“System wspomagający firmę zarządzającą konferencjami”

Projekt realizowany w ramach przedmiotu Podstawy Baz Danych

Akademia Górniczo-Hutnicza
Wydział Informatyki, Elektroniki i Telekomunikacji
Studia I stopnia Informatyka

Wykonawcy:
Adrian Beściak
Dawid Białka

Grupa nr 8 - prowadzący: dr Leszek Siwik

Spis treści

Opis funkcji systemu

Ogólne informacje o systemie

Firma organizuje konferencje, które mogą być jedno- lub kilkudniowe. Klienci rejestrują konto na stronie www, gdzie podają swoje dane i ew. zaznaczają, że są klientem firmowym. Klient prywatny ma możliwość z jednego konta zarejestrować na konferencję innych użytkowników, np. Swoją rodzinę. Każdy uczestnik konferencji otrzymuje identyfikator imienny (jak jest uczestnikiem firmowym to też informacja o firmie na nim). Dla konferencji kilkudniowych, uczestnicy mogą rejestrować się na dowolne z tych dni, ale dni te muszą być spójne (np. poniedziałek i wtorek, ale nie poniedziałek i środa). Na konferencję można rezerwować się do ostatniego dnia przed konferencją do godziny 24:00. Termin na uiszczenie opłaty jest taki sam.

Warsztaty

Aby uczestniczyć w danym warsztacie, trzeba być zapisanym w tym dniu na konferencję . Kilka warsztatów może trwać równocześnie, ale uczestnik nie może zarejestrować się na więcej niż jeden warsztat, który trwa w tym samym czasie. Jest także ograniczona ilość miejsc na każdy warsztat i na każdy dzień konferencji. Część warsztatów może być płatna, a część jest darmowa. Na warsztaty zapisujemy się przed konferencją, w dniu konferencji nie ma możliwości rezygnacji z uczestnictwa z warsztatów i zapisywania się na nowe. Mogą istnieć konferencje bez żadnych warsztatów.

Opłaty

Opłata za udział w konferencji zależy nie tylko od zarezerwowanych usług, ale także od terminu ich rezerwacji - jest kilka progów ceny (progi ceny dotyczą tylko udziału w konferencji, cena warsztatów jest stała) i im bliżej rozpoczęcia konferencji, tym cena jest wyższa (jest także zniżka procentowa dla studentów i w takim wypadku przy rezerwacji trzeba podać nr legitymacji studenckiej). Progi ceny są następujące:

- 7 dni przed konferencją – 100 % ceny
- 14 dni przed konferencją – 85 % ceny
- 21 dni przed konferencją – 75 % ceny

Na zapłatę klienci mają tydzień od rezerwacji na konferencję - jeśli do tego czasu nie pojawi się opłata, rezerwacja jest anulowana. Zniżka dla studenta – 10 % od ceny obniżonej, czyli np. na 14 dni przed konferencją student płaci 0,9 * 0,85 czyli 76,5 %.

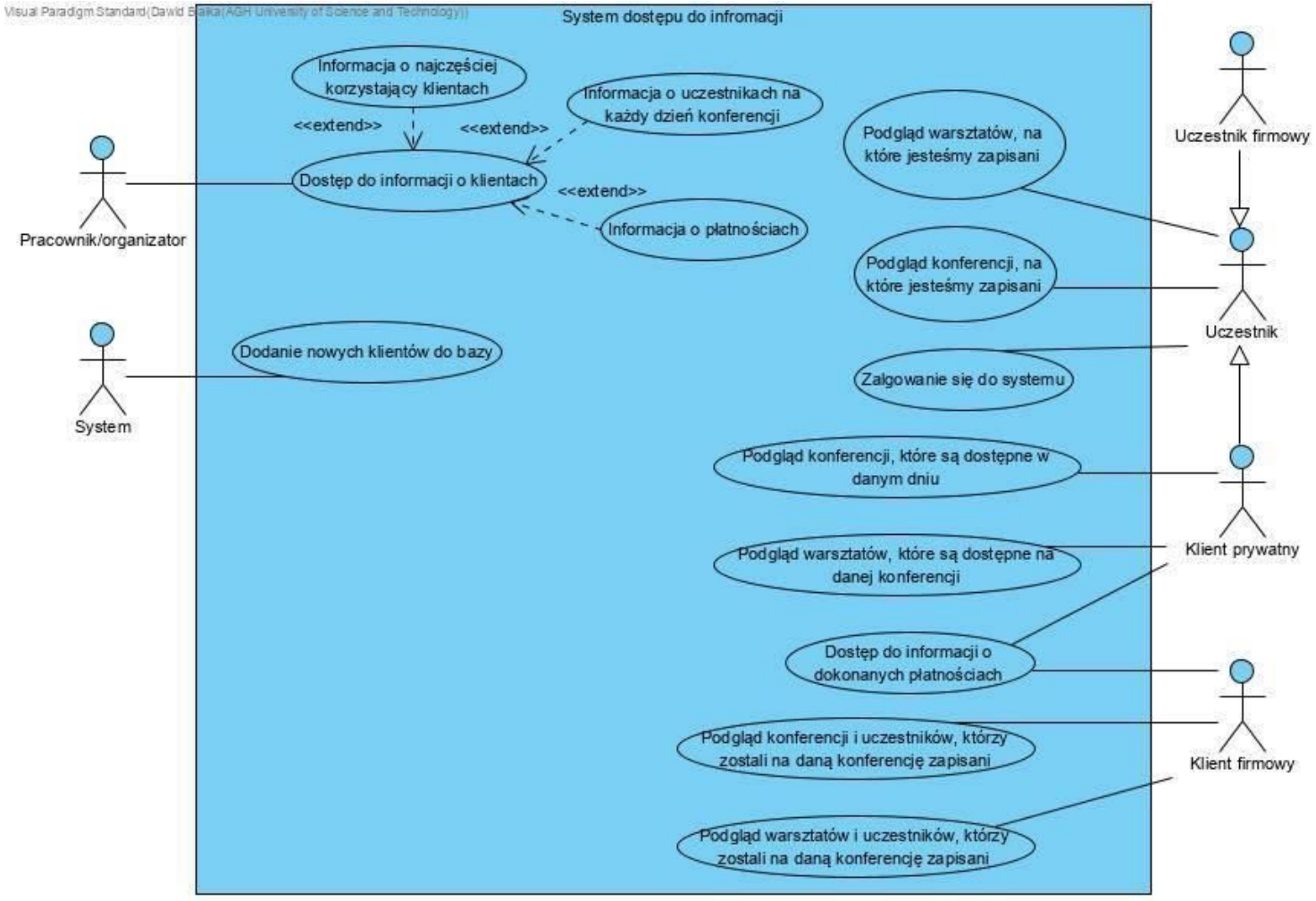
Raporty

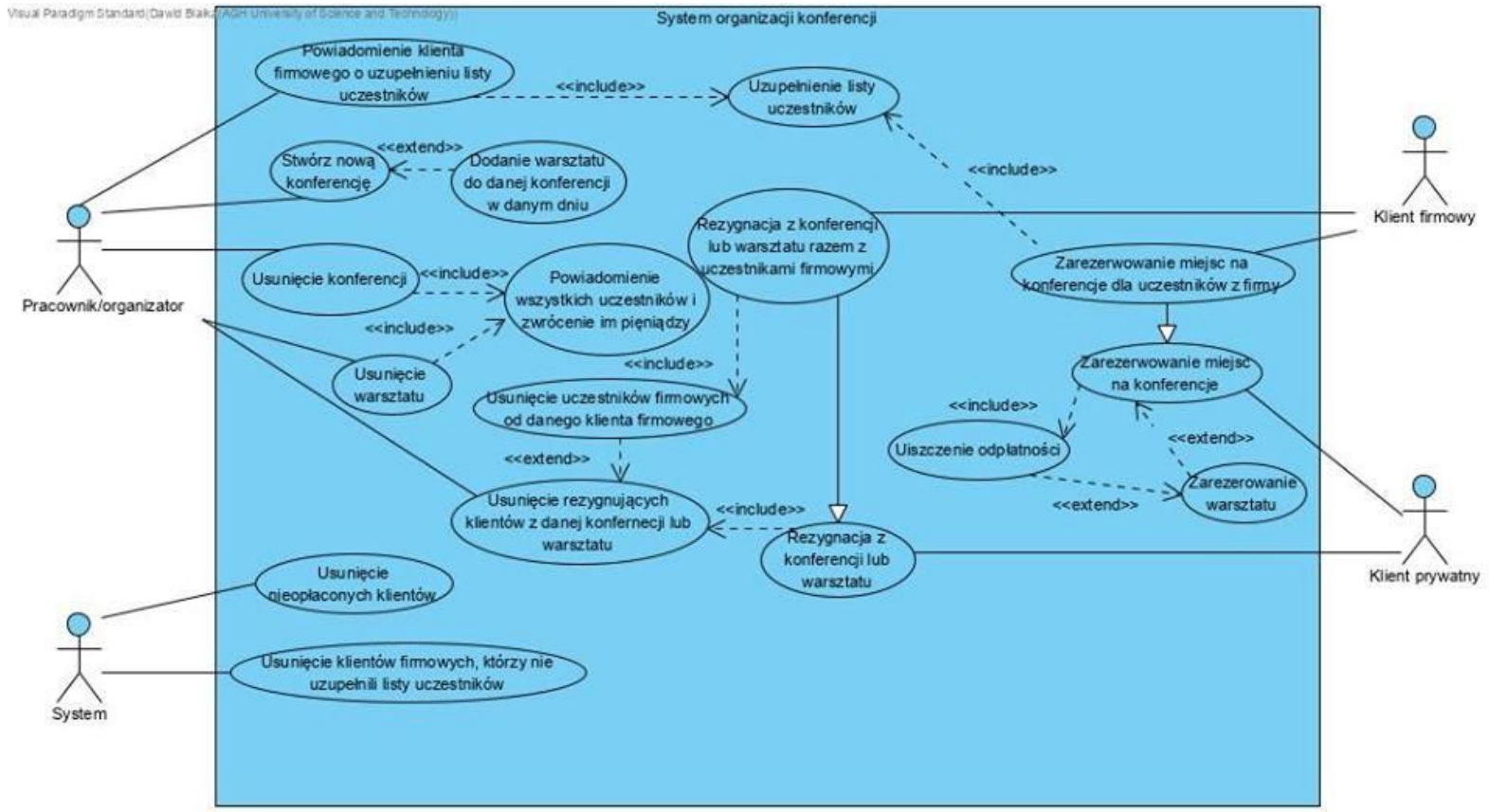
Dla organizatora najbardziej istotne są listy osobowe uczestników na każdy dzień konferencji i na każdy warsztat, a także informacje o płatnościach klientów. Ponadto organizator chciałby mieć informację o klientach, którzy najczęściej korzystają z jego usług. Organizator ma dostęp do listy osobowej uczestników na każdy dzień konferencji i każdy warsztat oraz informacji o płatnościach klientów.

W systemie wyróżniamy następujących użytkowników:

- Klient prywatny
- Klient firmowy
- Uczestnik od klienta firmowego
- Pracownik
- System

Role poszczególnych użytkowników w systemie zostały przedstawione na poniższych diagramach:





UC - Pracownik/organizator

- otrzymanie powiadomienia, jeśli na 2 tygodnie przed konferencją firma nie podała pełnej listy uczestników z systemu i ustalenie danych uczestników firmowych, które nie zostały podane
- generowanie raportów zawierających listy osobowe uczestników na każdy dzień konferencji i na każdy warsztat, informacje o płatnościach klientów
- informacja o klientach najczęściej korzystających z usług
- tworzenie i usuwanie konferencji i warsztatów
- usuwa rezygnujących uczestników opłaconych i nieopłaconych z danej konferencji

UC - Klient prywatny

- stworzenie konta na stronie www wraz podaniem następujących informacji : login, hasło, imię, nazwisko, e-mail, telefon, ulica, miasto, kod, kraj, ewentualnie nr legitymacji studenckiej
- podgląd konferencji, na które można się zapisać w danym dniu i informacja o liczbie wolnych miejsc
- podgląd warsztatów, na które można się zapisać, będąc na danej konferencji
- rezerwacja na daną konferencję na dany dzień (można wybrać wiele dni, jeśli konferencja jest wielodniowa, ale terminy muszą być spójne)
- rezerwacja na warsztat, który jest dostępny dla wcześniej wybranej przez klienta konferencji (można wybrać wiele warsztatów z danej konferencji, ale nie takich, które odbywają się w tym samym czasie)
- rezygnacja z rezerwacji na konferencję lub warsztat (po dokonaniu zapłaty nie ma możliwości uzyskania zwrotu pieniędzy, gdy chcemy zrezygnować z uczestnictwa w konferencji lub warsztacie)
- uiszczenie opłaty za udział w konferencji i warsztatach w terminie 7 dni od daty rezerwacji na konferencję (im później rezerwujemy się na konferencję, tym więcej trzeba płacić. Opłaty za warsztaty są niezależne od terminu rezerwacji)
- podgląd informacji o konferencjach, na które jesteśmy zapisani i informacja o zapłacie, którą trzeba jeszcze dokonać i do kiedy lub informacja o tym, że zapłata została uiszczona
- dla danej konferencji podgląd informacji o warsztatach, na które jesteśmy zapisani

UC - Klient firmowy

- stworzenie konta na stronie www z zaznaczeniem, że klientem jest firma, wraz podaniem następujących informacji : login, hasło, e-mail, telefon, ulica, miasto, kod, kraj, nazwa firmy
- podgląd konferencji, na które można się zapisać w danym dniu i informacja o liczbie wolnych miejsc
- podgląd warsztatów, na które można się zapisać, będąc na danej konferencji
- rezerwacja wybranej liczby miejsc na daną konferencję (nie trzeba od razu podawać listy uczestników) na dany dzień (można wybrać wiele dni, jeśli konferencja jest wielodniowa, ale te dni muszą być spójne)
- rezerwacja wybranej liczby miejsc na warsztat, który jest dostępny dla wcześniej wybranej przez klienta konferencji (można wybrać wiele warsztatów z danej konferencji, ale nie takich, które odbywają się w tym samym czasie. Nie można zarezerwować większej liczby miejsc na dany warsztat niż zostało zarezerwowanych uczestników na tą konferencję)
- rezygnacja z rezerwacji na konferencję lub warsztat (po dokonaniu zapłaty nie ma możliwości uzyskania zwrotu pieniędzy, gdy chcemy zrezygnować z uczestnictwa w konferencji lub warsztacie. Można anulować rezerwację dla pojedynczych osób lub dla całej firmy.)
- w przypadku nie podania pełnej listy uczestników na 2 tygodnie przed konferencją, klient zostaje powiadomiony o tym przez pracownika. W liście podajemy imię i

nazwisko, e-mail, telefon dla każdego uczestnika. Jeśli na 7 dni przed konferencją dane nie zostaną uzupełnione, to rezerwacja jest usuwana automatycznie
- uiszczenie opłaty za udział w konferencji i warsztatach w terminie 7 dni od daty rezerwacji na konferencję (im później rezerwujemy się na konferencję, tym więcej trzeba płacić. Opłaty za warsztaty są niezależne od terminu rezerwacji)
- podgląd informacji o wybranych przez nas konferencjach i uczestnikach, których przypisaliśmy do danej konferencji i informacja o zapłacie, którą trzeba jeszcze dokonać i do kiedy lub informacja o tym, że zapłata została uiszczona
- dla danej konferencji podgląd informacji o wybranych przez nas warsztatach i uczestnikach, których przypisaliśmy do danego warsztatu

UC - Uczestnik
- zalogowanie się na stronę www używając otrzymanych od swojej firmy loginu i hasła (brak zniżki, jeśli uczestnik jest studentem)
- podgląd konferencji, na które jesteśmy zapisani w danym dniu
- podgląd warsztatów, na które jesteśmy zapisani w danym dniu

UC - System
- usuwanie uczestników nieopłaconych w ciągu 7 dni od rezerwacji
- usuwanie uczestników firmowych, którzy nie uzupełnili listy uczestników na 7 dni przed konferencją
- dodanie nowych klientów do bazy (konto nie może zostać utworzone na tą samą osobę/firmę więcej niż raz)

Schemat bazy danych

Na poniższym schemacie został przedstawiony diagram bazy danych używanej w systemie:

Relacje znajdujące się w systemie:

1. **Address**
    - **AddressID: int -** pole będące kluczem głównym z autoinkrementacją
    - **Country: varchar(50) -** pole przechowujące kraj
    - **Region: varchar(50) -** pole przechowujące region
    - **City: varchar(50) -** pole przechowujące miasto
    - **PostalCode: varchar(50) -** pole przechowujące kod pocztowy
    - **Street: varchar(50) -** pole przechowujące ulicę
    - **Address: varchar(50) -** pole przechowujące numer domu/mieszkania

    Ta relacja służy do przechowywania adresu, używanego później przez relacje Conference, PrivateClients oraz CompanyClients. Żadne jej pole nie może być nullem.
    Kod generujący tabelę:

```
create table Address
(
  AddressID int identity,
```

```
  Country varchar(50) not null,
  Region varchar(50) not null,
  City varchar(50) not null,
  PostalCode varchar(50) not null,
  Street varchar(50) not null,
  Address varchar(50) not null
CONSTRAINT [PK_AddressID_CINEX] PRIMARY KEY CLUSTERED  (  AddressID ASC )WITH
(PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [Address_city_street_city_postalcode_uindex] ON
dbo.Address
      (  City ASC,
       Street ASC,
       PostalCode ASC )
   WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


ALTER TABLE dbo.Address WITH NOCHECK ADD CONSTRAINT [CHK_Address_city] CHECK
((ltrim([city])<>'')) GO
ALTER TABLE dbo.Address CHECK CONSTRAINT [CHK_Address_city] GO

ALTER TABLE dbo.Address  WITH NOCHECK ADD CONSTRAINT [CHK_Address_street] CHECK
((ltrim([Street])<>'')) GO
ALTER TABLE dbo.Address CHECK CONSTRAINT [CHK_Address_street] GO

ALTER TABLE dbo.Address  WITH NOCHECK ADD CONSTRAINT [CHK_Address_PostalCode]
CHECK ((Address.PostalCode like '[0-9][0-9]-[0-9][0-9][0-9]')) GO
ALTER TABLE [dbo].[Address] CHECK CONSTRAINT [CHK_Address_PostalCode] GO
```

2. **PrivateClients**
    - **PrivateClientID: int -** pole będące kluczem głównym z autoinkrementacją
    - **AddressID: int -** pole będące kluczem obcym, odwołującym się do tabeli Address
    - **Firstname: varchar(50) -** pole przechowujące imię klienta prywatnego
    - **Lastname: varchar(50) -** pole przechowujące nazwisko klienta prywatnego
    - **Email: varchar(50) -** pole przechowujące adres email klienta prywatnego
    - **Phone: varchar(50) -** pole przechowujące numer telefonu klienta prywatnego

    Ta relacja przechowuje klientów prywatnych, którzy korzystają z naszych usług. Do przechowywania adresu danego klienta wykorzystuje krotkę zawartą w tabeli Address. Żadne jej pole nie może być nullem.
    Kod generujący tabelę:

```
create table PrivateClients
(
  PrivateClientID int identity,
```

```
  AddressID int not null
    constraint FKPrivateCli914143
      references Address,
  Firstname varchar(50) not null,
  Lastname varchar(50) not null,
  Email varchar(50) not null,
  Phone varchar(50) not null
CONSTRAINT [PK_PrivateClients_CINDEX] PRIMARY KEY CLUSTERED  (  PrivateClientID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)


CREATE NONCLUSTERED INDEX [PrivateClients_phone_studentcardid_uindex] ON
dbo.PrivateClients
      (AddressID ASC,
       Lastname ASC,
       Phone ASC
       )
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


ALTER TABLE dbo.PrivateClients WITH NOCHECK ADD CONSTRAINT
[CHK_PrivateClients_firstname] CHECK  ((ltrim(FirstName)<>'')) GO
ALTER TABLE dbo.PrivateClients CHECK CONSTRAINT [CHK_PrivateClients_firstname] GO

ALTER TABLE dbo.PrivateClients WITH NOCHECK ADD CONSTRAINT
[CHK_PrivateClients_lastname] CHECK  ((ltrim(LastName)<>'')) GO
ALTER TABLE dbo.PrivateClients CHECK CONSTRAINT [CHK_PrivateClients_lastname] GO

ALTER TABLE dbo.PrivateClients WITH NOCHECK ADD CONSTRAINT [CHK_PrivateClients_phone]
CHECK  ((ltrim(Phone)<>'')) GO
ALTER TABLE dbo.PrivateClients CHECK CONSTRAINT [CHK_PrivateClients_phone] GO

ALTER TABLE dbo.PrivateClients WITH NOCHECK ADD CONSTRAINT [CHK_PrivateClients_email]
CHECK  ((Email like '%_@_%_.__%')) GO
ALTER TABLE dbo.PrivateClients CHECK CONSTRAINT [CHK_PrivateClients_email] GO
```

3. **CompanyClients**
    - **CompanyID: int -** pole będące kluczem głównym tabeli z autoinkrementacją
    - **AddressID: int -** pole będące kluczem obcym, odwołującym się do tabeli Address, gdzie jest przechowywany adres siedziby klienta
    - **CompanyName: varchar(50) -** pole zawierające nazwę firmy
    - **ContactName: varchar(50) -** pole zawierające dane osoby z tej firmy, która jest odpowiedzialna za kontakt z nami
    - **Phone: varchar(50) -** pole zawierające numer telefonu do klienta
    - **Fax: varchar(50) -** pole zawierające numer faxu klienta
    - **Email: varchar(50) -** pole zawierające adres email klienta

Ta relacja przechowuje klientów firmowych, którzy korzystają z naszych usług. Do przechowywania adresu firmy wykorzystujemy krotkę zawartą w tabeli Address. Żadne pole tej tabeli nie może być nullem.
Kod generujący tabelę:

```
create table CompanyClients
(
  CompanyID int identity,
  AddressID int not null
    constraint FKCompanyCli848384
      references Address,
  CompanyName varchar(50) not null,
  ContactName varchar(50) not null,
  Phone varchar(50) not null,
  Fax varchar(50) not null,
  Email varchar(50) not null
CONSTRAINT [PK_CompanyClients_CINDEX] PRIMARY KEY CLUSTERED  (  CompanyID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [CompanyClients_addressid_contactname_phone_email_uindex]
ON
dbo.CompanyClients
      (  AddressID ASC,
       ContactName ASC,
       Phone ASC
      )
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


ALTER TABLE dbo.CompanyClients WITH NOCHECK ADD CONSTRAINT
[CHK_CompanyClients_email] CHECK  ((Email like '%_@_%_.__%')) GO
ALTER TABLE dbo.CompanyClients CHECK CONSTRAINT [CHK_CompanyClients_email] GO

ALTER TABLE dbo.CompanyClients  WITH NOCHECK ADD CONSTRAINT
[CHK_CompanyClients_companyname] CHECK  ((ltrim(CompanyName)<>'')) GO
ALTER TABLE dbo.CompanyClients CHECK CONSTRAINT [CHK_CompanyClients_companyname]
GO

ALTER TABLE dbo.CompanyClients  WITH NOCHECK ADD CONSTRAINT
[CHK_CompanyClients_contactname] CHECK  ((ltrim(ContactName)<>'')) GO
ALTER TABLE dbo.CompanyClients CHECK CONSTRAINT [CHK_CompanyClients_contactname] GO

ALTER TABLE dbo.CompanyClients  WITH NOCHECK ADD CONSTRAINT
[CHK_CompanyClients_phone] CHECK  ((ltrim(Phone)<>'')) GO
ALTER TABLE dbo.CompanyClients CHECK CONSTRAINT [CHK_CompanyClients_contactname] GO

ALTER TABLE dbo.CompanyClients  WITH NOCHECK ADD CONSTRAINT
[CHK_CompanyClients_fax] CHECK  ((ltrim(Fax)<>'')) GO
```

```
ALTER TABLE dbo.CompanyClients CHECK CONSTRAINT [CHK_CompanyClients_fax] GO
```

4. **Clients**
    - **ClientID: int -** pole będące kluczem głównym tabeli, autoinkrementowane
    - **PrivateClientID: int -** pole będące kluczem obcym tabeli, odnoszącym się do tabeli PrivateClients
    - **CompanyID: int -** pole będące kluczem obcym tabeli, odnoszącym się do krotki w tabeli CompanyClients

    Ta relacja pozwala nam w dalszych kwerendach korzystać z jednego klucza, niezależnego od tego czy klient jest klientem prywatnym, czy firmowym. Zawsze dokładnie jeden z kluczy obcych będzie pusty, ponieważ to obiekt z tego drugiego będzie reprezentowany przez klucz główny tabeli Clients.
    Kod generujący tabelę:

```
create table Clients
(
  ClientID int identity,
  PrivateClientID int
     constraint FKClients521617
       references PrivateClients,
  CompanyID int
     constraint FKClients597535
       references CompanyClients
CONSTRAINT [PK_Clients_CINDEX] PRIMARY KEY CLUSTERED  (  ClientID ASC )WITH
(PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [Clients_privateclientid_companyid_uindex] ON
dbo. Clients
      (PrivateClientID ASC,
       CompanyID ASC
       )
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)
```

5. **Conference**
    - **ConferenceID: int -** pole będące kluczem głównym tabeli Conference, autoinkrementowane
    - **AddressID: int -** pole będące kluczem obcym, odnoszącym się do tabeli Address, gdzie jest przechowywany adres pod jakim odbywa się konferencja.
    - **Name: varchar(50) -** pole przechowujące nazwę konferencji
    - **Begin: date -** pole przechowujące datę rozpoczęcia się konferencji
    - **End: date -** pole przechowujące datę zakończenia się konferencji

- **Cancelled: bit -** pole przechowujące informację czy konferencja jest odwołana - jeśli tak, to przyjmuje wartość 1, a jeśli nie to 0

Ta relacja przechowuje podstawowe dane dotyczące całej konferencji takie jak nazwa, adres i czas, w jakim się odbywa.
Kod generujący tabelę:

```
create table Conference
(
  ConferenceID int identity,
  AddressID int not null
    constraint FKConference177987
      references Address,
  Name varchar(50) not null,
  Begin date not null,
  End date not null,
  Cancelled bit constraint DF_Cancelled default 0 not null
CONSTRAINT [PK_Conference_CINDEX] PRIMARY KEY CLUSTERED  (  ConferenceID ASC )WITH
(PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [Conference_addressid_name_begin_end__cancelled_uindex] ON
dbo.Conference
      (  AddressID ASC,
       'Begin' ASC,
       End ASC,
       Cancelled ASC)
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
     DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


ALTER TABLE dbo.Conference WITH NOCHECK ADD CONSTRAINT [CHK_Conference_name]
CHECK  ((ltrim(Name)<>'')) GO
ALTER TABLE dbo.Conference CHECK CONSTRAINT [CHK_Conference_name] GO

-- ALTER TABLE dbo.Conference  WITH NOCHECK ADD CONSTRAINT [CHK_Conference_begin]
CHECK ((Conference.Begin > GETDATE())) GO
-- ALTER TABLE dbo.Conference CHECK CONSTRAINT [CHK_Conference_begin] GO --do triggera

ALTER TABLE dbo.Conference  WITH NOCHECK ADD CONSTRAINT [CHK_Conference_end]
CHECK  (Conference.End > Conference.[Begin]) GO
ALTER TABLE dbo.Conference CHECK CONSTRAINT [CHK_Conference_end] GO
```

6. **ConferenceDay**
   - **ConferenceDayID: int -** pole będące kluczem głównym relacji, autoinkrementowane
   - **ConferenceID: int -** pole będące kluczem obcym, odnoszącym się do tabeli Conference

- **Date: date -** pole przechowujące dzień, w którym odbywa się ten dzień konferencji. Jest to ważny element, ponieważ konferencje mogą odbywać się przez kilka dni i musimy jakoś odróżnić pojedyncze dni od siebie
- **StartTime: time -** godzina rozpoczęcia konferencji w tym dniu, każdego dnia może być inna
- **EndTime: time -** godzina zakończenia konferencji w tym dniu, każdego dnia może być inna
- **MaxAmountOfPeople: int -** maksymalna ilość osób, które mogą się zapisać na dany dzień konferencji
- **Cancelled: bit -** informacja czy dany dzień konferencji nie został odwołany - jeśli tak, to przyjmuje wartość 1, a jeśli nie to 0. Domyślnie to pole przyjmuje wartość 0.
- **FeeForDay: numeric(16,2) -** pełna należność za dany dzień konferencji - podstawa do naliczenia zniżek przy obliczaniu kwoty do zapłaty

Ta relacja przechowuje informacje dotyczące szczegółów organizacyjnych, które mogą być odmienne dla poszczególnych dni konferencji. Nowe krotki w tej tabeli są automatycznie tworzone po utworzeniu krotki w tabeli Conference.
Kod generujący tabelę:

```
create table ConferenceDay
(
  ConferenceDayID int identity,
  ConferenceID int not null
    constraint FKConference184794
      references Conference,
  Date date not null,
  StartTime time not null,
  EndTime time not null,
  MaxAmountOfPeople int not null,
  Cancelled bit default 0 not null,
  FeeForDay numeric(16,2) not null
CONSTRAINT [PK_ConferenceDay_CINDEX] PRIMARY KEY CLUSTERED  (  ConferenceDayID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [ConferenceDay_conferenceid_cancelled_uindex] ON
dbo.ConferenceDay
      (  ConferenceID ASC,
       Cancelled ASC
      )
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


--ALTER TABLE dbo.ConferenceDay WITH NOCHECK ADD CONSTRAINT
[CHK_ConferenceDay_date] CHECK  ((ConferenceDay.Date > GETDATE())) GO
```

```
--ALTER TABLE dbo.ConferenceDay CHECK CONSTRAINT [CHK_ConferenceDay_date] GO do
triggera

ALTER TABLE dbo.ConferenceDay  WITH NOCHECK ADD CONSTRAINT
[CHK_Conference_endtime] CHECK ((ConferenceDay.StartTime < ConferenceDay.EndTime)) GO
ALTER TABLE dbo.ConferenceDay CHECK CONSTRAINT [CHK_Conference_endtime] GO

ALTER TABLE dbo.ConferenceDay  WITH NOCHECK ADD CONSTRAINT
[CHK_Conference_maxamountofpeople] CHECK ((ConferenceDay.MaxAmountOfPeople > 0)) GO
ALTER TABLE dbo.ConferenceDay CHECK CONSTRAINT [CHK_Conference_maxamountofpeople]
GO

ALTER TABLE dbo.ConferenceDay  WITH NOCHECK ADD CONSTRAINT
[CHK_Conference_feeforday] CHECK ((ConferenceDay.FeeForDay > 0)) GO
ALTER TABLE dbo.ConferenceDay CHECK CONSTRAINT [CHK_Conference_feeforday] GO
```

7. **CDReservation**
   - **CDReservationID: int -** klucz główny tabeli, autoinkrementowany
   - **ClientsClientID: int -** klucz obcy, odnoszący się do tabeli Clients, informujący który klient dokonuje rezerwacji
   - **ConferenceDayID: int -** klucz obcy, odnoszący się do tabeli ConferenceDay, informujący na jaki dzień jest dokonywana rezerwacja
   - **ReservationDate: date -** pole przechowujące dzień, w którym dokonano rezerwacji - potrzebna informacja do obliczenia należności za rezerwację
   - **CancellationDate: date -** pole domyślnie będące wartością null, w razie anulowania rezerwacji, zostanie wpisana data anulowania
   - **NumOfParicipants: int -** pole informujące o ilości osób, dla których dokonana została ta rezerwacja
   - **Cancelled: bit -** pole informujące o anulowaniu rezerwacji - jeśli anulowano, to przyjmuje wartość 1, a jeśli nie to 0. Domyślnie to pole przyjmuje wartość 0.
   - **PaidAmount: numeric(16,2) -** pole informujące wysokości wpłaty, która wpłynęła za daną rezerwację.

   Tabela służy do przechowywania rezerwacji na poszczególne dni konferencji oraz kojarzenia ich z klientami, którzy dokonali zakupu.
   Kod generujący tabelę:

```
create table CDReservation
(
  CDReservationID int identity,
  ClientsClientID int not null
    constraint FKCDReservat850490
      references Clients,
  ConferenceDayID int not null
    constraint FKCDReservat710097
      references ConferenceDay,
  ReservationDate date not null,
  CancellationDate date,
  NumOfParicipants int not null,
  Cancelled bit default 'false' not null,
  PaidAmount numeric(16,2) not null
CONSTRAINT [PK_CDReservation_CINDEX] PRIMARY KEY CLUSTERED  (  CDReservationID ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
```

```
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [CDReservation_clientsclientid_conferencedayid_uindex] ON
dbo.CDReservation
      (ClientsClientID ASC,
       ConferenceDayID ASC
       )
   WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


ALTER TABLE dbo.CDReservation WITH NOCHECK ADD CONSTRAINT
[CHK_CDReservation_cancellationdate] CHECK  ((CancellationDate > ReservationDate)) GO
ALTER TABLE dbo.CDReservation CHECK CONSTRAINT [CHK_CDReservation_cancellationdate] GO

ALTER TABLE dbo.CDReservation WITH NOCHECK ADD CONSTRAINT
[CHK_CDReservation_paidamount] CHECK  ((PaidAmount >= 0)) GO
ALTER TABLE dbo.CDReservation CHECK CONSTRAINT [CHK_CDReservation_paidamount] GO

ALTER TABLE dbo.CDReservation WITH NOCHECK ADD CONSTRAINT
[CHK_CDReservation_numofparticipants] CHECK  ((NumOfParicipants >= 0)) GO
ALTER TABLE dbo.CDReservation CHECK CONSTRAINT [CHK_CDReservation_numofparticipants]
GO
```

8. **Workshop**
    - **WorkshopID: int -** klucz główny tabeli, autoinkrementowany
    - **ConferenceDayID: int -** klucz obcy tabeli, odnoszący się do tabeli ConferenceDay, łączący warsztat z konkretnym dniem konferencji oraz samą konferencją
    - **Name: varchar(50) -** nazwa warsztatu
    - **StartTime: time -** godzina rozpoczęcia warsztatu
    - **EndTime: time -** godzina zakończenia warsztatu
    - **MaxAmountOfPeople: int -** maksymalna ilość osób, która może uczesniczyć w warsztacie
    - **Cancelled: bit -** informacja o odwołaniu warsztatu - jeśli anulowano, to przyjmuje wartość 1, a jeśli nie to 0. Domyślnie to pole przyjmuje wartość 0.
    - **Fee: numeric(16,2) -** opłata za udział w warsztacie

    Tabela służy do przechowywania podstawowych informacji o warsztatach prowadzonych w trakcie konferencji.
    Kod generujący relację:

```
create table Workshop
(
  WorkshopID int identity,
  ConferenceDayID int not null
    constraint FKWorkshop354008
      references ConferenceDay,
```

```
  Name varchar(50) not null,
  StartTime time not null,
  EndTime time not null,
  MaxAmountOfPeople int not null,
  Cancelled bit default 0 not null,
  Fee numeric(16,2) not null
CONSTRAINT [PK_Workshop_CINDEX] PRIMARY KEY CLUSTERED  (  WorkshopID ASC )WITH
(PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [Workshop_conferencedayid_cancelled_uindex] ON
dbo.Workshop
      (ConferenceDayID ASC,
       Cancelled ASC
       )
   WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


--ALTER TABLE dbo.ConferenceDay WITH NOCHECK ADD CONSTRAINT
[CHK_ConferenceDay_date] CHECK  ((ConferenceDay.Date > GETDATE())) GO
--ALTER TABLE dbo.ConferenceDay CHECK CONSTRAINT [CHK_ConferenceDay_date] GO do
triggera

ALTER TABLE dbo.Workshop  WITH NOCHECK ADD CONSTRAINT [CHK_Workshop_endtime]
CHECK ((ConferenceDay.StartTime < ConferenceDay.EndTime)) GO
ALTER TABLE dbo.Workshop CHECK CONSTRAINT [CHK_Workshop_endtime] GO

ALTER TABLE dbo.Workshop  WITH NOCHECK ADD CONSTRAINT
[CHK_Workshop_maxamountofpeople] CHECK ((Workshop.MaxAmountOfPeople > 0)) GO
ALTER TABLE dbo.Workshop CHECK CONSTRAINT [CHK_Workshop_maxamountofpeople] GO

ALTER TABLE dbo.Workshop  WITH NOCHECK ADD CONSTRAINT [CHK_Workshop_fee] CHECK
((Workshop.Fee > 0)) GO
ALTER TABLE dbo.Workshop CHECK CONSTRAINT [CHK_Workshop_fee] GO
```

9. **WorkshopReservation**
    - **WorkshopReservationID: int -** klucz główny tabeli, autoinkrementowany
    - **CDReservationID: int -** klucz obcy odnoszący się do tabeli CDReservationID, dzięki czemu wymusza istnienie rezerwacji na dany dzień konferencji, aby uczestnik mógł uczestniczyć w warsztacie
    - **WorkshopID: int -** klucz obcy odnoszący się do tabeli Workshop
    - **ReservationDate: date** - data rezerwacji na warsztat
    - **CancellationDate: date -** data rezygnacji z rezerwacji, jest nullem do momentu anulowania
    - **NumOfParicipants: int -** ilość osób, dla których dokonywana jest rezerwacja
    - **PaidAmount: numeric(16,2) -** wysokość kwoty już wpłaconej za warsztat

Tabela służąca do przechowywania rezerwacji na dany warsztat. Rezerwacji mogą dokonywać wyłącznie osoby, które posiadają rezerwację na dzień konferencji, w którym odbywa się ten warsztat.
Kod generujący tabelę:

```
create table WorkshopReservation
(
  WorkshopReservationID int identity,
  CDReservationID int not null
    constraint FKWorkshopRe912629
      references CDReservation,
  WorkshopID int not null
    constraint FKWorkshopRe425767
      references Workshop,
  ReservationDate date not null,
  CancellationDate date,
  NumOfParicipants int not null,
  PaidAmount numeric(16,2) not null
CONSTRAINT [PK_WorkshopReservation_CINDEX] PRIMARY KEY CLUSTERED  (
WorkshopReservationID ASC )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [WorkshopReservation_clientsclientid_conferencedayid_uindex]
ON
dbo.WorkshopReservation
      (CDReservationID ASC,
       WorkshopID ASC
       )
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


ALTER TABLE dbo.WorkshopReservation WITH NOCHECK ADD CONSTRAINT
[CHK_WorkshopReservation_cancellationdate] CHECK  ((CancellationDate > ReservationDate)) GO
ALTER TABLE dbo.WorkshopReservation CHECK CONSTRAINT
[CHK_WorkshopReservation_cancellationdate] GO

ALTER TABLE dbo.WorkshopReservation WITH NOCHECK ADD CONSTRAINT
[CHK_WorkshopReservation_paidamount] CHECK  ((PaidAmount >= 0)) GO
ALTER TABLE dbo.WorkshopReservation CHECK CONSTRAINT
[CHK_WorkshopReservation_paidamount] GO

ALTER TABLE dbo.WorkshopReservation WITH NOCHECK ADD CONSTRAINT
[CHK_WorkshopReservation_numofparticipants] CHECK  ((NumOfParicipants >= 0)) GO
ALTER TABLE dbo.WorkshopReservation CHECK CONSTRAINT
[CHK_WorkshopReservation_numofparticipants] GO
```

10. Participant

- **ParicipantID: int -** klucz główny tabeli, autoinkrementowany
- **FirstName: varchar(50) -** imię uczestnika konferencji
- **LastName: varchar(50) -** nazwisko uczestnika konferencji
- **Email: varchar(50) -** adres email uczestnika konferencji
- **Phone: varchar(50) -** numer telefonu uczestnika konferencji
- **StudentCardID: varchar(50)** - jeśli uczestnik jest studentem, to tutaj podany jest numer jego legitymacji studenckiej, dzięki czemu można naliczyć stosowna zniżkę. W przeciwnym wypadku pole to jest puste.

Tabela przechowująca dane o uczestnikach konferencji potrzebne aby ich zidentyfikować.

Kod generujący tabelę:

```
create table Participant
(
  ParicipantID int identity,
  FirstName varchar(50) not null,
  LastName varchar(50) not null,
  Email varchar(50) not null,
  Phone varchar(50) not null,
  StudentCardID varchar(50)
CONSTRAINT [PK_Participant_CINDEX] PRIMARY KEY CLUSTERED  (  ParicipantID ASC )WITH
(PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [Participant_phone_studentcardid_uindex] ON
dbo.Participant
      (Phone ASC,
       StudentCardID ASC
       )
   WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)


ALTER TABLE dbo.Participant WITH NOCHECK ADD CONSTRAINT [CHK_Participant_firstname]
CHECK  ((ltrim(FirstName)<>'')) GO
ALTER TABLE dbo.Participant CHECK CONSTRAINT [CHK_Participant_firstname] GO

ALTER TABLE dbo.Participant WITH NOCHECK ADD CONSTRAINT [CHK_Participant_lastname]
CHECK  ((ltrim(LastName)<>'')) GO
ALTER TABLE dbo.Participant CHECK CONSTRAINT [CHK_Participant_lastname] GO

ALTER TABLE dbo.Participant WITH NOCHECK ADD CONSTRAINT [CHK_Participant_phone]
CHECK  ((ltrim(Phone)<>'')) GO
ALTER TABLE dbo.Participant CHECK CONSTRAINT [CHK_Participant_phone] GO

ALTER TABLE dbo.Participant WITH NOCHECK ADD CONSTRAINT [CHK_Participant_studentcardid]
CHECK  ((ltrim(StudentCardID)<>'')) GO
ALTER TABLE dbo.Participant CHECK CONSTRAINT [CHK_Participant_studentcardid] GO
```

```
ALTER TABLE dbo.Participant WITH NOCHECK ADD CONSTRAINT [CHK_Participant_email] CHECK
((Email like '%_@_%_.__%')) GO
ALTER TABLE dbo.Participant CHECK CONSTRAINT [CHK_Participant_email] GO
```

11. **ConferenceDayParticipant**
    - **ConferenceDayParicipantID: int -** klucz główny tabeli, autoinkrementowany
    - **CDReservationID: int -** klucz obcy odnoszący się do tabeli CDReservation
    - **ParicipantID: int -** klucz obcy odnoszący się do tabeli Participant
    - **AddDate: date -** pole przechowujące informację o dniu powiązania uczestnika z dniem rezerwacji na ten dzień konferencji
    - **RemoveDate: date -** pole przechowujące informację o dniu usunięcia uczestnika z rezerwacji. Domyślnie pole to jest puste.

    Tabela służąca do powiązania uczestnika z rezerwacją na dzień konferencji, w którym bierze udział.
    Kod generujący tabelę:

```
create table ConferenceDayParicipant
(
 ConferenceDayParicipantID int identity,
 CDReservationID int not null
   constraint FKConference489919
     references CDReservation,
 ParicipantID int
   constraint FKConference233755
     references Participant,
 AddDate date not null,
 RemoveDate date
CONSTRAINT [PK_ConferenceDayParicipantID_CINDEX] PRIMARY KEY CLUSTERED  (
ParicipantID ASC )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)


CREATE NONCLUSTERED INDEX [ConferenceDayParicipant_cdreservationid_participantid_uindex]
ON
dbo.ConferenceDayParicipant
     (CDReservationID ASC,
      ParicipantID ASC
      )
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
     DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)

Go
```

12. **WorkshopParticipant**
    - **WorkshopParicipantID: int -** klucz główny tabeli, autoinkrementowany
    - **ConferenceDayParicipantID: int -** klucz obcy odnoszący się do tabeli ConferenceDayParticipant

- **WorkshopReservationID int -** klucz obcy odnoszący się do tabeli WorkshopReservation
- **AddDate date -** pole przechowujące informację o dniu powiązania uczestnika z dniem rezerwacji na ten warsztat
- **RemoveDate date -** pole przechowujące informację o dniu usunięcia uczestnika z rezerwacji. Domyślnie pole to jest puste.

Tabela służąca do powiązania uczestnika z rezerwacją na warsztat, w którym bierze udział. Robi to pośrednio, poprzez tabelę ConferenceDayParticipant, ponieważ uczestnik warsztatu musi być też uczestnikiem danego dnia konferencji.
Kod generujący tabelę:

```
create table WorkshopParticipant
(
  WorkshopParicipantID int identity,
  ConferenceDayParicipantID int not null
    constraint FKWorkshopPa625473
      references ConferenceDayParicipant,
  WorkshopReservationID int not null
    constraint FKWorkshopPa761914
      references WorkshopReservation,
  AddDate date not null,
  RemoveDate date
CONSTRAINT [PK_WorkshopParticipant_CINDEX] PRIMARY KEY CLUSTERED  (
WorkshopParicipantID ASC )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF,
   ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON)
)
go


CREATE NONCLUSTERED INDEX [WorkshopParticipant_clientsclientid_conferencedayid_uindex] ON
dbo.WorkshopParticipant
      (ConferenceDayParicipantID ASC,
       WorkshopReservationID ASC
       )
  WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF,
IGNORE_DUP_KEY = OFF,
      DROP_EXISTING = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS
= ON)
```

Widoki

1. **ConferenceDayParticipants**
   Zwraca wszystkie dni konferencji i uczestników, którzy są na te dni zapisani. Nie bierzemy pod uwagę dni konferencji, które zostały anulowane i uczestników, których rezerwacja została anulowana.

```
CREATE VIEW ConferenceDayParticipants
AS
  SELECT C.Name, CD.Date, FirstName, Lastname from Conference C
     INNER JOIN ConferenceDay CD on C.ConferenceID = CD.ConferenceID
     INNER JOIN CDReservation CDR on CD.ConferenceDayID = CDR.ConferenceDayID
     INNER JOIN ConferenceDayParicipant CDP on CDR.CDReservationID = CDP.CDReservationID
     INNER JOIN Participant P on CDP.ParicipantID = P.ParicipantID
     WHERE CD.Cancelled = 0 and CDP.RemoveDate IS NULL
```

2. **WorkshopParticipants**
   Zwraca nam wszystkie warsztaty i uczestników, którzy są na te warsztaty zapisani. Nie bierzemy pod uwagę warsztatów, które zostały anulowane i uczestników, których rezerwacja została anulowana.

```
CREATE VIEW WorkshopParticipants
AS
  SELECT W.Name, FirstName, LastName from Workshop W
     INNER JOIN WorkshopReservation WR on W.WorkshopID = WR.WorkshopID
     INNER JOIN WorkshopParticipant WP on WR.WorkshopReservationID =
WP.WorkshopReservationID
     INNER JOIN ConferenceDayParicipant CDP on WP.ConferenceDayParicipantID =
CDP.ConferenceDayParicipantID
     INNER JOIN Participant P on CDP.ParicipantID = P.ParicipantID
     WHERE W.Cancelled = 0 and WP.RemoveDate IS NULL
```

3. MostActiveClients

Zwraca nam 15 klientów, którzy byli najbardziej aktywni, czyli zarezerwowali największą liczbę miejsc na dni konferencji i rezerwacje te nie zostały anulowane.

```
SELECT TOP 15 LastName as Name, 'PrivateClient' as Who, COUNT(*) as NumOfParticipants from
PrivateClients
  INNER JOIN Clients C on PrivateClients.PrivateClientID = C.PrivateClientID
  INNER JOIN CDReservation CDR on C.ClientID = CDR.ClientsClientID
  INNER JOIN ConferenceDayParicipant CDP on CDR.CDReservationID = CDP.CDReservationID
  WHERE CDR.Cancelled = 0 AND CDP.RemoveDate IS NULL
  GROUP BY PrivateClients.LastName
UNION
SELECT CompanyName as Name, 'CompanyClient' as Who, COUNT(*) as NumOfParticipants from
CompanyClients
  INNER JOIN Clients C on CompanyClients.CompanyID = C.PrivateClientID
  INNER JOIN CDReservation CDR on C.ClientID = CDR.ClientsClientID
  INNER JOIN ConferenceDayParicipant CDP on CDR.CDReservationID = CDP.CDReservationID
  WHERE CDR.Cancelled = 0 AND CDP.RemoveDate IS NULL
  GROUP BY CompanyName
```

4. ClientsDebts

Zwraca nam wszystkich klientów i informację, ile muszą zapłacić za zarezerwowane miejsca na konferencjach i warsztatach oraz ile już zapłacili.

```
CREATE VIEW ClientsDebts
AS
  SELECT ConfPrivate.ID, ConfPrivate.Name, ConfPrivate.Who, ConfPrivate.Paid + PrivateWork.Paid
as Paid, ConfPrivate.TotalToPay + PrivateWork.TotalToPay as TotalToPay, ConfPrivate.LeftToPay +
PrivateWork.LeftToPay as LeftToPay FROM
  (SELECT PrivateClients.PrivateClientID as ID, LastName as Name, 'PrivateClient' as Who,
SUM(Paidw) as Paid, SUM(ToPay) as TotalToPay, SUM(Paidw)-SUM(ToPay) as LeftToPay
  from PrivateClients
  INNER JOIN Clients C on PrivateClients.PrivateClientID = C.PrivateClientID
  INNER JOIN CDReservation CDR on C.ClientID = CDR.ClientsClientID
  INNER JOIN (SELECT CDRw.CDReservationID, CDRw.PaidAmount as Paidw,
dbo.finalPrice(CDw.FeeForDay,
        (SELECT COUNT(*) from ConferenceDay CDw
          inner join CDReservation CDRww on CDw.ConferenceDayID = CDRww.ConferenceDayID
          inner join ConferenceDayParicipant CDPw on CDRww.CDReservationID =
CDPw.CDReservationID
          inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
          where Pw.StudentCardID IS NOT NULL and CDRww.CDReservationID =
CDRw.CDReservationID
          group by CDRww.CDReservationID
        ),
        (SELECT COUNT(*) from ConferenceDay CDw
          inner join CDReservation CDRww on CDw.ConferenceDayID = CDRww.ConferenceDayID
          inner join ConferenceDayParicipant CDPw on CDRww.CDReservationID =
CDPw.CDReservationID
```

```
            inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
            where Pw.StudentCardID IS NULL and CDRww.CDReservationID = CDRw.CDReservationID
            group by CDRww.CDReservationID
        ),
        CDw.Date, CDRw.ReservationDate) as ToPay from ConferenceDay CDw
            INNER JOIN CDReservation CDRw on CDw.ConferenceDayID = CDRw.ConferenceDayID
            WHERE CDw.Date > GETDATE()) as TMP on TMP.CDReservationID =
CDR.CDReservationID
        group by PrivateClients.PrivateClientID, Lastname) as ConfPrivate
  INNER JOIN

  (SELECT PrivateClients.PrivateClientID as ID, LastName as Name, 'PrivateClient' as Who,
SUM(Paidw) as Paid, SUM(ToPay) as TotalToPay, SUM(Paidw)-SUM(ToPay) as LeftToPay
  from PrivateClients
  INNER JOIN Clients C on PrivateClients.PrivateClientID = C.PrivateClientID
  INNER JOIN CDReservation CDR on C.ClientID = CDR.ClientsClientID
  INNER JOIN WorkshopReservation WR on CDR.CDReservationID = WR.CDReservationID
  INNER JOIN (SELECT WRw.WorkshopReservationID, WRw.PaidAmount as Paidw,
dbo.finalPrice(Ww.Fee,
        (SELECT COUNT(*) from WorkshopReservation WRww
            inner join WorkshopParticipant WPww on WPww.WorkshopReservationID =
WRww.WorkshopReservationID
            inner join ConferenceDayParicipant CDPww on WPww.ConferenceDayParicipantID =
CDPww.ConferenceDayParicipantID
            inner join Participant Pww on CDPww.ParicipantID = Pww.ParicipantID
            where Pww.StudentCardID IS NOT NULL and WRww.WorkshopReservationID =
WRw.CDReservationID
            group by WRww.WorkshopReservationID
        ),
        (SELECT COUNT(*) from WorkshopReservation WRww
            inner join WorkshopParticipant WPww on WPww.WorkshopReservationID =
WRww.WorkshopReservationID
            inner join ConferenceDayParicipant CDPww on WPww.ConferenceDayParicipantID =
CDPww.ConferenceDayParicipantID
            inner join Participant Pww on CDPww.ParicipantID = Pww.ParicipantID
            where Pww.StudentCardID IS NOT NULL and WRww.WorkshopReservationID =
WRw.CDReservationID
            group by WRww.WorkshopReservationID
        ),
        CDw.Date, WRw.ReservationDate) as ToPay from Workshop Ww
            INNER JOIN WorkshopReservation WRw on Ww.WorkshopID = WRw.WorkshopID
            INNER JOIN ConferenceDay CDw on Ww.ConferenceDayID = CDw.ConferenceDayID
            WHERE CDw.Date > GETDATE()) as TMP on TMP.WorkshopReservationID =
WR.WorkshopReservationID
            group by PrivateClients.PrivateClientID, Lastname) as PrivateWork on ConfPrivate.ID =
PrivateWork.ID

  UNION

  ----------

  SELECT ConfCompany.ID, ConfCompany.Name, ConfCompany.Who, ConfCompany.Paid +
ConfCompany.Paid as Paid, ConfCompany.TotalToPay + ConfCompany.TotalToPay as TotalToPay,
ConfCompany.LeftToPay + ConfCompany.LeftToPay as LeftToPay FROM
```

```
  (SELECT CompanyClients.CompanyID as ID, CompanyName as Name, 'PrivateClient' as Who,
SUM(Paidw) as Paid, SUM(ToPay) as TotalToPay, SUM(Paidw)-SUM(ToPay) as LeftToPay
  from CompanyClients
  INNER JOIN Clients C on CompanyClients.CompanyID = C.PrivateClientID
  INNER JOIN CDReservation CDR on C.ClientID = CDR.ClientsClientID
  INNER JOIN (SELECT CDRw.CDReservationID, CDRw.PaidAmount as Paidw,
dbo.finalPrice(CDw.FeeForDay,
        (SELECT COUNT(*) from ConferenceDay CDw
          inner join CDReservation CDRww on CDw.ConferenceDayID = CDRww.ConferenceDayID
          inner join ConferenceDayParicipant CDPw on CDRww.CDReservationID =
CDPw.CDReservationID
          inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
          where Pw.StudentCardID IS NOT NULL and CDRww.CDReservationID =
CDRw.CDReservationID
          group by CDRww.CDReservationID
        ),
        (SELECT COUNT(*) from ConferenceDay CDw
          inner join CDReservation CDRww on CDw.ConferenceDayID = CDRww.ConferenceDayID
          inner join ConferenceDayParicipant CDPw on CDRww.CDReservationID =
CDPw.CDReservationID
          inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
          where Pw.StudentCardID IS NULL and CDRww.CDReservationID = CDRw.CDReservationID
          group by CDRww.CDReservationID
        ),
        CDw.Date, CDRw.ReservationDate) as ToPay from ConferenceDay CDw
          INNER JOIN CDReservation CDRw on CDw.ConferenceDayID = CDRw.ConferenceDayID
          WHERE CDw.Date > GETDATE()) as TMP on TMP.CDReservationID =
CDR.CDReservationID
      group by CompanyClients.CompanyID, CompanyName) as ConfCompany
  INNER JOIN

  (SELECT CompanyClients.CompanyID as ID, CompanyName as Name, 'PrivateClient' as Who,
SUM(Paidw) as Paid, SUM(ToPay) as TotalToPay, SUM(Paidw)-SUM(ToPay) as LeftToPay
  from CompanyClients
  INNER JOIN Clients C on CompanyClients.CompanyID = C.PrivateClientID
  INNER JOIN CDReservation CDR on C.ClientID = CDR.ClientsClientID
  INNER JOIN WorkshopReservation WR on CDR.CDReservationID = WR.CDReservationID
  INNER JOIN (SELECT WRw.WorkshopReservationID, WRw.PaidAmount as Paidw,
dbo.finalPrice(Ww.Fee,
        (SELECT COUNT(*) from WorkshopReservation WRww
          inner join WorkshopParticipant WPww on WPww.WorkshopReservationID =
WRww.WorkshopReservationID
          inner join ConferenceDayParicipant CDPww on WPww.ConferenceDayParicipantID =
CDPww.ConferenceDayParicipantID
          inner join Participant Pww on CDPww.ParicipantID = Pww.ParicipantID
          where Pww.StudentCardID IS NOT NULL and WRww.WorkshopReservationID =
WRw.CDReservationID
          group by WRww.WorkshopReservationID
        ),
        (SELECT COUNT(*) from WorkshopReservation WRww
          inner join WorkshopParticipant WPww on WPww.WorkshopReservationID =
WRww.WorkshopReservationID
          inner join ConferenceDayParicipant CDPww on WPww.ConferenceDayParicipantID =
CDPww.ConferenceDayParicipantID
```

```
            inner join Participant Pww on CDPww.ParicipantID = Pww.ParicipantID
            where Pww.StudentCardID IS NOT NULL and WRww.WorkshopReservationID =
WRw.CDReservationID
            group by WRww.WorkshopReservationID
        ),
        CDw.Date, WRw.ReservationDate) as ToPay from Workshop Ww
            INNER JOIN WorkshopReservation WRw on Ww.WorkshopID = WRw.WorkshopID
            INNER JOIN ConferenceDay CDw on Ww.ConferenceDayID = CDw.ConferenceDayID
            WHERE CDw.Date > GETDATE()) as TMP on TMP.WorkshopReservationID =
WR.WorkshopReservationID
            group by CompanyClients.CompanyID, CompanyName) as CompanyWork on
ConfCompany.ID = CompanyWork.ID
```

5. UpcomingConferences

Zwraca nam wszystkie konferencje, które mają się odbyć w przyszłości.

```
CREATE VIEW UpcomingConferences
AS
  SELECT Conference.ConferenceID, ConferenceDayID, Name, Date,
dbo.freePlacesOnConferenceDay(ConferenceDayID) as freeSlots from Conference
    INNER JOIN ConferenceDay CD on Conference.ConferenceID = CD.ConferenceID
    WHERE CD.Date > GETDATE()
```

6. UpcomingWorkshops

Zwraca nam wszystkie warsztaty, które mają się odbyć w przyszłości.

```
 CREATE VIEW UpcomingWorkshops
AS
  SELECT WorkshopID, Workshop.Name, Date, dbo.freePlacesOnWorkshop(Workshop.WorkshopID)
as freeSlots from Workshop
    INNER JOIN ConferenceDay CD on Workshop.ConferenceDayID = CD.ConferenceDayID
    WHERE CD.Date > GETDATE()
```

Procedury składowane, funkcje, triggery

1. Procedury do wprowadzania danych

1.1 addCompanyClient

```
CREATE PROCEDURE dbo.addCompanyClient
  @AddressID int,
  @CompanyName varchar(50),
  @ContactName varchar(50),
  @Phone varchar(50),
  @Fax varchar(50),
  @Email varchar(50)
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS(SELECT * from Address where AddressID = @AddressID)
     THROW 51000, '@AddresID does not exist in Address', 1

  INSERT INTO CompanyClients VALUES (@AddressID, @CompanyName, @ContactName,
@Phone, @Fax, @Email)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.2 addAddress

```
CREATE PROCEDURE dbo.addAddress
  @Country varchar(50),
  @Region varchar(50),
  @City varchar(50),
  @PostalCode varchar(50),
  @Street varchar(50),
  @Address varchar(50)
AS
```

```
INSERT INTO dbo.Address VALUES (@Country, @Region, @City, @PostalCode, @Street,
@Address)

go
```

1.3 addConference

```
CREATE PROCEDURE dbo.addConference
  @Address int,
  @Name nvarchar(50),
  @Begin date,
  @End date
AS

INSERT INTO dbo.Conference VALUES (@Address, @Name, @Begin, @End, 0);
go
```

1.4 addConferenceDayReservation

```
CREATE PROCEDURE dbo.addConferenceDayReservation
  @ConferenceDayID int,
  @ClientID int,
  @ReservationDate date,
  @NumberOfParticipants int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS(SELECT * from ConferenceDay where ConferenceDayID = @ConferenceDayID)
     THROW 51000, '@ConferenceDayID does not exist in ConferenceDay', 1

  IF NOT EXISTS(SELECT * from Clients where ClientID = @ClientID)
     THROW 51000, '@ClientID does not exist in Clients', 1

  IF @NumberOfParticipants <= 0
     THROW 51000, '@NumberOfParticipants is not proper value of count of peple', 1

  IF @NumberOfParticipants > (SELECT dbo.freePlacesOnConferenceDay (@ConferenceDayID))
     THROW 51000, '@NumberOfParticipants is higher than available amount of places', 1

  INSERT INTO CDReservation VALUES (@ClientID, @ConferenceDayID, @ReservationDate,
null, @NumberOfParticipants, 0, 0)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.5 addDefaultConferenceDay

```
CREATE PROCEDURE dbo.addDefaultConferenceDay
  @ConferenceID int,
  @Date date

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS ((SELECT * from Conference
     where ConferenceID = @ConferenceID))
     THROW 51000, '@@ConferenceID does not exists', 1

  IF @Date > (SELECT [End] FROM Conference WHERE ConferenceID = @ConferenceID)
          OR @Date < (SELECT [Begin] FROM Conference WHERE ConferenceID =
@ConferenceID)
     THROW 5100, '@Date is out of Conference', 1


  INSERT INTO ConferenceDay
     VALUES (@ConferenceID, @Date, (TIMEFROMPARTS(9,0,0,0,7)),
(TIMEFROMPARTS(19,0,0,0,7)), 200, 0, 100)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.6 addManyDaysConferenceReservation

```
CREATE PROCEDURE dbo.addManyDaysConferenceReservation
  @ConferenceID int,
  @ClientID int,
  @FirstDay date,
  @LastDay date,
  @NumberOfParticipants int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS(SELECT * from Conference where ConferenceID = @ConferenceID)
     THROW 51000, '@ConferenceID does not exist in Conference', 1

  IF NOT EXISTS(SELECT * from Clients where ClientID = @ClientID)
     THROW 51000, '@ClientID does not exist in Clients', 1
```

```
  IF @NumberOfParticipants <= 0
     THROW 51000, '@NumberOfParticipants is not proper value of count of peple', 1

  IF @FirstDay > @LastDay
     THROW 5100, '@FirstDay is later then @LastDay', 1

  DECLARE @ItDay AS DATE = @FirstDay;

  WHILE @ItDay <= @LastDay
     BEGIN
        IF EXISTS(SELECT * from ConferenceDay where Date = @ItDay AND ConferenceID =
@ConferenceID)
           INSERT INTO CDReservation VALUES (@ClientID,
                (SELECT ConferenceDayID from ConferenceDay where Date = @ItDay AND
ConferenceID = @ConferenceID),
                                    @ItDay, null, @NumberOfParticipants, 0, 0)
     END



  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.7 addParticipant

```
CREATE PROCEDURE dbo.addParticipant
  @FirstName varchar(50),
  @LastName varchar(50),
  @Phone varchar(50),
  @Email varchar(50),
  @StudentCardID varchar(50)


AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  INSERT INTO Participant
    VALUES (@FirstName, @LastName, @Phone, @Email, @StudentCardID)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.8 addParticipantToCDReservation

```
CREATE PROCEDURE dbo.addParticipantToCDReservation
  @CDReservationID int,
  @FirstName varchar(50),
  @LastName varchar(50),
  @Email varchar(50),
  @Phone varchar(50),
  @StudentCardID varchar(50)

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS(SELECT * from CDReservation where CDReservationID = @CDReservationID)
    THROW 51000, '@CDReservationID does not exist in CDReservation', 1

  IF NOT EXISTS(SELECT * FROM Participant WHERE Email = @Email)
    EXECUTE dbo.addParticipant @FirstName, @LastName, @Phone, @Email, @StudentCardID

  DECLARE @ParticipantID int;

  SELECT @ParticipantID = ParicipantID FROM Participant WHERE Email = @Email

  EXECUTE dbo.connectParticipantToCDReservation @ParticipantID, @CDReservationID
```

```
  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.9 addParticipantToWorkshopReservation

```
CREATE PROCEDURE dbo.addParticipantToWorkshopReservation
  @ParticipantID int,
  @WorkshopReservationID int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ParticipantID IS NULL or @WorkshopReservationID IS NULL
     THROW 51000, '@ParticipantID or @WorkshopReservationID is null', 1

  IF NOT EXISTS ((SELECT ParicipantID from Participant
     where ParicipantID = @ParticipantID))
     THROW 51000, '@ParticipantID does not exists', 1

  IF NOT EXISTS ((SELECT WorkshopReservationID from WorkshopReservation
     where WorkshopReservationID = @WorkshopReservationID))
     THROW 51000, '@WorkshopReservationID does not exists', 1

  IF (SELECT NumOfParicipants FROM WorkshopReservation
     where WorkshopReservationID = @WorkshopReservationID)
     =
     (SELECT COUNT(*) from WorkshopReservation WR
        inner join WorkshopParticipant WPw on WR.WorkshopReservationID =
WPw.WorkshopReservationID
        where WR.WorkshopReservationID = @WorkshopReservationID
        group by WR.WorkshopReservationID)
     THROW 51000, 'maximum numer of participant for this reservation reached', 1

  DECLARE @ConferenceParticipantID int

  SET @ConferenceParticipantID = (SELECT CDP.ConferenceDayParicipantID from
ConferenceDayParicipant CDP
     inner join WorkshopParticipant WP on CDP.ConferenceDayParicipantID =
WP.ConferenceDayParicipantID
     inner join WorkshopReservation WR on WP.WorkshopReservationID =
WR.WorkshopReservationID
     inner join Participant P on CDP.ParicipantID = P.ParicipantID
     where WR.WorkshopReservationID = @WorkshopReservationID and P.ParicipantID =
@ParticipantID)
```

```
  DECIARE @WorkshopDate date;
  DECLARE @WorkshopStartTime time;
  DECLARE @WorkshopEndTime time;
  SELECT @WorkshopDate = CD.Date, @WorkshopStartTime = W.StartTime,
@WorkshopEndTime = W.EndTime FROM WorkshopReservation WR
          INNER JOIN Workshop W ON W.WorkshopID = WR.WorkshopID
          INNER JOIN ConferenceDay CD ON CD.ConferenceDayID = W.ConferenceDayID
    WHERE WR.WorkshopReservationID = @WorkshopReservationID

  IF dbo.countOfParticipantActivitiesInTimeRange(@ParticipantID, @WorkshopDate,
@WorkshopStartTime,
    @WorkshopEndTime) > 0
    BEGIN
      THROW 5100, 'Customer has different workshops in this time', 1
    end

  INSERT INTO WorkshopParticipant
    VALUES (@ConferenceParticipantID, @WorkshopReservationID, GETDATE(), NULL)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.10 addPrivateClient

```
CREATE PROCEDURE dbo.addPrivateClient
  @AddressID int,
  @FirstName varchar(50),
  @LastName varchar(50),
  @Email varchar(50),
  @Phone varchar(50)
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS(SELECT * from Address where AddressID = @AddressID)
    THROW 51000, '@AddresID does not exist in Address', 1

  INSERT INTO PrivateClients VALUES (@AddressID, @FirstName, @LastName, @Email,
@Phone)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
```

1.11 addWorkshop

```
CREATE PROCEDURE dbo.addWorkshop
  @ConferenceDayID int,
  @Name varchar(50),
  @StartTime time,
  @EndTime time,
  @MaxAmountOfPeople int,
  @Fee int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS(SELECT * from ConferenceDay where ConferenceDayID = @ConferenceDayID)
     THROW 51000, '@ConferenceDay does not exist in Address', 1

  INSERT INTO Workshop VALUES (@ConferenceDayID, @Name, @StartTime, @EndTime,
@MaxAmountOfPeople, 0, @Fee)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

1.12 addWorkshopReservation

```
CREATE PROCEDURE dbo.addWorkshopReservation
  @CDReservationID int,
  @WorkshopID int,
  @ReservationDate date,
  @NumberOfParticipants int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS(SELECT * from Workshop where WorkshopID = @WorkshopID)
     THROW 51000, '@WorkshopID does not exist in Workshop', 1

  IF NOT EXISTS(SELECT * from CDReservation where CDReservationID = @CDReservationID)
     THROW 51000, '@CDReservation does not exist in CDReservation', 1

  IF @NumberOfParticipants <= 0
     THROW 51000, '@NumberOfParticipants is not proper value of count of pe0ple', 1
```

```
  IF @NumberOfParticipants > dbo.freePlacesOnWorkshop(@WorkshopID)
     THROW 51000, '@NumberOfParticipants is higher than available amount of places', 1

  INSERT INTO WorkshopReservation VALUES (@CDReservationID, @WorkshopID,
@ReservationDate, null, @NumberOfParticipants, 0)

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
Go
```

2. Procedury modyfikujące dane

2.1 cancelConference

```
CREATE PROCEDURE dbo.cancelConference
  @ConferenceID int
AS BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ConferenceID IS NULL
     THROW 51000, '@ConferenceID is null', 1

  UPDATE Conference
     SET Cancelled = 1
     WHERE ConferenceID = @ConferenceID

  IF @@ROWCOUNT = 0
     THROW 51000, 'no conference with that id', 1

  UPDATE ConferenceDay
     SET Cancelled =1
     WHERE ConferenceID = @ConferenceID


  UPDATE Workshop
     SET Cancelled = 1
     WHERE Workshop.ConferenceDayID in
        (SELECT ConferenceDayID from ConferenceDay
           where ConferenceID = @ConferenceID)

  SELECT ClientsClientID
  INTO ClientsToRemove
  FROM Conference C
     inner join ConferenceDay CD on C.ConferenceID = CD.ConferenceID
     inner join CDReservation CR on CD.ConferenceDayID = CR.ConferenceDayID
```

```
  where Date > GETDATE()

  IF @@ROWCOUNT = 0
    THROW 51000, 'no clients signed to that conference', 1

  DECLARE @ClientID int
    DECLARE ClientsCursor CURSOR LOCAL FAST_FORWARD
    FOR SELECT * FROM ClientsToRemove

    OPEN ClientsCursor
    FETCH NEXT FROM ClientsCursor INTO @ClientID
    WHILE @@FETCH_STATUS = 0
      BEGIN
        EXEC cancelConferenceReservationClient @ClientID, @ConferenceID
        FETCH NEXT FROM ClientsCursor
        INTO @ClientID
      END
    CLOSE ClientsCursor
    DEALLOCATE ClientsCursor

  COMMIT TRANSACTION;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.2 cancelConferenceDayReservationClient

```
CREATE PROCEDURE dbo.cancelConferenceDayReservationClient
  @ClientID int,
  @ConferenceDayID int
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ConferenceDayID IS NULL or @ClientID IS NULL
    THROW 51000, '@ConferenceID or @ParicipantID is null', 1

  SELECT CDReservationID
  INTO ConferenceDayReservations
  from Clients
    inner join CDReservation CR on Clients.ClientID = CR.ClientsClientID
    inner join ConferenceDay CD on CR.ConferenceDayID = CD.ConferenceDayID
    where CD.ConferenceDayID = @ConferenceDayID and Date > GETDATE() and
Clients.ClientID = @ClientID

  IF @@ROWCOUNT = 0
    THROW 51000, '@Client has no reservation on @ConferenceDayID', 1

  UPDATE CDReservation
```

```
    SET CancellationDate = GETDATE()
    where CDReservation.CDReservationID in
    (SELECT CDReservationID from ConferenceDayReservations)

  SELECT P.ParicipantID
  INTO ParticipantsFromClient
  from ConferenceDayReservations
    inner join ConferenceDayParicipant CDP on ConferenceDayReservations.CDReservationID =
CDP.CDReservationID
    inner join Participant P on CDP.ParicipantID = P.ParicipantID

  IF @@ROWCOUNT = 0
    THROW 51000, '@Client has no participants on @ConferenceDayID', 1

  DECLARE @ParticipantID int
    DECLARE ParticipantsCursor CURSOR LOCAL FAST_FORWARD
    FOR SELECT * FROM ParticipantsFromClient

    OPEN ParticipantsCursor
    FETCH NEXT FROM ParticipantsCursor INTO @ParticipantID
    WHILE @@FETCH_STATUS = 0
      BEGIN
        EXEC cancelConferenceDayReservationParticipant @ParticipantID, @ConferenceDayID
        FETCH NEXT FROM ParticipantsCursor
        INTO @ParticipantID
      END
    CLOSE ParticipantsCursor
    DEALLOCATE ParticipantsCursor

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.3 cancelConferenceDayReservationParticipant

```
CREATE PROCEDURE dbo.cancelConferenceDayReservationParticipant
  @ParticipantID int,
  @ConferenceDayID int
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ConferenceDayID IS NULL or @ParticipantID IS NULL
    THROW 51000, '@ConferenceDayID or @ParicipantID is null', 1


  DECLARE @CDParticipantID int
  SET @CDParticipantID =
```

```
    (SELECT ConferenceDayParicipantID from Participant P
      inner join ConferenceDayParicipant CDP on CDP.ParicipantID = P.ParicipantID
      inner join CDReservation CDR on CDR.CDReservationID = CDP.CDReservationID
      inner join ConferenceDay CD on CDR.ConferenceDayID = CD.ConferenceDayID
      where CDR.ConferenceDayID = @ConferenceDayID and P.ParicipantID = @ParticipantID
and Date > GETDATE())

  IF @@ROWCOUNT = 0
    THROW 51000, '@ParticipantID not signed on @ConferenceDayID', 1

  UPDATE ConferenceDayParicipant
    SET RemoveDate = GETDATE()
    where ConferenceDayParicipantID = @CDParticipantID

  UPDATE CDReservation
    SET NumOfParicipants = NumOfParicipants -1
    where CDReservationID in
    (SELECT CDReservation.CDReservationID from CDReservation
      inner join ConferenceDayParicipant CDP on CDReservation.CDReservationID =
CDP.CDReservationID
      where ConferenceDayParicipantID = @CDParticipantID)

  SELECT WorkshopID
  INTO Workshops
  FROM ConferenceDay CD
    inner join Workshop W on CD.ConferenceDayID = W.ConferenceDayID

  DECLARE @WorkshopID int
    DECLARE WorkshopsCursor CURSOR LOCAL FAST_FORWARD
    FOR SELECT * FROM Workshops

    OPEN WorkshopsCursor
    FETCH NEXT FROM WorkshopsCursor INTO @WorkshopID
    WHILE @@FETCH_STATUS = 0
      BEGIN
        EXEC cancelWorkshopReservationParticipant @ParticipantID, @WorkshopID
        FETCH NEXT FROM WorkshopsCursor
        INTO @WorkshopID
      END
    CLOSE WorkshopsCursor
    DEALLOCATE WorkshopsCursor

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.4 cancelConferenceReservationClient

```
CREATE PROCEDURE dbo.cancelConferenceReservationClient
  @ClientID int,
  @ConferenceID int
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ConferenceID IS NULL or @ClientID IS NULL
    THROW 51000, '@ConferenceID or @ParicipantID is null', 1

  SELECT CD.ConferenceDayID
  INTO ConferenceDays
  from ConferenceDay CD
    inner join CDReservation CR on CD.ConferenceDayID = CR.ConferenceDayID
    where ConferenceID = @ConferenceID and Date > GETDATE() and CR.ClientsClientID =
@ClientID

  IF @@ROWCOUNT = 0
    THROW 51000, '@Client is not signed on @ConferenceID', 1

  DECLARE @ConferenceDayID int
    DECLARE ConferenceDaysCursor CURSOR LOCAL FAST_FORWARD
    FOR SELECT * FROM ConferenceDays

    OPEN ConferenceDaysCursor
    FETCH NEXT FROM ConferenceDaysCursor INTO @ConferenceDayID
    WHILE @@FETCH_STATUS = 0
      BEGIN
        EXEC cancelConferenceDayReservationClient @ClientID, @ConferenceDayID
        FETCH NEXT FROM ConferenceDaysCursor
        INTO @ConferenceDayID
      END
    CLOSE ConferenceDaysCursor
    DEALLOCATE ConferenceDaysCursor


  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.5 cancelConferenceReservationParticipant

```
CREATE PROCEDURE dbo.cancelConferenceReservationParticipant
  @ParticipantID int,
  @ConferenceID int
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ConferenceID IS NULL or @ParticipantID IS NULL
     THROW 51000, '@ConferenceID or @ParicipantID is null', 1

  SELECT CD.ConferenceDayID
  INTO ConferenceDays
  from ConferenceDay CD
     inner join CDReservation CR on CD.ConferenceDayID = CR.ConferenceDayID
     inner join ConferenceDayParicipant CDP on CR.CDReservationID = CDP.CDReservationID
     inner join Participant P on CDP.ParicipantID = P.ParicipantID
     where ConferenceID = @ConferenceID and Date > GETDATE() and P.ParicipantID =
@ParticipantID

  IF @@ROWCOUNT = 0
     THROW 51000, '@ParticipantID not signed on @ConferenceID', 1

  DECLARE @ConferenceDayID int
     DECLARE ConferenceDaysCursor CURSOR LOCAL FAST_FORWARD
     FOR SELECT * FROM ConferenceDays

     OPEN ConferenceDaysCursor
     FETCH NEXT FROM ConferenceDaysCursor INTO @ConferenceDayID
     WHILE @@FETCH_STATUS = 0
        BEGIN
           EXEC cancelConferenceDayReservationParticipant @ParticipantID, @ConferenceDayID
           FETCH NEXT FROM ConferenceDaysCursor
           INTO @ConferenceDayID
        END
     CLOSE ConferenceDaysCursor
     DEALLOCATE ConferenceDaysCursor

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.6 cancelWorkshop

```
CREATE PROCEDURE dbo.cancelWorkshop
  @WorkshopID int
AS BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
  BEGIN TRANSACTION;

  IF @WorkshopID IS NULL
     THROW 51000, '@WorkshopID is null', 1

  UPDATE Workshop
     SET Cancelled = 1
     WHERE WorkshopID = @WorkshopID

  IF @@ROWCOUNT = 0
     THROW 51000, 'no workshop with that id', 1

  SELECT ClientsClientID
  INTO ClientsToRemove
  FROM Workshop W
     inner join ConferenceDay CD on W.ConferenceDayID = CD.ConferenceDayID
     inner join CDReservation CR on CD.ConferenceDayID = CR.ConferenceDayID
     where Date > GETDATE() and WorkshopID = @WorkshopID

  IF @@ROWCOUNT = 0
     THROW 51000, 'no clients signed to workshop', 1

  DECLARE @ClientID int
     DECLARE ClientsCursor CURSOR LOCAL FAST_FORWARD
     FOR SELECT * FROM ClientsToRemove

     OPEN ClientsCursor
     FETCH NEXT FROM ClientsCursor INTO @ClientID
     WHILE @@FETCH_STATUS = 0
        BEGIN
           EXEC cancelWorkshopReservationClient @ClientID, @WorkshopID
           FETCH NEXT FROM ClientsCursor
           INTO @ClientID
        END
     CLOSE ClientsCursor
     DEALLOCATE ClientsCursor

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.7 cancelWorkshopReservationClient

```
CREATE PROCEDURE dbo.cancelWorkshopReservationClient
  @ClientID int,
  @WorkshopID int
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @WorkshopID IS NULL or @ClientID IS NULL
     THROW 51000, '@WorkshopID or @ParicipantID is null', 1

  SELECT WorkshopReservationID, WorkshopID
  INTO WorkshopReservations
  from Clients
     inner join CDReservation CR on Clients.ClientID = CR.ClientsClientID
     inner join ConferenceDay CD on CR.ConferenceDayID = CD.ConferenceDayID
     inner join WorkshopReservation WR on CR.CDReservationID = WR.CDReservationID
     where WR.WorkshopID = @WorkshopID and Date > GETDATE() and Clients.ClientID =
@ClientID

  IF @@ROWCOUNT = 0
     THROW 51000, '@Client has no reservation on @WorkshopID', 1

  UPDATE WorkshopReservation
     SET CancellationDate = GETDATE()
     where WorkshopReservationID in
     (SELECT WorkshopReservationID from WorkshopReservations)

  SELECT P.ParicipantID
  INTO ParticipantsFromClient
  from WorkshopReservations WR
     inner join WorkshopParticipant WP on WR.WorkshopReservationID =
WP.WorkshopReservationID
     inner join ConferenceDayParicipant CDP on WP.ConferenceDayParicipantID =
CDP.ConferenceDayParicipantID
     inner join Participant P on CDP.ParicipantID = P.ParicipantID

  IF @@ROWCOUNT = 0
     THROW 51000, '@Client has no participants on @WorkshopID', 1

  DECLARE @ParticipantID int
     DECLARE ParticipantsCursor CURSOR LOCAL FAST_FORWARD
     FOR SELECT * FROM ParticipantsFromClient

     OPEN ParticipantsCursor
     FETCH NEXT FROM ParticipantsCursor INTO @ParticipantID
     WHILE @@FETCH_STATUS = 0
        BEGIN
           EXEC cancelWorkshopReservationParticipant @ParticipantID, @WorkshopID
           FETCH NEXT FROM ParticipantsCursor
           INTO @ParticipantID
```

```
        END
    CLOSE ParticipantsCursor
    DEALLOCATE ParticipantsCursor

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.8 cancelWorkshopReservationParticipant

```
CREATE PROCEDURE dbo.cancelWorkshopReservationParticipant
  @ParticipantID int,
  @WorkshopID int
AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @WorkshopID IS NULL or @ParticipantID IS NULL
    THROW 51000, '@WorkshopID or @ParicipantID is null', 1

  SELECT WR.WorkshopReservationID, WP.WorkshopParicipantID
  INTO WorkshopReservations
  from WorkshopReservation WR
    inner join WorkshopParticipant WP on WR.WorkshopReservationID =
WP.WorkshopReservationID
    inner join ConferenceDayParicipant CDP on WP.ConferenceDayParicipantID =
CDP.ConferenceDayParicipantID
    inner join Participant P on CDP.ParicipantID = P.ParicipantID
    inner join CDReservation CR on WR.CDReservationID = CR.CDReservationID
    inner join ConferenceDay CD on CR.ConferenceDayID = CD.ConferenceDayID
    where WorkshopID = @WorkshopID and Date > GETDATE() and P.ParicipantID =
@ParticipantID

  IF @@ROWCOUNT = 0
    THROW 51000, '@ParticipantID not signed on @WorkshopID', 1

  UPDATE WorkshopReservation
    SET NumOfParicipants = NumOfParicipants -1
    where WorkshopReservationID in
    (SELECT WorkshopReservationID FROM WorkshopReservations)

  UPDATE WorkshopParticipant
    SET RemoveDate = GETDATE()
    where WorkshopParicipantID in
    (SELECT WorkshopParicipantID from WorkshopReservations)


  COMMIT TRANSACTION ;
```

```
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.9 payForConferenceDay

```
CREATE PROCEDURE dbo.payForConferenceDay
  @ConferenceID int,
  @AmountSend numeric(16, 2)

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;


  IF @ConferenceID IS NULL
     THROW 51000, '@ConferenceID is null', 1

  IF @AmountSend IS NULL OR @AmountSend < 0
     THROW 51000, '@AmountSend is null or less then 0', 1

  UPDATE CDReservation
     SET CDReservation.PaidAmount = CDReservation.PaidAmount + @AmountSend
     where ConferenceDayID = @ConferenceID

  IF @@ROWCOUNT = 0
     THROW 51000, 'no reservation with such id', 1

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.10 payForWorkshop

```
CREATE PROCEDURE dbo.payForWorkshop
  @WorkshopID int,
  @AmountSend numeric(16, 2)

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;


  IF @WorkshopID IS NULL
     THROW 51000, '@WorkshopID is null', 1

  IF @AmountSend IS NULL OR @AmountSend < 0
     THROW 51000, '@AmountSend is null or less then 0', 1

  UPDATE WorkshopReservation
     SET PaidAmount = PaidAmount + @AmountSend
     where WorkshopID = @WorkshopID

  IF @@ROWCOUNT = 0
     THROW 51000, 'no reservation with such id', 1

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.11 removeConferenceReservationSlot

```
CREATE PROCEDURE dbo.removeConferenceDayReservationSlot
  @ConferenceDayID int,
  @ClientID int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ConferenceDayID IS NULL or @ClientID is NULL
     THROW 51000, '@ConferenceDayID or @ClientID is null', 1

  SELECT NumOfParicipants
  INTO NumOfPart
  from CDReservation CDR
        where CDR.ConferenceDayID = @ConferenceDayID and CDR.ClientsClientID = @ClientID
```

```
  IF @@ROWCOUNT = 0
     THROW 51000, '@ClintID has no reservation on @ConferenceDayID', 1

  IF (SELECT * FROM NumOfPart)
     =
    (SELECT COUNT(*) from CDReservation CDR
        inner join ConferenceDayParicipant CDPw on CDR.CDReservationID =
CDPw.CDReservationID
        where CDR.ConferenceDayID = @ConferenceDayID and CDR.ClientsClientID = @ClientID
        group by CDR.CDReservationID)
     THROW 51000, 'all slots are filled with signed participants', 1
  ELSE IF (SELECT NumOfParicipants from CDReservation CDR
        where CDR.ConferenceDayID = @ConferenceDayID and CDR.ClientsClientID = @ClientID)
= 1
     THROW 51000, 'only 1 slot reserved. You have to cancel the whole reservation', 1
  ELSE
     UPDATE CDReservation
        SET NumOfParicipants = NumOfParicipants -1
        where ConferenceDayID = @ConferenceDayID and ClientsClientID = @ClientID


  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.12 removeReservationsNoParticipantsList

Procedura, która ma za zadanie usunąć klientów, którzy nie podali pełnej listy uczestników na swoje rezerwacje na 7 dni przed dniem konferencji.

```
CREATE PROCEDURE dbo.removeReservationsNoParticipantsList

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  Select DISTINCT CD.ConferenceDayID, CDR.ClientsClientID
  INTO ConferenceDayReservationToCanel
  from CDReservation CDR
     inner join ConferenceDay CD on CDR.ConferenceDayID = CD.ConferenceDayID
     where CDR.NumOfParicipants > (SELECT COUNT(*) from CDReservation CDRw
        inner join ConferenceDayParicipant CDPw on CDRw.CDReservationID =
CDPw.CDReservationID
        where CDRw.CDReservationID = CDR.CDReservationID
        group by CDRw.CDReservationID) and DATEDIFF(day ,CD.Date, GETDATE()) <= 7 and
CD.Date > GETDATE()

  iF @@ROWCOUNT > 0
```

```
BEGIN
   DECLARE @ConferenceDayID int
   DECLARE @ConfClientID int
   DECLARE ConferenceDaysCursor CURSOR LOCAL FAST_FORWARD
   FOR SELECT * FROM ConferenceDayReservationToCanel

   OPEN ConferenceDaysCursor
   FETCH NEXT FROM ConferenceDaysCursor INTO @ConferenceDayID, @ConfClientID
   WHILE @@FETCH_STATUS = 0
      BEGIN
         EXEC cancelConferenceDayReservationClient @ConfClientID, @ConferenceDayID
         FETCH NEXT FROM ConferenceDaysCursor
         INTO @ConferenceDayID, @ConfClientID
      END
   CLOSE ConferenceDaysCursor
   DEALLOCATE ConferenceDaysCursor
END

SELECT DISTINCT W.WorkshopID, CDR.ClientsClientID
INTO WorkshopReservationToCancel
from WorkshopReservation WR
   inner join CDReservation CDR on WR.CDReservationID = CDR.CDReservationID
   inner join ConferenceDay CD on CDR.ConferenceDayID = CD.ConferenceDayID
   inner join Workshop W on WR.WorkshopID = W.WorkshopID
   where WR.NumOfParicipants > (SELECT COUNT(*) from WorkshopReservation WRw
      inner join WorkshopParticipant WPw on WRw.WorkshopReservationID =
WPw.WorkshopReservationID
      where WRw.WorkshopReservationID = WR.WorkshopReservationID
      group by WRw.WorkshopReservationID) and DATEDIFF(day ,CD.Date, GETDATE()) <= 7
and CD.Date > GETDATE()

iF @@ROWCOUNT > 0
BEGIN
   DECLARE @WorkshopID int
   DECLARE @WorkClientID int
   DECLARE WorkshopCursor CURSOR LOCAL FAST_FORWARD
   FOR SELECT * FROM WorkshopReservationToCancel

   OPEN WorkshopCursor
   FETCH NEXT FROM WorkshopCursor INTO @WorkshopID, @WorkClientID
   WHILE @@FETCH_STATUS = 0
      BEGIN
         EXEC cancelWorkshopReservationClient @WorkClientID, @WorkshopID
         FETCH NEXT FROM WorkshopCursor
         INTO @WorkshopID, @WorkClientID
      END
   CLOSE WorkshopCursor
   DEALLOCATE WorkshopCursor
END


COMMIT TRANSACTION ;
END TRY
BEGIN CATCH
```

```
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.13 removeReservationsNotPaid

Procedura do usuwania rezerwacji klientów, którzy nie opłacili ich na 7 dni przed dniem konferencji.

```
CREATE PROCEDURE dbo.removeReservationsNotPaid

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  Select DISTINCT CD.ConferenceDayID, CDR.ClientsClientID,  CD.Date, CDR.ReservationDate
  INTO ConferenceDayReservationToCancel
  from CDReservation CDR
     inner join ConferenceDay CD on CDR.ConferenceDayID = CD.ConferenceDayID
     where CDR.PaidAmount <
     (SELECT dbo.finalPrice(CD.FeeForDay,
        (SELECT COUNT(*) from ConferenceDay CDw
           inner join CDReservation CDRw on CDw.ConferenceDayID = CDRw.ConferenceDayID
           inner join ConferenceDayParicipant CDPw on CDRw.CDReservationID =
CDPw.CDReservationID
           inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
           where Pw.StudentCardID IS NOT NULL and CDRw.CDReservationID =
CDR.CDReservationID
           group by CDRw.CDReservationID
        ),
        (SELECT COUNT(*) from ConferenceDay CDw
           inner join CDReservation CDRw on CDw.ConferenceDayID = CDRw.ConferenceDayID
           inner join ConferenceDayParicipant CDPw on CDRw.CDReservationID =
CDPw.CDReservationID
           inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
           where Pw.StudentCardID IS NULL and CDRw.CDReservationID = CDR.CDReservationID
           group by CDRw.CDReservationID
        ),
        CD.Date, CDR.ReservationDate) as price) and DATEDIFF(day ,CD.Date, GETDATE()) <= 7
and CD.Date > GETDATE()

  iF @@ROWCOUNT > 0
  BEGIN
     DECLARE @ConferenceDayID int
     DECLARE @ConfClientID int
     DECLARE ConferenceDaysCursor CURSOR LOCAL FAST_FORWARD
     FOR SELECT * FROM ConferenceDayReservationToCancel

     OPEN ConferenceDaysCursor
     FETCH NEXT FROM ConferenceDaysCursor INTO @ConferenceDayID, @ConfClientID
     WHILE @@FETCH_STATUS = 0
        BEGIN
```

```
            EXEC cancelConferenceDayReservationClient @ConfClientID, @ConferenceDayID
            FETCH NEXT FROM ConferenceDaysCursor
            INTO @ConferenceDayID, @ConfClientID
        END
    CLOSE ConferenceDaysCursor
    DEALLOCATE ConferenceDaysCursor
END


Select DISTINCT WR.WorkshopID, CDR.ClientsClientID,  CD.Date, CDR.ReservationDate
INTO WorkshopReservationToCancel
from WorkshopReservation WR
    inner join CDReservation CDR on WR.CDReservationID = CDR.CDReservationID
    inner join ConferenceDay CD on CDR.ConferenceDayID = CD.ConferenceDayID
    inner join Workshop W on WR.WorkshopID = W.WorkshopID
    where WR.PaidAmount <
    (SELECT dbo.finalPrice(W.Fee,
        (SELECT COUNT(*) from WorkshopReservation WRw
            inner join WorkshopParticipant WPw on WRw.WorkshopReservationID =
WPw.WorkshopReservationID
            inner join ConferenceDayParicipant CDPw on WPw.ConferenceDayParicipantID =
CDPw.ConferenceDayParicipantID
            inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
            where Pw.StudentCardID IS NOT NULL and WRw.WorkshopReservationID =
WR.WorkshopReservationID
            group by WRw.WorkshopReservationID
        ),
        (SELECT COUNT(*) from WorkshopReservation WRw
            inner join WorkshopParticipant WPw on WRw.WorkshopReservationID =
WPw.WorkshopReservationID
            inner join ConferenceDayParicipant CDPw on WPw.ConferenceDayParicipantID =
CDPw.ConferenceDayParicipantID
            inner join Participant Pw on CDPw.ParicipantID = Pw.ParicipantID
            where Pw.StudentCardID IS NULL and WRw.WorkshopReservationID =
WR.WorkshopReservationID
            group by WRw.WorkshopReservationID
        ),
        CD.Date, WR.ReservationDate) as price) and DATEDIFF(day ,CD.Date, GETDATE()) <= 7
and CD.Date > GETDATE()

  iF @@ROWCOUNT > 0
  BEGIN
    DECLARE @WorkshopID int
    DECLARE @WorkClientID int
    DECLARE WorkshopCursor CURSOR LOCAL FAST_FORWARD
    FOR SELECT * FROM WorkshopReservationToCancel

    OPEN WorkshopCursor
    FETCH NEXT FROM WorkshopCursor INTO @WorkshopID, @WorkClientID
    WHILE @@FETCH_STATUS = 0
        BEGIN
            EXEC cancelWorkshopReservationClient @WorkClientID, @WorkshopID
            FETCH NEXT FROM WorkshopCursor
            INTO @WorkshopID, @WorkClientID
```

```
        END
      CLOSE WorkshopCursor
      DEALLOCATE WorkshopCursor
  END

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.14 removeWorkshopReservationSlot

```
CREATE PROCEDURE dbo.removeWorkshopReservationSlot
  @WorkshopID int,
  @ClientID int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @WorkshopID IS NULL or @ClientID IS NULL
     THROW 51000, '@WorkshopID or @ClientID is null', 1

  SELECT WR.NumOfParicipants
  INTO NumOfPart
  from WorkshopReservation WR
     inner join CDReservation CDR on WR.CDReservationID = CDR.CDReservationID
        where WR.WorkshopID = @WorkshopID and CDR.ClientsClientID = @ClientID

  IF @@ROWCOUNT = 0
     THROW 51000, '@ClientID has no reservation on @WorkshopID', 1

  IF (SELECT * FROM NumOfPart)
     =
    (SELECT COUNT(*) from WorkshopReservation WR
        inner join WorkshopParticipant WPw on WR.WorkshopReservationID =
WPw.WorkshopReservationID
        where WR.WorkshopID = @WorkshopID
        group by WR.WorkshopReservationID)

     THROW 51000, 'all slots are filled with signed participants', 1

  ELSE IF (SELECT * FROM NumOfPart) = 1
     THROW 51000, 'only 1 slot reserved. You have to cancel the whole reservation', 1
  ELSE
     UPDATE WorkshopReservation
        SET NumOfParicipants = NumOfParicipants -1
        where WorkshopID = @WorkshopID and WorkshopReservationID IN
        (SELECT WR.WorkshopReservationID FROM WorkshopReservation WR
```

```
        inner join CDReservation CDR on WR.CDReservationID = CDR.CDReservationID
        where WR.WorkshopID = @WorkshopID and CDR.ClientsClientID = @ClientID)


  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.15 updateCompanyClient

```
CREATE PROCEDURE dbo.updateCompanyClient
  @CompanyID int,
  @AddressID int,
  @CompanyName varchar(50),
  @ContactName varchar(50),
  @Phone varchar(50),
  @Fax varchar(50),
  @Email varchar(50)

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @CompanyID IS NULL
    THROW 51000, '@CompanyID is null', 1

  IF @CompanyID not in (SELECT CompanyID from CompanyClients)
    THROW 51000, 'there is no company on @CompanyID', 1

  IF @AddressID IS NOT NULL
    UPDATE CompanyClients
      SET AddressID = @AddressID
      where CompanyID = @CompanyID

  IF @CompanyName IS NOT NULL
    UPDATE CompanyClients
      SET CompanyName = @CompanyName
      where CompanyID = @CompanyID

  IF @ContactName IS NOT NULL
    UPDATE CompanyClients
      SET ContactName = @ContactName
      where CompanyID = @CompanyID

  IF @Phone IS NOT NULL
    UPDATE CompanyClients
      SET Phone = @Phone
      where CompanyID = @CompanyID
```

```
  IF @Fax IS NOT NULL
    UPDATE CompanyClients
      SET Fax = @Fax
      where CompanyID = @CompanyID

  IF @Email IS NOT NULL
    UPDATE CompanyClients
      SET Email = @Email
      where CompanyID = @CompanyID

  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.16 updateConferenceDayData

```
CREATE PROCEDURE dbo.updateConferenceDayData
  @ConferenceDayID int,
  @StartTime time,
  @EndTime time,
  @MaxAmounfOfPeople int,
  @FeeForDay int

AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF NOT EXISTS ((SELECT * from ConferenceDay
    where ConferenceDayID = @ConferenceDayID))
    THROW 51000, '@ConferenceDayID does not exists in ConferenceDay', 1

  IF @StartTime > @EndTime
    THROW 5100, 'Conference is starting that day later than it ends', 1

  IF @MaxAmounfOfPeople <= 0
    THROW 5100, 'Number of people is lower than 0', 1

  IF @FeeForDay <= 0
    THROW 5100, 'Fee for this day is lower than 0', 1

  UPDATE ConferenceDay
    SET StartTime = @StartTime, EndTime = @EndTime, MaxAmountOfPeople =
@MaxAmounfOfPeople, FeeForDay = @FeeForDay
    WHERE ConferenceDayID = @ConferenceDayID

  COMMIT TRANSACTION ;
  END TRY
```

```
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.17 updateParticipant

```
CREATE PROCEDURE dbo.updateParticipant
  @ParticipantID int,
  @FirstName varchar(50),
  @LastName varchar(50),
  @Phone varchar(50),
  @Email varchar(50),
  @StudentCardID varchar(50)


AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @ParticipantID IS NULL
     THROW 51000, '@PrivateClientID is null', 1

  IF @ParticipantID not in (SELECT ParicipantID from Participant)
     THROW 51000, 'there is no participant on @ParticipantID', 1


  IF @FirstName IS NOT NULL
     UPDATE Participant
        SET Firstname= @FirstName
        where @ParticipantID = @ParticipantID

  IF @LastName IS NOT NULL
     UPDATE Participant
        SET Lastname= @LastName
        where @ParticipantID = @ParticipantID

  IF @Phone IS NOT NULL
     UPDATE Participant
        SET Phone = @Phone
        where @ParticipantID = @ParticipantID

  IF @Email IS NOT NULL
     UPDATE Participant
        SET Email = @Email
        where @ParticipantID = @ParticipantID

   IF @StudentCardID IS NOT NULL
     UPDATE Participant
        SET StudentCardID= @StudentCardID
        where @ParticipantID = @ParticipantID
```

```
  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

2.18 updatePrivateClient

```
CREATE PROCEDURE dbo.updatePrivateClient
  @PrivateClientID int,
  @AddressID int,
  @FirstName varchar(50),
  @LastName varchar(50),
  @Phone varchar(50),
  @Email varchar(50)


AS BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
  BEGIN TRANSACTION;

  IF @PrivateClientID IS NULL
     THROW 51000, '@PrivateClientID is null', 1

  IF @PrivateClientID not in (SELECT PrivateClientID from PrivateClients)
     THROW 51000, 'there is no client on @PrivateClientID', 1

  IF @AddressID IS NOT NULL
     UPDATE PrivateClients
        SET AddressID= @AddressID
        where PrivateClientID = @PrivateClientID

  IF @FirstName IS NOT NULL
     UPDATE PrivateClients
        SET Firstname= @FirstName
        where PrivateClientID = @PrivateClientID

  IF @LastName IS NOT NULL
     UPDATE PrivateClients
        SET Lastname= @LastName
        where PrivateClientID = @PrivateClientID

  IF @Phone IS NOT NULL
     UPDATE PrivateClients
        SET Phone = @Phone
        where PrivateClientID = @PrivateClientID

  IF @Email IS NOT NULL
     UPDATE PrivateClients
```

```
        SET Email = @Email
        where PrivateClientID = @PrivateClientID


  COMMIT TRANSACTION ;
  END TRY
  BEGIN CATCH
  ROLLBACK TRANSACTION;
  THROW
  END CATCH
END
go
```

3. Procedury zwracające dane

3.1 clientsNotUpdatedParticipantList

```
CREATE PROCEDURE ClientsNotUpdatedParticipantList

AS
  BEGIN
     SET NOCOUNT ON;

  Select DISTINCT ClientsClientID from CDReservation CDR
     inner join ConferenceDay CD on CDR.ConferenceDayID = CD.ConferenceDayID
     where CDR.NumOfParicipants > (SELECT COUNT(*) from CDReservation CDRw
     inner join ConferenceDayParicipant CDPw on CDRw.CDReservationID =
CDPw.CDReservationID
     where CDRw.CDReservationID = CDR.CDReservationID
     group by CDRw.CDReservationID) and DATEDIFF(day ,CD.Date, GETDATE()) <= 14 and
CD.Date > GETDATE()
  end
go
```

3.2 ParticipantIndentificators

```
CREATE PROCEDURE ParticipantsIndentificators
  @ConferenceID int
AS
  BEGIN
      IF @ConferenceID IS NULL
         THROW 51000, '@@ConferenceID is null', 1

  SELECT
     CONVERT(NVARCHAR, C.ConferenceID) + '/' +
     CONVERT(NVARCHAR, CDR.ClientsClientID) + '/' +
     CONVERT(NVARCHAR, P.ParicipantID) AS id
     FROM Conference C
     INNER JOIN ConferenceDay CD on C.ConferenceID = CD.ConferenceID
     INNER JOIN CDReservation CDR on CD.ConferenceDayID = CDR.ConferenceDayID
     INNER JOIN ConferenceDayParicipant CDP on CDR.CDReservationID =
CDP.CDReservationID
     INNER JOIN Participant P on CDP.ParicipantID = P.ParicipantID
```

```
  end
go
```

4. Funkcje

4.1 finalPrice

Funkcja do obliczania ostatecznej ceny za warsztat lub dzień konferencji.

```
CREATE FUNCTION finalPrice(
  @BasePrice numeric(16,2),
  @StudentRes int,
  @NormalRes int,
  @ReservationDate Date,
  @StartDate Date)
  RETURNS numeric(16, 2)
  WITH RETURNS NULL ON NULL INPUT, SCHEMABINDING
AS BEGIN

  DECLARE @PriceModifier numeric(3,2)
  DECLARE @DaysDiff int
  DECLARE @StudentDiscount numeric(3,2)

  SET @DaysDiff = DATEDIFF(day, @StartDate, @ReservationDate)
  SET @StudentDiscount = 0.1

  IF @DaysDiff >= 21
     SET @PriceModifier = 0.75
  ELSE IF @DaysDiff >=14
     SET @PriceModifier = 0.85
  ELSE
     SET @PriceModifier = 1.00

  DECLARE @price NUMERIC(20, 6) = @NormalRes * @BasePrice * @PriceModifier +
                        @StudentRes * @BasePrice * @PriceModifier * @StudentDiscount
  DECLARE @result NUMERIC(16, 2) = CONVERT(NUMERIC(16, 2), ROUND(@price, 2))
  RETURN @result
END
go
```

4.2 countOfParticipantActivitiesInRange

Funkcja, która zlicza, ile dany uczestnik w zadanym przedziale czasowym ma warsztatów.

```
CREATE FUNCTION dbo.countOfParticipantActivitiesInTimeRange (
  @ParticipantID int,
  @Day date,
  @Begin time,
  @End time
)
RETURNS int AS
```

```
  BEGIN
    DECLARE @Result int;
    SELECT @Result = COUNT(W.WorkshopID) FROM Participant P INNER JOIN
ConferenceDayParicipant CDP on P.ParicipantID = CDP.ParicipantID
        INNER JOIN  WorkshopParticipant WP on CDP.ConferenceDayParicipantID =
WP.ConferenceDayParicipantID
        INNER JOIN WorkshopReservation WR on WP.WorkshopReservationID =
WR.WorkshopReservationID
        INNER JOIN Workshop W on WR.WorkshopID = W.WorkshopID
        INNER JOIN ConferenceDay CD on W.ConferenceDayID = CD.ConferenceDayID
        WHERE CD.Date = @Day AND P.ParicipantID = @ParticipantID AND (
            (W.EndTime BETWEEN @Begin AND @End) OR
            (W.StartTime BETWEEN @Begin AND @End) OR
            (@Begin > W.StartTime AND @End < W.EndTime)
          )
    RETURN @Result
  end
go
```

4.3 freePlacesOnConfereneDay

```
CREATE FUNCTION freePlacesOnConferenceDay(
  @ConferenceDayID int)
  RETURNS int
  WITH RETURNS NULL ON NULL INPUT, SCHEMABINDING
AS BEGIN

  DECLARE @MaxAmount int;
  DECLARE @OccupiedPlaces int;

  SELECT @MaxAmount = MaxAmountOfPeople FROM dbo.ConferenceDay WHERE
ConferenceDayID = @ConferenceDayID
  SELECT @OccupiedPlaces =  SUM(NumOfParicipants) FROM dbo.CDReservation WHERE
ConferenceDayID = @ConferenceDayID

  IF @OccupiedPlaces IS NULL
    RETURN @MaxAmount

  RETURN @MaxAmount - @OccupiedPlaces
END
go
```

4.4 freePlacesOnWorkshop

```
CREATE FUNCTION freePlacesOnWorkshop(
  @WorkshopID int)
  RETURNS int
  WITH RETURNS NULL ON NULL INPUT, SCHEMABINDING
AS BEGIN

  DECLARE @MaxAmount int;
  DECLARE @OccupiedPlaces int;
```

```
  SET @MaxAmount = (SELECT MaxAmountOfPeople FROM dbo.Workshop WHERE
WorkshopID = @WorkshopID)
  SET @OccupiedPlaces =  (SELECT SUM(NumOfParicipants) FROM dbo.WorkshopReservation
WHERE WorkshopID = @WorkshopID GROUP BY WorkshopReservationID)

  IF @OccupiedPlaces IS NULL
    RETURN @MaxAmount

  RETURN @MaxAmount - @OccupiedPlaces
END
go
```

5. Triggery

5.1 tr_CreateClientRecord1

Automatyczne dodawanie rekordu w tabeli Clients w momencie dodawania rekordu w tabeli PrivateClients i połączenie ich ze sobą.

```
CREATE TRIGGER dbo.tr_CreateClientRecord1
  ON dbo.PrivateClients
  AFTER INSERT
AS BEGIN
  DECLARE @PrivateClientID int

  IF @@ROWCOUNT = 0
    RETURN

  IF @@ROWCOUNT = 1
    BEGIN
      SELECT @PrivateClientID = PrivateClientID
      FROM INSERTED

      INSERT INTO Clients
        VALUES (@PrivateClientID, NULL)
    END
  ELSE BEGIN

    DECLARE ClientsCursor CURSOR LOCAL FAST_FORWARD
    FOR SELECT DISTINCT PrivateClientID FROM INSERTED

    OPEN ClientsCursor
    FETCH NEXT FROM ClientsCursor INTO @PrivateClientID
    WHILE @@FETCH_STATUS = 0
      BEGIN
        INSERT INTO Clients
          VALUES (@PrivateClientID, NULL)
        FETCH NEXT FROM ClientsCursor
        INTO @PrivateClientID
      END
    CLOSE ClientsCursor
    DEALLOCATE ClientsCursor
```

```
  END
END
go
```

5.2 tr_CreateClientRecord2

Automatyczne dodawanie rekordu w tabeli Clients w momencie dodawania rekordu w tabeli CompanyClients i połączenie ich ze sobą.

```
CREATE TRIGGER dbo.tr_CreateClientRecord2
  ON dbo.CompanyClients
  AFTER INSERT
AS BEGIN
  DECLARE @CompanyID int

  IF @@ROWCOUNT = 0
    RETURN

  IF @@ROWCOUNT = 1
    BEGIN
      SELECT @CompanyID = CompanyID
      FROM INSERTED

      INSERT INTO Clients
        VALUES (NULL, @CompanyID)
    END
  ELSE BEGIN

    DECLARE ClientsCursor CURSOR LOCAL FAST_FORWARD
    FOR SELECT DISTINCT CompanyID FROM INSERTED

    OPEN ClientsCursor
    FETCH NEXT FROM ClientsCursor INTO @CompanyID
    WHILE @@FETCH_STATUS = 0
      BEGIN
        INSERT INTO Clients
          VALUES (@CompanyID, NULL)
        FETCH NEXT FROM ClientsCursor
        INTO @CompanyID
      END
    CLOSE ClientsCursor
    DEALLOCATE ClientsCursor

  END
END
go
```

5.3 addConferenceDays

Po utworzeniu rekordu z nową konferencją, są automatycznie tworzone rekordy w tabeli ConferenceDay odpowiadające każdemu dniu w trakcie konferencji.

```
CREATE TRIGGER dbo.addConferenceDays ON Conference
  FOR INSERT
  NOT FOR REPLICATION
  AS
  BEGIN

     DECLARE @DateIt date;
     SELECT  @DateIt = [Begin] FROM inserted;
     DECLARE @ConferenceID int;
     SELECT @ConferenceID = ConferenceID FROM inserted;

     WHILE @DateIt <= (SELECT [End] FROM inserted)
     BEGIN
        EXEC dbo.addDefaultConferenceDay @ConferenceID , @DateIt
        SET @DateIt = DATEADD(day, 1, @DateIt)
     END

  end
go
```

5.4 checkPossibilityToReserveWorkshop

Po dodaniu rezerwacji warsztatu, są sprawdzane warunki czy można zarezerwować ten warsztat, jeśli tak - pozostaje on zarezerwowany, jeśli nie - transakcja jest odrzucana.

```
CREATE TRIGGER dbo.checkPossibilityToReserveWorkshop ON WorkshopReservation
  FOR INSERT AS
  BEGIN

     DECLARE @Workshop int;
     DECLARE @CDReservation int;
     SELECT @Workshop = WorkshopID, @CDReservation = CDReservationID FROM inserted;
     IF (SELECT ConferenceDayID FROM Workshop WHERE Workshop.WorkshopID = @Workshop)
        != (SELECT ConferenceDayID FROM CDReservation WHERE CDReservation.CDReservationID
= @CDReservation)
        BEGIN
           ROLLBACK TRANSACTION
           RAISERROR('Podany warsztat nie należy do tego dnia konferencji, co rezerwacja', 16, 1)
        end

     IF (SELECT MaxAmountOfPeople FROM Workshop WHERE Workshop.WorkshopID =
@Workshop) - (SELECT SUM(NumOfParicipants)
        FROM WorkshopReservation WHERE WorkshopID = @Workshop) < 0
        BEGIN
           ROLLBACK TRANSACTION
           RAISERROR('Na warsztat jest dostępnych mniej miejsc niż chcesz zarezerwować', 16, 1)
        end

  end
go
```

Wygenerowane dane

Generator

Baza została wypełniona przez dane generowane z dwóch źródeł:

1. Strona generatedata.com - do wygenerowania prostych danych dla relacji Address, Conference(podstawowa część), PrivateClients oraz CompanyClients
2. Własny generator napisany w języku C++ - do danych w pozostałych tabelach oraz przetworzenie dat w wygenerowanym wcześniej Conference

Pierwszy generator posiada tą zaletę, że zawiera rozbudowany słownik, dzięki któremu można mieć dużą różnorodność wśród wygenerowanych danych, jednak nie sprawdza się przy bardziej skomplikowanych obostrzeniach nakładanych na dane, które muszą być zgodne z krotkami zawartymi w pozostałych relacjach.

Kod własnego generatora został dołączony jako osobny plik do skryptu generującego bazę, a wyniki pracy obu generatorów zostały połączone w pliku generated.sql.

Wygenerowane dane zostały utworzone tak, aby odpowiadały zawartości bazy po 3 latach użytkowania.