'There is no Conference with such ID.', 1
   END

   INSERT INTO PricePerDay (ConferenceID, Price,
ReservationIntervalStart, ReservationIntervalEnd)
   VALUES (@ConferenceID, @Price, @ReservationIntervalStart,
@ReservationIntervalEnd)

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
       = 'Cannot add Price Per Day. Error message: ' + ERROR_MESSAGE();
   ;THROW 51000, @errorMsg, 1
 END CATCH

END
GO
```

e) **AddWorkshopParticipant** - procedura dodawania uczestnika warsztatów

```
CREATE PROCEDURE AddWorkshopParticipant
 @WorkshopReservationID    int,
 @ConferenceParticipantsID int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

   IF NOT EXISTS
     (
       SELECT * FROM ConferenceParticipants
       WHERE ConferenceParticipantsID = @ConferenceParticipantsID
     )
   BEGIN
     ;THROW 51000, 'There is no ConferenceParticipants with such ID.', 1
   END

   IF NOT EXISTS
     (
       SELECT * FROM WorkshopReservations
       WHERE WorkshopReservationID = @WorkshopReservationID
     )
   BEGIN
     ;THROW 51000, 'There is no WorkshopReservation with such ID.', 1
   END

   IF
     (
       (SELECT ConferenceDayID FROM ConferenceDayReservations AS cdr
        INNER JOIN ConferenceParticipants AS cp
          ON cp.ReservationID = cdr.ReservationID
        WHERE cp.ConferenceParticipantsID = @ConferenceParticipantsID)
        <>
       (SELECT ConferenceDayID FROM Workshop AS w
        INNER JOIN WorkshopReservations AS wr
          ON w.WorkshopID = wr.WorkshopID
        WHERE wr.WorkshopReservationID = @WorkshopReservationID)
     )
   BEGIN
     ;THROW 51000, 'Participant having this ID does not have reservation on
this day of conference,
            so he cannot participate in this workshop.', 1
```

```
    END

    INSERT INTO   WorkshopParticipants(WorkshopReservationID,
ConferenceParticipantsID)
    VALUES (@WorkshopReservationID, @ConferenceParticipantsID)

  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    DECLARE @errorMsg nvarchar(2048)
        = 'Cannot add Workshop Participant. Error message: ' +
ERROR_MESSAGE();
    ;THROW 51000, @errorMsg, 1
  END CATCH
END
GO
```

f) **AddConferenceDayReservation** - procedura dodająca rezerwację na dzień konferencji

```
CREATE PROCEDURE AddConferenceDayReservation
 @CustomerID      int,
 @ConferenceDayID  int,
 @ReservationDate  datetime,
 @ReservedSeats    int,
 @NumberOfStudents int,
 @isPaid           bit

AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

   IF NOT EXISTS
     (
       SELECT * FROM Customers
       WHERE CustomerID = @CustomerID
     )
   BEGIN
     ;THROW 51000, 'There is no customer with such ID.', 1
   END

   DECLARE @TakenSeats int;
   SET @TakenSeats = (SELECT SUM(ReservedSeats) FROM
ConferenceDayReservations
             WHERE ConferenceDayID = @ConferenceDayID
             GROUP BY ConferenceDayID);


   IF
     (
       (SELECT MaxParticipants FROM Conference AS conf
        INNER JOIN ConferenceDay AS cd
          ON conf.ConferenceID = cd.ConferenceID
        WHERE cd.ConferenceDayID = @ConferenceDayID) < @ReservedSeats
+ @TakenSeats
     )
   BEGIN
     ;THROW 51000, 'Conference participants limit exceeded.', 1
   END
```

```
    INSERT INTO ConferenceDayReservations (CustomerID,
ConferenceDayID, ReservationDate, ReservedSeats, NumberOfStudents,
isPaid)
    VALUES (@CustomerID, @ConferenceDayID, @ReservationDate,
@ReservedSeats, @NumberOfStudents, @isPaid)

  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    DECLARE @errorMsg nvarchar(2048)
        = 'Cannot add Conference Day Reservation. Error message: ' +
ERROR_MESSAGE();
    ;THROW 51000, @errorMsg, 1
  END CATCH
END
GO
```

g) AddCustomer - procedura dodająca klienta

```
CREATE PROCEDURE AddCustomer
 @Name      nvarchar(60),
 @isCompany  bit,
 @Address    nvarchar(95),
 @City      nvarchar(60),
 @Region     nvarchar(60),
 @Email     nvarchar(60),
 @Phone     nvarchar(20)
AS
BEGIN

 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

   IF EXISTS
     (
       SELECT * FROM Customers
       WHERE Email = @Email
     )
   BEGIN
     ;THROW 51000, 'Email already used.', 1
   END

   INSERT INTO Customers (Name, isCompany, Address, City, Region,
Email, Phone)
   VALUES (@Name, @isCompany, @Address, @City, @Region, @Email,
@Phone)

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
       = 'Cannot add Customer. Error message: ' + ERROR_MESSAGE();
   ;THROW 51000, @errorMsg, 1
 END CATCH
END
GO
```

h) **AddWorkshopReservation** - procedura dodająca rezerwację na warsztaty

```
CREATE PROCEDURE AddWorkshopReservation
 @WorkshopID         int,
 @ReservationID      int,
 @ReservedSeats      int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

   IF NOT EXISTS
     (
       SELECT * FROM Workshop
       WHERE WorkshopID = @WorkshopID
     )
   BEGIN
     ;THROW 51000, 'There is no workshop with such ID', 1
   END

   IF NOT EXISTS
     (
       SELECT * FROM ConferenceDayReservations
       WHERE ReservationID = @ReservationID
     )
   BEGIN
     ;THROW 51000, 'There is no reservation with such ID', 1
   END

   DECLARE @TakenSeats INT;
   SET @TakenSeats = (SELECT SUM(ReservedSeats) FROM
WorkshopReservations
             WHERE WorkshopID = @WorkshopID
             GROUP BY WorkshopID);


   IF
     (
       (SELECT MaxParticipants FROM Workshop
        WHERE WorkshopID = @WorkshopID) < @ReservedSeats +
@TakenSeats
     )
   BEGIN
     ;THROW 51000, 'Workshop participants limit exceeded.', 1
```

```
    END

    IF
      (
        (SELECT ConferenceDayID FROM ConferenceDayReservations
         WHERE ReservationID = @ReservationID)
         <>
        (SELECT ConferenceDayID FROM Workshop
         WHERE WorkshopID = @WorkshopID)
      )
    BEGIN
      ;THROW 51000, 'There was no reservation for this day of conference.', 1
    END

    INSERT INTO   WorkshopReservations (WorkshopID, ReservationID,
ReservedSeats)
    VALUES (@WorkshopID, @ReservationID, @ReservedSeats)

  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    DECLARE @errorMsg nvarchar(2048)
        = 'Cannot add workshop reservation. Error message: ' +
ERROR_MESSAGE();
    ;THROW 51000, @errorMsg, 1
  END CATCH
END
GO
```

i) **AddConferenceParticipant** - procedura dodawania uczestnika konferencji

```
CREATE PROCEDURE AddConferenceParticipant
 @ReservationID        int,
 @ParticipantID        int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

   IF NOT EXISTS
     (
       SELECT * FROM Participants
       WHERE ParticipantID = @ParticipantID
     )
   BEGIN
     ;THROW 51000, 'There is no participant with such ID.', 1
   END

   IF NOT EXISTS
     (
       SELECT * FROM ConferenceDayReservations
       WHERE ReservationID = @ReservationID
     )
   BEGIN
     ;THROW 51000, 'There is no reservation with such ID.', 1
   END

   INSERT INTO ConferenceParticipants (ReservationID, ParticipantID)
   VALUES (@ReservationID, @ParticipantID)

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
       = 'Cannot add Conference Participant. Error message: ' +
ERROR_MESSAGE();
   ;THROW 51000, @errorMsg, 1
 END CATCH
END
GO
```

j) **AddWorkshop** - procedura dodawania warsztatu

```
CREATE PROCEDURE AddWorkshop
 @ConferenceDayID int,
 @Name          nvarchar(100),
 @StartDate     datetime,
 @EndDate       datetime,
 @Price         money,
 @MaxParticipants int,
 @Description    ntext
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION
   IF NOT EXISTS
     (
       SELECT * FROM ConferenceDay
       WHERE ConferenceDayID = @ConferenceDayID
     )
   BEGIN
     ;THROW 51000, 'There is no ConferenceDay with such ID.', 1
   END

   INSERT INTO Workshop (ConferenceDayID, Name, StartDate, EndDate,
Price, MaxParticipants, Description)
   VALUES (@ConferenceDayID, @Name, @StartDate, @EndDate, @Price,
@MaxParticipants, @Description)

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
       = 'Cannot add Workshop. Error message: ' + ERROR_MESSAGE();
   ;THROW 51000, @errorMsg, 1
 END CATCH
END
GO
```

Procedury uaktualniające dane

k) **UpdateCustomerInfo** - procedura uaktualniająca informacje o kliencie

```
CREATE PROCEDURE UpdateCustomerInfo
 @CustomerID int,
 @Name nvarchar(60),
 @Address nvarchar(95),
 @City nvarchar(60),
 @Region nvarchar(60),
 @Email nvarchar(60),
 @Phone nvarchar(20)
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

  IF NOT EXISTS
   (
    SELECT * FROM Customers
    WHERE CustomerID = @CustomerID
   )
  BEGIN
   ;THROW 51000, 'There is no customer with such ID.', 1
  END

  IF @Name IS NOT NULL
  BEGIN
   UPDATE Customers
   SET Name = @Name
   WHERE CustomerID = @CustomerID
  END

  IF @Address IS NOT NULL
  BEGIN
   UPDATE Customers
   SET Address = @Address
   WHERE CustomerID = @CustomerID
  END

  IF @City IS NOT NULL
  BEGIN
   UPDATE Customers
```

```
    SET City = @City
    WHERE CustomerID = @CustomerID
  END

  IF @Region IS NOT NULL
  BEGIN
    UPDATE Customers
    SET Region = @Region
    WHERE CustomerID = @CustomerID
  END

  IF @Email LIKE '%@%'
  BEGIN
    IF @Email IS NOT NULL
    BEGIN
      UPDATE Customers
      SET Email = @Email
      WHERE CustomerID = @CustomerID
    END
  END

  DECLARE @ShortestPhonePossibleLen int = 9;  -- '123 456 789'
  DECLARE @LongestPhonePossibleLen  int = 12; -- '+48 123 456 789'

  IF LTRIM(@Phone) <> '' AND LEN(@Phone) <=
@LongestPhonePossibleLen AND LEN(@Phone) >=
@ShortestPhonePossibleLen
  BEGIN
    IF @Phone IS NOT NULL
    BEGIN
      UPDATE Customers
      SET Phone = @Phone
      WHERE CustomerID = @CustomerID
    END
  END
 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
  ROLLBACK TRANSACTION
  DECLARE @errorMsg nvarchar(2048)
    = 'Cannot add Conference. Error message: ' + ERROR_MESSAGE();
  ;THROW 52000, @errorMsg, 1
 END CATCH

END
GO
```

l) **UpdateParticipantInfo** - procedura uaktualniająca informacje o uczestniku

```
CREATE PROCEDURE UpdateParticipantInfo
 @ParticipantID int,
 @StudentIDCard NVARCHAR(10),
 @FirstName NVARCHAR(60),
 @LastName NVARCHAR(60)
AS
BEGIN

 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION

  IF NOT EXISTS
   (
    SELECT * FROM Participants
    WHERE ParticipantID = @ParticipantID
   )
  BEGIN
   ;THROW 51000, 'There is no participant with such ID.', 1
  END

  IF @FirstName IS NOT NULL
  BEGIN
   UPDATE Participants
   SET FirstName = @FirstName
   WHERE ParticipantID = @ParticipantID
  END

  IF @StudentIDCard IS NOT NULL
  BEGIN
   UPDATE Participants
   SET StudentIDCard = @StudentIDCard
   WHERE ParticipantID = @ParticipantID
  END

  IF @LastName IS NOT NULL
  BEGIN
   UPDATE Participants
   SET LastName = @LastName
   WHERE ParticipantID = @ParticipantID
  END
```

```
 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
   ROLLBACK TRANSACTION
   DECLARE @errorMsg nvarchar(2048)
     = 'Cannot add Conference. Error message: ' + ERROR_MESSAGE();
   ;THROW 52000, @errorMsg, 1
 END CATCH

END
GO
```

Procedury anulujące

m) **CancelWorkshopReservation** - procedura anulująca rezerwację na warsztaty

```
CREATE PROCEDURE CancelWorkshopReservation
 @WorkshopReservationID int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION
   IF NOT EXISTS
     (
       SELECT * FROM WorkshopReservations
       WHERE WorkshopReservationID = @WorkshopReservationID
     )
   BEGIN
     ;THROW 51000, 'There is no workshop reservation with such ID.', 1
   END

   IF
     (
       (SELECT isCancelled FROM WorkshopReservations
        WHERE WorkshopReservationID = @WorkshopReservationID) = 1
     )
   BEGIN
     ;THROW 51000, 'Workshop reservation already cancelled.', 1
   END

   UPDATE WorkshopReservations
   SET isCancelled = 1
   WHERE WorkshopReservationID = @WorkshopReservationID

   DECLARE cursor_WorkshopParticipants CURSOR LOCAL
FAST_FORWARD FOR
     SELECT DISTINCT WorkshopParticipantsID FROM
WorkshopParticipants
     WHERE WorkshopReservationID = @WorkshopReservationID;

   DECLARE @WorkshopParticipantID int;

   OPEN cursor_WorkshopParticipants
   FETCH NEXT FROM cursor_WorkshopParticipants INTO
@WorkshopParticipantID
```

```
    WHILE @@FETCH_STATUS = 0
     BEGIN
      EXEC CancelWorkshopParticipation @WorkshopParticipantID;

      FETCH NEXT FROM cursor_WorkshopParticipants INTO
@WorkshopParticipantID
     END
    CLOSE cursor_WorkshopParticipants
    DEALLOCATE cursor_WorkshopParticipants

  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    DECLARE @errorMsg nvarchar(2048)
      = 'Cannot add Conference. Error message: ' + ERROR_MESSAGE();
    ;THROW 52000, @errorMsg, 1
  END CATCH
END
GO
```

n) **CancelWorkshopParticipation** -procedura anulująca uczestnictwo w warsztatach

```
CREATE PROCEDURE CancelWorkshopParticipation
 @WorkshopParticipantID int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION
  IF NOT EXISTS
    (
      SELECT * FROM WorkshopParticipants
      WHERE WorkshopParticipantsID = @WorkshopParticipantID
    )
  BEGIN
    ;THROW 51000, 'There is no workshop participant with such ID.', 1
  END

  IF
    (
      (SELECT isCancelled FROM WorkshopParticipants
       WHERE WorkshopParticipantsID = @WorkshopParticipantID) = 1
    )
  BEGIN
    ;THROW 51000, 'Workshop participation already cancelled.', 1
  END

  UPDATE WorkshopParticipants
  SET isCancelled = 1
  WHERE WorkshopParticipantsID = @WorkshopParticipantID

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
  ROLLBACK TRANSACTION
  DECLARE @errorMsg nvarchar(2048)
    = 'Cannot add Conference. Error message: ' + ERROR_MESSAGE();
  ;THROW 52000, @errorMsg, 1
 END CATCH
END
GO
```

o) **CancelConferenceParticipation** - procedura anulująca uczestnictwo w konferencji

```
CREATE PROCEDURE CancelConferenceParticipation
 @ConferenceParticipantsID int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION
  IF NOT EXISTS
    (
      SELECT * FROM ConferenceParticipants
      WHERE ConferenceParticipantsID = @ConferenceParticipantsID
    )
  BEGIN
    ;THROW 51000, 'There is no conference participant with such ID.', 1
  END

  IF
    (
      (SELECT isCancelled FROM dbo.ConferenceParticipants
       WHERE ConferenceParticipantsID = @ConferenceParticipantsID) = 1
    )
  BEGIN
    ;THROW 51000, 'Conference participation already cancelled.', 1
  END

  UPDATE ConferenceParticipants
  SET isCancelled = 1
  WHERE ConferenceParticipantsID = @ConferenceParticipantsID

 COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
  ROLLBACK TRANSACTION
  DECLARE @errorMsg nvarchar(2048)
    = 'Cannot add Conference. Error message: ' + ERROR_MESSAGE();
  ;THROW 52000, @errorMsg, 1
 END CATCH

END
GO
```

p) **CancelConferenceDayReservation** - procedura anulowania rezerwacji na dany dzień konferencji

```
CREATE PROCEDURE CancelConferenceDayReservation
 @ReservationID int
AS
BEGIN
 SET NOCOUNT ON;

 BEGIN TRY
 BEGIN TRANSACTION
  IF NOT EXISTS
   (
    SELECT * FROM ConferenceDayReservations
    WHERE ReservationID = @ReservationID
   )
  BEGIN
   ;THROW 51000, 'There is no reservation with such ID.', 1
  END

  IF
   (
    (SELECT isCancelled FROM ConferenceDayReservations
     WHERE ReservationID = @ReservationID) = 1
   )
  BEGIN
   ;THROW 51000, 'Reservation already cancelled.', 1
  END

  UPDATE ConferenceDayReservations
  SET isCancelled = 1
  WHERE ReservationID = @ReservationID

  DECLARE cursor_ConferenceParticipants CURSOR LOCAL
FAST_FORWARD FOR
    SELECT DISTINCT ConferenceParticipantsID FROM
ConferenceParticipants
    WHERE ReservationID = @ReservationID;

  DECLARE @ConferenceParticipantID int;

  OPEN cursor_ConferenceParticipants
  FETCH NEXT FROM cursor_ConferenceParticipants INTO
@ConferenceParticipantID
  WHILE @@FETCH_STATUS = 0
   BEGIN
```

```
      EXEC CancelConferenceParticipation @ConferenceParticipantID;

      FETCH NEXT FROM cursor_ConferenceParticipants INTO
@ConferenceParticipantID
     END
    CLOSE cursor_ConferenceParticipants
    DEALLOCATE cursor_ConferenceParticipants



    DECLARE cursor_WorkshopsReservation CURSOR LOCAL
FAST_FORWARD FOR
      SELECT DISTINCT WorkshopReservationID FROM
WorkshopReservations
      WHERE ReservationID = @ReservationID;

    DECLARE @WorkshopReservationID int;

    OPEN cursor_WorkshopsReservation
    FETCH NEXT FROM cursor_WorkshopsReservation INTO
@WorkshopReservationID
    WHILE @@FETCH_STATUS = 0
     BEGIN
      EXEC CancelWorkshopReservation @WorkshopReservationID;

      FETCH NEXT FROM cursor_ConferenceParticipants INTO
@WorkshopReservationID
     END
    CLOSE cursor_WorkshopsReservation
    DEALLOCATE cursor_WorkshopsReservation

  COMMIT TRANSACTION
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION
    DECLARE @errorMsg nvarchar(2048)
      = 'Cannot add Conference. Error message: ' + ERROR_MESSAGE();
    ;THROW 52000, @errorMsg, 1
  END CATCH
END
GO
```

F. Indeksy

Utworzyliśmy indeksy do kluczy będących kluczami głównymi w danych tabelach.

```
➔ create index Conference_ConferenceID on Conference(ConferenceID);
➔ create index ConferenceDay_ConferenceDayID on
  ConferenceDay(ConferenceDayID)
➔ create index PricePerDay_PriceID on PricePerDay(PriceID);
➔ create index Customers_CustomerID on Customers(CustomerID);
➔ create index Workshop_WorkshopID on Workshop(WorkshopID);
➔ create index ConferenceDayReservations_ReservationID on
  ConferenceDayReservations(ReservationID);
➔ create index Participants_ParticipantID on Participants(ParticipantID);
➔ create index WorkshopReservations_WorkshopReservationID on
  WorkshopReservations(WorkshopReservationID);
➔ create index ConferenceParticipants_ConferenceParticipantsID on
  ConferenceParticipants(ConferenceParticipantsID);
➔ create index WorkshopParticipants_WorkshopParticipantsID on
  WorkshopParticipants(WorkshopParticipantsID);
```