Projekt na bazy danych

Autorzy :

Sebastian Kuśnierz, Mateusz Surjak

1. Aktorzy systemu:

- administrator systemu,
- przedstawiciel firmy rezerwującej miejsca na konferencje,
- osoba prywatna rezerwująca miejsca na konferencje,
- pracownik firmy rezerwującej miejsca na konferencje
- organizator konferencji (firma)
- osoba odpowiedzialna za zarządzanie konferencjami
- system

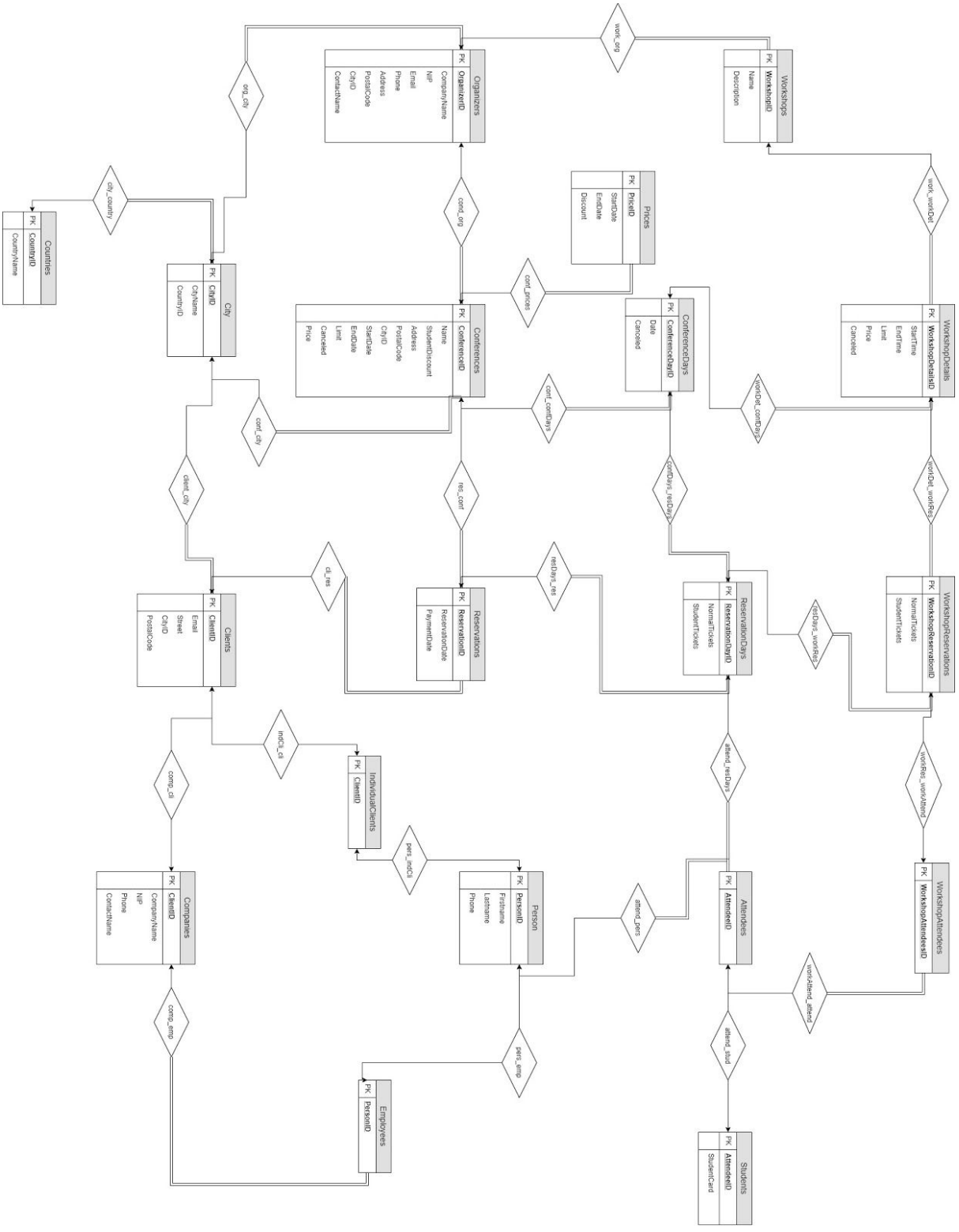
2. Funkcje systemu

- przedstawiciel firmy rezerwującej miejsce na konferencje,
    - zarezerwowanie określonej liczby miejsc na konferencji
    - wprowadzenie użytkowników którzy mają wziąć udział w konferencji
    - opłacenie udziału w konferencji
    - zmiana liczby uczestników
    - zapisanie pracowników na warsztaty
        - określenie liczby pracowników którym przysługuje zniżka
- osoba prywatna rezerwująca miejsca na konferencje,
    - rejestracja w systemie
    - zarezerwowanie miejsca na konferencję
    - opłacenie udziału w konferencji
    - podanie danych do rezerwacji
        - określenie czy będzie korzystał ze zniżki studenckiej
    - sprawdzenie bieżącej oferty konferencji
    - zapisanie się na warsztaty
    - sprawdzenie liczby wolnych miejsc na dany dzień konferencji
    - sprawdzenie liczby wolnych miejsc na dany warsztat
    - wyświetlenie opłaconych rezerwacji
    - wyświetlenie nieopłaconych rezerwacji
    - zmiana danych osobowych
- pracownik firmy rezerwującej miejsca na konferencje
    - potwierdzenie uczestnictwa w konferencji poprzez wprowadzenie danych do konferencji których nie podał jego przełożony
- organizator konferencji (firma)
    - utworzenie konta organizatora
    - utworzenie nowych konferencji
        - podanie nazwy konferencji
        - określenie czasu trwania konferencji
        - zdefiniowanie opłaty za jeden dzień konferencji
        - określenie liczby uczestników

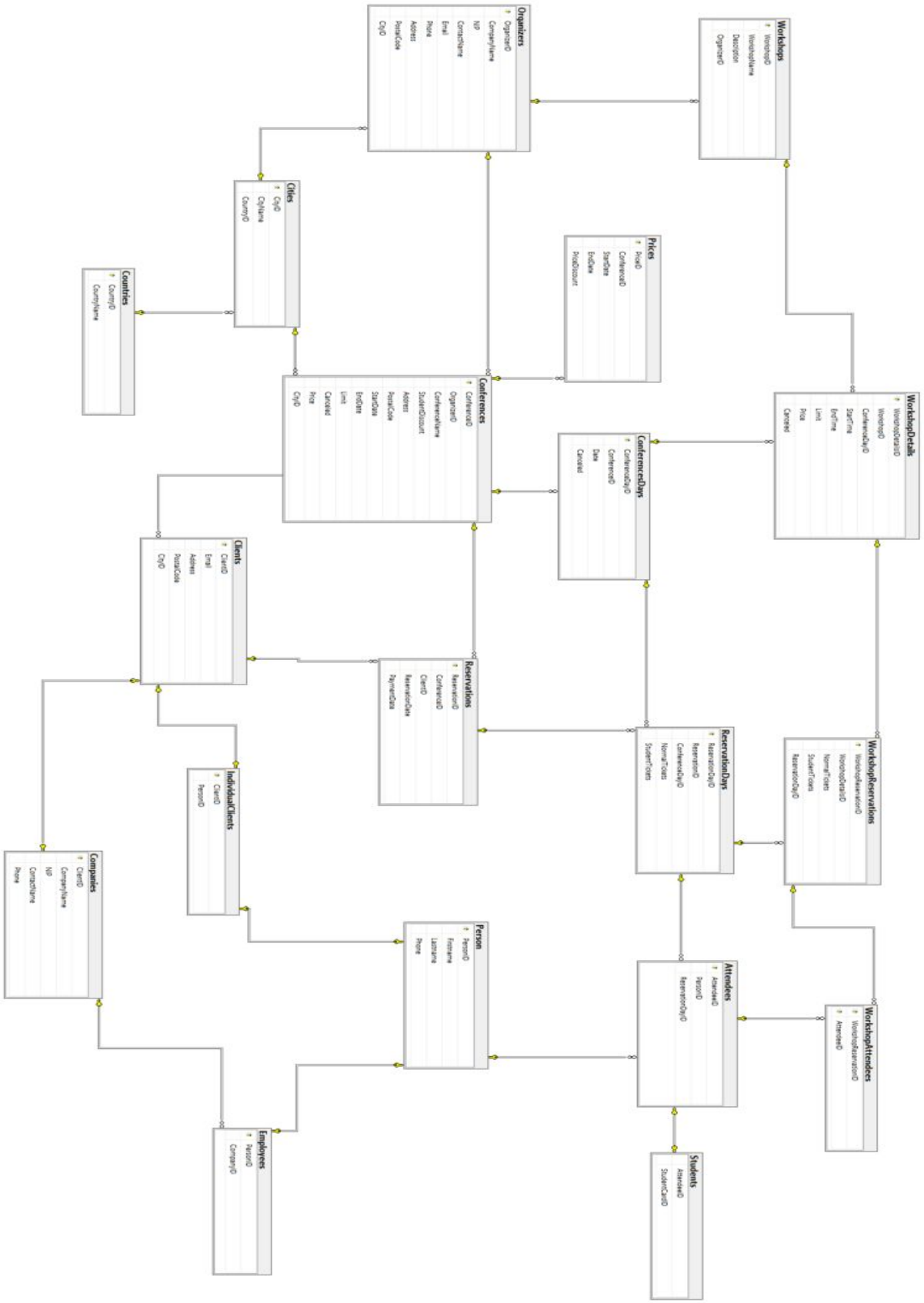
        - określenie miejsca konferencji
        - określenie zniżki studenckiej
    - dodanie nowego warsztatu na konferencji
        - wprowadzenie daty warsztatu
        - wprowadzenie liczby miejsc
    - zmiana danych dotyczących konferencji/warsztatu
        - maksymalna liczba uczestników
        - miejsce konferencji
        - dni konferencji
        - opłata za dzień konferencji
        - zniżka studencka
    - odwołanie konferencji
    - dostęp do informacji o wpłatach uczestników
    - dodanie dnia konferencji
    - ustalenie progów cenowych
    - dodanie progu cenowego konferencji
    - anulowanie warsztatu
    - wyświetlenie listy organizowanych konferencji
    - sprawdzenie ilości zarezerwowanych i wolnych miejsc na dany dzień konferencji i dany warsztat
    - wyświetlenie warsztatów dla danej konferencji

- administrator systemu
    - całkowita ingerencja w system
- osoba odpowiedzialna za zarządzanie konferencjami
    - odwołanie konferencji
    - zatwierdzanie konferencji
    - wygenerowanie listy uczestników na dany dzień konferencji
    - wygenerowanie listy uczestników na dany warsztat
    - wprowadzenie informacji o opłaceniu rezerwacji
    - wygenerowanie liczby wolnych miejsc na dany dzień konferencji
    - wygenerowanie liczby wolnych miejsc na dany warsztat
    - wygenerowanie listy rezerwacji nieopłaconych
    - wygenerowanie listy opłaconych rezerwacji
    - wygenerowanie listy przypisanych do siebie konferencji i warsztatów
    - wygenerowanie identyfikatorów dla uczestników konferencji
- system
    - naliczanie zniżek za konferencję
    - sprawdzenie czy dany uczestnik może się zapisać na konferencję
    - wyliczenie kosztu rezerwacji
    - sprawdzenie ilości zajętych/wolnych miejsc na dany dzień konferencji
    - sprawdzenie ilości zajętych/wolnych miejsc na dany warsztat
    - anulowanie rezerwacji które nie zostały opłacone na czas

3. Diagram ER


work_org
org_city
city_country
cond_org
cond_prices
cont_city
cont_contDays
workDet_contDays
client_city
res_cont
contDays_resDays
workDet_workRes
cli_res
resDays_res
resDays_workRes
indCl_cli
comp_cli
attend_resDays
workRes_workAttend
pers_indCl
attend_pers
workAttend_attend
comp_emp
pers_emp
attend_stud
Countries
CountryID
CountryName
City
CityID
CityName
CountryID
Organizers
OrganizerID
CompanyName
NIP
Email
Phone
Address
PostalCode
CityID
ContactName
Prices
PriceID
StartDate
EndDate
Discount
Workshops
WorkshopID
Name
Description
Conferences
ConferenceID
Name
StudentDiscount
Address
PostalCode
CityID
StartDate
EndDate
Limit
Canceled
Price
ConferenceDays
ConferenceDayID
Date
Canceled
WorkshopDetails
WorkshopDetailsID
StartTime
EndTime
Limit
Price
Canceled
Clients
ClientID
Email
Street
CityID
PostalCode
Reservations
ReservationID
ReservationDate
PaymentDate
ReservationDays
ReservationDayID
NormalTickets
StudentTickets
WorkshopReservations
WorkshopReservationID
NormalTickets
StudentTickets
IndividualClients
ClientID
Companies
ClientID
CompanyName
NIP
Phone
ContactName
Person
PersonID
Firstname
Lastname
Phone
AttendeeDays
AttendeeID
WorkshopAttendees
WorkshopAttendeesID
Employees
PersonID
Students
AttendeeID
StudentCard

4. Schemat Bazy Danych


Organizers
OrganizerID
CompanyName
NIP
ContactName
Email
Phone
Address
PostalCode
CityID
Workshops
WorkshopID
WorkshopName
Description
OrganizerID
Cities
CityID
CityName
CountryID
Countries
CountryID
CountryName
Prices
PriceID
ConferenceID
StartDate
EndDate
PriceDiscount
Conferences
ConferenceID
OrganizerID
ConferenceName
StudentDiscount
Address
PostalCode
StartDate
EndDate
Limit
Canceled
Price
CityID
WorkshopDetails
WorkshopDetailsID
WorkshopID
ConferenceDayID
StartTime
EndTime
Limit
Price
Canceled
ConferenceDays
ConferenceDayID
ConferenceID
Date
Canceled
Clients
ClientID
Email
Address
PostalCode
CityID
Reservations
ReservationID
ConferenceID
ClientID
ReservationDate
PaymentDate
IndividualClients
ClientID
PersonID
Companies
ClientID
CompanyName
NIP
ContactName
Phone
Person
PersonID
Firstname
Lastname
Phone
ReservationDays
ReservationDayID
ReservationID
ConferenceDayID
NormalTickets
StudentTickets
WorkshopReservations
WorkshopReservationID
WorkshopDetailsID
NormalTickets
StudentTickets
ReservationDayID
Attendees
AttendeeID
PersonID
ReservationDayID
WorkshopAttendees
WorkshopReservationID
AttendeeID
Employees
PersonID
CompanyID
Students
AttendeeID
StudentCardID

5. Tabele

a. **ATTENDEES -** tabela reprezentująca uczestników konferencji.
AttendeeID - Klucz główny
PersonID - Klucz obcy
ReservationDayID - Klucz obcy

```
CREATE TABLE [dbo].[Attendees](
	[AttendeeID] [int] IDENTITY(1,1) NOT NULL,
	[PersonID] [int] NOT NULL,
	[ReservationDayID] [int] NOT NULL,
 CONSTRAINT [Attendees_pk] PRIMARY KEY CLUSTERED
(
	[AttendeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Attendees]  WITH CHECK ADD  CONSTRAINT
[Attendees_Person_PersonID_fk] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[Attendees] CHECK CONSTRAINT [Attendees_Person_PersonID_fk]
GO

ALTER TABLE [dbo].[Attendees]  WITH CHECK ADD  CONSTRAINT
[Attendees_ReservationDays_ReservationDayID_fk] FOREIGN KEY([ReservationDayID])
REFERENCES [dbo].[ReservationDays] ([ReservationDayID]) on delete cascade
GO

ALTER TABLE [dbo].[Attendees] CHECK CONSTRAINT
[Attendees_ReservationDays_ReservationDayID_fk]
GO
```

b. **CITIES - tabela reprezentująca miasta**
CityID - Klucz główny
City Name - Nazwa miasta

```
CREATE TABLE [dbo].[Cities](
	[CityID] [int] IDENTITY(1,1) NOT NULL,
	[CityName] [varchar](255) NOT NULL,
	[CountryID] [int] NOT NULL,
 CONSTRAINT [Cities_pk] PRIMARY KEY CLUSTERED
(
	[CityID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT
[Cities_Countries_CountryID_fk] FOREIGN KEY([CountryID])
REFERENCES [dbo].[Countries] ([CountryID])
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [Cities_Countries_CountryID_fk]
GO
```

c. **CLIENTS - tabela reprezentująca klientów**
ClientID - Klucz główny
Email - Adres email, musi posiadać @
Address - Adres klienta
PostalCode - Kod pocztowy, musi być postaci “XXXXX” lub “XX-XXX”
CityID - klucz obcy

```
CREATE TABLE [dbo].[Clients](
	[ClientID] [int] IDENTITY(1,1) NOT NULL,
	[Email] [varchar](255) NOT NULL,
	[Address] [varchar](255) NULL,
	[PostalCode] [varchar](255) NULL,
	[CityID] [int] NOT NULL,
 CONSTRAINT [Clients_pk] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY],
 CONSTRAINT [Clients_Email_UQ] UNIQUE NONCLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT
[Clients_Cities_CityID_fk] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [Clients_Cities_CityID_fk]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT [Clients_Email_CK] CHECK
(([Email] like '%@%'))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [Clients_Email_CK]
GO

ALTER TABLE [dbo].[Clients]  WITH CHECK ADD  CONSTRAINT [Clients_PostalCode_CK]
CHECK  (([PostalCode] like '[0-9][0-9]-[0-9][0-9][0-9]' OR [PostalCode] like
'[0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [Clients_PostalCode_CK]
GO
```

d. **COMPANIES - tabela reprezentująca firmy**
ClientId - Klucz główny
CompanyName - Nazwa firmy
NIP - Numer Identyfikacji Podatkowej firmy, musi być unikatowy
ContactName - Dane pracownika (reprezentanta firmy) do kontaktu
Phone - numer telefonu do firmy, musi być podany w postaci numerycznej

```
CREATE TABLE [dbo].[Companies](
	[ClientID] [int] NOT NULL,
	[CompanyName] [varchar](255) NOT NULL,
	[NIP] [char](10) NOT NULL,
	[ContactName] [varchar](255) NOT NULL,
	[Phone] [varchar](255) NOT NULL,
 CONSTRAINT [Companies_pk] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY],
 CONSTRAINT [UQ_Companies_NIP] UNIQUE NONCLUSTERED
(
	[NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT
[Companies_Clients_ClientID_fk] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [Companies_Clients_ClientID_fk]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT [CK_Companies_Phone]
CHECK  ((isnumeric([Phone])=(1)))
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [CK_Companies_Phone]
GO
```

e. **CONFERENCES - tabela reprezentująca konferencje**
ConferenceID - Klucz główny
OrganizerID - Klucz obcy
ConferenceName - Nazwa konferencji
StudentDiscount - Wartości zniżki studenckiej wyrażona z przedziału [0,1]
Address - Adres konferencji
PostalCode - Kod pocztowy, musi być postaci “XXXXX” lub “XX-XXX”
StartDate - Data rozpoczęcia się konferencji DD/MM/YYYY
EndDate - Data zakończenia się konferencji DD/MM/YYYY, musi następować po dacie rozpoczęcia się konferencji
Limit - Limit miejsc na konferencję, musi być większy od zera
Canceled - flaga do określenia czy konferencja się odbędzie, (0 - konferencja odbędzie się, 1 - konferencja nie odbędzie się)
Price - cena bazowa za konferencję
CityID - klucz obcy

```
CREATE TABLE [dbo].[Conferences](
	[ConferenceID] [int] IDENTITY(1,1) NOT NULL,
	[OrganizerID] [int] NOT NULL,
	[ConferenceName] [varchar](255) NOT NULL,
	[StudentDiscount] [float] NOT NULL,
	[Address] [varchar](255) NOT NULL,
	[PostalCode] [varchar](255) NOT NULL,
	[StartDate] [date] NOT NULL,
	[EndDate] [date] NOT NULL,
	[Limit] [int] NOT NULL,
	[Canceled] [bit] NOT NULL,
	[Price] [money] NOT NULL,
	[CityID] [int] NOT NULL,
 CONSTRAINT [Conferences_pk] PRIMARY KEY CLUSTERED
(
	[ConferenceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
```

```
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Conferences] ADD  CONSTRAINT [DF_Conferences_Canceled]
DEFAULT ((0)) FOR [Canceled]
GO

ALTER TABLE [dbo].[Conferences] ADD  CONSTRAINT [DF_Conferences_Price]  DEFAULT
((0)) FOR [Price]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[Conferences_Cities_CityID_fk] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [Conferences_Cities_CityID_fk]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[Conferences_Organizers_OrganizerID_fk] FOREIGN KEY([OrganizerID])
REFERENCES [dbo].[Organizers] ([OrganizerID])
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT
[Conferences_Organizers_OrganizerID_fk]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[CK_Conferences_Dates] CHECK  (([StartDate]<=[EndDate]))
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [CK_Conferences_Dates]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[CK_Conferences_Limit] CHECK  (([Limit]>(0)))
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [CK_Conferences_Limit]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[CK_Conferences_StuDis] CHECK  (([StudentDiscount]>=(0) and [StudentDiscount] <=
1 ))
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [CK_Conferences_StuDis]
GO
```

```
ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[CK_Conferences_PostalCode] CHECK  (([PostalCode] like
'[0-9][0-9]-[0-9][0-9][0-9]' OR [PostalCode] like '[0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [CK_Conferences_PostalCode]
GO
```

Trigger : AddConferencesDays - dodaje każdy dzień konferencji

```
CREATE TRIGGER Add_Conference_Days
   ON Conferences
   AFTER INSERT
   AS
BEGIN
   DECLARE @ConID AS int
   declare @startDate DATE
   declare @endDate DATE


           Select @ConID = ConferenceID, @startDate = StartDate, @endDate =
EndDate From inserted


           DECLARE @i date = @startDate
           WHILE @i <= @endDate
               BEGIN
                   INSERT INTO [ConferencesDays](ConferenceID, Date)
                   VALUES (@ConID, @i)
                   SET @i = DATEADD(d, 1, @i)

               END
END
go
```

f. **CONFERENCES_DAYS - tabele reprezentująca dni konferencji**
ConferenceDayID - Klucz główny
ConferenceID - Klucz obcy oznaczający daną konferencje
Date - Data w której odbywa się dana konferencja DD/MM/YYYY
Canceled - flaga do określenia czy konferencja się odbędzie, (0 - konferencja odbędzie się w danym dniu, 1 - konferencja nie odbędzie się w danym dniu)

```
CREATE TABLE [dbo].[ConferencesDays](
       [ConferenceDayID] [int] IDENTITY(1,1) NOT NULL,
       [ConferenceID] [int] NOT NULL,
       [Date] [date] NOT NULL,
       [Canceled] [bit] NOT NULL,
 CONSTRAINT [ConferencesDays_pk] PRIMARY KEY CLUSTERED
```

```
(
	[ConferenceDayID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ConferencesDays] ADD  CONSTRAINT
[DF_ConferencesDays_Canceled]  DEFAULT ((0)) FOR [Canceled]
GO

ALTER TABLE [dbo].[ConferencesDays]  WITH CHECK ADD  CONSTRAINT
[ConferencesDays_Conferences_ConferenceID_fk] FOREIGN KEY([ConferenceID])
REFERENCES [dbo].[Conferences] ([ConferenceID])
GO

ALTER TABLE [dbo].[ConferencesDays] CHECK CONSTRAINT
[ConferencesDays_Conferences_ConferenceID_fk]
GO
```

g. **COUNTRIES - tabela reprezentująca kraje**
CountryID - Klucz główny
CountyName - Nazwa kraju, musi być unikalna

```
CREATE TABLE [dbo].[Countries](
	[CountryID] [int] IDENTITY(1,1) NOT NULL,
	[CountryName] [varchar](255) NOT NULL,
 CONSTRAINT [Countries_pk] PRIMARY KEY CLUSTERED
(
	[CountryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY],
 CONSTRAINT [UQ_Countries_CountryName] UNIQUE NONCLUSTERED
(
	[CountryName] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO
```

h. **EMPLOYEES - tabela reprezentująca pracowników firm**
PersonID - Klucz obcy, przechowuje informacje o danym pracowniku
CompanyID - Klucz obcy, przechowuje informacje do jakiej firmy należy pracownik

```
CREATE TABLE [dbo].[Employees](
	[PersonID] [int] NOT NULL,
	[CompanyID] [int] NOT NULL,
 CONSTRAINT [Employees_pk] PRIMARY KEY CLUSTERED
```

```
(
	[PersonID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[Employees_Companies_ClientID_fk] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([ClientID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employees_Companies_ClientID_fk]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[Employees_Person_PersonID_fk] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [Employees_Person_PersonID_fk]
GO
```

i. **INDIVIDUAL_CLIENTS - tabela reprezentująca klientów indywidualnych**
ClientID - Klucz obcy
PersonID - Klucz obcy, przechowuje informacje o danym kliencie indywidualnym

```
CREATE TABLE [dbo].[IndividualClients](
	[ClientID] [int] NOT NULL,
	[PersonID] [int] NOT NULL,
 CONSTRAINT [IndividualClients_pk] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY],
 CONSTRAINT [UQ_IndividualClients_PersonID] UNIQUE NONCLUSTERED
(
	[PersonID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndividualClients]  WITH CHECK ADD  CONSTRAINT
[IndividualClients_Clients_ClientID_fk] FOREIGN KEY([ClientID])
```

```
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT
[IndividualClients_Clients_ClientID_fk]
GO

ALTER TABLE [dbo].[IndividualClients]  WITH CHECK ADD  CONSTRAINT
[IndividualClients_Person_PersonID_fk] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[IndividualClients] CHECK CONSTRAINT
[IndividualClients_Person_PersonID_fk]
GO
```

j. **ORGANIZERS - tabela reprezentująca organizatorów konferencji**
OrganizerID - Klucz główny
CompanyName - Nazwa firmy
NIP - Numer Identyfikacji Podatkowej firmy, musi być unikatowy
Email - Adres email, musi posiadać @
Phone - Numer telefonu, musi być podany w postaci liczbowej
Address - Adres firmy
PostalCode - Kod pocztowy, musi być postaci “XXXXX” lub “XX-XXX”
CityID - Klucz obcy

```
CREATE TABLE [dbo].[Organizers](
	[OrganizerID] [int] IDENTITY(1,1) NOT NULL,
	[CompanyName] [varchar](255) NOT NULL,
	[NIP] [char](10) NOT NULL,
	[ContactName] [varchar](255) NOT NULL,
	[Email] [varchar](255) NOT NULL,
	[Phone] [varchar](255) NOT NULL,
	[Address] [varchar](255) NULL,
	[PostalCode] [varchar](255) NULL,
	[CityID] [int] NOT NULL,
 CONSTRAINT [Organizers_pk] PRIMARY KEY CLUSTERED
(
	[OrganizerID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY],
 CONSTRAINT [UQ_Organizers_Email] UNIQUE NONCLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY],
 CONSTRAINT [UQ_Organizers_NIP] UNIQUE NONCLUSTERED
(
```

```
      [NIP] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Organizers]  WITH CHECK ADD  CONSTRAINT
[Organizers_Cities_CityID_fk] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO

ALTER TABLE [dbo].[Organizers] CHECK CONSTRAINT [Organizers_Cities_CityID_fk]
GO

ALTER TABLE [dbo].[Organizers]  WITH CHECK ADD  CONSTRAINT [CK_Organizers_Email]
CHECK  (([Email] like '%@%'))
GO

ALTER TABLE [dbo].[Organizers] CHECK CONSTRAINT [CK_Organizers_Email]
GO

ALTER TABLE [dbo].[Organizers]  WITH CHECK ADD  CONSTRAINT [CK_Organizers_Phone]
CHECK  ((isnumeric([Phone])=(1)))
GO

ALTER TABLE [dbo].[Organizers] CHECK CONSTRAINT [CK_Organizers_Phone]
GO

ALTER TABLE [dbo].[Organizers]  WITH CHECK ADD  CONSTRAINT
[CK_Organizers_PostalCode] CHECK  (([PostalCode] like
'[0-9][0-9]-[0-9][0-9][0-9]' OR [PostalCode] like '[0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Organizers] CHECK CONSTRAINT [CK_Organizers_PostalCode]
GO
```

k. **PERSON - tabela reprezentująca osoby**
   PersonID - Klucz główny
   FirstName - Imię danej osoby
   LastName - Nazwisko danej osoby
   Phone - Numer telefonu, musi być podany w postaci liczbowej

```
CREATE TABLE [dbo].[Person](
	[PersonID] [int] IDENTITY(1,1) NOT NULL,
	[Firstname] [varchar](255) NULL,
	[Lastname] [varchar](255) NULL,
	[Phone] [varchar](255) NULL,
 CONSTRAINT [Person_pk] PRIMARY KEY CLUSTERED
(
	[PersonID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Person]  WITH CHECK ADD  CONSTRAINT [CK_Person_Phone] CHECK
((isnumeric([Phone])=(1)))
GO

ALTER TABLE [dbo].[Person] CHECK CONSTRAINT [CK_Person_Phone]
GO
```

I. **PRICES - tabela reprezentująca zniżki na konferencje**
PriceID - Klucz główny
ConferenceID - Klucz obcy, wskazujący na daną konferencję
StartTime - Data rozpoczęcia obowiązywania zniżki
EndTime - Data zakończenia obowiązywania zniżk, musi następować po dacie rozpoczęcia StartTime
PriceDiscount - zniżka, jej wartość musi zawierać się pomiędzy [0,1]

```
CREATE TABLE [dbo].[Prices](
	[PriceID] [int] IDENTITY(1,1) NOT NULL,
	[ConferenceID] [int] NOT NULL,
	[StartDate] [date] NOT NULL,
	[EndDate] [date] NOT NULL,
	[PriceDiscount] [float] NOT NULL,
 CONSTRAINT [Prices_pk] PRIMARY KEY CLUSTERED
(
	[PriceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Prices]  WITH CHECK ADD  CONSTRAINT
[Prices_Conferences_ConferenceID_fk] FOREIGN KEY([ConferenceID])
REFERENCES [dbo].[Conferences] ([ConferenceID])
GO

ALTER TABLE [dbo].[Prices] CHECK CONSTRAINT [Prices_Conferences_ConferenceID_fk]
GO
```

```
ALTER TABLE [dbo].[Prices]  WITH CHECK ADD  CONSTRAINT [CK_Prices_Dates] CHECK
(([StartDate]<=[EndDate]))
GO

ALTER TABLE [dbo].[Prices] CHECK CONSTRAINT [CK_Prices_Dates]
GO

ALTER TABLE [dbo].[Prices]  WITH CHECK ADD  CONSTRAINT [CK_Prices_Dis] CHECK
(([PriceDiscount]>=(0) and [PriceDiscount] <= 1 ))
GO

ALTER TABLE [dbo].[Prices] CHECK CONSTRAINT [CK_Prices_Dis]
GO
```

m. **RESERVATION_DAYS -** tabela reprezentująca rezerwacje na dany dzień konferencji
ReservationDayID - Klucz główny
ReservationID - Klucz obcy
ConferenceDayID - Klucz obcy, oznaczający dzień konferencji, który rezerwujemy
NormalTickets - ilości biletów normalnych, musi być większa lub równa zero
StudentTickets - ilości biletów studenckich, musi być większa lub równa zero

```
CREATE TABLE [dbo].[ReservationDays](
	[ReservationDayID] [int] IDENTITY(1,1) NOT NULL,
	[ReservationID] [int] NOT NULL,
	[ConferenceDayID] [int] NOT NULL,
	[NormalTickets] [int] NOT NULL,
	[StudentTickets] [int] NOT NULL,
 CONSTRAINT [ReservationDays_pk] PRIMARY KEY CLUSTERED
(
	[ReservationDayID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ReservationDays]  WITH CHECK ADD  CONSTRAINT
[ReservationDays_ConferencesDays_ConferenceDayID_fk] FOREIGN
KEY([ConferenceDayID])
REFERENCES [dbo].[ConferencesDays] ([ConferenceDayID])
GO

ALTER TABLE [dbo].[ReservationDays] CHECK CONSTRAINT
[ReservationDays_ConferencesDays_ConferenceDayID_fk]
GO

ALTER TABLE [dbo].[ReservationDays]  WITH CHECK ADD  CONSTRAINT
[ReservationDays_Reservations_ReservationID_fk] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID]) on delete cascade
GO
```

```
ALTER TABLE [dbo].[ReservationDays] CHECK CONSTRAINT
[ReservationDays_Reservations_ReservationID_fk]
GO

ALTER TABLE [dbo].[ReservationDays]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationDays_NormalTickets] CHECK  (([NormalTickets]>=(0)))
GO

ALTER TABLE [dbo].[ReservationDays] CHECK CONSTRAINT
[CK_ReservationDays_NormalTickets]
GO

ALTER TABLE [dbo].[ReservationDays]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationDays_StudentTickets] CHECK  (([StudentTickets]>=(0)))
GO

ALTER TABLE [dbo].[ReservationDays] CHECK CONSTRAINT
[CK_ReservationDays_StudentTickets]
GO
```

n. **RESERVATIONS -** tabela reprezentująca rezerwacje na daną konferencję
ReservationID - Klucz główny
ConferenceID - Klucz obcy, oznaczający konferencje, której dotyczy rezerwacja
ClientID - Klucz obcy, oznaczający klienta, który składa rezerwacje
ReservationDate - Data rezerwacji DD/MM/YYYY
PaymentDate - Data opłacenia rezerwacji DD/MM/YYYY, musi następować po dacie rezerwacji

```
CREATE TABLE [dbo].[Reservations](
	[ReservationID] [int] IDENTITY(1,1) NOT NULL,
	[ConferenceID] [int] NOT NULL,
	[ClientID] [int] NOT NULL,
	[ReservationDate] [date] NOT NULL,
	[PaymentDate] [date] NULL,
 CONSTRAINT [Reservations_pk] PRIMARY KEY CLUSTERED
(
	[ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[Reservations_Clients_ClientID_fk] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
```

```
[Reservations_Clients_ClientID_fk]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[Reservations_Conferences_ConferenceID_fk] FOREIGN KEY([ConferenceID])
REFERENCES [dbo].[Conferences] ([ConferenceID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT
[Reservations_Conferences_ConferenceID_fk]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[CK_Reservations_Dates] CHECK
((datediff(day,[PaymentDate],[ReservationDate])<=(0)))
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [CK_Reservations_Dates]
GO
```

o. **STUDENTS -** tabela reprezentująca studentów w bazie
AttendeeID - Klucz obcy, musi być unikatowy
StudentCardID - Numer karty studenckiej

```
CREATE TABLE [dbo].[Students](
	[AttendeeID] [int] NOT NULL,
	[StudentCardID] [char](10) NOT NULL,
 CONSTRAINT [Student_ParticipantID_UQ] UNIQUE NONCLUSTERED
(
	[AttendeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Students]  WITH CHECK ADD  CONSTRAINT
[Students_Attendees_AttendeeID_fk] FOREIGN KEY([AttendeeID])
REFERENCES [dbo].[Attendees] ([AttendeeID]) on delete cascade
GO

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [Students_Attendees_AttendeeID_fk]
GO
```

p. **WORKSHOP_ATTENDEES**
WorkshopReservationID - Razem z AtendeeID tworzą klucz główny, ID rezerwacji warsztatu, w którym dany uczestnik będzie brał udział
AtendeeID - Razem z WorkshopReservationID tworzą klucz główny, ID uczestnika który bierze udział w warsztacie

```
CREATE TABLE [dbo].[WorkshopAttendees](
	[WorkshopReservationID] [int] NOT NULL,
	[AttendeeID] [int] NOT NULL,
 CONSTRAINT [WorkshopAttendees_pk] PRIMARY KEY CLUSTERED
(
	[WorkshopReservationID] ASC,
	[AttendeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WorkshopAttendees]  WITH CHECK ADD  CONSTRAINT
[WorkshopAttendees_Attendees_AttendeeID_fk] FOREIGN KEY([AttendeeID])
REFERENCES [dbo].[Attendees] ([AttendeeID]) on delete cascade
GO

ALTER TABLE [dbo].[WorkshopAttendees] CHECK CONSTRAINT
[WorkshopAttendees_Attendees_AttendeeID_fk]
GO

ALTER TABLE [dbo].[WorkshopAttendees]  WITH CHECK ADD  CONSTRAINT
[WorkshopAttendees_WorkshopReservations_WorkshopReservationID_fk] FOREIGN
KEY([WorkshopReservationID])
REFERENCES [dbo].[WorkshopReservations] ([WorkshopReservationID])
GO

ALTER TABLE [dbo].[WorkshopAttendees] CHECK CONSTRAINT
[WorkshopAttendees_WorkshopReservations_WorkshopReservationID_fk]
GO
```

q. **WORKSHOP_DETAILS**
WorkshopDetailsID - Klucz główny
WorkshopID - Klucz obcy
ConferenceDayID - Klucz obcy
StartTime - Godzina rozpoczęcia warsztatu
EndTime - Godzina zakończenia warsztatu
Limit - Limit miejsc, musi być większy od zera
Price - Cena za udział w konferencji, musi być większa od zera
Canceled - Flaga mówiąca nam czy dany warsztat się odbędzie (0 - odbędzie się, 1 - nie odbędzie się)

```
CREATE TABLE [dbo].[WorkshopDetails](
	[WorkshopDetailsID] [int] IDENTITY(1,1) NOT NULL,
	[WorkshopID] [int] NOT NULL,
	[ConferenceDayID] [int] NOT NULL,
	[StartTime] [time](7) NOT NULL,
```

```
	[EndTime] [time](7) NOT NULL,
	[Limit] [int] NOT NULL,
	[Price] [money] NOT NULL,
	[Canceled] [bit] NOT NULL,
 CONSTRAINT [WorkshopDetails_pk] PRIMARY KEY CLUSTERED
(
	[WorkshopDetailsID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WorkshopDetails]  WITH CHECK ADD  CONSTRAINT
[WorkshopDetails_ConferencesDays_ConferenceDayID_fk] FOREIGN
KEY([ConferenceDayID])
REFERENCES [dbo].[ConferencesDays] ([ConferenceDayID])
GO

ALTER TABLE [dbo].[WorkshopDetails] CHECK CONSTRAINT
[WorkshopDetails_ConferencesDays_ConferenceDayID_fk]
GO

ALTER TABLE [dbo].[WorkshopDetails]  WITH CHECK ADD  CONSTRAINT
[WorkshopDetails_Workshops_WorkshopID_fk] FOREIGN KEY([WorkshopID])
REFERENCES [dbo].[Workshops] ([WorkshopID])
GO

ALTER TABLE [dbo].[WorkshopDetails] CHECK CONSTRAINT
[WorkshopDetails_Workshops_WorkshopID_fk]
GO

ALTER TABLE [dbo].[WorkshopDetails]  WITH CHECK ADD  CONSTRAINT
[CK_WorkshopDetails_Dates] CHECK  (([StartTime]<[EndTime]))
GO

ALTER TABLE [dbo].[WorkshopDetails] CHECK CONSTRAINT [CK_WorkshopDetails_Dates]
GO

ALTER TABLE [dbo].[WorkshopDetails]  WITH CHECK ADD  CONSTRAINT
[CK_WorkshopDetails_Limit] CHECK  (([Limit]>(0)))
GO

ALTER TABLE [dbo].[WorkshopDetails] CHECK CONSTRAINT [CK_WorkshopDetails_Limit]
GO

ALTER TABLE [dbo].[WorkshopDetails]  WITH CHECK ADD  CONSTRAINT
[CK_WorkshopDetails_Price] CHECK  (([Price]>=(0)))
GO
```

```
ALTER TABLE [dbo].[WorkshopDetails] CHECK CONSTRAINT [CK_WorkshopDetails_Price]
GO
```

r. **WORKSHOP_RESERVATIONS**
WorkshopReservationID - Klucz główny
WorkshopDetailsID - Klucz obcy, wskazujący na warsztat którego dotyczy rezerwacja
NormalTickets - Ilość biletów normalnych, musi być większa lub równa zero
StudentTickets - Ilość biletów studenckich, musi być większa lub równa zero
ReservationDayID - Klucz obcy, wskazuje rezerwacje na dany dzień konferencji

```
CREATE TABLE [dbo].[WorkshopReservations](
	[WorkshopReservationID] [int] IDENTITY(1,1) NOT NULL,
	[WorkshopDetailsID] [int] NOT NULL,
	[NormalTickets] [int] NULL,
	[StudentTickets] [int] NULL,
	[ReservationDayID] [int] NOT NULL,
 CONSTRAINT [WorkshopReservations_pk] PRIMARY KEY CLUSTERED
(
	[WorkshopReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WorkshopReservations]  WITH CHECK ADD  CONSTRAINT
[WorkshopReservations_ReservationDays_ReservationDayID_fk] FOREIGN
KEY([ReservationDayID])
REFERENCES [dbo].[ReservationDays] ([ReservationDayID]) on delete cascade
GO

ALTER TABLE [dbo].[WorkshopReservations] CHECK CONSTRAINT
[WorkshopReservations_ReservationDays_ReservationDayID_fk]
GO

ALTER TABLE [dbo].[WorkshopReservations]  WITH CHECK ADD  CONSTRAINT
[WorkshopReservations_WorkshopDetails_WorkshopDetailsID_fk] FOREIGN
KEY([WorkshopDetailsID])
REFERENCES [dbo].[WorkshopDetails] ([WorkshopDetailsID])
GO

ALTER TABLE [dbo].[WorkshopReservations] CHECK CONSTRAINT
[WorkshopReservations_WorkshopDetails_WorkshopDetailsID_fk]
GO

ALTER TABLE [dbo].[WorkshopReservations]  WITH CHECK ADD  CONSTRAINT
[CK_WorkshopReservations_Normal] CHECK  (([NormalTickets]>=(0)))
GO

ALTER TABLE [dbo].[WorkshopReservations] CHECK CONSTRAINT
```

```
[CK_WorkshopReservations_Normal]
GO

ALTER TABLE [dbo].[WorkshopReservations]  WITH CHECK ADD  CONSTRAINT
[CK_WorkshopReservations_Student] CHECK  (([StudentTickets]>=(0)))
GO

ALTER TABLE [dbo].[WorkshopReservations] CHECK CONSTRAINT
[CK_WorkshopReservations_Student]
GO
```

s. **WORKSHOPS**
WorkshopID - Klucz główny
WorkshopName - Nazwa warsztatu
Description - Opis warsztatu
OrganizerID - Klucz obcy, wskazuje na organizatora, który organizuje warsztat

```
CREATE TABLE [dbo].[Workshops](
	[WorkshopID] [int] IDENTITY(1,1) NOT NULL,
	[WorkshopName] [varchar](255) NOT NULL,
	[Description] [varchar](255) NOT NULL,
	[OrganizerID] [int] NOT NULL,
 CONSTRAINT [Workshops_pk] PRIMARY KEY CLUSTERED
(
	[WorkshopID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Workshops]  WITH CHECK ADD  CONSTRAINT
[Workshops_Organizers_OrganizerID_fk] FOREIGN KEY([OrganizerID])
REFERENCES [dbo].[Organizers] ([OrganizerID])
GO

ALTER TABLE [dbo].[Workshops] CHECK CONSTRAINT
[Workshops_Organizers_OrganizerID_fk]
GO
```

6. Widoki

a. Canceled_Conferences - widok reprezentujący anulowane konferencje

```
CREATE VIEW CanceledConferences AS
```

```
SELECT C.ConferenceName,
       O.CompanyName,
       CIT.CityName,
       C2.CountryName,
       C.StartDate,
       C.EndDate
FROM Conferences C
         INNER JOIN Organizers O
                    ON C.OrganizerID = O.OrganizerID
         INNER JOIN Cities CIT
                    ON C.CityID = CIT.CityID
         INNER JOIN Countries C2
                    on CIT.CountryID = C2.CountryID
WHERE C.Canceled = 1
go
```

b. Canceled_Workshops - widok reprezentujący anulowane warsztaty

```
CREATE VIEW CanceledWorkshops AS
SELECT O.CompanyName,
       W.WorkshopName,
       W.Description,
       WD.StartTime,
       WD.EndTime
FROM WorkshopDetails WD
         INNER JOIN Workshops W
                    ON WD.WorkshopID = W.WorkshopID
         INNER JOIN Organizers O
                    ON W.OrganizerID = O.OrganizerID
WHERE WD.Canceled = 1
go
```

c. Count_Of_Paid_Reservations_From_Companies_Clients - widok reprezentujący liczbę zapłaconych rezerwacji dla każdej firmy

```
CREATE VIEW CountOfPaidReservationsFromCompaniesClients AS
SELECT Companies.CompanyName,
       (SELECT COUNT(ReservationID)
        FROM Reservations
        WHERE Clients.ClientID = Reservations.ClientID
          AND PaymentDate IS NOT NULL) as Count
FROM Clients
         INNER JOIN Reservations
                    ON Clients.ClientID = Reservations.ClientID
         INNER JOIN Companies
                    ON Clients.ClientID = Companies.ClientID
go
```

d. Free_Places_In_Conferences - widok reprezentujący liczbę wolnych miejsc dla każdej konferencji

```
CREATE VIEW FreePlacesInConferences AS
SELECT Conferences.ConferenceID,
      Conferences.ConferenceName,
      ConferencesDays.Date,
      Conferences.Limit -
      ((SELECT ISNULL(SUM(NormalTickets), 0)
        FROM ReservationDays
        WHERE (ConferencesDays.ConferenceDayID = ConferenceDayID))
          +
       (SELECT ISNULL(SUM(StudentTickets), 0)
        FROM ReservationDays
        WHERE (ConferencesDays.ConferenceDayID = ConferenceDayID)))
          AS TicketsLeft
FROM Conferences
        INNER JOIN ConferencesDays
                   ON ConferencesDays.ConferenceID = Conferences.ConferenceID
go
```

e. Free_Places_In_Workshops - widok reprezentujący liczbę wolnych miejsc na warsztat dla każdego warsztatu

```
CREATE VIEW FreePlacesInWorkshops AS
SELECT Workshops.WorkshopName,
      Workshops.Description,
      ConferencesDays.Date,
      WorkshopDetails.StartTime,
      WorkshopDetails.EndTime,
      WorkshopDetails.Limit - ((SELECT ISNULL(SUM(NormalTickets), 0)
                                FROM WorkshopReservations
                                WHERE (WorkshopDetails.WorkshopDetailsID =
                                       WorkshopDetailsID))
          +
                               (SELECT ISNULL(SUM(StudentTickets), 0)
                                FROM WorkshopReservations
                                WHERE (WorkshopDetails.WorkshopDetailsID =
                                       WorkshopDetailsID))) AS TicketsLeft
FROM WorkshopDetails
        INNER JOIN Workshops
                   ON Workshops.WorkshopID = WorkshopDetails.WorkshopID
        INNER JOIN ConferencesDays
                   ON ConferencesDays.ConferenceDayID =
                      WorkshopDetails.ConferenceDayID
go
```

f. List_Conferences_In_Cities - widok reprezentujący listę konferencji odbywających się w miastach

```
CREATE VIEW ListConferencesInCities AS
SELECT Cities.CityName,
      COUNT(Conferences.ConferenceID) AS ConferencesCount
FROM Cities
        INNER JOIN Conferences
                   ON Cities.CityID = Conferences.CityID
GROUP BY Cities.CityName
go
```

g. List_Conferences_In_Countries - widok reprezentujący listę konferencji odbywających się krajach

```
CREATE VIEW ListOfAttendeesIdentificators AS
SELECT Person.Firstname,
      Person.Lastname,
      { fn CONCAT(SUBSTRING(Person.Firstname, 0, 5),
                  SUBSTRING(Person.Lastname, 0, 5)) } AS Identificator
FROM Person
        INNER JOIN Attendees
                   ON Person.PersonID = Attendees.PersonID
        INNER JOIN ReservationDays
                   ON Attendees.ReservationDayID =
                      ReservationDays.ReservationDayID
go
```

h. List_Of_Attendees_Identificators - widok reprezentujący identyfikatory uczestników

```
CREATE VIEW ListOfAttendeesIdentificators AS
SELECT Person.Firstname,
      Person.Lastname,
      { fn CONCAT(SUBSTRING(Person.Firstname, 0, 5),
                  SUBSTRING(Person.Lastname, 0, 5)) } AS Identificator
FROM Person
        INNER JOIN Attendees
                   ON Person.PersonID = Attendees.PersonID
        INNER JOIN ReservationDays
                   ON Attendees.ReservationDayID =
                      ReservationDays.ReservationDayID
go
```

i. Popularity_Of_Conferences - widok reprezentujący liczbę miejsc zarezerwowanych na konferencje

```
CREATE VIEW PopularityOfConferences AS
SELECT TOP 2147483647 C.ConferenceID,
                      C.ConferenceName,
                      SUM(RD.NormalTickets) +
                      SUM(RD.StudentTickets) AS TicketsCount
```

```
FROM Conferences C
        INNER JOIN Reservations
                   ON C.ConferenceID = Reservations.ConferenceID
        INNER JOIN ReservationDays RD
                   ON Reservations.ReservationID =
                      RD.ReservationID
GROUP BY C.ConferenceID, C.ConferenceName
ORDER BY TicketsCount DESC
go
```

j. Popularity_Of_Workshops - widok reprezentujący liczbę miejsc zarezerwowanych na warsztaty

```
CREATE VIEW PopularityOfWorkshops AS
SELECT TOP 2147483647 W.WorkshopName,
                      W.Description,
                      SUM(WR.NormalTickets) +
                      SUM(WR.StudentTickets) AS TicketsCount
FROM Workshops W
        INNER JOIN WorkshopDetails WD
                   ON W.WorkshopID = WD.WorkshopID
        INNER JOIN WorkshopReservations WR
                   ON WD.WorkshopDetailsID =
                      WR.WorkshopDetailsID
GROUP BY W.WorkshopName, W.Description
ORDER BY TicketsCount DESC
go
```

k. Unpaid_Company_Reservations - widok reprezentujący nieopłacone rezerwacje firm

```
CREATE VIEW UnpaidCompanyReservations AS
SELECT COM.CompanyName,
       C.ConferenceName,
       R.ReservationDate,
       C.StartDate,
       C.EndDate
FROM Conferences C
        INNER JOIN Reservations R
                   ON C.ConferenceID = R.ConferenceID
        INNER JOIN Companies COM
                   ON R.ClientID = COM.ClientID
WHERE R.PaymentDate IS NULL
go
```

l. Unpaid_Individual_Reservations - widok reprezentujący nieopłacone rezerwacje osób indywidualnych

```
CREATE VIEW UnpaidIndividualReservations AS
```

```
SELECT P.Firstname,
       P.Lastname,
       R.ReservationDate,
       C.ConferenceName,
       C.StartDate,
       C.EndDate
FROM Conferences C
         INNER JOIN Reservations R
                    ON C.ConferenceID = R.ConferenceID
         INNER JOIN IndividualClients IC
                    ON R.ClientID = IC.ClientID
         INNER JOIN Person P
                    ON IC.PersonID = P.PersonID
WHERE R.PaymentDate IS NULL
go
```

m. Unpaid_Reservations_That_Shoold_Be_Paid_Tomorrow - widok reprezentujący nieopłacone rezerwacje które powinny być opłacone do następnego dnia

```
CREATE VIEW UnpaidReservationsThatShouldBePaidTomorrow AS
SELECT COM.CompanyName,
       C.ConferenceName,
       R.ReservationDate,
       C.StartDate,
       C.EndDate
FROM Reservations R
         INNER JOIN Clients
                    ON R.ClientID = Clients.ClientID
         INNER JOIN Companies COM
                    ON Clients.ClientID = COM.ClientID
         INNER JOIN Conferences C
                    ON R.ConferenceID = C.ConferenceID
WHERE R.PaymentDate IS NULL
 AND DATEDIFF(day, R.ReservationDate, GETDATE()) = 15
go
```

7. Funkcje

a. **GetConferenceDayConferenceID - Funkcje zwracająca ID konferencji do której należy dany dzień konferencji**

```
CREATE FUNCTION [dbo].[sf_GetConferenceDayConferenceID](
   @ConferenceDayID int
)
   RETURNS int
AS
BEGIN
   RETURN (SELECT ConferenceID
           FROM [ConferencesDays]
           WHERE ConferenceDayID = @ConferenceDayID)
```

```
END
go
```

b. **GetConferenceDayFreePlaces - Funkcja zwracająca ilość wolnych miejsc na dany dzień konferencji**

```
CREATE FUNCTION [dbo].[sf_GetConferenceDayFreePlaces](
   @conferenceDayID INT
)
   RETURNS INT
AS
BEGIN
   DECLARE @limit int =

dbo.sf_GetConferenceLimit(dbo.sf_GetConferenceDayConferenceID(@conferenceDayID))
   DECLARE @used int =
       dbo.sf_GetConferenceDayUsedPlaces(@conferenceDayID)
   RETURN @limit - @used
END
go
```

c. **GetConferenceDayID - Funkcja zwracająca ID dnia konferencji na podstawie ID konferencji i daty**

```
CREATE FUNCTION [dbo].[sf_GetConferenceDayID](@conferenceID int,
                                              @date date)
   RETURNS int
AS
BEGIN
   RETURN (Select ConferenceDayID
           From [ConferencesDays]
           WHERE ConferenceID = @conferenceID
             AND Date = @date)
END
go
```

d. **GetConferenceDayUsedPlaces - Funkcja zwracająca miejsc zajęte na dany dzień konferencji**

```
CREATE FUNCTION [dbo].[sf_GetConferenceDayUsedPlaces](
   @conferenceDayID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT SUM(NormalTickets) + SUM(StudentTickets)
                  FROM [ReservationDays]
                  Where ConferenceDayID = @conferenceDayID), 0)
```

```
END
go
```

e. **GetConferenceLimit - Funkcja zwracająca limit miejsc na konferencje**

```
CREATE FUNCTION [dbo].[sf_GetConferenceLimit](
   @conferenceID int)
   RETURNS int
AS
BEGIN
   RETURN (ISNULL((SELECT Limit
                   FROM Conferences
                   WHERE ConferenceID = @conferenceID), 0))
END
go
```

f. **GetConferenceOrganizerIDv - Funkcja zwracająca ID organizatora koferencji**

```
CREATE FUNCTION [dbo].[sf_GetConferenceOrganizerID](
   @conferenceID int
)
   RETURNS int
AS
BEGIN
   Return (
       Select OrganizerID
       From Conferences
       Where ConferenceID = @conferenceID)
END
go
```

g. **GetConferencePriceDiscount - Funkcja zwracająca procent zniżki dla danej konferencji i zadanej daty**

```
CREATE FUNCTION [dbo].[sf_GetConferencePriceDiscount](@conferenceID int,
                                                      @date date)
   RETURNS decimal(3, 2)
AS
BEGIN
   RETURN ISNULL((SELECT PriceDiscount
                  FROM Prices
                  WHERE ConferenceID = @conferenceID
                    AND StartDate <= @date
                    AND @date <= EndDate), 0)
END
go
```

h. GetReservationCost - funkcja zwracająca koszt danej rezerwacji

```
CREATE FUNCTION [dbo].[sf_GetReservationCost](
   @reservationID int
)
   RETURNS int
AS

BEGIN
   DECLARE @normalprice MONEY =
       dbo.sf_GetReservationNormalTicketPrice(@reservationID)
   DECLARE @discount decimal(3, 2) =
       (SELECT C.StudentDiscount
        FROM Reservations as R
                 JOIN Conferences as C
                      ON C.ConferenceID = R.ConferenceID
        WHERE R.ReservationID = @reservationID)
   DECLARE @reservationCost MONEY =
       (Select Sum(NormalTickets) * @normalprice +
               Sum(StudentTickets) * @normalprice * (1 - @discount)
        From [ReservationDays]
        WHERE ReservationID = @reservationID)
   DECLARE @workshopCost MONEY =
       (Select sum(value)
        From (Select (Select SUM(WR.NormalTickets * WI.Price) +
                             SUM(WR.StudentTickets * (1 - @discount) * WI.Price)
                      FROM [WorkshopReservations] as WR
                               JOIN [WorkshopDetails] as WI
                                    ON WI.WorkshopDetailsID =
WR.WorkshopDetailsID
                      WHERE WR.ReservationDayID = RD.ReservationDayID) as value
              From [ReservationDays] as RD
              WHERE RD.ReservationID = @reservationID) src)
   RETURN ISNULL(@reservationCost, 0) + ISNULL(@workshopCost, 0)
END
go
```

i. GetReservationDayNormal - funkcja zwracająca liczbę normalnych biletów dla danego dnia rezerwacji

```
CREATE FUNCTION [dbo].[sf_GetReservationDayNormal](
   @reservationDayID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT NormalTickets
                  FROM [ReservationDays]
                  WHERE ReservationDayID = @reservationDayID), 0)
END
go
```

j. **GetReservationDayNormalUsed - funkcja zwracająca ilość wykorzystanych miejsc normalnych w danym dniu rezerwacji**

```
CREATE FUNCTION [dbo].[sf_GetReservationDayNormalUsed](
   @reservationDayID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT Count(PersonID)
                  FROM Attendees
                  WHERE ReservationDayID = @reservationDayID) -
                 dbo.sf_GetReservationDayStudentUsed(@reservationDayID), 0)
END
go
```

k. **GetReservationDayStudent - funkcja zwracająca ilość zarezerwowanych miejsc dla studentów na dany dzień konferencji**

```
CREATE FUNCTION [dbo].[sf_GetReservationDayStudent](
   @reservationDayID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT StudentTickets
                  FROM [ReservationDays]
                  WHERE ReservationDayID = @reservationDayID), 0)
END
go
```

l. **GetReservationDayStudentUsed - funkcja zwracająca ilość wykorzystanych miejsc dla studentów na dany dzień konferencji**

```
CREATE FUNCTION [dbo].[sf_GetReservationDayStudentUsed](
   @reservationDayID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT Count(A.PersonID)
                  FROM Attendees as A
                           JOIN Students as S
                                ON S.AttendeeID = A.AttendeeID
                  WHERE A.ReservationDayID = @reservationDayID), 0)
END
go
```

m. **GetReservationNormalTicketPrice - Funkcja zwracająca cenę normalnego biletu w danej rezerwacji**

```
CREATE FUNCTION [dbo].[sf_GetReservationNormalTicketPrice](
   @reservationID INT
)
   RETURNS MONEY
AS
BEGIN
   DECLARE @normalPrice MONEY =
       (SELECT C.Price *
               (1 - dbo.sf_GetConferencePriceDiscount(C.ConferenceID,
R.ReservationDate))
        FROM Reservations as R
                 JOIN Conferences as C
                      ON C.ConferenceID = R.ConferenceID
        WHERE R.ReservationID = @reservationID)
   RETURN ISNULL(@normalprice, 0
       )
END
go
```

n. **GetWorkshopDetailsFreePlaces - Funkcja zwracająca ilość wolnych miejsc na dany warsztat**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopDetailsFreePlaces](
   @workshopDetailsID INT
)
   RETURNS INT
AS
BEGIN
   RETURN (dbo.sf_GetWorkshopDetailsLimit(@workshopDetailsID) -
           dbo.sf_GetWorkshopDetailsUsedPlaces(@workshopDetailsID))
END
go
```

o. **GetWorkshopDetailsLimit - Funkcja zwracająca limit miejsc na dany warsztat**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopDetailsLimit](
   @workshopInstanceID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT Limit
                  FROM [WorkshopDetails]
                  WHERE WorkshopDetailsID = @workshopInstanceID),
                 0)
END
```

```
go
```

p. **GetWorkshopDetailsUsesPlaces - Funkcja zwracająca liczbę zajętych miejsc na dany warsztat**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopDetailsUsedPlaces](
   @workshopDetailsID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT SUM(NormalTickets) + SUM(StudentTickets)
                  FROM [WorkshopReservations]
                  Where WorkshopDetailsID = @workshopDetailsID),
                 0)
END
go
```

q. **GetWorkshopOrganizerID - Funkcja zwracająca ID organizatora warsztatu**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopOrganizerID](
   @workshopID int
)
   RETURNS int
AS
BEGIN
   Return (
       Select OrganizerID
       From Workshops
       Where WorkshopID = @workshopID)
END
go
```

r. **GetWorkshopReservationNormal - funkcja zwracająca ilość zarezerwowanych miejsc normalnych na warsztat**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopReservationNormal](
   @workshopReservationID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT NormalTickets
                  FROM [WorkshopReservations]
                  WHERE WorkshopReservationID = @workshopReservationID), 0)
END
go
```

s. **GetWorkshopReservationNormalUsed - Funkcja zwracająca ilość zajętych miejsc normalnych na warsztat**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopReservationNormalUsed](
   @workshopReservationID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT Count(WP.AttendeeID)
                  FROM [WorkshopReservations] as WR
                           JOIN [WorkshopAttendees] as WP
                                ON WP.WorkshopReservationID =
WR.WorkshopReservationID
                  WHERE WP.WorkshopReservationID = @workshopReservationID)
                     -
dbo.sf_GetWorkshopReservationStudentUsed(@workshopReservationID), 0)
END
go
```

t. **GetWorkshopReservationStudent - Funkcja zwracająca ilość zarezerwowanych miejsc dla studentów na dany warsztat**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopReservationStudent](
   @workshopReservationID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT StudentTickets
                  FROM [WorkshopReservations]
                  WHERE WorkshopReservationID = @workshopReservationID), 0)
END
go
```

u. **GetWorkshopReservationStudentUsed - Funkcja zwracająca ilość zajętych miejsc dla studentów na dany warsztat**

```
CREATE FUNCTION [dbo].[sf_GetWorkshopReservationStudentUsed](
   @workshopReservationID int
)
   RETURNS int
AS
BEGIN
   RETURN ISNULL((SELECT Count(P.PersonID)
                  FROM [WorkshopReservations] as WR
                           JOIN [WorkshopAttendees] as WP
                                ON WP.WorkshopReservationID =
WR.WorkshopReservationID
```

```
                                        JOIN Attendees as P
                                             ON WP.AttendeeID = P.AttendeeID
                                        JOIN Students as S
                                             ON S.AttendeeID = P.AttendeeID
                                WHERE WP.WorkshopReservationID =
            @workshopReservationID), 0)
            END
            go
```

v. ConferencesDaysWithFreePlaces

```
CREATE FUNCTION fp_ConferencesDaysWithFreePlaces(
   @OrganizerID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT DISTINCT ConferenceName, StartDate, EndDate
               FROM dbo.Conferences
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceID =
Conferences.ConferenceID
                        INNER JOIN dbo.Reservations
                                   ON Reservations.ConferenceID = Conferences.ConferenceID
                        INNER JOIN dbo.[ReservationDays] ON [ReservationDays].ConferenceDayID =
                                                            [ConferencesDays].ConferenceDayID
               WHERE Limit >
                     (SELECT SUM(NormalTickets) + SUM(StudentTickets)
                      FROM dbo.[ReservationDays]
                      WHERE dbo.Conferences.OrganizerID = @OrganizerID)
           )
go
```

w. ListAttendeesInConference

```
CREATE FUNCTION fp_ListAttendeesInConference(
   @ConferenceID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT Firstname, Lastname
               FROM dbo.Clients
                        INNER JOIN dbo.Companies
                                   ON Companies.ClientID = Clients.ClientID
                        INNER JOIN dbo.Employees
                                   ON Employees.CompanyID = Companies.ClientID
                        INNER JOIN dbo.Person
                                   ON Person.PersonID = Employees.PersonID
                        INNER JOIN dbo.Reservations
                                   ON Reservations.ClientID = Clients.ClientID
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = Reservations.ConferenceID
                                       AND Conferences.ConferenceID = @ConferenceID
               WHERE dbo.Conferences.Canceled = 0
```

```
                UNION ALL
                SELECT Firstname, Lastname
                FROM dbo.Clients
                        INNER JOIN dbo.[IndividualClients] ON [IndividualClients].ClientID =
Clients.ClientID
                        INNER JOIN dbo.Person
                                ON Person.PersonID = [IndividualClients].PersonID
                        INNER JOIN dbo.Reservations
                                ON Reservations.ClientID = Clients.ClientID
                        INNER JOIN dbo.Conferences
                                ON Conferences.ConferenceID = Reservations.ConferenceID
                                    AND Conferences.ConferenceID = @ConferenceID
                WHERE dbo.Conferences.Canceled = 0
            )
go
```

x. ListAttendeesInConferenceDay

```
CREATE FUNCTION fp_ListAttendeesInConferenceDay(
   @ConferenceDayID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT Firstname, Lastname
               FROM dbo.Clients
                        INNER JOIN dbo.Companies ON Companies.ClientID = Clients.ClientID
                        INNER JOIN dbo.Employees
                                ON Employees.CompanyID = Companies.ClientID
                        INNER JOIN dbo.Person
                                ON Person.PersonID = Employees.PersonID
                        INNER JOIN dbo.Reservations
                                ON Reservations.ClientID = Clients.ClientID
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceID =
Reservations.ConferenceID
                    AND ConferenceDayID = @ConferenceDayID
               UNION ALL
               SELECT Firstname, Lastname
               FROM dbo.Clients
                        INNER JOIN dbo.[IndividualClients] ON [IndividualClients].ClientID =
Clients.ClientID
                        INNER JOIN dbo.Person
                                ON Person.PersonID = [IndividualClients].PersonID
                        INNER JOIN dbo.Reservations
                                ON Reservations.ClientID = Clients.ClientID
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceID =
Reservations.ConferenceID
                    AND ConferenceDayID = @ConferenceDayID
           )
go
```

y. ListAttendeesInWorkshop

```
CREATE FUNCTION fp_ListAttendeesInWorkshop(
   @WorkshopDetailsID int
)
   RETURNS TABLE
```

```
       AS
       RETURN
           (
               SELECT Firstname, Lastname
               FROM dbo.[WorkshopAttendees]
                        INNER JOIN dbo.Attendees
                                   ON Attendees.AttendeeID = [WorkshopAttendees].AttendeeID
                        INNER JOIN dbo.Person
                                   ON Person.PersonID = Attendees.PersonID
                        INNER JOIN dbo.[WorkshopReservations] ON
[WorkshopReservations].WorkshopReservationID =

[WorkshopAttendees].WorkshopReservationID
                        INNER JOIN dbo.[WorkshopDetails] ON [WorkshopDetails].WorkshopDetailsID =
                                                            [WorkshopReservations].WorkshopDetailsID
                   AND [WorkshopDetails].WorkshopDetailsID = @WorkshopDetailsID
           )
go
```

z. ListEmployeesInCompany

```
CREATE FUNCTION [dbo].[fp_ListEmployeesInCompany](
    @CompanyID int
)
    RETURNS TABLE
        AS
        RETURN
            (
                SELECT Firstname, Lastname
                FROM dbo.Employees
                         INNER JOIN dbo.Companies
                                    ON Companies.ClientID = Employees.CompanyID
                                        AND ClientID = @CompanyID
                         INNER JOIN dbo.Person
                                    ON Person.PersonID = Employees.PersonID
            )
go
```

aa. PrintActualReservationsForClient

```
CREATE FUNCTION fp_PrintActualReservationsForClient(
   @ClientID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT ReservationDate, PaymentDate, ConferenceName, StartDate
               FROM dbo.Reservations
                        INNER JOIN dbo.Clients
                                   ON Clients.ClientID = Reservations.ClientID
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = Reservations.ConferenceID
               WHERE (DATEDIFF(day, StartDate, dbo.Reservations.ReservationDate) < 0)
                 AND (DATEDIFF(DAY, StartDate, GETDATE()) < 0) and Clients.ClientID = @ClientID
           )
go
```

bb. PrintActualReservationsForPerson

```
CREATE FUNCTION fp_PrintActualReservationsForPerson(
   @PersonID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT ReservationDate,
                      PaymentDate,
                      ConferenceName,
                      StartDate
               FROM dbo.Reservations
                        INNER JOIN dbo.[ReservationDays] ON [ReservationDays].ReservationID =
                                                            Reservations.ReservationID
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceDayID =
                                                            [ReservationDays].ConferenceDayID
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = [ConferencesDays].ConferenceID
                        INNER JOIN dbo.Attendees
                                   ON Attendees.ReservationDayID =
[ReservationDays].ReservationDayID
                        INNER JOIN dbo.Person
                                   ON Person.PersonID = Attendees.PersonID
                                       AND Person.PersonID = @PersonID
               WHERE (DATEDIFF(day, StartDate, dbo.Reservations.ReservationDate) < 0)
                 AND (DATEDIFF(DAY, StartDate, GETDATE()) < 0)
           )
go
```

cc. PrintActualWorkshopsForPerson

```
CREATE FUNCTION fp_PrintActualWorkshopsForPerson(
   @PersonID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT WorkshopName,
                      Description,
                      Date,
                      StartTime,
                      EndTime,
                      Address,
                      CityName
               FROM dbo.[WorkshopDetails]
                        INNER JOIN dbo.Workshops
                                   ON Workshops.WorkshopID = [WorkshopDetails].WorkshopID
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceDayID =
                                                            [WorkshopDetails].ConferenceDayID
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = [ConferencesDays].ConferenceID
                        INNER JOIN dbo.Cities ON Cities.CityID = Conferences.CityID
                        INNER JOIN dbo.[WorkshopReservations] ON
[WorkshopReservations].WorkshopDetailsID =
```

```
                                                        [WorkshopDetails].WorkshopDetailsID
                    INNER JOIN dbo.[ReservationDays] ON [ReservationDays].ConferenceDayID =
                                                        [ConferencesDays].ConferenceDayID
                    INNER JOIN dbo.Attendees
                               ON Attendees.ReservationDayID =
                                  [ReservationDays].ReservationDayID
                    INNER JOIN dbo.Person
                               ON Person.PersonID = Attendees.PersonID
                                   AND Attendees.PersonID = @PersonID
        )
go
```

dd. PrintNotPaidReservationsForClients

```
CREATE FUNCTION [dbo].[fp_PrintNotPaidReservationsForClient](
   @ClientID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT ReservationID, ConferenceName, StartDate, EndDate
               FROM dbo.Reservations
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = Reservations.ConferenceID
                        INNER JOIN dbo.Clients
                                   ON Clients.ClientID = Reservations.ClientID
               WHERE Clients.ClientID = @ClientID
                 AND PaymentDate IS NULL
           )
go
```

ee. PrintNotPaidReservationsForOrganizer

```
CREATE FUNCTION [dbo].[fp_PrintNotPaidReservationsForOrganizer](
   @Organizer int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT ReservationID, ConferenceName, StartDate, EndDate
               FROM dbo.Reservations
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = Reservations.ConferenceID
                        INNER JOIN dbo.Organizers
                                   ON Organizers.OrganizerID = Conferences.OrganizerID
               WHERE Conferences.OrganizerID = @Organizer
                 AND PaymentDate IS NULL
           )
go
```

ff. WorkshopsWithFreePlaces

```
CREATE FUNCTION [dbo].[fp_WorkshopsWithFreePlaces](
   @OrganizerID int
```

```
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT WorkshopDetailsID,
                      WorkshopName,
                      Description,
                      StartTime,
                      Date
               FROM dbo.[WorkshopDetails]
                        INNER JOIN dbo.Workshops
                                   ON Workshops.WorkshopID = [WorkshopDetails].WorkshopID
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceDayID =
                                                            [WorkshopDetails].ConferenceDayID
               WHERE Limit > (SELECT SUM(NormalTickets) + SUM(StudentTickets)
                              FROM dbo.[WorkshopReservations]
                              WHERE OrganizerID = @OrganizerID)
           )
go
```

gg. BestClientsForOrganizer - wyświetla listę liczbę “najlepszych” klientów dla danego klienta, czyli tych którzy brali najwięcej razy udział w konferencjach danego organizatora

```
CREATE FUNCTION fp_BestClientsForOrganizer(
   @OrganizerID int
)
   RETURNS TABLE
       AS RETURN
           (
               SELECT ClientID,
                      (SELECT SUM(StudentTickets) + SUM(NormalTickets)
                       FROM dbo.[ReservationDays]
                                INNER JOIN dbo.Reservations
                                           ON Reservations.ReservationID =
[ReservationDays].ReservationID
                                INNER JOIN dbo.Conferences
                                           ON Conferences.ConferenceID = Reservations.ConferenceID
                       WHERE OrganizerID = @OrganizerID
                         AND dbo.Clients.ClientID = dbo.Reservations.ClientID)
                        AS TOTAL
               FROM dbo.Clients
               WHERE (SELECT COUNT(*)
                      FROM dbo.[ReservationDays]
                               INNER JOIN dbo.Reservations
                                          ON Reservations.ReservationID =
[ReservationDays].ReservationID
                               INNER JOIN dbo.Conferences
                                          ON Conferences.ConferenceID = Reservations.ConferenceID
                      WHERE OrganizerID = @OrganizerID
                        AND dbo.Clients.ClientID = dbo.Reservations.ClientID) > 0
               GROUP BY ClientID
           )
go
```

hh. ListCompanyAttendesInConference - wyświetla listę uczestników firmowych danej konferencji

```
CREATE FUNCTION fp_ListCompanyAttendeesInConference(
    @ConferenceID int
)
    RETURNS TABLE
        AS
        RETURN
            (
                SELECT Firstname, Lastname
                FROM dbo.Clients
                        INNER JOIN dbo.Companies
                                ON Companies.ClientID = Clients.ClientID
                        INNER JOIN dbo.Employees
                                ON Employees.CompanyID = Companies.ClientID
                        INNER JOIN dbo.Person
                                ON Person.PersonID = Employees.PersonID
                        INNER JOIN dbo.Reservations
                                ON Reservations.ClientID = Clients.ClientID
                        INNER JOIN dbo.Conferences
                                ON Conferences.ConferenceID =
    Reservations.ConferenceID
                                        AND Conferences.ConferenceID = @ConferenceID
                    WHERE dbo.Conferences.Canceled = 0 and Firstname is not null and
    Lastname is not null
                )
    go
```

ii. ListIndividualAttendesInConference - wyświetla listę uczestników indywidualnych danej konferencji

```
CREATE FUNCTION fp_ListIndividualAttendeesInConference(
   @ConferenceID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT Firstname, Lastname
               FROM dbo.Clients
                       INNER JOIN dbo.[IndividualClients] ON [IndividualClients].ClientID =
Clients.ClientID
                       INNER JOIN dbo.Person
                                  ON Person.PersonID = [IndividualClients].PersonID
                       INNER JOIN dbo.Reservations
                                  ON Reservations.ClientID = Clients.ClientID
                       INNER JOIN dbo.Conferences
                                  ON Conferences.ConferenceID = Reservations.ConferenceID
                                      AND Conferences.ConferenceID = @ConferenceID
               WHERE dbo.Conferences.Canceled = 0
           )
go
```

jj. Split - funkcja która z podanego stringa jako argument wejściowy tworzy tabele, którą uzupełnia “pociętymi” strgingami

```
CREATE FUNCTION [dbo].[Split] (@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
```

8. Procedury

a. AddAttendee - procedura dodanie pracownika firmy do rezerwacji

```
CREATE PROCEDURE [dbo].[sp_AddAttendee] @reservationDayID int,
                                        @personID int,
                                        @studentCardID char(10) = NULL,
                                        @attendeeID int out
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN AddAttendee
            IF ((SELECT ClientID
                 FROM [ReservationDays] as RD
                          JOIN Reservations as R
                               ON R.ReservationID = RD.ReservationID
                 WHERE RD.ReservationDayID = @reservationDayID) <>
                (SELECT CompanyID
                 FROM Employees
                 WHERE PersonID = @personID))
                BEGIN
                    ;
                    THROW 52000,
                        'Pracownik nie należy do firmy do której należy dana rezerwacja', 1;
                END
            IF (SELECT Count(personID)
                FROM Attendees as P
                         JOIN [ReservationDays] as RD
                              On RD.ReservationDayID = P.ReservationDayID
                WHERE P.PersonID = @personID and RD.ReservationDayID = @reservationDayID) > 0
                BEGIN
                    ;
                    THROW 52000,
                        'Pracownik jest już przypisany do danej rezerwacji', 1;
                END
            IF (@studentCardID is not null)
                BEGIN
                    IF (dbo.sf_GetReservationDayStudent
                            (@reservationDayID) -
```

```
                    dbo.sf_GetReservationDayStudentUsed
                        (@reservationDayID) < 1)
                    BEGIN
                        ;
                        THROW 52000,
                            'Brak wolnych miejsc dla studentów
                            w rezerwacji', 1;
                    END
                INSERT INTO Attendees(PersonID, ReservationDayID)
                VALUES (@personID, @reservationDayID)
                SET @attendeeID = @@IDENTITY
                INSERT INTO Students(AttendeeID, StudentCardID)
                VALUES (@attendeeID, @studentCardID)
            END
        ELSE
            BEGIN
                IF (dbo.sf_GetReservationDayNormal
                        (@reservationDayID) -
                    dbo.sf_GetReservationDayNormalUsed(@reservationDayID) < 1)
                    BEGIN
                        ;
                        THROW 52000,
                            'Brak wolnych miejsc normalnych w rezerwacji', 1;
                    END
                INSERT INTO Attendees(PersonID, ReservationDayID)
                VALUES (@personID, @reservationDayID)
                SET @attendeeID = @@IDENTITY
            END
        COMMIT TRAN AddAttendee
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN AddAttendee
        DECLARE @msg NVARCHAR(2048) = 'Nie udało się dodać uczestnika:'
            +
                                    CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

b. AddCompanyClient - dodanie firmy jako klienta

```
CREATE PROCEDURE [dbo].[sp_AddCompanyClient]
    @companyName varchar(255),
        @nip char(10),
        @contactName varchar(255),
        @phone varchar(255),
        @email varchar(255),
        @address varchar(255) = NULL,
        @cityName varchar(255) = NULL,
        @countryName varchar(255) = NULL,
        @postalCode varchar(255) = NULL,
        @clientID int OUTPUT
AS
BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
                BEGIN TRAN ADD_COMPANY_CLIENT
                        EXEC sp_InsertClient
```

```
                                @email,
                                @address,
                                @cityName,
                                @countryName,
                                @postalCode,
                                @clientID = @clientID OUTPUT
                        INSERT INTO Companies(ClientID, CompanyName, NIP,
                                                ContactName, Phone)
                        VALUES(
                                @clientID,
                                @companyName,
                                @nip,
                                @contactName,
                                @phone
                        );
                COMMIT TRAN ADD_COMPANY_CLIENT
        END TRY
        BEGIN CATCH
                ROLLBACK TRAN ADD_COMPANY_CLIENT
                DECLARE @msg NVARCHAR(2048) =  'Nie udało się dodać firmy:' +
                CHAR(13) + CHAR(10) + ERROR_MESSAGE();
                THROW 52000,@msg, 1;
        END CATCH
END
go
```

c. AddConference - dodanie konferencji

```
CREATE PROCEDURE [dbo].[sp_AddConference] @organizerID int,
                                        @conferenceName varchar(255),
                                        @studentDiscount decimal(3, 2) = 0,
                                        @address varchar(255),
                                        @cityName varchar(255),
                                        @countryName varchar(255),
                                        @postalCode varchar(255),
                                        @startDate date,
                                        @endDate date,
                                        @limit int,
                                        @price money,
                                        @conferenceID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN ADD_Conference
           IF (@startDate < GETDATE())
               BEGIN
                   ;
                   THROW 52000,
                       'Data startu konferencji jest wcześniejsza niż obecna data', 1;
               END
           DECLARE @cityID int
           EXEC sp_FindCity
                @cityName,
                @countryName,
                @cityID = @cityID out
           INSERT INTO Conferences(OrganizerID,
                                   ConferenceName,
                                   StudentDiscount,
```

```
                                Address,
                                CityID,
                                PostalCode,
                                StartDate,
                                EndDate,
                                Limit,
                                Price)
          VALUES (@organizerID,
                  @conferenceName,
                  @studentDiscount,
                  @address,
                  @cityID,
                  @postalCode,
                  @startDate,
                  @endDate,
                  @limit,
                  @price);
          SET @conferenceID = @@IDENTITY;
          DECLARE @i date = @startDate
          WHILE @i <= @endDate
              BEGIN
                  INSERT INTO [ConferencesDays](ConferenceID, Date)
                  VALUES (@conferenceID, @i)
                  SET @i = DATEADD(d, 1, @i)
              END
       COMMIT TRAN ADD_Conference
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN ADD_Conference
       DECLARE @msg NVARCHAR(2048) =
               'Nie udało się stworzyć konferencji:' +
               CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

d. AddEmployee - dodanie pracownika

```
CREATE PROCEDURE [dbo].[sp_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 ADD_Employee
           EXEC [dbo].sp_InsertPerson
                @firstname,
                @lastname,
                @phone,
                @personID = @personID OUTPUT
           INSERT INTO Employees (PersonID, CompanyID)
           VALUES (@personID, @companyID);
       COMMIT TRAN ADD_Employee
   END TRY
   BEGIN CATCH
```

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

e. AddIndividualClient - dodanie klienta indywidualnego

```
CREATE PROCEDURE [dbo].[sp_AddIndividualClient] @firstname varchar(255),
                                                @lastname varchar(255),
                                                @phone varchar(255),
                                                @email varchar(255),
                                                @address varchar(255) = NULL,
                                                @cityName varchar(255) = NULL,
                                                @countryName varchar(255) = NULL,
                                                @postalCode varchar(255) = NULL,
                                                @clientID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN ADD_INDIVIDUAL_CLIENT
            DECLARE @personID int
            EXEC sp_InsertClient
                 @email,
                 @address,
                 @cityName,
                 @countryName,
                 @postalCode,
                 @clientID = @clientID OUTPUT

            EXEC sp_InsertPerson
                 @firstname,
                 @lastname,
                 @phone,
                 @personID = @personID OUTPUT

            INSERT INTO [IndividualClients] (ClientID, PersonID)
            VALUES (@clientID,
                    @personID);

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

f. AddOrganizer - dodanie organizatora

```
CREATE PROCEDURE [dbo].[sp_AddOrganizer] @companyName varchar(255),
                                         @nip char(10),
                                         @contactName varchar(255),
                                         @email varchar(255),
                                         @phone varchar(255),
                                         @address varchar(255) = NULL,
                                         @cityName varchar(255) = NULL,
                                         @countryName varchar(255) = NULL,
                                         @postalCode varchar(255) = NULL,
                                         @organizerID INT OUTPUT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN ADD_Organizer
           DECLARE @cityID int = NULL

           EXEC sp_FindCity
                @cityName,
                @countryName,
                @cityID = @cityID OUTPUT
           INSERT INTO Organizers(CompanyName, NIP,
                                  ContactName, Email, Phone,
                                  Address, PostalCode, CityID)
           VALUES (@companyName,
                   @nip,
                   @contactName,
                   @email,
                   @phone,
                   @address,
                   @postalCode,
                   @cityID);
           SET @organizerID = @@IDENTITY
       COMMIT TRAN ADD_Organizer
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN ADD_Organizer
       DECLARE @msg NVARCHAR(2048) =
           'Nie udało się dodać organizatora do bazy:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

g. AddPrice - dodanie progu cenowego ko konferencji

```
CREATE PROCEDURE [dbo].[sp_AddPrice] @conferenceID int,
                                     @startDate date,
                                     @endDate date,
                                     @priceDiscount decimal(3, 2)
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN ADD_PRICE_TO_CONFERENCE
           IF (Convert(date, getdate()) > @startDate)
               BEGIN
                   ;
                   THROW 52000,
```

```
                    'Data progu cenowego juz byla', 1;
            END
        IF (@endDate >= (SELECT StartDate
                         FROM Conferences
                         WHERE ConferenceID = @conferenceID))
            BEGIN
                ;
                THROW 52000,
                    'Progi cenowe nie mogą kończyć się po rozpoczęciu konferencji', 1;
            END
        IF (0 < (SELECT Count(PriceID)
                 FROM Prices
                 WHERE ConferenceID = @conferenceID
                   and ((StartDate <= @endDate and @endDate <= EndDate) or
                        (StartDate <= @startDate and @startDate <= EndDate))))
            BEGIN
                ;
                THROW 52000,
                    'Konferencja ma już próg cenowy pokrywający się z tym okresem czasu', 1;
            END
        INSERT INTO Prices(ConferenceID, StartDate, EndDate,
                           PriceDiscount)
        VALUES (@conferenceID,
                @startDate,
                @endDate,
                @priceDiscount)
    COMMIT TRAN ADD_PRICE_TO_CONFERENCE
END TRY
BEGIN CATCH
    ROLLBACK TRAN ADD_PRICE_TO_CONFERENCE
    DECLARE @msg NVARCHAR(2048) =
            'Nie udało się dodać progu cenowego do konferencji:' +
            CHAR(13) + CHAR(10) + ERROR_MESSAGE();
    THROW 52000,@msg, 1;
END CATCH
END
go
```

h. AddReservation - utworzenie rezerwacji

```
CREATE PROCEDURE [dbo].[sp_AddReservation] @conferenceID int,
                                           @clientID int,
                                           @reservationID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN ADD_RESERVATION
           DECLARE @date date = GETDATE()
           IF ((SELECT Count(ConferenceID)
                FROM Conferences
                WHERE ConferenceID = @conferenceID) = 0)
               BEGIN
                   ;THROW 52000,
                       'Podana konferencja nie istnieje', 1;
               END
           IF ((SELECT Count(ReservationID)
                FROM Reservations
                WHERE ClientID = @clientID
```

```
                and ConferenceID = @conferenceID) > 0)
            BEGIN
                ;THROW 52000,
                    'Podany klient już posiada rezerwacje na daną konferencje', 1;
            END
        IF ((SELECT StartDate
             FROM Conferences
             WHERE ConferenceID = @conferenceID) <= @date)
            BEGIN
                ;THROW 52000,
                    'Nie mozna juz dokonywac rezerwacji na podana konferencje, konferencja już
sie zaczęła', 1;
            END
        IF ((SELECT Canceled
             FROM Conferences
             WHERE ConferenceID = @conferenceID) = 1)
            BEGIN
                ;THROW 52000,
                    'Konferencja została anulowana',
                    1;
            END
        INSERT INTO Reservations(ConferenceID, ClientID,
                                 ReservationDate)
        VALUES (@conferenceID, @clientID, @date)
        SET @reservationID = @@IDENTITY
    COMMIT TRAN ADD_RESERVATION
  END TRY
  BEGIN CATCH
    ROLLBACK TRAN ADD_RESERVATION
    DECLARE @msg NVARCHAR(2048) =
            'Nie udało się dodać rezerwacji:' +
            CHAR(13) + CHAR(10) + ERROR_MESSAGE();
    THROW 52000,@msg, 1;
  END CATCH
END
go
```

i. AddReservationDay - zarezerwowanie dnia konferencji dla firm

```
CREATE PROCEDURE [dbo].[sp_AddReservationDay] @reservationID int,
                                              @conferenceDayID int,
                                              @normalTickets int = 0,
                                              @studentTickets int = 0,
                                              @studentCardIDs varchar(3000) = null,
                                              @reservationDayID int out
AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
    BEGIN TRAN ADD_ReservationDay
      IF (dbo.sf_GetConferenceDayConferenceID(@conferenceDayID) <> (Select ConferenceID
                                                                    FROM Reservations
                                                                    WHERE ReservationID =
@reservationID))
            BEGIN
                ;
                THROW 52000,
                    'Rezerwacja jest na inna konferencje', 1;
            END
```

```
        IF ((SELECT PaymentDate
             FROM Reservations
             WHERE ReservationID = @reservationID) is not null)
            BEGIN
                ;
                THROW 52000,'Rezerwacja została już opłacona', 1;
            END
        IF (@normalTickets + @studentTickets = 0)
            BEGIN
                ;
                THROW 52000,
                    'Trzeba rezerwować przynajmniej jedno miejsce', 1;
            END
        IF ((SELECT count(ReservationID)
             FROM [ReservationDays]
             WHERE ReservationID = @reservationID
               AND ConferenceDayID = @conferenceDayID) = 1)
            BEGIN
                ;
                THROW 52000,
                    'Klient posiada już rezerwacje na dany dzień konferencji', 1;
            END
        IF ((SELECT Canceled
             FROM [ConferencesDays]
             WHERE ConferenceDayID = @conferenceDayID) =
            1)
            BEGIN
                ;
                THROW 52000,
                    'Ten dzień konferencji został anulowany', 1;
            END
        IF (dbo.sf_GetConferenceDayFreePlaces
                (@conferenceDayID) < @normalTickets + @studentTickets)
            BEGIN
                ;
                THROW 52000,
                    'Nie ma wystarczającej ilości wolnych miejsc', 1;
            END
        INSERT INTO [ReservationDays](ReservationID,
                                      ConferenceDayID,
                                      NormalTickets,
                                      StudentTickets)
        VALUES (@reservationID,
                @conferenceDayID,
                @normalTickets,
                @studentTickets)
        SET @reservationDayID = @@IDENTITY

        if @studentCardIDs is not null
            begin
                if (select count(*) from Split(' ', @studentCardIDs)) != @studentTickets
                    begin
                        ;
                        THROW 52000,
                            'Liczba student IDs jest różna od liczby biletów studenckich', 1;
                    end
                Select *
                Into #Temp
                From Split(' ', @studentCardIDs)

                Declare @Id int
```

```
                declare @scID varchar(100)

                While (Select Count(*) From #Temp) > 0
                    Begin

                        Select Top 1 @Id = pn, @scID = s From #Temp


                        declare @personId int
                        declare @companyID int
                        set @companyID = (select ClientID from Reservations where ReservationID =
@reservationID)

                        exec sp_AddEmployee @companyID, null, null, null, @personId out
                        exec sp_AddAttendee @reservationDayID, @personId, @scID, null

                        Delete #Temp Where pn = @Id

                    End
            end
        COMMIT TRAN ADD_ReservationDay
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN ADD_ReservationDay
        DECLARE @msg NVARCHAR(2048) =
            'Błąd dodania rezerwacji:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

j. AddReservationDayIndividual - zarezerwowanie dnia konferencji dla osoby indywidualnej

```
CREATE PROCEDURE [dbo].[sp_AddReservationDayIndividual] @clientID int,
                                                        @reservationID int,
                                                        @conferenceDayID int,
                                                        @studentCardID char(10) = null,
                                                        @reservationDayID int out
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN ADD_ReservationDayIndividual
            DECLARE @personID int =
                (SELECT top 1 IC.PersonID
                 FROM Reservations as R
                          JOIN [IndividualClients] as IC
                               ON IC.ClientID = R.ClientID
                 where IC.ClientID = @clientID)
            IF (@personID is null)
                BEGIN
                    ;THROW 52000,'Nie istnieje klient o takim ID', 1;
                END
            DECLARE @normal int = 1
            DECLARE @student int = 0
            IF (@studentCardID is not null)
                BEGIN
                    SET @normal = 0
                    SET @student = 1
```

```
            END
        EXEC sp_AddReservationDay
             @reservationID,
             @conferenceDayID,
             @normal,
             @student,
            null,
             @reservationDayID = @reservationDayID out
        INSERT INTO Attendees(PersonID, ReservationDayID)
        VALUES (@personID, @reservationDayID)
        DECLARE @attendeeID int = @@IDENTITY
        IF (@studentCardID is not null)
            BEGIN
                INSERT INTO Students(AttendeeID, StudentCardID)
                VALUES (@attendeeID, @studentCardID)
            END
    COMMIT TRAN ADD_ReservationDayIndividual
END TRY
BEGIN CATCH
    ROLLBACK TRAN ADD_ReservationDayIndividual
    DECLARE @msg NVARCHAR(2048) =
            'Błąd dodania rezerwacji indywidualnej:' +
            CHAR(13) + CHAR(10) + ERROR_MESSAGE();
    THROW 52000,@msg, 1;
END CATCH
END
go
```

k. AddReservationWorkshop - zarezerwowanie warsztatu dla firm

```
CREATE PROCEDURE [dbo].[sp_AddReservationWorkshop] @reservationDayID int,
                                                   @workshopDetailsID int,
                                                   @normalTickets int = 0,
                                                   @studentTickets int = 0,
                                                   @workshopReservationID int out
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN ADD_WorkshopReservation
           IF (@normalTickets + @studentTickets = 0)
               BEGIN
                   THROW 52000,
                       'Liczba rezerwowanych miejsc musi być większa od 0', 1;
               END
           IF ((SELECT R.PaymentDate
                FROM Reservations as R
                         JOIN [ReservationDays] as RD
                              ON RD.ReservationID = R.ReservationID
                WHERE RD.ReservationDayID = @reservationDayID) is not null)
               BEGIN
                   THROW 52000,
                       'Rezerwacja została już opłacona', 1;
               END
           IF ((SELECT count(ReservationDayID)
                FROM [WorkshopReservations]
                WHERE ReservationDayID = @reservationDayID
                  and @workshopDetailsID = WorkshopDetailsID)
               > 0)
```

```
            BEGIN
                THROW 52000,
                    'Klient posiada już rezerwacje na dany warsztat', 1;
            END
        IF ((SELECT ConferenceDayID
             FROM [WorkshopDetails]
             WHERE WorkshopDetailsID = @workshopDetailsID) <>
            (SELECT ConferenceDayID
             FROM [ReservationDays]
             WHERE ReservationDayID = @reservationDayID))
            BEGIN
                ;
                THROW 52000,
                    'Rezerwacja i warsztat odwołują się do innego dnia konferencji', 1;
            END
        IF ((SELECT Canceled
             FROM [WorkshopDetails]
             WHERE WorkshopDetailsID = @workshopDetailsID) = 1)
            BEGIN
                THROW 52000,
                    'Ten warsztat został anulowany',
                    1;
            END
        IF (dbo.sf_GetWorkshopDetailsFreePlaces(@workshopDetailsID) < @normalTickets +
@studentTickets)
            BEGIN
                ;
                THROW 52000,
                    'Nie ma wystarczającej ilości wolnych miejsc na ten warsztat', 1;
            END
        IF (dbo.sf_GetReservationDayNormal(@reservationDayID) < @normalTickets
            or dbo.sf_GetReservationDayStudent(@reservationDayID) < @studentTickets)
            BEGIN
                ;
                THROW 52000,
                    'Nie można rezerwować większej ilości miejsc niż w rezerwacji na dzien
konferencji', 1;
            END
        INSERT INTO [WorkshopReservations](WorkshopDetailsID,
                                           NormalTickets,
                                           StudentTickets,
                                           ReservationDayID)
        VALUES (@workshopDetailsID,
                @normalTickets,
                @studentTickets,
                @reservationDayID)
        SET @workshopReservationID = @@IDENTITY
    COMMIT TRAN ADD_WorkshopReservation
END TRY
BEGIN CATCH
    ROLLBACK TRAN ADD_WorkshopReservation
    DECLARE @msg NVARCHAR(2048) =
        'Nie udało się dodać rezerwacji:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
    THROW 52000,@msg, 1;
END CATCH
END
go
```

l. AddReservationWorkshopIndividual - zarezerwowanie warsztatu dla osoby prywatnej

```
CREATE PROCEDURE [dbo].[sp_AddReservationWorkshopIndividual] @reservationDayID int,
                                                             @workshopDetailsID int,
                                                             @workshopReservationID int out
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN ADD_ReservationWorkshopInd
           DECLARE @attendeeID int =
               (SELECT AttendeeID
                FROM Attendees
                WHERE ReservationDayID = @reservationDayID)

           DECLARE @normal int =
               dbo.sf_GetReservationDayNormal(@reservationDayID)
           DECLARE @student int =
               dbo.sf_GetReservationDayStudent(@reservationDayID)
           EXEC sp_AddReservationWorkshop
                @reservationDayID,
                @workshopDetailsID,
                @normal,
                @student,
                @workshopReservationID = @workshopReservationID out
           EXEC sp_AddWorkshopAttendee
                @workshopReservationID,
                @attendeeID
       COMMIT TRAN ADD_ReservationWorkshopInd
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN ADD_ReservationWorkshopInd
       DECLARE @msg NVARCHAR(2048) =
               'Nie udało się dodać rezerwacji indywidualnej:' +
               CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

m. AddWorkshop - dodanie warsztatu

```
CREATE PROCEDURE [dbo].[sp_AddWorkshop] @organizerID int,
                                        @workshopName varchar(255),
                                        @workshopDescription varchar(255),
                                        @workshopID int out
AS
BEGIN
   SET NOCOUNT ON;
   INSERT INTO Workshops(OrganizerID,
                         WorkshopName, Description)
   VALUES (@organizerID,
           @workshopName,
           @workshopDescription)
   SET @workshopID = @@IDENTITY
END
go
```

n. AddWorkshopAttendee - dodanie uczestnika warsztatu

```
CREATE PROCEDURE [dbo].[sp_AddWorkshopAttendee] @workshopReservationID int,
                                                @attendeeID int
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       BEGIN TRAN ADD_WorkshopAttendee
           print @workshopReservationID
           print @attendeeID
           IF (SELECT Count(RD.ReservationDayID)
               FROM [ReservationDays] as RD
                        JOIN Attendees as P
                             ON P.ReservationDayID = RD.ReservationDayID
                        JOIN [WorkshopReservations] as WR
                             ON WR.ReservationDayID = RD.ReservationDayID
               where WR.WorkshopReservationID = @workshopReservationID
                 and P.AttendeeID = @attendeeID) = 0
               BEGIN
                   print '1';
                   THROW 52000,'Klient zapisał się na inny dzień konferencji niż dzień odbywania sie
warsztatu', 1;
               END
           print '12'
           DECLARE @workshopDetailsID INT = (SELECT WorkshopDetailsID
                                             FROM [WorkshopReservations]
                                             WHERE workshopReservationID = @workshopReservationID)
           DECLARE @ConferenceDayID INT = (SELECT ConferenceDayID
                                           FROM [WorkshopDetails]
                                           WHERE WorkshopDetailsID = @workshopDetailsID)
           DECLARE @startTime time = (SELECT StartTime
                                      FROM [WorkshopDetails]
                                      WHERE WorkshopDetailsID = @workshopDetailsID)
           DECLARE @endTime time = (SELECT EndTime
                                    FROM [WorkshopDetails]
                                    WHERE WorkshopDetailsID = @workshopDetailsID)
           IF ((SELECT COUNT(WP.AttendeeID)
                FROM [WorkshopDetails] as WI
                         JOIN [WorkshopReservations] as WR
                              ON WR.WorkshopDetailsID = WI.WorkshopDetailsID
                         JOIN [WorkshopAttendees] as WP
                              ON WP.WorkshopReservationID = WR.WorkshopReservationID
                                  and WP.AttendeeID = @attendeeID
               WHERE ((WI.StartTime <= @startTime and @startTime <= WI.EndTime) or
                      (WI.StartTime <= @endTime and @endTime <= WI.EndTime))
                 and WI.ConferenceDayID = @ConferenceDayID) > 0)
               BEGIN
                   print '2';THROW 52000,'W czasie odbywania sie warsztatu uczestnik bierze udział w
innym warsztacie', 1;
               END
           IF ((Select Count(AttendeeID)
                FROM Students
                WHERE AttendeeID = @attendeeID) > 0)
               BEGIN
                   IF (dbo.sf_GetWorkshopReservationStudent(@workshopReservationID) -
                       dbo.sf_GetWorkshopReservationStudentUsed(@workshopReservationID) < 1)
                       BEGIN
                           print '3';THROW 52000,'Brak wolnych miejsc dla studentów w rezerwacji',
1;
                       END
```

```
            END
        ELSE
            BEGIN
                IF (dbo.sf_GetWorkshopReservationNormal(@workshopReservationID) -
                    dbo.sf_GetWorkshopReservationNormalUsed(@workshopReservationID) < 1)
                    BEGIN
                        print '4';THROW 52000,'Brak wolnych miejsc normalnych w rezerwacji', 1;
                    END
            END
        INSERT INTO [WorkshopAttendees] (WorkshopReservationID, AttendeeID)
        VALUES (@workshopReservationID, @attendeeID)
    COMMIT TRAN ADD_WorkshopAttendee
END TRY
BEGIN CATCH
    ROLLBACK TRAN ADD_WorkshopAttendee
    DECLARE @msg NVARCHAR(2048) = 'Nie udało się dodać uczestnika do warsztatu:' + CHAR(13) +
CHAR(10) + ERROR_MESSAGE();
    THROW 52000,@msg, 1;
END CATCH
END
go
```

o. AddWorksopDetails - dodanie warsztatu do konferencji

```
CREATE PROCEDURE [dbo].[sp_AddWorkshopDetails] @workshopID int,
                                               @conferenceID int,
                                               @date date,
                                               @startTime time(7),
                                               @endTime time(7),
                                               @limit int,
                                               @price money = 0,
                                               @workshopInstanceID int out
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN Add_WorkshopDetails
            IF (@date < GETDATE())
                BEGIN
                    ;
                    THROW 52000,'Data warsztatu jest niepoprawna', 1;
                END
            IF (dbo.sf_GetWorkshopOrganizerID(@workshopID) <>
                dbo.sf_GetConferenceOrganizerID(@conferenceID))
                BEGIN
                    ;
                    THROW 52000,'Warsztat i konferencja należą do innych organizatorow', 1;
                END
            IF (dbo.sf_GetConferenceLimit(@conferenceID) < @limit)
                BEGIN
                    ;
                    THROW 52000,'Limit miejsc nie może być większy od liczby miejsc na konferencje',
1;
                END
            DECLARE @conferenceDayID int = dbo.sf_GetConferenceDayID(@conferenceID, @date)
            IF (@conferenceDayID is null)
                BEGIN
                    ;
                    THROW 52000,'Konferencja nie odbywa sie w podanym dniu', 1;
```

```
            END
        INSERT INTO [WorkshopDetails](WorkshopID,
                                      ConferenceDayID,
                                      StartTime,
                                      EndTime,
                                      Limit,
                                      Price,Canceled)
        VALUES (@workshopID,
                @conferenceDayID,
                @startTime,
                @endTime,
                @limit,
                @price,'')
        SET @workshopInstanceID = @@IDENTITY
    COMMIT TRAN Add_WorkshopDetails
END TRY
BEGIN CATCH
    ROLLBACK TRAN Add_WorkshopDetails
    DECLARE @msg NVARCHAR(2048) =
            'Nie udało się dodać warsztatu do konferencji:' +
            CHAR(13) + CHAR(10) + ERROR_MESSAGE();
    THROW 52000,@msg, 1;
END CATCH
END
go
```

p. CancelConference - anulowanie konferencji

```
CREATE PROCEDURE [dbo].[sp_CancelConference] @conferenceID int
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN CancelConference
           IF ((Select Canceled
                FROM Conferences
                WHERE conferenceID = @conferenceID) != 0)
               BEGIN
                   ;THROW 52000,'Konferencja została wczesniej anulowana', 1;
               END

           UPDATE Conferences
           SET Canceled = 1
           WHERE conferenceID = @conferenceID

           DELETE Reservations
           WHERE conferenceID = @conferenceID
             and PaymentDate is null

           UPDATE ConferencesDays
           set Canceled = 1
           where ConferenceID = @conferenceID

           UPDATE WorkshopDetails
           set Canceled = 1
           where ConferenceDayID in
                 (select ConferencesDays.ConferenceDayID from ConferencesDays where ConferenceID =
@conferenceID)
       COMMIT TRAN CancelConference
   END TRY BEGIN CATCH
```

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

q. CancelConferenceDay - anulowanie dnia konferencji

```
CREATE PROCEDURE [dbo].[sp_CancelConferenceDay] @conferenceDayID int
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN CancelConferenceDay
           IF ((Select Canceled
                FROM [ConferencesDays]
                WHERE conferenceDayID = @conferenceDayID) != 0)
               BEGIN
                   ;THROW 52000,'Dzień został wczesniej anulowany', 1;
               END

           UPDATE [ConferencesDays]
           SET Canceled = 1
           WHERE conferenceDayID = @conferenceDayID

           DELETE RD
           FROM [ReservationDays] as RD
                    JOIN Reservations as R
                         ON R.ReservationID = RD.ReservationID
           WHERE RD.conferenceDayID = @conferenceDayID
             and R.PaymentDate is null
       COMMIT TRAN CancelConferenceDay
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN CancelConferenceDay
       DECLARE @msg NVARCHAR(2048) = 'Nie udało sie anulować dnia konferencji:' + CHAR(13) +
CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

r. CancelWorkshop - anulowanie warsztatu

```
CREATE PROCEDURE [dbo].[sp_CancelWorkshop] @WorkshopDetailsID int
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN CancelWorkshop
           IF ((Select Canceled
                FROM [WorkshopDetails]
                WHERE WorkshopDetailsID = @WorkshopDetailsID) != 0)
               BEGIN
                   ;THROW 52000,'Warsztat został wcześniej anulowany', 1;
               END

           UPDATE [WorkshopDetails]
```

```
            SET Canceled = 1
            WHERE WorkshopDetailsID = @WorkshopDetailsID

            DELETE WA
            FROM [WorkshopAttendees] as WA
                     JOIN [WorkshopReservations] as WR
                          ON WA.WorkshopReservationID = WR.WorkshopReservationID
                     JOIN [ReservationDays] as RD
                          ON RD.ReservationDayID = WR.ReservationDayID
                     JOIN Reservations as R
                          ON R.ReservationID = RD.ReservationID
            WHERE WR.WorkshopDetailsID = @WorkshopDetailsID
              and R.PaymentDate is null

            DELETE WR
            FROM [WorkshopReservations] as WR
                     JOIN [ReservationDays] as RD
                          ON RD.ReservationDayID = WR.ReservationDayID
                     JOIN Reservations as R
                          ON R.ReservationID = RD.ReservationID
            WHERE WR.WorkshopDetailsID = @WorkshopDetailsID
              and R.PaymentDate is null

        COMMIT TRAN CancelWorkshop
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN CancelWorkshop
        DECLARE @msg NVARCHAR(2048) = 'Nie udało się anulować warsztatu:' + CHAR(13) + CHAR(10) +
ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

s. FindCity - znalezienie miasta i jeśli nie istnieje to dodanie go do bazy

```
CREATE PROCEDURE [dbo].[sp_FindCity] @cityName varchar(255),
                                     @countryName varchar(255),
                                     @cityID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN FIND_CITY_BY_NAME
            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 ', 1;
                END
            IF (@cityName is not null and @countryName is not null)
                BEGIN
                    DECLARE @countryID int
                    EXEC sp_FindCountry
                         @countryName,
                         @countryID = @countryID out
```

```
                SET @cityID = (Select cityID
                               From Cities
                               Where CountryID = @countryID and CityName = @cityName)
                IF (@cityID is null)
                    BEGIN
                        INSERT INTO Cities(CityName, CountryID)
                        VALUES (@cityName, @countryID);
                        SET @cityID = @@IDENTITY;
                    END
            END
        COMMIT TRAN FIND_CITY_BY_NAME
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN FIND_CITY_BY_NAME
        DECLARE @msg NVARCHAR(2048) =
            'Nie udało się znaleźć miesta miasta:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

t. FindCountry - znalezienie kraju i jeśli nie istnieje to dodanie go do bazy

```
CREATE PROCEDURE [dbo].[sp_FindCountry] @countryName varchar(255),
                                        @countryID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRAN FIND_COUNTRY
            SET @countryID = (Select CountryID
                              From Countries
                              Where CountryName = @countryName)
            IF (@countryID is null)
                BEGIN
                    INSERT INTO Countries(CountryName)
                    VALUES (@countryName);
                    SET @countryID = @@IDENTITY;
                END
        COMMIT TRAN FIND_COUNTRY
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN FIND_COUNTRY
        DECLARE @msg NVARCHAR(2048) =
            'Nie udało się znaleźć kraju:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

u. InsertClient - dodanie klienta do bazy

```
CREATE PROCEDURE [dbo].[sp_InsertClient] @email varchar(255),
                                         @address varchar(255) = NULL,
                                         @cityName varchar(255) = NULL,
                                         @countryName varchar(255) = NULL,
                                         @postalCode varchar(255) = NULL,
                                         @clientID int OUTPUT
```

```
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       DECLARE @cityID int
       EXEC sp_FindCity
            @cityName,
            @countryName,
            @cityID = @cityID OUTPUT
       INSERT INTO Clients(Email, Address, PostalCode, CityID)
       VALUES (@email,
               @address,
               @postalCode,
               @cityID);
       SET @clientID = @@IDENTITY
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) =
           'Nie udało się dodać klienta do bazy:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

v. InsertPerson - wstawienie osoby do bazy

```
CREATE PROCEDURE [dbo].[sp_InsertPerson]
        @firstname varchar(255) = NULL,
        @lastname varchar(255) = NULL,
        @phone varchar(255) = NULL,
        @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) =
                'Nie udało się dodać osoby do bazy:' + CHAR(13)+CHAR(10) + ERROR_MESSAGE ();
                THROW 52000,@msg, 1;
        END CATCH
END
go
```

w. PayReservation - płacenie za rezerwacje

```
CREATE PROCEDURE [dbo].[sp_PayReservation] @reservationID int
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN PayReservation
```

```
            IF ((SELECT PaymentDate
                 FROM Reservations
                 WHERE ReservationID = @reservationID) is not null)
                BEGIN
                    ;THROW 52000,'Rezerwacja jest opłacona',1;
                END
            UPDATE Reservations
            SET PaymentDate = GETDATE()
            WHERE ReservationID = @reservationID
        COMMIT TRAN PayReservation
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN PayReservation
        DECLARE @msg NVARCHAR(2048) = 'Nie udało się opłacić rezerwacji rezerwacji:' + CHAR(13) +
CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
end
go
```

x. RemoveAttendee - usunięcie uczestnika

```
CREATE PROCEDURE [dbo].[sp_RemoveAttendee] @attendeeID int
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN RemoveAttendee
            IF ((Select Count(AttendeeID)
                 From Attendees
                 WHERE AttendeeID = @attendeeID) < 1)
                BEGIN
                    ;
                    THROW 52000,
                        'Nie znaleziono uczestika o podanym ID',
                        1;
                END
            DELETE from Attendees
            WHERE AttendeeID = @attendeeID
        COMMIT TRAN RemoveAttendee
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN RemoveAttendee
        DECLARE @msg NVARCHAR(2048) =
                'Nie udało się usunąć uczestnika:' +
                CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

y. RemoveOldReservations - usunięcie rezerwacji nieopłaconych do dwóch tygodni

```
CREATE PROCEDURE [dbo].[sp_RemoveOldReservations]
AS
BEGIN
    BEGIN TRY
```

```
        BEGIN TRAN RemoveOldReservations
            DELETE
            FROM Reservations
            WHERE PaymentDate is null
              and DATEDIFF(d, ReservationDate, GETDATE()) >= 14
        COMMIT TRAN RemoveOldReservations
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN RemoveOldReservations
        DECLARE @msg NVARCHAR(2048) = 'Nie udało sie usunąć rezerwacji rezerwacji:' + CHAR(13) +
CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

z. RemoveReservation - usunięcie danej rezerwacji

```
CREATE PROCEDURE [dbo].[sp_RemoveReservation] @reservationID int
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN RemoveReservation
            IF ((SELECT PaymentDate
                 FROM Reservations
                 WHERE ReservationID = @reservationID) is not null)
                BEGIN
                    ;THROW 52000,'Rezerwacja jest opłacona',1;
                END
            IF ((SELECT COUNT(ReservationID)
                 FROM Reservations
                 WHERE ReservationID = @reservationID) < 1)
                BEGIN
                    ;THROW 52000,'Nie znaleziono rezerwacji',1;
                END
            DELETE
            FROM Reservations
            WHERE ReservationID = @reservationID
        COMMIT TRAN RemoveReservation
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN RemoveReservation
        DECLARE @msg NVARCHAR(2048) =
            'Nie udało się usunąć rezerwacji:' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

aa. RemoveReservationDay - usunięcie danego dnia rezerwacji

```
CREATE PROCEDURE [dbo].[sp_RemoveReservationDay] @reservationDayID int
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN RemoveReservationDay
            IF ((SELECT PaymentDate
                 FROM Reservations as R
```

```
                        JOIN [ReservationDays] as RD
                            ON RD.ReservationID = R.ReservationID
                 WHERE @reservationDayID = RD.ReservationDayID) is not null)
                BEGIN
                    ;
                    THROW 52000,'Rezerwacja jest opłacona',
                        1;
                END
            IF ((SELECT COUNT(RD.ReservationDayID)
                 From [ReservationDays] as RD
                 WHERE @reservationDayID = RD.ReservationDayID) < 1)
                BEGIN
                    ;
                    THROW 52000,'Nie znaleziono rezerwacji',
                        1;
                END
            DELETE
            FROM [ReservationDays]
            WHERE @reservationDayID = ReservationDayID
        COMMIT TRAN RemoveReservationDay
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN RemoveReservationDay
        DECLARE @msg NVARCHAR(2048) =
                'Nie udało się usunąć rezerwacji:' +
                CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

bb. RemoveReservationWorkshop - usunięcie rezerwacji na warsztat

```
CREATE PROCEDURE [dbo].[sp_RemoveReservationWorkshop] @workshopReservationID int
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN RemoveReservationWorkshop
            IF ((SELECT PaymentDate
                 FROM Reservations as R
                        JOIN [ReservationDays] as RD
                            ON RD.ReservationID = R.ReservationID
                        JOIN [WorkshopReservations] as WR
                            ON WR.ReservationDayID = RD.ReservationDayID
                 WHERE WR.WorkshopReservationID = @workshopReservationID) is not null)
                BEGIN
                    ;
                    THROW 52000,'Rezerwacja jest opłacona',
                        1;
                END
            IF ((SELECT COUNT(WR.WorkshopReservationID)
                 FROM [WorkshopReservations] as WR
                 WHERE WR.WorkshopReservationID = @workshopReservationID) < 1)
                BEGIN
                    ;
                    THROW 52000,'Nie znaleziono rezerwacji',
                        1;
                END
            DELETE
```

```
            FROM [WorkshopAttendees]
            WHERE WorkshopReservationID = @workshopReservationID
            DELETE
            FROM [WorkshopReservations]
            WHERE WorkshopReservationID = @workshopReservationID
        COMMIT TRAN RemoveReservationWorkshop
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN RemoveReservationWorkshop
        DECLARE @msg NVARCHAR(2048) =
                'Nie udało się usunąć rezerwacji:' +
                CHAR(13) + CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
end
go
```

cc. RemoveWorkshopAttendee - usunięcie uczestnika warsztatu

```
CREATE PROCEDURE [dbo].[sp_RemoveWorkshopAttendee] @attendeeID int,
                                                   @WorkshopReservationID int
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN RemoveWorkshopAttendee
            IF ((SELECT COUNT(AttendeeID)
                 FROM [WorkshopAttendees]
                 WHERE AttendeeID = @attendeeID
                   and @WorkshopReservationID = WorkshopReservationID) < 1)
                BEGIN
                    ;THROW 52000,'Nie znaleziono połączenia między warsztatem a uczestnikiem', 1;
                END
            DELETE from [WorkshopAttendees]
            WHERE AttendeeID = @attendeeID
              and @WorkshopReservationID = WorkshopReservationID
        COMMIT TRAN RemoveWorkshopAttendee
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN RemoveWorkshopAttendee
        DECLARE @msg NVARCHAR(2048) = 'Nie udało się usunąć rezerwacji:' + CHAR(13) + CHAR(10) +
ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

dd. UpdateConference - zaktualizowanie danych konferencji

```
CREATE PROCEDURE [dbo].[sp_UpdateConference] @conferenceID int,
                                             @ConferenceName varchar(255) = null,
                                             @studentDiscount float = null,
                                             @Address varchar(255) = null,
                                             @postalCode varchar(255) = null,
                                             @limit int = null,
                                             @price money = null,
                                             @city varchar(255) = null,
                                             @country varchar(255) = null
AS
```

```
BEGIN
   BEGIN TRY
       BEGIN TRAN UpdateConference
           if @limit is not null
               begin
                   IF ((Select COUNT(ConferenceDayID)
                        FROM [ConferencesDays]
                        WHERE conferenceID = @conferenceID
                          and dbo.sf_GetConferenceDay Used Places(ConferenceDayID) > @limit) > 0)
                       BEGIN
                           ;THROW 52000,'Nie mozna zmniejszc ilosc miejsc poniżej liczby osób które
już zarezerwowały konferencje', 1;
                       END


                   UPDATE Conferences
                   SET Limit = @limit
                   WHERE ConferenceID = @conferenceID
               end
           if @ConferenceName is not null
               begin
                   Update Conferences set ConferenceName = @ConferenceName where ConferenceID =
@conferenceID
               end
           if @studentDiscount is not null
               begin
                   Update Conferences set StudentDiscount = @studentDiscount where ConferenceID =
@conferenceID
               end
           if @Address is not null
               begin
                   Update Conferences set Address = @Address where ConferenceID = @conferenceID
               end
           if @postalCode is not null
               begin
                   Update Conferences set PostalCode = @postalCode where ConferenceID =
@conferenceID
               end
           if @price is not null
               begin
                   Update Conferences set Price = @price where ConferenceID = @conferenceID
               end
           if @city is not null and @country is not null
               begin
                   DECLARE @cityID int
                   EXEC sp_FindCity
                        @city,
                        @country,
                        @cityID = @cityID out
                   Update Conferences set CityID = @cityID where ConferenceID = @conferenceID

               end
       COMMIT TRAN UpdateConference
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN UpdateConference
       DECLARE @msg NVARCHAR(2048) = 'Nie udało się zmienić danych konferencji:' + CHAR(13) +
CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

```
go
```

ee. UpdateReservationDay - zaktualizowanie danych rezerwacji dnia konferencji

```
CREATE PROCEDURE [dbo].[sp_UpdateReservationDay] @reservationDayID int,
                                                 @normalTickets int,
                                                 @studentTickets int
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN UpdateReservationDay
            IF ((select count(*) from ReservationDays join Reservations R2 on
ReservationDays.ReservationID = R2.ReservationID join IndividualClients IC on R2.ClientID =
IC.ClientID
               where ReservationDayID = @reservationDayID
               )>0)
                  BEGIN
                   ;THROW 52000,'Nie można zmienić ilości miejsc dla osoby prywatnej', 1;
               END

           IF (@normalTickets + @studentTickets = 0)
               BEGIN
                   ;THROW 52000,'Trzeba rezerwować przynajmniej jedno miejsce', 1;
               END

           IF ((SELECt R.PaymentDate
                FROM REservations as R
                         JOIN [ReservationDays] as RD
                              on RD.ReservationID = R.ReservationID
                WHERE RD.ReservationDayID = @reservationDayID) is not null)
               BEGIN
                   ;THROW 52000,'Rezerwacja już opłacona', 1;
               END

           IF (dbo.sf_GetReservationDayNormalUsed(@reservationDayID) > @normalTickets
               or dbo.sf_GetReservationDayStudentUsed(@reservationDayID) > @studentTickets)
               BEGIN
                   ;THROW 52000,'Nie można zmienić na ilość mniejsza niż ilość przypisanych już
użytkowników', 1;
               END
           DECLARE @conferenceDayID int = (SELECT ConferenceDayID
                                           FROM [ReservationDays] as RD
                                           WHERE RD.ReservationDayID = @reservationDayID)
           IF (dbo.sf_GetConferenceDayFreePlaces(@conferenceDayID) <
               @normalTickets + @studentTickets -
dbo.sf_GetReservationDayNormalUsed(@reservationDayID) -
               dbo.sf_GetReservationDayStudentUsed(@reservationDayID))
               BEGIN
                   ;THROW 52000,'Niestety nie ma wystarczającej ilości wolnych miejsc', 1;
               END
           UPDATE [ReservationDays]
           SET NormalTickets  = @normalTickets,
               StudentTickets = @studentTickets
           WHERE ReservationDayID = @reservationDayID
       COMMIT TRAN UpdateReservationDay
   END TRY
   BEGIN CATCH
       ROLLBACK TRAN UpdateReservationDay
```

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

ff. UpdateReservationWorkshop - zaktualizowanie danych rezerwacji warsztatu

```
CREATE PROCEDURE [dbo].[sp_UpdateReservationWorkshop] @workshopReservationID int,
                                                      @normalTickets int,
                                                      @studentTickets int
AS
BEGIN
   BEGIN TRY
       BEGIN TRAN UpdateReservationWorkshop
           IF (@normalTickets + @studentTickets = 0)
               BEGIN
                   ;THROW 52000,'Trzeba rezerwować przynajmniej jedno miejsce', 1;
               END
           IF ((SELECt R.PaymentDate
                FROM REservations as R
                         JOIN [ReservationDays] as RD
                              on RD.ReservationID = R.ReservationID
                         JOIN [WorkshopReservations] as WR
                              ON WR.ReservationDayID = RD.ReservationDayID
                WHERE WR.workshopReservationID = @workshopReservationID) is not null)
               BEGIN
                   ;THROW 52000,'Rezerwacja już opłacona', 1;
               END
           IF (dbo.sf_GetWorkshopReservationNormalUsed(@workshopReservationID) > @normalTickets
               or dbo.sf_GetWorkshopReservationStudentUsed(@workshopReservationID) >
@studentTickets)
               BEGIN
                   ;THROW 52000,'Nie można zmienić na ilość mniejsza niż ilość przypisanych już
użytkowników', 1;
               END
           DECLARE @workshopDetailsID int = (SELECT WorkshopDetailsID
                                             FROM [WorkshopReservations] as WR
                                             WHERE WR.WorkshopReservationID =
@workshopReservationID)
           IF (dbo.sf_GetWorkshopDetailsFreePlaces(@workshopDetailsID) <
               @normalTickets + @studentTickets -
dbo.sf_GetWorkshopReservationNormalUsed(@workshopReservationID) -
               dbo.sf_GetWorkshopReservationStudentUsed(@workshopReservationID))
               BEGIN
                   ;THROW 52000,'Niestety nie ma wystarczającej ilości wolnych miejsc', 1;
               END
           DECLARE @reservationDayID int = (SELECT WR.reservationDayID
                                            FROM [WorkshopReservations] as WR
                                            WHERE WR.workshopReservationID = @workshopReservationID)
           IF (dbo.sf_GetReservationDayNormal(@reservationDayID) < @normalTickets or
               dbo.sf_GetReservationDayStudent(@reservationDayID) < @studentTickets)
               BEGIN
                   ;THROW 52000,'Nie można rezerwować większej ilości miejsc niż w rezerwacji na
dzień konferencji', 1;
               END
           UPDATE [WorkshopReservations]
           SET NormalTickets  = @normalTickets,
```

```
                StudentTickets = @studentTickets
            WHERE workshopReservationID = @workshopReservationID
        COMMIT TRAN UpdateReservationWorkshop
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN UpdateReservationWorkshop
        DECLARE @msg NVARCHAR(2048) = 'Nie udało się zmienić danych rezerwacji:' + CHAR(13) +
CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
go
```

gg. UpdateWorkshop - zaktualizowanie danych warsztatu

```
CREATE PROCEDURE [dbo].[sp_UpdateWorkshop] @WorkshopDetailsID int,
                                           @limit int = null,
                                           @price money = null
AS
BEGIN
    BEGIN TRY
        BEGIN TRAN UpdateConference
            if @limit is not null
                begin

                    IF (dbo.sf_GetWorkshopDetailsUsedPlaces(@workshopDetailsID) > @limit)
                        BEGIN
                            ;THROW 52000,'Nie mozna zmniejszc ilosc miejsc poniżej liczby
zarezerwowanych juz miejsc', 1;
                        END
                    DECLARE @conferenceID int = (SELECT conferenceID
                                                 FROM [WorkshopDetails] as WI
                                                          JOIN [ConferencesDays] as CD
                                                               on CD.ConferenceDayID =
WI.ConferenceDayID
                                                 WHERE WI.WorkshopDetailsID = @WorkshopDetailsID)
                    IF (dbo.sf_GetConferenceLimit(@conferenceID) < @limit)
                        BEGIN
                            ;THROW 52000,'Limit miejsc nie może być większa od liczby miejsc na
konferencje', 1;
                        END

                    UPDATE [WorkshopDetails]
                    SET Limit = @limit
                    WHERE WorkshopDetailsID = @WorkshopDetailsID
                end
            if @price is not null
                begin
                    update WorkshopDetails set Price = @price where WorkshopDetailsID =
@WorkshopDetailsID
                end
        COMMIT TRAN UpdateConference
    END TRY
    BEGIN CATCH
        ROLLBACK TRAN UpdateConference
        DECLARE @msg NVARCHAR(2048) = 'Nie udało się zaktualizować warsztatu:' + CHAR(13) + CHAR(10)
+ ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
```

```
END
go
```

hh. GenerateInvoiceForReservationID - generuje fakturę dla danego zamówienia

```
CREATE FUNCTION fp_GenerateInvoiceForReservationID(
   @ReservationID int
)
   RETURNS TABLE
       AS
       RETURN
           (
               SELECT CONCAT('Konferencja: ', ConferenceName,
                             ',Data: ', Date, ' - ', NormalTickets, ' biletów normalnych')   AS
Name,
                      dbo.sf_GetReservationNormalTicketPrice(@ReservationID) * NormalTickets AS COST
               FROM dbo.Reservations
                        INNER JOIN dbo.[ReservationDays]
                                   ON [ReservationDays].ReservationID = Reservations.ReservationID
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = Reservations.ConferenceID
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceDayID =
                                                            [ReservationDays].ConferenceDayID
               WHERE Reservations.ReservationID = @ReservationID
                 AND NormalTickets > 0
                 AND EXISTS(SELECT *
                            FROM [dbo].[Reservations]
                            WHERE ReservationID = @ReservationID)
               UNION ALL
               SELECT CONCAT('Konferencja: ', ConferenceName, ',Data: ', Date, ' - ',
StudentTickets,
                             ' biletów ulgowych')            AS Name,
                      dbo.sf_GetReservationNormalTicketPrice(@ReservationID) *
                      StudentTickets * (1 - StudentDiscount) AS COST
               FROM dbo.Reservations
                        INNER JOIN dbo.[ReservationDays]
                                   ON [ReservationDays].ReservationID = Reservations.ReservationID
                        INNER JOIN dbo.Conferences
                                   ON Conferences.ConferenceID = Reservations.ConferenceID
                        INNER JOIN dbo.[ConferencesDays] ON [ConferencesDays].ConferenceDayID =
                                                            [ReservationDays].ConferenceDayID
               WHERE Reservations.ReservationID = @ReservationID
                 AND StudentTickets > 0
                 AND EXISTS(SELECT *
                            FROM [dbo].[Reservations]
                            WHERE ReservationID = @ReservationID)
               UNION ALL
               SELECT CONCAT('Warsztat: ', WorkshopName,
                             ',Data: ', Date, ' - ', WorkshopReservations.NormalTickets,
                             ' biletów normalnych')                              AS Name,
                      WorkshopReservations.NormalTickets * WorkshopDetails.Price AS Cost
               FROM dbo.[WorkshopReservations]
                        INNER JOIN dbo.[WorkshopDetails] ON [WorkshopDetails].WorkshopDetailsID =
                                                            [WorkshopReservations].WorkshopDetailsID
                        join ReservationDays RD on WorkshopReservations.ReservationDayID =
RD.ReservationDayID
                        join Reservations R2 on RD.ReservationID = R2.ReservationID
                        join Workshops W on WorkshopDetails.WorkshopID = W.WorkshopID
                        join ConferencesDays CD on RD.ConferenceDayID = CD.ConferenceDayID
               WHERE R2.ReservationID = @ReservationID
```

```
                AND WorkshopReservations.NormalTickets > 0
                AND EXISTS(SELECT *
                           FROM [dbo].[Reservations]
                           WHERE ReservationID = @ReservationID)
            UNION ALL
            SELECT CONCAT('Warsztat: ', WorkshopName,
                          ',Data: ', Date, ' - ', WorkshopReservations.StudentTickets, ' biletów
ulgowych') AS Name,
                   WorkshopReservations.StudentTickets * WD.Price * (1 - StudentDiscount)
AS Cost
            FROM dbo.[WorkshopReservations]
                     join ReservationDays D on WorkshopReservations.ReservationDayID =
D.ReservationDayID
                     join Reservations R3 on D.ReservationID = R3.ReservationID
                     join Conferences C on R3.ConferenceID = C.ConferenceID
                     join ConferencesDays CD2 on C.ConferenceID = CD2.ConferenceID
                     join WorkshopDetails WD on CD2.ConferenceDayID = WD.ConferenceDayID
                     join Workshops W2 on WD.WorkshopID = W2.WorkshopID

            WHERE R3.ReservationID = @ReservationID
              AND WorkshopReservations.StudentTickets > 0
              AND EXISTS(SELECT *
                         FROM [dbo].[Reservations]
                         WHERE ReservationID = @ReservationID)
            UNION ALL
            SELECT 'Suma' AS NAME,
                   dbo.sf_GetReservationCost(@ReservationID)
                          AS COST
        )
go
```

ii. Update Person

```
CREATE PROCEDURE [dbo].[sp_InsertPerson]
        @firstname varchar(255) = NULL,
        @lastname varchar(255) = NULL,
        @phone varchar(255) = NULL,
        @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) =
                'Błąd dodania osoby do bazy:' + CHAR(13)+CHAR(10) + ERROR_MESSAGE ();
                THROW 52000,@msg, 1;
        END CATCH
END
go
```

8.Generator :

Do wygenerowania danych dla naszej bazy użyliśmy generatora SQL Data Generator firmy RedGate, poprawności wprowadzanych danych zapewniliśmy naszymi funkcjami pomocniczymi, które usunęły sprzeczne krotki.

Podsumowanie dodanych danych :

Attendees - 5600,
Cities - 1000,
Clients - 4700,
Companies - 300,
Conferences - 300,
Conferences Days - 1080,
Countries - 100,
Employees - 5000,
Individual Clients - 4400,
Organizers - 50,
Person - 9700,
Prices - 930,
Reservation Days - 5620,
Reservation - 3000,
Students - 500,
Workshop Attendees - 2000,
Workshop Details - 1280,
Workshop Reservations - 2400,
Workshops - 900

9.Przykładowe użycia funkcji :

a. Dodanie nowego organizatora

```
use u_surjak
exec sp_AddOrganizer 'Akamai', '1231231234','Jakub
Kowalski','perzylo1@gmail.com','123141123','Bujaka
1','Krakow','Poland','30-498',@organizerID out
```

b. Dodanie nowego warsztatu

```
exec sp_AddWorkshop @organizerID,'React in 1 hour','Lorem ipsum ',@workshopID
out
print @workshopID
```

c. Dodanie nowej konferencji

```
exec sp_AddConference @organizerID,'React',0.3,'Bajkowa
11','Krakow','Poland','30-133','20200303','20200305',100,100,null
set @conferenceID = (select ConferenceID from Conferences where ConferenceName =
'Angular')
```

d. Dodanie danych do nowo utworzonego warsztatu

```
exec sp_AddWorkshopDetails
901,301,'20200304','15:00','16:30',30,50,@workshopDetailsID out
print @workshopDetailsID
```

e. Dodanie progu cenowego do danej konferencji

```
exec sp_AddPrice 301,'20200220','20200302',0.3
```

f. Dodanie klienta z danej firmowego

```
exec sp_AddCompanyClient 'Facebook','1431134244','Bill
Gates','113123424','goog22lae@gmail.com','Krakowska
4','Krakow','Poland','31-133',null
print @clientID
```

g. Dodanie rezerwacji na dany warsztat dla firmy

```
exec sp_AddIndividualClient 'Adam','Małysz','333444555','ala23@interia.pl','Ala
12','Krakow', 'Poland','33-444',null

exec sp_AddEmployee 4747031,'Kamila','Jakubinska','321616777',@personId out

exec sp_AddReservation 301,4747040, @resID out

exec sp_AddReservation 301,4747031,@resIdFirm out

exec sp_AddReservationDay 3003,1076,3,2,'123123 123555',@redDayIdFirm out

declare @wResIdForm int
exec sp_AddReservationWorkshop 42496,60001,2,1,@wResIdForm out
exec sp_AddWorkshopAttendee 2402,5603
```

h. Dodanie rezerwacji na dany warsztat dla osoby indywidualnej

```
exec sp_AddReservationDayIndividual 4747040,3002,1076,'444222',@resDayId out

exec sp_AddReservationWorkshopIndividual 42495,60001,@resWorkId out
```