Projekt i implementacja systemu bazodanowego

Błażej Kustra, Mateusz Mastalerczyk

Opis:

Aktorzy:

1. Administrator systemu
2. Pracownik firmy organizującej
3. Firma organizująca konferencje
4. Osoba prywatna (rezerwuje miejsce na konferencje)
5. Firma (rezerwuje miejsca dla pracowników na konferencje)

Funkcje realizowane przez system dla poszczególnych użytkowników:

Administrator

- **dostęp do danych użytkowników - brak możliwości usunięcia**
- **tworzenie/edycja/usuwanie kont pracownika/organizatora/administratora**

Pracownik firmy organizującej

- **Generowanie listy uczestników którzy nie mają wypełnionych danych**
- **Anulowanie zamówień nieopłaconych w terminie**
    - Codzienne uruchamianie procedury sprawdzającej status płatności klientów, usuwa nieopłacone rezerwacje które są spóźnione z zapłatą oraz zwalnia zarezerwowane miejsce dla innych uczestników.

Firma organizująca konferencje

- **Stworzenie konferencji**
    - nazwanie konferencji
    - ustalenie tematu konferencji
    - utworzenie progów cenowych dla konferencji
    - zdefiniowanie czasu konferencji
    - ustalenie limitu uczestników
    - określenie miejsca konferencji
    - ustawienie opłaty za jeden dzień konferencji
    - ustalenie zniżek dla studentów

- **Stworzenie warsztatu**
    - dodanie warsztatu do konkretnego dnia konferencji
    - nazwanie warsztatu
    - określenie trwania warsztatu
    - wprowadzenie daty warsztatów
    - ustalenie tematu warsztatu
    - ustalenie limitu uczestników
    - ustawienie ceny warsztatu
    - dodanie osoby prowadzącej warsztaty
- **Anulowanie konferencji/warsztatów**
    - *anulowanie konferencji w wypadku gdy nikt się nie zapisał, przed rozpoczęciem konferencji*
    - *anulowanie warsztatów w wypadku gdy nikt się nie zapisał, przed rozpoczęciem warsztatów*
- **Generowanie raportu z listą uczestników konferencji**
- **Generowanie statystyk najbardziej aktywnych klientów**

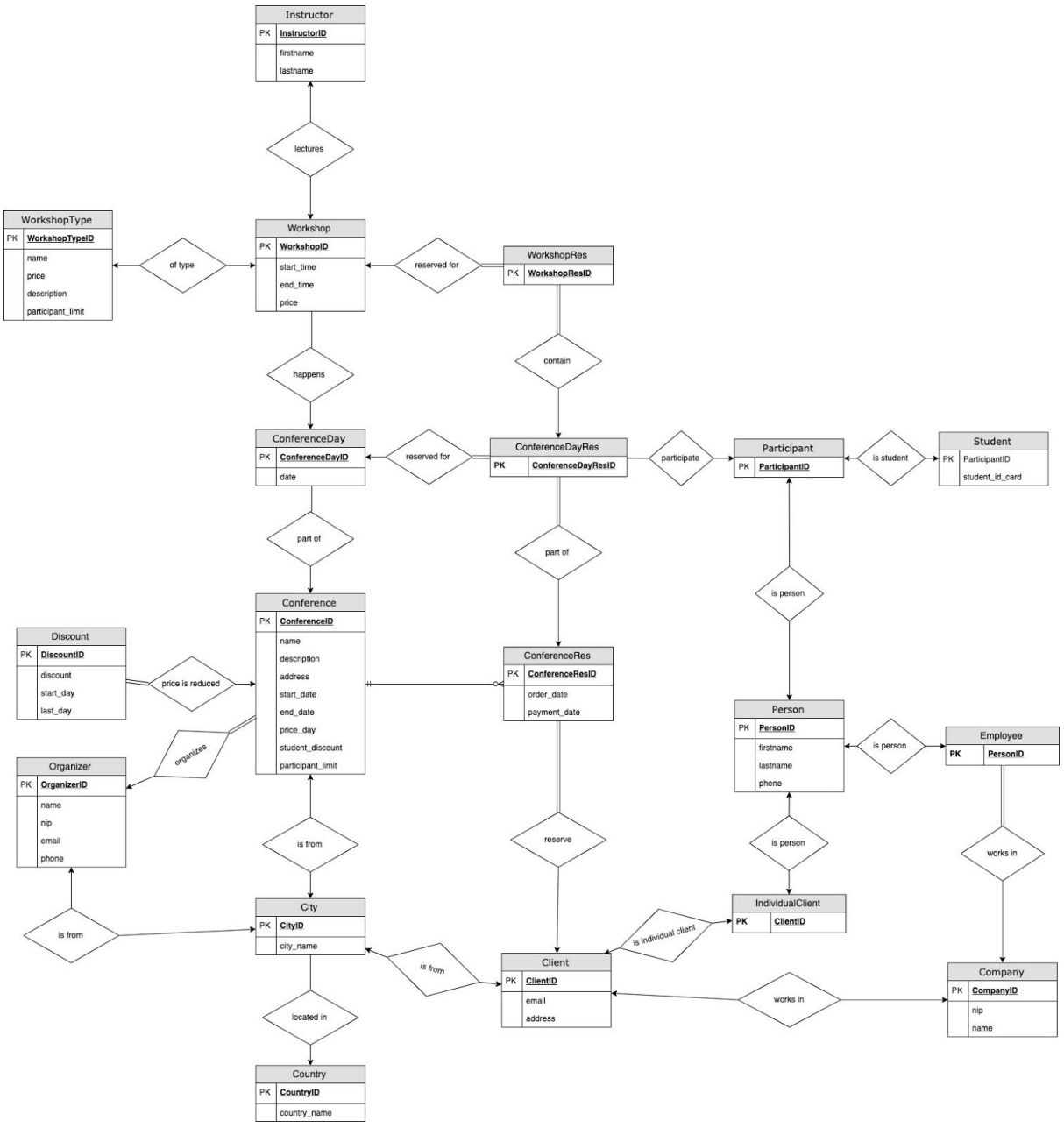
Osoba prywatna (rezerwuje miejsce na konferencje)

- **Zarezerwowanie miejsca na konferencje**
- **Zarezerwowanie miejsca na warsztaty**
    - klient musi posiadać rezerwacje na dzień konferencji w którym odbywają się warsztaty
- **Podanie danych osobowych**
    - podanie numeru legitymacji studenckiej
- **Zmiana danych osobowych**
- **Dostęp do informacji na temat konferencji/warsztatu**

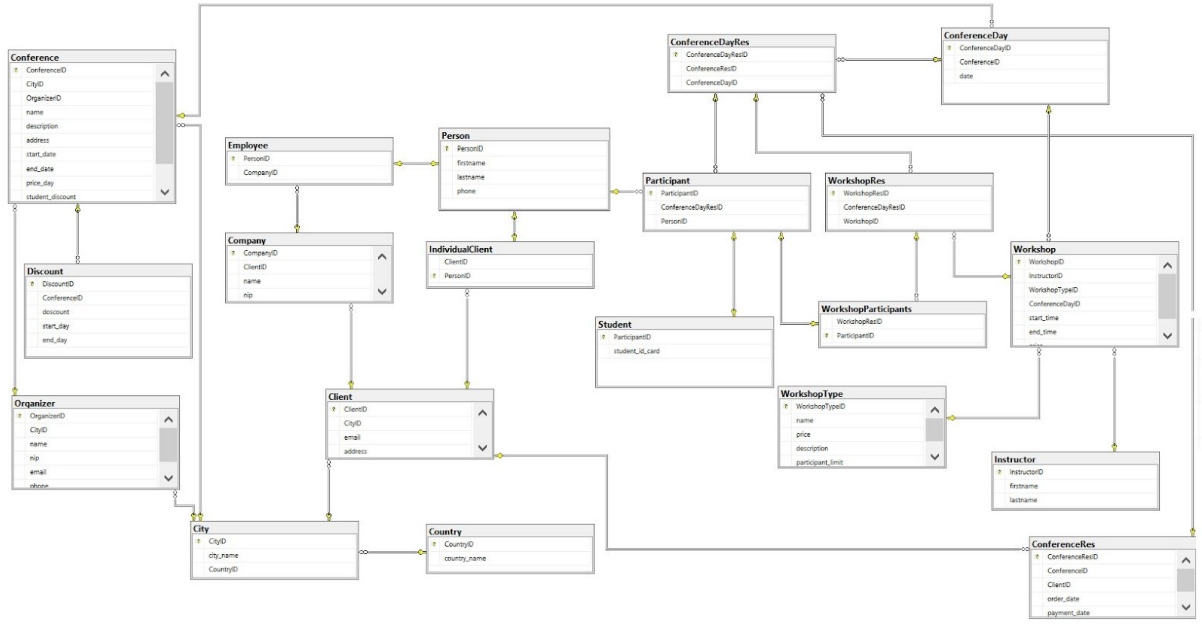
Firma (rezerwuje miejsca dla pracowników na konferencje)

- **Zarezerwowanie liczby miejsc dla pracowników na konferencje**
- **Zarezerwowanie liczby miejsc dla pracowników na warsztaty**
- **Opłacenie udziału pracowników**
- **Podanie danych osobowych pracowników**
    - wpisanie numerów legitymacji studenckich
- **Zmiana danych osobowych pracowników**
- **Sprawdzenie tematu warsztatów**

Schemat ER:


Instructor
PK InstructorID
firstname
lastname
lectures
WorkshopType
PK WorkshopTypeID
name
price
description
participant_limit
of type
Workshop
PK WorkshopID
start_time
end_time
price
reserved for
WorkshopRes
PK WorkshopResID
contain
happens
ConferenceDay
PK ConferenceDayID
date
reserved for
ConferenceDayRes
PK ConferenceDayResID
participate
Participant
PK ParticipantID
is student
Student
PK ParticipantID
student_id_card
part of
part of
is person
Conference
PK ConferenceID
name
description
address
start_date
end_date
price_day
student_discount
participant_limit
Discount
PK DiscountID
discount
start_day
last_day
price is reduced
ConferenceRes
PK ConferenceResID
order_date
payment_date
Person
PK PersonID
firstname
lastname
phone
is person
Employee
PK PersonID
works in
organizes
Organizer
PK OrganizerID
name
nip
email
phone
is from
reserve
is person
IndividualClient
PK ClientID
City
PK CityID
city_name
is from
is individual client
Client
PK ClientID
email
address
works in
Company
PK CompanyID
nip
name
is from
located in
Country
PK CountryID
country_name

Schemat logiczny:



Opisy tabel:

Tabela City: reprezentacja miast w bazie danych

Klucz główny: **CityID**
Klucz obcy: **CountryID**
nazwa miasta: **city_name**
Warunki integralności:

- city_name unikalne

```
create table City(
  CityID int not null
     constraint City_pk
        primary key nonclustered,
  city_name varchar(32) not null,
  CountryID int not null
     constraint City_CountryID
        references Country
)
```

Tabela Client: reprezentacja klientów w bazie danych

Klucz główny: **ClientID**
Klucz obcy: **CityID**
email klienta: **email**

adres klienta: **address**
Warunki integralności:

- adres email musi posiadać znak ‘@’
- adres email unikalny

```
create table Client
(
   ClientID int identity
       constraint Client_pk
           primary key,
   CityID   int
       constraint Client_CityID
           references City,
   email    varchar(64)
       constraint CK_Client_Email
           check ([email] like '%@%'),
   address  varchar(64)
)

create unique index Client_email_uindex
   on Client (email)
```

Tabela Company: reprezentacja firmy w bazie danych

Klucz główny: **CompanyID**
Klucz obcy: **ClienID**
nazwa firmy: **name**
numer NIP: **nip**
Warunki integralności:

- nip unikalny

```
create table Company
(
   CompanyID int identity
       constraint Company_pk
           primary key,
   ClientID  int         not null
       constraint Company_ClientID
           references Client,
   name      varchar(32) not null,
   nip       char(10)    not null
)

create unique index Company_nip_uindex
   on Company (nip)
```

Tabela Conference: reprezentacja konferencji w bazie danych

Klucz główny: **ConferenceID**
Klucz obcy: **CityID**
Klucz obcy: **OrganizerID**
nazwa konferencji: **name**
opis konferencji: **description**
miejsce organizacji konferencji: **address**
data rozpoczęcia konferencji: **start_date**
data zakończenia konferencji: **end_date**
cena za jeden dzień konferencji: **price_day**
zniżka dla studentów: **student_discount**
limit miejsc na konferencje: **participant_limit**
Warunki integralności:

- data zakończenia konferencji nie może być wcześniejsza od jej rozpoczęcia
- limit uczestników na konferencji musi być większy od zera
- zniżka studencka nie może być ujemna
- description default = “brak opisu”

```
create table Conference
(
   ConferenceID      int identity
       constraint Conference_pk
           primary key,
   CityID            int           not null
       constraint Conference_CityID
           references City,
   OrganizerID       int           not null
       constraint Conference_OrganizeID
           references Organizer,
   name              varchar(32)   not null,
   description       varchar(256) default 'brak opisu',
   address           varchar(64)   not null,
   start_date        date          not null,
   end_date          date          not null,
   price_day         money         not null,
   student_discount  decimal(3, 3) not null
       constraint CK_Conference_StudentDiscount
           check ([student_discount] >= 0),
   participant_limit int           not null
       constraint CK_Conference_ParticipantLimit
           check ([participant_limit] > 0),
   constraint CK_Conference_EndDate
       check ([end_date] >= [start_date])
)
```

Tabela ConferenceDay: reprezentacja poszczególnych dni konferencji w bazie danych

Klucz główny: **ConferenceDayID**
Klucz obcy: **ConferenceID**
data: **date**

```
create table ConferenceDay
(
   ConferenceDayID int identity
       constraint ConferenceDay_pk
           primary key,
   ConferenceID    int not null
       constraint ConferenceDay_ConferenceID
           references Conference,
   date            date
)
```

Tabela ConferenceDayRes: reprezentacja rezerwacji na poszczególne dni konferencji w bazie danych
Klucz główny: **ConferenceDayResID**
Klucz obcy: **ConferenceResID**
Klucz obcy: **ConferenceDayID**
Warunki integralności:

- student_tickets default = 0
- normal_tickets default = 0

```
create table ConferenceDayRes
(
   ConferenceDayResID int identity
       constraint ConferenceDayRes_pk
           primary key,
   ConferenceResID    int not null
       constraint ConferenceDayRes_ConferenceResID
           references ConferenceRes,
   ConferenceDayID    int not null
       constraint ConferenceDayRes_ConferenceDayID
           references ConferenceDay,
   normal_tickets     int default 0,
   student_tickets    int default 0
)
```

Tabela ConferenceRes: reprezentacja rezerwacji na daną konferencję w bazie danych
Klucz główny: **ConferenceResID**
Klucz obcy: **ClientID**
data złożenia rezerwacji: **order_date**
data zapłaty za rezerwację: **payment_date**

Warunki integralności:

- data zapłaty nie może być wcześniejsza niż data złożenia rezerwacji
- order_date default = GETDATE()

```
create table ConferenceRes
(
   ConferenceResID int identity
       constraint ConferenceRes_pk
           primary key,
   ClientID        int                    not null
       constraint ConferenceRes_ClientID
           references Client,
   order_date      date default getdate() not null,
   payment_date    date,
   constraint CK_ConferenceRes_PaymentDay
       check ([payment_date] >= [order_date])
)
```

Tabela Country: reprezentacja krajów w bazie danych

Klucz główny: **CountryID**
nazwa kraju: **country_name**
Warunki integralności:

- country_name unikalne

```
create table Country
(
   CountryID    int identity
       constraint Country_pk
           primary key nonclustered,
   country_name varchar(32) not null
)
```

Tabela Discount: reprezentacja progów zniżek na konferencje w bazie danych

Klucz główny: **DiscountID**
Klucz obcy: **ConferenceID**
wielkość zniżki: **discount**
data rozpoczęcia zniżki: **start_day**
data zakończenia zniżki: **end_day**
Warunki integralności:

- data zakończenia zniżki nie może być wcześniejszy od daty rozpoczęcia

```
create table Discount
(
   DiscountID   int identity
       constraint Discount_pk
```

```
            primary key,
    ConferenceID int            not null
        constraint Discount_ConferenceID
            references Conference,
    discount     decimal(3, 3) not null,
    start_day    date           not null,
    end_day      date           not null,
    constraint CK_Discount_EndDay
        check ([end_day] >= [start_day])
)
```

Tabela Employee: reprezentacja pracowników zarejestrowanych przez firmę na konferencję w bazie danych

Klucz główny: **PersonID, CompanyID**

Klucz obcy:**PersonID, CompanyID**

```
create table Employee(
  PersonID int not null
     constraint Employee_pk
        primary key nonclustered
     constraint Employee_PersonID
        references Person,
  CompanyID int
     constraint Employee_CompanyID
        references Company
)
```

Tabela IndividualClient: reprezentacja klientów indywidualnych konferencji w bazie danych

Klucz główny: **ClientID, PersonID**

Klucz obcy: **ClientID**

Klucz obcy: **PersonID**

```
create table IndividualClient(
  ClientID int
     constraint IndividualClient_ClientID
        references Client,
  PersonID int not null
     constraint IndividualClient_pk
        primary key nonclustered
     constraint IndividualClient_PersonID
        references Person
)
```

Tabela Instructor: reprezentacja instruktorów prowadzących warsztaty w bazie danych

Klucz główny: **InstructorID**
imię instruktora: **firstname**
nazwisko instruktora: **lastname**

```
create table Instructor
(
   InstructorID int identity
       constraint Instructor_pk
           primary key,
   firstname    varchar(32) not null,
   lastname     varchar(32) not null
)
```

Tabela Organizer: reprezentacja organizatorów konferencji w bazie danych

Klucz główny: **OrganizerID**
Klucz obcy: **CityID**
nazwa firmy: **name**
numer NIP: **nip**
adres email firmy: **email**
numer kontaktowy firmy: **phone**
Warunki integralności:

- adres email musi zawierać znak '@'
- numer telefonu musi się składać z samych cyfr
- nip unikalny
- email unikalny

```
create table Organizer
(
   OrganizerID int identity
       constraint Organizer_pk
           primary key,
   CityID      int         not null
       constraint Organizer_CityID
           references City,
   name        varchar(32) not null,
   nip         char(10)    not null,
   email       varchar(64) not null
       constraint CK_Organizer_Email
           check ([email] like '%@%'),
   phone       varchar(16) not null
       constraint CK_Organizer_phone
           check (isnumeric([phone]) = 1)
```

```
)
go

create unique index Organizer_email_uindex
   on Organizer (email)
```

Tabela Participant: reprezentacja uczestników konferencji w bazie danych

Klucz główny: **ParticipantID**
Klucz obcy:**ConferenceDayResID**
Klucz obcy: **PersonID**

```
create table Participant
(
   ParticipantID      int identity
       constraint Participant_pk
           primary key,
   ConferenceDayResID int not null
       constraint Participant_ConferenceDayResID
           references ConferenceDayRes,
   PersonID           int not null
       constraint Participant_PersonID
           references Person
)
```

Tabela Person: reprezentacja osób w bazie danych

Klucz główny: **PersonID**
imię osoby: **firstname**
nazwisko osoby: **lastname**
numer kontaktowy osoby: **phone**
Warunki integralności:

- numer telefony musi składać się z samych cyfr

```
create table Person
(
   PersonID  int identity
       constraint Person_pk
           primary key,
   firstname varchar(32),
   lastname  varchar(32),
   phone     int
       constraint CK_Person_Phone
           check (isnumeric([phone]) = 1)
)
```

Tabela Student: reprezentuje studenta w bazie danych

Klucz główny: **ParticipantID**
Klucz obcy: **ParticipantID**
numer karty studenckiej: **student_id_card**

```
create table Student(
  ParticipantID int not null
     constraint Student_pk
        primary key nonclustered
     constraint Student_ParticipantID
        references Participant,
  student_id_card varchar(32) not null
)
```

Tabela Workshop: reprezentuje instancje warsztatów w bazie danych

Klucz główny: **WorkshopID**
Klucz obcy: **InstructorID**
Klucz obcy: **WorkshopTypeID**
Klucz obcy: **ConferenceDayID**
czas rozpoczęcia warsztatu: **start_time**
czas zakończenia warsztatu: **end_time**
koszt warsztatu: **price**
Warunki integralności:

- koszt uczestnictwa w warsztacie nie może być ujemny
- godzina zakończenia warsztatu nie może być wcześniej niż rozpoczęcie

```
create table Workshop
(
   WorkshopID      int identity
       constraint Workshop_pk
           primary key,
   InstructorID    int     not null
       constraint Workshop_InstructorID
           references Instructor,
   WorkshopTypeID  int     not null
       constraint Workshop_WorkshopTypeID
           references WorkshopType,
   ConferenceDayID int     not null
       constraint Workshop_ConferenceDayID
           references ConferenceDay,
   start_time      time(0) not null,
   end_time        time(0) not null,
   price           money   not null
       constraint CK_Workshop_Price
           check ([price] >= 0),
   constraint CK_Workshop_EndTime
```

```
        check ([end_time] > [start_time])
)
```

Tabela WorkshopParticipant: reprezentuje uczestnika warsztatu
Klucz główny: **WorkshopResID, ParticipantID**
Klucz obcy: **WorkshopResID**
Klucz obcy: **ParticipantID**

```
create table WorkshopParticipants
(
   WorkshopResID int
       constraint WorkshopParticipants_WorkshopResID
           references WorkshopRes,
   ParticipantID int not null
       constraint WorkshopParticipants_pk
           primary key
       constraint WorkshopParticipants_ParticipantID
           references Participant
)
```

Tabela WorkshopRes: rezerwacje na konkretne instancje warsztatów w bazie danych.
Klucz główny: **WorkshopResID**
Klucz obcy: **ConferenceDayResID**
Klucz obcy: **WorkshopID**
Warunki integralności:

- normal_tickets default = 0
- student_tickets default = 0

```
create table WorkshopRes
(
   WorkshopResID      int identity
       constraint WorkshopRes_pk
           primary key,
   ConferenceDayResID int not null
       constraint WorkshopRes_ConferenceDayResID
           references ConferenceDayRes,
   WorkshopID         int not null
       constraint WorkshopRes_WorkshopID
           references Workshop,
   normal_tickets     int default 0,
   student_tickets    int default 0
)
```

Tabela WorkshopType: tabela reprezentuje warsztaty w bazie danych
Klucz główny: **WorkshopTypeID**
nazwa warsztatu: **name**
koszt warsztatu: **price**
opis warsztatu: **description**
limit uczestników danego warsztatu: **participant_limit**
Warunki integralności:

- koszt uczestnictwa w warsztacie nie może być ujemny
- limit uczestników musi być większy od zera
- description default = ‘brak opisu’

```
create table WorkshopType
(
   WorkshopTypeID    int identity
       constraint WorkshopType_pk
           primary key,
   name              varchar(32)                       not null,
   price             money                             not null
       constraint CK_WorkshopType_Price
           check ([price] >= 0),
   description       varchar(256) default 'brak opisu' not null,
   participant_limit int                               not null
       constraint CK_WorkshopType_ParticipantLimit
           check ([participant_limit] > 0)
)
```

Widoki:

Widok ConferenceParticipants: Wypisuje listę uczestników na każdy dzień konferencji

```
Create view ConferenceParticipants as
select C.ConferenceID,
      C.name
as [Conference Name],
      CDR.ConferenceDayID,
      firstname + ' ' + lastname
as Participant,
      CONCAT(SUBSTRING(P.firstname, 0, 5), substring(P.lastname, 0, 5))
as ID
from Person P
        inner join Participant P2 on P.PersonID = P2.PersonID
        inner join ConferenceDayRes CDR on P2.ConferenceDayResID =
CDR.ConferenceDayResID
        inner join ConferenceDay CD on CDR.ConferenceDayID =
CD.ConferenceDayID
```

```
        inner join Conference C on CD.ConferenceID = C.ConferenceID
go
```

Widok WorkshopParticipants: wypisuje wszystkich uczestników na konkretne warsztaty

```
Create view WokrshopParticipantsList as
select W.WorkshopID,
      WT.name,
      W.start_time,
      W.end_time,
      firstname + ' ' + lastname
as Participant,
      CONCAT(SUBSTRING(P.firstname, 0, 5), substring(P.lastname, 0, 5))
as ID
from Person P
        inner join Participant P2 on P.PersonID = P2.PersonID
        inner join WorkshopParticipants WP on P2.ParticipantID =
WP.ParticipantID
        inner join WorkshopRes WR on WP.WorkshopResID = WR.WorkshopResID
        inner join Workshop W on WR.WorkshopID = W.WorkshopID
        inner join WorkshopType WT on W.WorkshopTypeID =
WT.WorkshopTypeID
go
```

Widok WorkshopsInConference: pokazuje wszystkie warsztaty podczas danej konferencji

```
Create view WorkshopsInConference as
   select C.ConferenceID, W.WorkshopID, WT.name, CD.date, W.start_time,
W.end_time
   from Conference C
   inner join ConferenceDay CD on C.ConferenceID = CD.ConferenceID
   inner join Workshop W on CD.ConferenceDayID = W.ConferenceDayID
   inner join WorkshopType WT on W.WorkshopTypeID = WT.WorkshopTypeID
go
```

Widok unpaidCompanyRes: pokazuje nieopłacone firmowe rezerwacje

```
Create view unpaidCompanyRes as
select distinct CO.name as CompanyName,
      C.name  as ConferenceName,
      CR.order_date,
      C.start_date,
      C.end_date
FROM ConferenceRes CR
        inner join ConferenceDayRes CDR on CDR.ConferenceResID =
CR.ConferenceResID
```

```
        inner join ConferenceDay CD on CDR.ConferenceDayID =
CD.ConferenceDayID
        inner join Conference C on CD.ConferenceID = C.ConferenceID
        inner join Company CO on CR.ClientID = CO.ClientID
where CR.payment_date is null
 and datediff(day, CR.order_date, GETDATE()) > 0
go
```

Widok unpaidIndividualRes: pokazuje nieopłacone indywidualny rezerwacje

```
Create view unpaidIndividualRes as
SELECT distinct P.firstname,
      P.lastname,
      CR.order_date,
      C.name as ConferenceName,
      C.start_date,
      C.end_date
FROM ConferenceRes CR
        inner join ConferenceDayRes CDR on CDR.ConferenceResID =
CR.ConferenceResID
        inner join ConferenceDay CD on CDR.ConferenceDayID =
CD.ConferenceDayID
        inner join Conference C on CD.ConferenceID = C.ConferenceID
        INNER JOIN IndividualClient IC
                   ON IC.ClientID = CR.ClientID
        INNER JOIN Person P
                   ON IC.PersonID = P.PersonID
WHERE CR.payment_date IS NULL
 AND DATEDIFF(day, CR.order_date, GETDATE()) > 0
go
```

Widok unpaidResForTomorrow: nieopłacone rezerwacje które muszą zostać opłacone jutro

```
Create VIEW unpaidResForTomorrow AS
select CO.name as CompanyName, C.name as ConferanceName,
CR.ConferenceResID, CR.order_date, C.start_date, C.end_date
from ConferenceRes CR
        inner join Client CI on CR.ClientID = CI.ClientID
        inner join Company CO on CI.ClientID = CO.ClientID
        inner join ConferenceDayRes CDR on CDR.ConferenceResID =
CR.ConferenceResID
        inner join ConferenceDay CD on CDR.ConferenceDayID =
CD.ConferenceDayID
```

```
        inner join Conference C on CD.ConferenceID = C.ConferenceID
where CR.payment_date is null
 and datediff(day, CR.order_date, GETDATE()) = 6
```

Widok allIndentificators: wyświetla identyfikatory uczestników konferencji

```
Create view allIdentificators as
select firstname,
      lastname,
      CONCAT(SUBSTRING(P.firstname, 0, 5), substring(P.lastname, 0, 5))
as ID
from Person P
        inner join Participant P2 on P.PersonID = P2.PersonID
        inner join ConferenceDayRes CDR on P2.ConferenceDayResID =
CDR.ConferenceDayResID
```

Widok bestClients: wyświetla klientów którzy najczęściej rezerwują konferencje (oraz płacą)

```
CREATE VIEW bestClient AS
select CO.name,
      (select count(CR.ConferenceResID)
       from ConferenceRes CR
       where CO.ClientID = CR.ClientID
         and CR.payment_date is not null) as Conferences
from Company CO
go
```

Widok allConferenceRes: wyświetla liczbę rezerwacji na daną konferencję

```
Create view allConferenceRes as
select C.ConferenceID,
      C.name,
      sum(CDR.normal_tickets + CDR.student_tickets) as ticketsCount
FROM ConferenceRes CR
        inner join ConferenceDayRes CDR on CDR.ConferenceResID =
CR.ConferenceResID
        inner join ConferenceDay CD on CDR.ConferenceDayID =
CD.ConferenceDayID
        inner join Conference C on CD.ConferenceID = C.ConferenceID
group by C.ConferenceID, C.name
```

Widok allWorkshopRes: wyświetla liczbę rezerwacji na dany warsztat

```
Create view allWorkshopRes as
select W.WorkshopID,
      WT.name,
      sum(WR.normal_tickets + WR.student_tickets) as ticketsCount
from WorkshopType WT
        inner join Workshop W on WT.WorkshopTypeID = W.WorkshopTypeID
        inner join WorkshopRes WR on W.WorkshopID = WR.WorkshopID
group by W.WorkshopID, WT.name
```

Widok conferenceVacancy: wyświetla listę konferencji wraz z liczbą miejsc wolnych

```
CREATE VIEW conferenceVacancy AS
select C.ConferenceID,
      C.name,
      CD.date,
      (C.participant_limit - (sum(CDR.normal_tickets +
CDR.student_tickets))) as ticketsLeft
from Conference C
        inner join ConferenceDay CD on C.ConferenceID = CD.ConferenceID
        inner join ConferenceDayRes CDR on CD.ConferenceDayID =
CDR.ConferenceDayID
group by C.ConferenceID, C.name, CD.date, C.participant_limit
```

Widok workshopVacancy: wyświetla listę warsztatów wraz z liczbą miejsc wolnych

```
Create view workshopVacancy as
select W.WorkshopID,
      WT.name,
      (WT.participant_limit - sum(WR.normal_tickets +
WR.student_tickets)) as ticketsLeft
from WorkshopType WT
        inner join Workshop W on WT.WorkshopTypeID = W.WorkshopTypeID
        inner join WorkshopRes WR on W.WorkshopID = WR.WorkshopID
group by W.WorkshopID, WT.name, WT.participant_limit
```

Widok countryConferenceList: wyświetla kraje wraz z liczbą konferencji, które odbyły się w danym kraju

```
Create view countryConferenceList as
```

```
select CO.country_name,
      count(C.ConferenceID) as ConferenceCount
from City CI
inner join Country CO on CI.CountryID = CO.CountryID
inner join Conference C on CI.CityID = C.CityID
group by CO.country_name
```

Widok cityConferenceList: wyświetla miasta wraz z liczbą konferencji, które odbyły się w danym mieście

```
Create view cityConferenceList as
select CI.city_name,
      count(C.ConferenceID) as ConferenceCount
from City CI
        inner join Conference C on CI.CityID = C.CityID
group by CI.city_name
```

Widok workshopInstructors:wyświetla konferencje wraz z prowadzącymi je instruktorami

```
Create view workshopInstruktor AS
select WT.name,
      W.WorkshopID,
      (select I2.firstname + ' ' + I2.lastname from Instructor I2 where
I2.InstructorID = W.InstructorID) as Instructor
from Workshop W
        inner join Instructor I on W.InstructorID = I.InstructorID
        inner join WorkshopType WT on W.WorkshopTypeID =
WT.WorkshopTypeID
```

Procedury:

Procedura FindCountry - dodaje nowy kraj do bazy

```
Create PROCEDURE [dbo].FindCountry @countryName varchar(255),
                            @countryID int OUTPUT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN FIND_COUNTRY
           SET @countryID = (SELECT CountryID
                             FROM Country
```

```
                            WHERE country_name = @countryName)
          IF (@countryID is null)
              Begin
                  INSERT INTO Country(country_name)
                  VALUES (@countryName);
                  SET @countryID = @@IDENTITY;
              END
      COMMIT TRAN FIND_COUNTRY
   END TRY
   BEGIN CATCH
      ROLLBACK TRAN FIND_COUNTRY
      DECLARE @msg NVARCHAR(2048) = 'Bład wyszukiwania kraju:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
      THROW 52000,@msg, 1;
   END CATCH
```

Procedura FindCity - dodaje nowe miasto do bazy

```
CREATE PROCEDURE FindCity @cityName varchar(255),
                          @countryName varchar(255),
                          @cityID int OUTPUT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
      BEGIN TRAN FindCity
          SET @cityID = null
          IF ((@cityName is not null and @countryName is null)
              OR (@cityName is null and @countryName is not null))
              BEGIN;
                  THROW 52000,'Nalezy podac nazwe miasta i nazwe
kraju', 0;
              END
          IF (@cityName is not null and @countryName is not null)
              BEGIN
                  DECLARE @countryID int
                  EXEC FindCountry
                       @countryName,
                       @countryID = @countryID out
                  SET @cityID = (Select cityID
                                 From City
                                 Where CountryID = @countryID and
city_name=@cityName)
                  print(@cityID)
                  IF (@cityID is null)
```

```
                        BEGIN
                            INSERT INTO City(city_name, CountryID)
                            VALUES (@cityName, @countryID);
                            SET @cityID = @@IDENTITY;
                        END
                END
        COMMIT TRAN FindCity
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN FindCity
        DECLARE @msg NVARCHAR(2048) = 'Bład wyszukiwania miasta:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura InsertClient - wstawienie klienta do bazy (systemowe)

```
CREATE PROCEDURE InsertClient @email varchar(255),
                              @address varchar(255) = NULL,
                              @cityName varchar(255) = NULL,
                              @countryName varchar(255) = NULL,
                              @clientID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        DECLARE @cityID int
        EXEC FindCity
             @cityName,
             @countryName,
             @cityID = @cityID OUTPUT
        INSERT INTO Client(email, address, cityID)
        VALUES (@email,
                @address,
                @cityID);
        SET @clientID = @@IDENTITY
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) =
            'Bład przy dodawaniu klienta:' + CHAR(13) + CHAR(10) +
ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura InsertPerson - wstawienie osoby do bazy (systemowa)

```
CREATE PROCEDURE InsertPerson @firstname varchar(255),
                              @lastname varchar(255),
                              @phone varchar(255),
                              @personID int OUTPUT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       INSERT INTO Person(firstname, lastname, phone)
       VALUES (@firstname,
               @lastname,
               @phone);
       SET @personID = @@IDENTITY
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = 'Bład przy dodawaniu osoby:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

Procedura AddBusinessClient - rejestracja klienta firmowego

```
CREATE PROCEDURE AddBusinessClient @companyName varchar(255),
                                   @email varchar(255),
                                   @cityName varchar(255),
                                   @countryName varchar(255),
                                   @address varchar(255) = NULL,
                                   @nip char(10),
                                   @clientID int OUTPUT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddBusinessClient
           EXEC InsertClient
                @email,
                @address,
                @cityName,
                @countryName,
                @clientID = @clientID OUTPUT
           INSERT INTO Company(ClientID, name, nip)
           VALUES (@clientID,
```

```
                    @companyName,
                    @nip);
        COMMIT TRAN AddBusinessClient
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddBusinessClient
        DECLARE @msg NVARCHAR(2048) ='Błąd dodania klienta biznesowego:'
+CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddIndividualClient - rejestracja klienta indywidualnego

```
CREATE PROCEDURE AddIndividualClient @firstname varchar(255),
                                     @lastname varchar(255),
                                     @phone varchar(255),
                                     @email varchar(255),
                                     @address varchar(255) = NULL,
                                     @CityName varchar(255),
                                     @CountryName varchar(255),
                                     @clientID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN AddIndividualClient
            DECLARE @personID int
            EXEC InsertClient
                 @email,
                 @address,
                 @CityName,
                 @CountryName,
                 @clientID = @clientID OUTPUT
            EXEC InsertPerson
                 @firstname,
                 @lastname,
                 @phone,
                 @personID = @personID OUTPUT
            INSERT INTO IndividualClient (ClientID, PersonID)
            VALUES (@clientID,
                    @personID);
        COMMIT TRAN AddIndividualClient
    END TRY
    BEGIN CATCH
```

```
        ROLLBACK TRAN AddIndividualClient
        DECLARE @msg NVARCHAR(2048) = 'Bład dodania klienta
indiwidualnego:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddOrganizer - rejestracja konta organizatora

```
CREATE PROCEDURE AddOrganizer @name varchar(255),
                              @nip char(10),
                              @email varchar(255),
                              @phone varchar(255),
                              @cityName varchar(255) = NULL,
                              @countryName varchar(255) = NULL,
                              @organizerID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN AddOrganizer
            DECLARE @cityID int = NULL
            EXEC FindCity
                 @cityName,
                 @countryName,
                 @cityID = @cityID OUTPUT
            INSERT INTO Organizer(name, nip, email,phone, CityID)
            VALUES (@name,
                    @nip,
                    @email,
                    @phone,
                    @cityID);
            SET @organizerID = @@IDENTITY
        COMMIT TRAN AddOrganizer
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddOrganizer
        DECLARE @msg NVARCHAR(2048) = 'Bład przy dodawaniu organizatora:'
+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddInstructor - rejestracja konta instruktora

```
CREATE PROCEDURE AddInstructor @firstname varchar(255),
                               @lastname varchar(255),
                               @InstructorID INT OUTPUT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddInstructor
           INSERT INTO Instructor(firstname, lastname)
           VALUES (@firstname,
                   @lastname);
           SET @InstructorID = @@IDENTITY
       COMMIT TRAN AddInstructor
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN AddInstructor
       DECLARE @msg NVARCHAR(2048) = 'Bład przy dodawaniu instruktora:'
+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

Procedura AddParticipantCompany - dodaje dokładne dane uczestnika konferencji przy rezerwacji przez firmę

```
CREATE PROCEDURE AddParticipant @ConferenceDayResID int,
                                    @personID int,
                                    @studentIDCard char(10) = NULL,
                                    @ParticipantID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddParticipant
           INSERT INTO Participant(ConferenceDayResID, PersonID)
           VALUES (@ConferenceDayResID, @personID)

           SET @ParticipantID = @@IDENTITY
           IF (@studentIDCard is not null)
               BEGIN
                   INSERT INTO Student(ParticipantID, student_id_card)
                   VALUES (@ParticipantID, @studentIDCard)
               END
       COMMIT TRAN AddParticipant
   END TRY
```

```
    BEGIN CATCH
        ROLLBACK TRAN AddParticipant
        DECLARE @msg NVARCHAR(2048) = 'Nie udało się dodać uczestnika:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

Procedura AddParticipantCompany - dodaje dokładne dane uczestnika konferencji przy rezerwacji przez firmę

```
CREATE PROCEDURE AddParticipantCompany @ConferenceDayResID int,
                                       @firstname varchar(255),
                                       @lastname varchar(255),
                                       @phone varchar(255),
                                       @studentIDCard char(10) = null
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN AddParticipantCompany
            IF (@firstname is null or @lastname is null or @phone is
null)
                BEGIN
                    THROW 52000, 'Podane dane nie są wystarczające', 1;
                END
            IF (@studentIDCard is null and
([dbo].GetResDayNormal(@ConferenceDayResID)

-[dbo].GetResDayNormalAlreadyDetailed(@ConferenceDayResID)) <= 0)
                BEGIN
                    THROW 52000, 'Wszytskie miejsca normalne juz
wykorzystane', 1;
                END

            IF(@studentIDCard is null)
                BEGIN
                    DECLARE @personID int
                    declare @companyID int = (select CompanyID
                                              from ConferenceDayRes CDR
                                                       inner join
ConferenceRes CR on CDR.ConferenceResID = CR.ConferenceResID
                                                       inner join Client
```

```
C on CR.ClientID = C.ClientID
                                                        inner join Company
C2 on C.ClientID = C2.ClientID
                                                where ConferenceDayResID =
@ConferenceDayResID)

                    exec AddEmployee @companyID, @firstname, @lastname,
@phone, @personID out
                    exec AddParticipant @ConferenceDayResID, @personID,
@studentIDCard, null
                END
            ELSE
                BEGIN
                    declare @personID1 int = (select P2.PersonID
                                              from Student S
                                                       inner join
Participant P on S.ParticipantID = P.ParticipantID
                                                       inner join Person
P2 on P.PersonID = P2.PersonID
                                                       inner join
ConferenceDayRes CDR on P.ConferenceDayResID = CDR.ConferenceDayResID
                                             where S.student_id_card =
@studentIDCard
                                               and CDR.ConferenceDayResID
= @ConferenceDayResID)

                    if(@PersonID1 is null)
                        BEGIN
                            THROW 52000, 'Ta osoba nie zostala podana
przy rezerwacji firmowej', 1;
                        END

                    UPDATE Person set firstname = @firstname,lastname =
@lastname,phone = @phone
                    where PersonID = @personID1

                END

        COMMIT TRAN AddParticipantCompany
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddParticipantCompany
        DECLARE @msg NVARCHAR(2048) = 'Bład dodania informacji
inwidualnego pracownika:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
```

```
   END CATCH
END
go
```

Procedura AddParticipantWorkshop - dodaje uczestnika konferencji do konkretnego warsztatu, przy rezerwacji firmowej

```
CREATE PROCEDURE AddParticipantWorkshop @WorkshopResID int,
                                        @ParticipantID int
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddParticipantWorkshop
           DECLARE @ConferenceDayResID int = (select ConferenceDayResID
from Participant where ParticipantID = @ParticipantID)
           DECLARE @ConferenceDayResID2 int = (select ConferenceDayResID
from WorkshopRes where WorkshopResID = @WorkshopResID)
           IF (@ConferenceDayResID is null or @ConferenceDayResID2 is
null or
               (@ConferenceDayResID <> @ConferenceDayResID2))
               BEGIN
                   THROW 52000, 'Ten uczestnik nie jest przypisany do
dnia rezerwacji', 1;
               END

           DECLARE @studentIDCard varchar(256) = (select student_id_card
from Student where ParticipantID = @ParticipantID)
           IF (@studentIDCard is null and
(dbo.GetWorkshopResNormal(@WorkshopResID)

-dbo.GetWorkshopResNormalAlreadyDetailed(@WorkshopResID)) <= 0)
               BEGIN
                   THROW 52000, 'Wszytskie miejsca normalne juz
wykorzystane', 1;
               END
           IF (@studentIDCard is not null and
(dbo.GetWorkshopResStudent(@WorkshopResID)

-dbo.GetWorkshopResStudentAlreadyDetailed(@WorkshopResID)) <= 0)
               BEGIN
                   THROW 52000, 'Wszytskie miejsca studenckie juz
wykorzystane', 1;
               END
```

```
            DECLARE @workshopID INT = (SELECT WorkshopID FROM WorkshopRes
WHERE WorkshopResID = @workshopResID)
            DECLARE @ConferenceDayID INT = (SELECT ConferenceDayID FROM
Workshop WHERE WorkshopID = @workshopID)
            DECLARE @startTime time = (SELECT start_time FROM Workshop
WHERE WorkshopID = @workshopID)
            DECLARE @endTime time = (SELECT end_time FROM Workshop WHERE
WorkshopID = @workshopID)

            IF ((SELECT COUNT(WP.ParticipantID)
                 FROM Workshop W
                          INNER JOIN WorkshopRes WR
                                     ON WR.WorkshopID = W.WorkshopID
                          INNER JOIN WorkshopParticipants WP
                                     ON WP.WorkshopResID =
WR.WorkshopResID and WP.ParticipantID = @ParticipantID
                WHERE ((W.start_time <= @startTime and @startTime <=
W.end_time) or
                       (W.start_time <= @endTime and @endTime <=
W.end_time))
                  and W.ConferenceDayID = @ConferenceDayID) > 0)
                BEGIN
                    THROW 52000,'W czasie odbywania sie warsztatu
uczestnik bierze udział w innym warsztacie', 1;
                END

            INSERT INTO WorkshopParticipants(workshopresid,
participantid)
            VALUES(@workshopResID,@participantID);
        COMMIT TRAN AddParticipantWorkshop
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddParticipantWorkshop
        DECLARE @msg NVARCHAR(2048) = 'Bład dodania informacji
inwidualnego pracownika do warsztatu:' + CHAR(13) + CHAR(10) +
ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

Procedura AddConference - dodanie nowe konferencji i dni konferencji

```
CREATE PROCEDURE AddConference @cityName varchar(255),
                               @organizerID int,
                               @conferenceName varchar(255),
                               @description varchar(255) = 'brak opisu',
                               @address varchar(255),
                               @startDate date,
                               @endDate date,
                               @studentDiscount decimal(3, 3) = 0,
                               @countryName varchar(255),
                               @participantLimit int,
                               @price money,
                               @conferenceID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddConference
           IF (@startDate < GETDATE())
               BEGIN
                   THROW 52000, 'Konferencje nie moga byc tworzone w
przeszlosci', 1;
               END
           DECLARE @cityID int
           EXEC FindCity
                @cityName,
                @countryName,
                @cityID = @cityID out
           IF (@description is null)
               BEGIN
                   SET @description = 'brak opisu'
               END
           INSERT INTO Conference(CityID, OrganizerID, name,
description, address, start_date, end_date, price_day,
                                  student_discount, participant_limit)
           VALUES (@cityID,
                   @organizerID,
                   @conferenceName,
                   @description,
                   @address,
                   @startDate,
                   @endDate,
                   @price,
                   @studentDiscount,
                   @participantLimit);
           SET @conferenceID = @@IDENTITY;
```

```
            DECLARE @i date = @startDate
            WHILE @i <= @endDate
                BEGIN
                    INSERT INTO ConferenceDay(ConferenceID, Date)
                    VALUES (@conferenceID, @i)
                    SET @i = DATEADD(d, 1, @i)
                END
        COMMIT TRAN AddConference
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddConference
        DECLARE @msg NVARCHAR(2048) = 'Błąd stworzenia konferencji:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddConferencePrice - dodanie progów cenowy do konferencji

```
CREATE PROCEDURE AddConferencePrice @conferenceID int,
                                    @startDate date,
                                    @endDate date,
                                    @priceDiscount decimal(3, 3)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN AddConferencePrice
            IF ((SELECT Count(ConferenceID)
                 FROM Conference
                 WHERE ConferenceID = @conferenceID) = 0)
                BEGIN;
                    THROW 52000,'Podana konferencja nie istnieje', 1;
                END
            IF (Convert(date, getdate()) > @startDate)
                BEGIN;
                    THROW 52000, 'Progi cenowe nie mogą być dodawane
wstecz', 1;
                END
            IF (@endDate >= (SELECT start_date
                             FROM Conference
                             WHERE ConferenceID = @conferenceID))
                BEGIN
                    THROW 52000,'Progi cenowe nie mogą kończyć sie po
```

```
rozpoczęciu konferencji', 1;
                END
            IF (0 < (SELECT Count(DiscountID)
                     FROM Discount
                     WHERE ConferenceID = @conferenceID
                       and ((start_day <= @endDate and @endDate <=
end_day)
                        or (start_day <= @startDate and @startDate <=
end_day))))
                BEGIN
                    THROW 52000,'Konferencja ma juz prog cenowy
zawierajacy czesc tego okresu', 1;
                END
            IF ((SELECT min(discount)
                     FROM Discount
                     WHERE ConferenceID = @conferenceID and
                           end_day < @startDate) < @priceDiscount)
                BEGIN
                    THROW 52000,'obnizka ceny jest za wysoka w porownaniu
ze wczesniejszymi znizkami', 1;
                END
            IF ((SELECT max(discount)
                 FROM Discount
                 WHERE ConferenceID = @conferenceID
                   and start_day > @endDate) > @priceDiscount)
                BEGIN
                    THROW 52000,'obnizka ceny jest za niska w porownaniu
z pozniejszymi znizkami', 1;
                END
            INSERT INTO Discount(ConferenceID, discount, start_day,
end_day)
            VALUES (@conferenceID,
                    @priceDiscount,
                    @startDate,
                    @endDate)
        COMMIT TRAN AddConferencePrice
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddConferencePrice
        DECLARE @msg NVARCHAR(2048) = 'Bład dodania progu cenowego do
konferencji:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddRes - tworzenie rezerwacji

```
CREATE PROCEDURE AddRes @conferenceID int,
                        @clientID int,
                        @ConferenceResID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddRes
           DECLARE @date date = GETDATE()
           IF ((SELECT Count(ConferenceID)
                FROM Conference
                WHERE ConferenceID = @conferenceID) = 0)
               BEGIN;
                   THROW 52000,'Podana konferencja nie istnieje', 1;
               END
           IF ((SELECT Count(ClientID)
                FROM Client
                WHERE ClientID = @clientID) = 0)
               BEGIN;
                   THROW 52000,'Podany klient nie istnieje', 1;
               END
           IF ((SELECT Count(ConferenceResID)
                FROM ConferenceRes
                WHERE ClientID = @clientID
                  and ConferenceID = @conferenceID) > 0)
               BEGIN;
                   THROW 52000,'Podany klient już posiada rezerwacje na
dana konferencje', 1;
               END
           IF ((SELECT start_date
                FROM Conference
                WHERE ConferenceID = @conferenceID) <= @date)
               BEGIN;
                   THROW 52000,'Niestety nie mozna juz dokonywac
rezerwacji na podana konferencje, konferencja juz sie zaczela', 1;
               END
           INSERT INTO ConferenceRes(ConferenceID,ClientID,order_date)
           VALUES (@conferenceID,@clientID, @date)
           SET @ConferenceResID = @@IDENTITY
       COMMIT TRAN AddRes
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN AddRes
       DECLARE @msg NVARCHAR(2048) = 'Bład przy składaniu rezerwacji:' +
```

```
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddEmployee - dodanie pracownika firmy rezerwującej konferencje

```
CREATE PROCEDURE AddEmployee @companyID int,
                             @firstname varchar(255) = NULL,
                             @lastname varchar(255) = NULL,
                             @phone varchar(255) = NULL,
                             @personID int output
AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      BEGIN TRAN AddEmployee
          EXEC InsertPerson
               @firstname,
               @lastname,
               @phone,
               @personID = @personID OUTPUT
          INSERT INTO Employee (PersonID, CompanyID)
          VALUES (@personID, @companyID);
      COMMIT TRAN AddEmployee
  END TRY
  BEGIN CATCH
      ROLLBACK TRAN AddEmployee
      DECLARE @msg NVARCHAR(2048) = 'Nie udało się dodać pracownika:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
      THROW 52000,@msg, 1;
  END CATCH
END
```

Procedura SplitIDs - dzieli stringa na tabele

```
CREATE FUNCTION SplitIDs(@sep char(1), @list varchar(3000))
   RETURNS table
       AS
       RETURN
           (
               WITH Pieces(pn, start, stop) AS (
                   SELECT 1, 1, CHARINDEX(@sep, @list)
                   UNION ALL
```

```
                    SELECT pn + 1, stop + 1, CHARINDEX(@sep, @list, stop
+ 1)
                    FROM Pieces
                    WHERE stop > 0
                )
                SELECT pn,
                       SUBSTRING(@list, start, CASE WHEN stop > 0 THEN
stop - start ELSE 5000 END) AS s
                FROM Pieces
            )
go
```

Procedura AddResDayCompany - rezerwacja dnia konferencji dla firm

```
CREATE PROCEDURE AddResDayCompany @ConferenceResID int,
                                  @conferenceDayID int,
                                  @normalTickets int = 0,
                                  @studentTickets int = 0,
                                  @StudentIDCards varchar(1000) = null,
                                  @ConferenceDayResID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddResDayCompany
           IF((select ConferenceResID from ConferenceRes where
ConferenceResID = @ConferenceResID) is null)
               BEGIN;
                   THROW 52000, 'Rezerwacja nie istnieje', 1;
               END
           IF ((select ConferenceID from ConferenceDay where
@conferenceDayID = ConferenceDayID) is null or
               ((select ConferenceID from ConferenceDay where
@conferenceDayID = ConferenceDayID) <>
                (select ConferenceID from ConferenceRes where
ConferenceResID = @ConferenceResID)))
               BEGIN;
                   THROW 52000, 'Rezerwacja jest na inna konferencje',
1;
               END
           IF ((SELECT payment_date
                FROM ConferenceRes
                WHERE ConferenceResID = @ConferenceResID) is not null)
               BEGIN;
                   THROW 52000,'Rezerwacja została już opłacona', 1;
```

```
                END
            IF (@normalTickets + @studentTickets = 0)
                BEGIN;
                    THROW 52000,'Trzeba rezerwowac przynajmniej jedno
miejsce', 1;
                END
            IF ((SELECT count(ConferenceResID)
                 FROM ConferenceDayRes
                 WHERE ConferenceResID = @ConferenceResID
                   AND ConferenceDayID = @conferenceDayID) = 1)
                BEGIN;
                    THROW 52000,'Klient posiada już rezerwacje na dany
dzień konferencji', 1;
                END
            IF ([dbo].GetConferenceDayFree(@conferenceDayID) <
@normalTickets + @studentTickets)
                BEGIN;
                    THROW 52000,'Niestety nie ma wystarczajacej ilosci
wolnych miejsc', 1;
                END

            INSERT INTO ConferenceDayRes(ConferenceResID,
                                         ConferenceDayID,
                                         normal_tickets,
                                         student_tickets)
            VALUES (@ConferenceResID,
                    @conferenceDayID,
                    @normalTickets,
                    @studentTickets)
            SET @ConferenceDayResID = @@IDENTITY

            if @StudentIDCards is not null
               begin
                   if (select count(*) from SplitIDs(',',
@StudentIDCards)) != @studentTickets
                       begin;
                           THROW 52000,'Liczba legitymacji studenckich
jest różna od liczby biletów studenckich', 1;
                       end
                   Select *
                   Into #List
                   From SplitIDs(',', @StudentIDCards)

                   Declare @i int
                   declare @studentIDCard varchar(100)
```

```
                While (Select Count(*) From #List) > 0
                    Begin

                        Select Top 1 @i = pn, @studentIDCard = s From
#List

                        declare @personID int
                        declare @companyID int = (select CompanyID
                                                  from ConferenceRes
CR
                                                                inner join
Client C on CR.ClientID = C.ClientID
                                                                inner join
Company C2 on C.ClientID = C2.ClientID
                                                  where
ConferenceResID = @ConferenceResID)

                        exec AddEmployee @companyID, null, null, null,
@personID out
                        exec AddParticipant @ConferenceDayResID,
@personID, @studentIDCard, null

                        delete #List Where pn = @i
                    end
            end
        COMMIT TRAN AddResDayCompany
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddResDayCompany
        DECLARE @msg NVARCHAR(2048) ='Bład przy składaniu rezerwacji:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

Procedura AddResDayIndividual - dodanie rezerwacji na dzień konferencji klientów indywidualnych

```
CREATE PROCEDURE AddResDayIndividual @ConferenceResID int,
                                     @conferenceDayID int,
                                     @studentIDCard char(10) = null,
                                     @ConferenceResDayID int out
```

```
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN AddResDayIndividual
           IF ((select ConferenceResID from ConferenceRes where
ConferenceResID = @ConferenceResID) is null)
               BEGIN;
                   THROW 52000, 'Rezerwacja nie istnieje', 1;
               END
           IF ((select ConferenceResID from ConferenceDayRes
               where ConferenceResID = @ConferenceResID and
ConferenceDayID = @conferenceDayID) is not null)
               BEGIN;
                   THROW 52000, 'Taka rezerwacja już istnieje', 1;
               END
           DECLARE @personID int =
               (SELECT IC.PersonID
                FROM ConferenceRes as CR
                    JOIN IndividualClient as IC ON IC.ClientID =
CR.ClientID
                WHERE CR.ConferenceResID = @ConferenceResID)
           DECLARE @normal int = 1
           DECLARE @student int = 0
           IF (@studentIDCard is not null)
               BEGIN
                   SET @normal = 0
                   SET @student = 1
               END
           EXEC AddResDayCompany
                @ConferenceResID,
                @conferenceDayID,
                @normal,
                @student,
                @ConferenceResDayID = @ConferenceResDayID out

           INSERT INTO Participant(ConferenceDayResID, PersonID)
           VALUES (@ConferenceResDayID, @personID)
           DECLARE @participantID int = @@IDENTITY

           IF (@studentIDCard is not null)
               BEGIN
                   INSERT INTO Student(ParticipantID, student_id_card)
                   VALUES (@participantID, @studentIDCard)
               END
       COMMIT TRAN AddResDayIndividual
```

```
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddResDayIndividual
        DECLARE @msg NVARCHAR(2048) = 'Bład dodania rezerwacji
inwidualnej:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura RemoveOldRes - usuwanie nieopłaconych rezerwacji

```
CREATE PROCEDURE RemoveOldRes
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN RemoveOldRes
            DELETE
            FROM ConferenceRes
            WHERE payment_date is null and DATEDIFF(d, order_date,
GETDATE()) >= 7
        COMMIT TRAN RemoveOldRes
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN RemoveOldRes
        DECLARE @msg NVARCHAR(2048) = 'Bład usuniecia rezerwacj i:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

Procedura PayRes - opłacenie danej rezerwacji

```
CREATE PROCEDURE PayRes @ConferenceResID int
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN PayRes
            IF ((SELECT payment_date
                 FROM ConferenceRes
                 WHERE ConferenceResID = @ConferenceResID) is not null)
                BEGIN;
                    THROW 52000,'Rezerwacja jest juz oplacona',1;
                END
```

```
            UPDATE ConferenceRes
            SET payment_date = GETDATE()
            WHERE ConferenceResID = @ConferenceResID
        COMMIT TRAN PayRes
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN PayRes
        DECLARE @msg NVARCHAR(2048) = 'Bład przy placeniu za rezerwacje:'
+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddWorkshop - dodanie warsztatu do konferencji

```
CREATE PROCEDURE AddWorkshop @workshopTypeID int,
                             @InstructorID int,
                             @conferenceDayID int,
                             @startTime time(0),
                             @endTime time(0),
                             @price money,
                             @workshopID int out
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN AddWorkshop
            IF ((select date from ConferenceDay where ConferenceDayID =
@conferenceDayID) < GETDATE())
                BEGIN
                    THROW 52000,'Nie można tworzyć warsztatów w
przeszłosci', 1;
                END
             IF (@price is null)
                BEGIN
                    SET @price = (select price from WorkshopType where
WorkshopTypeID = @workshopTypeID)
                END
            INSERT INTO Workshop(InstructorID,
                                 WorkshopTypeID,
                                 ConferenceDayID,
                                 start_time,
                                 end_time,
                                 Price)
            VALUES (@InstructorID,
```

```
                    @workshopTypeID,
                    @conferenceDayID,
                    @startTime,
                    @endTime,
                    @price)
            SET @workshopID = @@IDENTITY
        COMMIT TRAN AddWorkshop
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddWorkshop
        DECLARE @msg NVARCHAR(2048) = 'Błąd dodania warsztatu do
konferencji:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura AddWorkshopType - dodanie warsztatu

```
CREATE PROCEDURE AddWorkshopType @workshopName varchar(32),
                                 @wokshopPrice money,
                                 @description varchar(256),
                                 @workshopLimit int,
                                 @workshopID int out
AS
BEGIN
    SET NOCOUNT ON;
    IF (@description is null)
        BEGIN
            SET @description = 'brak opisu'
        END
    INSERT INTO WorkshopType(name, price, description, participant_limit)
    VALUES (@workshopName,
            @wokshopPrice,
            @description,
            @workshopLimit)
    SET @workshopID = @@IDENTITY
END
```

Procedura AddResWorkshopIndividual - dodanie rezerwacji indywidualnej na warsztat

```
CREATE PROCEDURE AddResWorkshopIndividual @ConferenceDayResID int,
                                          @workshopID int,
                                          @workshopResID int out
AS
```

```
BEGIN
   BEGIN TRY
       BEGIN TRAN AddResWorkshopIndividual
           DECLARE @participantID int =
               (SELECT ParticipantID
                FROM Participant
                WHERE ConferenceDayResID = @ConferenceDayResID)
           DECLARE @normal int =
dbo.GetResDayNormal(@ConferenceDayResID)
           DECLARE @student int =
dbo.GetResDayStudent(@ConferenceDayResID)
           EXEC AddResWorkshopCompany
                @ConferenceDayResID,
                @workshopID,
                @normal,
                @student,
                @workshopResID = @workshopResID out
           INSERT INTO WorkshopParticipants(workshopresid,
participantid)
           VALUES(@workshopResID,@participantID);
       COMMIT TRAN AddResWorkshopIndividual
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN AddResWorkshopIndividual
       DECLARE @msg NVARCHAR(2048) = 'Bład dodania rezerwacji
indywidualnej:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

Procedura AddResWorkshopCompany - dodanie rezerwacji jako firma na warsztat

```
CREATE PROCEDURE AddResWorkshopCompany @ConferenceDayResID int,
                                       @workshopID int,
                                       @normalTickets int = 0,
                                       @studentTickets int = 0,
                                       @workshopResID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN AddResWorkshopCompany
           IF (@normalTickets + @studentTickets = 0)
               BEGIN;
```

```
                    THROW 52000,'Trzeba rezerwowac przynajmniej jedno
miejsce', 1;
                END
            IF ((SELECT payment_date
                 FROM ConferenceRes
                          INNER JOIN ConferenceDayRes ON
ConferenceRes.ConferenceResID = ConferenceDayRes.ConferenceDayResID
                 WHERE ConferenceDayRes.ConferenceDayID =
@ConferenceDayResID) is not null)
                BEGIN;
                    THROW 52000,'Rezerwacja została już opłacona', 1;
                END
            IF ((SELECT count(ConferenceDayResID)
                 FROM WorkshopRes
                 WHERE ConferenceDayResID = @ConferenceDayResID
                   and @workshopID = WorkshopID) > 0)
                BEGIN;
                    THROW 52000,'Klient posiada już rezerwacje na dany
warsztat', 1;
                END
            IF ((SELECT ConferenceDayID
                 FROM Workshop
                 WHERE WorkshopID = @workshopID) <>
                (SELECT ConferenceDayID
                 FROM ConferenceDayRes
                 WHERE ConferenceDayResID = @ConferenceDayResID))
                BEGIN;
                    THROW 52000,'Rezerwacja oraz warsztat odwołują sie do
innego dnia konferencji', 1;
                END
            IF ([dbo].GetWorkshopFree(@workshopID) < @normalTickets +
@studentTickets)
                BEGIN;
                    THROW 52000,'Niestety nie ma wystarczajacej ilosci
wolnych miejsc', 1;
                END
            IF ([dbo].GetResDayNormal(@ConferenceDayResID) <
@normalTickets
                or [dbo].GetResDayStudent(@ConferenceDayResID) <
@studentTickets)
                BEGIN;
                    THROW 52000,'Nie mozna rezerwowacj wiekszej ilosci
miejsc niz w rezerwacji na dzien konferencji', 1;
                END
            INSERT INTO WorkshopRes(ConferenceDayResID,
```

```
                                    WorkshopID,
                                    normal_tickets,
                                    student_tickets)
            VALUES (@ConferenceDayResID,
                    @workshopID,
                    @normalTickets,
                    @studentTickets)
            SET @workshopResID = @@IDENTITY
        COMMIT TRAN AddResWorkshopCompany
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddResWorkshopCompany
        DECLARE @msg NVARCHAR(2048) = 'Błąd przy dodawaniu rezerwacji:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

Procedura RemoveConferenceRes - usuwa podaną nie opłaconą rezerwację

```
CREATE PROCEDURE RemoveConferenceRes @ConferenceResID int
AS
BEGIN
  BEGIN TRY
      BEGIN TRAN RemoveConferenceRes
          IF ((SELECT payment_date
               FROM ConferenceRes
               WHERE ConferenceResID = @ConferenceResID) is not null)
              BEGIN;
                   THROW 52000,'Rezerwacja jest opłacona',1;
              END
          IF ((SELECT COUNT(ConferenceResID )
               FROM ConferenceRes
               WHERE ConferenceResID  = @ConferenceResID) < 1)
              BEGIN;
                   THROW 52000,'Nie znaleziono rezerwacji',1;
              END
          DELETE
          FROM ConferenceRes
          WHERE ConferenceResID = @ConferenceResID
      COMMIT TRAN RemoveConferenceRes
  END TRY
  BEGIN CATCH
      ROLLBACK TRAN RemoveConferenceRes
```

```
      DECLARE @msg NVARCHAR(2048) = 'Nie udało się usunąć rezerwacji:' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
      THROW 52000,@msg, 1;
  END CATCH
END
go
```

Funkcje:

Funkcja GetConferenceDayWithConferenceID - zwraca ID konferencji do której należy dzień konferencji

```
CREATE FUNCTION GetConferenceDayWithConferenceID(@ConferenceDayID int)
   RETURNS int
AS
BEGIN
   RETURN (SELECT ConferenceID
           FROM ConferenceDay
           WHERE ConferenceDayID = @ConferenceDayID)
END
```

Funkcja GetConferenceDiscount - zwraca zniżkę dla konkretnej daty zakupu

```
CREATE FUNCTION [dbo].GetConferenceDiscount(
   @conferenceID int,
   @order_date date
)
  RETURNS decimal(3, 2)
AS
BEGIN
  RETURN ISNULL((SELECT discount
                 FROM Discount
                 WHERE ConferenceID = @conferenceID
                   AND start_day <= @order_date
                   AND @order_date <= end_day), 0)
END
```

Funkcja GetResCost - zwraca cenę całej rezerwacji

```
CREATE FUNCTION [dbo].[GetResCost](
  @ConferenceResID int
)
  RETURNS money
```

```
AS

BEGIN

  DECLARE @normalprice MONEY =
      ISNULL((SELECT C.price_day * (1 -
dbo.GetConferenceDiscount(C.ConferenceID, CR.order_date))
       FROM ConferenceRes as CR
                JOIN Conference as C ON C.ConferenceID = CR.ConferenceID
       WHERE CR.ConferenceResID = @ConferenceResID),0)

  DECLARE @student_discount decimal(3, 3) =
      (SELECT C.student_discount
       FROM ConferenceRes as CR
                JOIN Conference as C ON C.ConferenceID = CR.ConferenceID
       WHERE CR.ConferenceResID = @ConferenceResID)

  DECLARE @reservationCost MONEY =
      (Select Sum(normal_tickets) * @normalprice +
              Sum(student_tickets) * @normalprice * (1 -
@student_discount)
       From ConferenceDayRes
       WHERE ConferenceResID = @ConferenceResID)

  DECLARE @workshopCost MONEY =
      (select SUM(WR.normal_tickets * W.price) + SUM(WR.student_tickets
* W.price * (1 - @student_discount))
       from Workshop W
                join WorkshopRes WR on W.WorkshopID = WR.WorkshopID
                join ConferenceDayRes CDR on WR.ConferenceDayResID =
CDR.ConferenceDayResID
                join ConferenceRes CR on CDR.ConferenceResID =
CR.ConferenceResID
       where CR.ConferenceResID = @ConferenceResID)
  RETURN (ISNULL(@reservationCost, 0) + ISNULL(@workshopCost, 0))
END
```

Funkcja GetResDayNormal - zwraca ilość zarezerwowanych miejsc normalnych na dany dzien konferencji w danej rezerwacji

```
CREATE FUNCTION GetResDayNormal(
   @ConferenceDayResID int
)
   RETURNS int
AS
```

```
BEGIN
   RETURN ISNULL((SELECT normal_tickets
                  FROM ConferenceDayRes
                  WHERE ConferenceDayResID = @ConferenceDayResID), 0)
END
```

Funkcja GetResDayStudent - zwraca ilość zarezerowanych miejsc studenckich na dzień konferencji

```
CREATE FUNCTION GetResDayStudent(
   @ConferenceDayResID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT student_tickets
                  FROM ConferenceDayRes
                  WHERE ConferenceDayResID = @ConferenceDayResID), 0)
END
```

Funkcja GetConferenceDayFree - zwraca ilość wolnych miejsc na dzień konferencji

```
CREATE FUNCTION GetConferenceDayFree(
   @conferenceDayID int
)
   RETURNS int
AS
BEGIN
   RETURN (SELECT C.participant_limit
           FROM Conference C
                    inner join ConferenceDay CD on C.ConferenceID =
CD.ConferenceID
           Where CD.ConferenceDayID = @conferenceDayID) -
dbo.GetConferenceDayTaken(@conferenceDayID)
END
```

Funkcja GetConferenceDayTaken - zwraca ile jest zajętych miejsc na dzień konferencji

```
CREATE FUNCTION GetConferenceDayTaken(
   @conferenceDayID int
)
   RETURNS int
```

```
AS
BEGIN
   RETURN ISNULL((SELECT SUM(normal_tickets) + SUM(student_tickets)
                  FROM ConferenceDayRes
                  Where ConferenceDayID = @conferenceDayID), 0)
END
```

Funkcja GetWorkshopFree - zwraca ilość wolnych miejsc na dany warsztat

```
CREATE FUNCTION GetWorkshopFree(
   @workshopID INT
)
   RETURNS INT
AS
BEGIN
   RETURN ((select participant_limit
            from Workshop
                     inner join WorkshopType WT on
Workshop.WorkshopTypeID = WT.WorkshopTypeID
            where WorkshopID = @workshopID) -
           dbo.GetWorkshopTaken(@workshopID))
END
```

Funkcja GetWorkshopTaken - zwraca ile jest zajętych miejsc na warsztat

```
CREATE FUNCTION GetWorkshopTaken(
   @workshopID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT SUM(normal_tickets) + SUM(student_tickets)
                  FROM WorkshopRes
                  Where WorkshopID = @workshopID), 0)
END
```

Funkcja GetWorkshopResStudent - zwraca ilość zarezerwowanych miejsc studenckich na dany warsztat

```
CREATE FUNCTION GetWorkshopResStudent(
   @workshopReservationID int
)
```

```
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT student_tickets
                  FROM WorkshopRes
                  WHERE WorkshopResID = @workshopReservationID), 0)
END
```

funkcja GetWorkshopResNormal - zwraca ilość zarezerwowanych miejsc normalnych na warsztat w rezerwacji

```
CREATE FUNCTION GetWorkshopResNormal(
   @workshopResID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT normal_tickets
                  FROM WorkshopRes
                  WHERE WorkshopResID = @workshopResID), 0)
END
```

Funkcja GetResDayNormalAlreadyDetailed - zwraca ilość osób bez zniżki studenckiej już zapisanych w bazie razem z danymi

```
CREATE FUNCTION GetResDayNormalAlreadyDetailed(
   @ConferenceDayResID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((select count(*)
                  from ConferenceDayRes CDR
                           inner join Participant P on
P.ConferenceDayResID = CDR.ConferenceDayResID
                           left outer join Student S on P.ParticipantID
= S.ParticipantID
                  where CDR.ConferenceDayResID = @ConferenceDayResID
                    and student_id_card is null
                 ), 0)
END
```

Funkcja GetResDayStudentIAlreadyDetailed - zwraca ilość osób z zniżką studencką już zapisanych w bazie razem z danymi

```
CREATE FUNCTION GetResDayStudentAlreadyDetailed(
   @ConferenceDayResID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((select count(*)
                  from ConferenceDayRes CDR
                           inner join Participant P on
P.ConferenceDayResID = CDR.ConferenceDayResID
                           left outer join Student S on P.ParticipantID
= S.ParticipantID
                  where CDR.ConferenceDayResID = @ConferenceDayResID
                    and student_id_card is not null
                 ), 0)
END
```

Funkcja GetWorkshopResNormalAlreadyDetailed - zwraca ilość osób bez zniżki studenckiej już przypisanych do konkretnego warsztatu

```
CREATE FUNCTION GetWorkshopResNormalAlreadyDetailed(
   @WorkshopResID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((select count(*)
                  from WorkshopRes WR
                           inner join WorkshopParticipants WP on
WR.WorkshopResID = WP.WorkshopResID
                           inner join Participant P on WP.ParticipantID
= P.ParticipantID
                           left outer join Student S on P.ParticipantID
= S.ParticipantID
                  where WR.WorkshopResID = @WorkshopResID
                    and student_id_card is null), 0)
END
```

Funkcja GetWorkshopResStudentAlreadyDetailed - zwraca ilość osób ze zniżką studencką już przypisanych do konkretnego warsztatu

```
CREATE FUNCTION GetWorkshopResStudentAlreadyDetailed(
   @WorkshopResID int
)
```

```
   RETURNS int
AS
BEGIN
   RETURN ISNULL((select count(*)
                  from WorkshopRes WR
                           inner join WorkshopParticipants WP on
WR.WorkshopResID = WP.WorkshopResID
                           inner join Participant P on WP.ParticipantID
= P.ParticipantID
                           left outer join Student S on P.ParticipantID
= S.ParticipantID
                  where WR.WorkshopResID = @WorkshopResID
                    and student_id_card is not null), 0)
END
```

Triggery:

Trigger DeleteConference - trigger przy usuwaniu konferencji

```
create trigger DeleteConference
   on Conference
   for delete
   as
begin
   set nocount on
   declare @ConferenceID int = (select ConferenceID from deleted)
   delete
   from ConferenceDay
   where ConferenceID = @ConferenceID
   delete
   from Discount
   where ConferenceID = @ConferenceID
   delete
   from ConferenceRes
   where ConferenceID = @ConferenceID
end
```

Trigger DeleteConferenceDay - trigger przy usuwaniu dnia konferencji

```
create trigger DeleteConferenceDay
   on ConferenceDay
   for delete
   as
```

```
begin
   set nocount on
   declare @DayID int = (select ConferenceDayID from deleted)
   delete
   from ConferenceDayRes
   where ConferenceDayID = @DayID
   delete
   from Workshop
   where ConferenceDayID = @DayID
end
```

Trigger DeleteConferenceDayRes - trigger przy usuwaniu rezerwacji na konferencje

```
create trigger DeleteConferenceDayRes
   on ConferenceDayRes
   for delete
   as
begin
   set nocount on
   declare @ConferenceDayResID int = (select ConferenceDayResID from
deleted)
   delete
   from Participant
   where ConferenceDayResID = @ConferenceDayResID
end
```

Trigger DeleteWorkshop - trigger przy usuwaniu warsztatu

```
create trigger DeleteWorkshop
   on Workshop
   for delete
   as
begin
   set nocount on
   declare @WorkshopID int= (select WorkshopID from deleted)
   delete
   from WorkshopRes
   where WorkshopID = @WorkshopID
end
```

Trigger DeleteWorkshopRes - trigger przy usuwaniu rezerwacji na warsztat

```
create trigger DeleteWorkshopRes
   on WorkshopRes
   for delete
   as
begin
   set nocount on
   declare @WorkshopResID int= (select WorkshopResID from deleted)
   delete
   from WorkshopParticipants
   where WorkshopResID = @WorkshopResID
end
```

Trigger DeleteParticipant - trigger przy usuwaniu uczestnika konferencji

```
create trigger DeleteParticipant
   on Participant
   for delete
   as
begin
   set nocount on
   delete
   from Person
   where PersonID = (select PersonID from deleted)
end
```

Indeksy:

Indeks `City_city_name_index`: ustawienie indexu na city_name w tabeli City

```
create index City_city_name_index
   on City (city_name)
go
```

Indeks `Country_country_name_index`: ustawienie indexu na country_name w tabeli Country

```
create index Country_country_name_index
  on Country (country_name)
go
```

Indeks ConferenceDay_ConferenceID_index: ustawienie indexu na ConferenceID w tabeli ConferenceDay

```
create index ConferenceDay_ConferenceID_index
```

```
  on ConferenceDay (ConferenceID)
go
```

Indeks Discount_ConferenceID_index: ustawienie indexu na ConferenceID w tabeli Discount

```
create index Discount_ConferenceID_index
  on Discount (ConferenceID)
go
```

Indeks WorkshopRes_WorkshopID_ConferenceDayResID_index: ustawienie indexów na WorkshopID oraz ConferenceDayResID w tabeli WorkshopRes

```
create index WorkshopRes_WorkshopID_ConferenceDayResID_index
  on WorkshopRes (WorkshopID, ConferenceDayResID)
go
```

Indeks Employee_CompanyID_index: ustawienie indexu na CompanyID w tabeli Employee

```
create index Employee_CompanyID_index
  on Employee (CompanyID)
go
```

Indeks ConferenceRes_ClientID_index: ustawienie indexu na ClientID w tabeli ConferenceRes

```
create index ConferenceRes_ClientID_index
  on ConferenceRes (ClientID)
go
```

Indeks ConferenceDayRes_ConferenceResID_ConferenceDayID_index: ustawienie indexów na ConferenceResID oraz ConferenceDayID w tabeli ConferenceD**ayRes**

```
create index ConferenceDayRes_ConferenceResID_ConferenceDayID_index
  on ConferenceDayRes (ConferenceResID, ConferenceDayID)
go
```

Indeks Student_ParticipantID_index: ustawienie indexu naParticipantID w tabeli Student

```
create index Student_ParticipantID_index
  on Student (ParticipantID)
go
```

Generator

Dane w naszej bazie danych wygenerowaliśmy przy pomocy generatora SQL Data Generator firmy RedGate, podczas generowania zapewniliśmy usuwanie danych, które są sprzeczne z warunkami integralnościowymi.