Akademia Górniczo-Hutnicza w Krakowie
Wydział Informatyki, Elektroniki i Telekomunikacji

Podstawy baz danych - projekt i implementacja systemu bazodanowego

Miłosz Galas, Paweł Kocimski

Katedra Informatyki
Studia inżynierskie
Drugi rok

Dokumentacja projektu baz danych konferencje

Miłosz Galas, Paweł Kocimski

Aktorzy:

- Administrator
- Organizator
- Klient indywidualny
- Klient zbiorowy

Funkcje systemu:

- Administrator

- Zautomatyzowane funkcje systemu
    - Raz dziennie sprawdzenie czy nie ma nieopłaconych rezerwacji złożonych ponad tydzień temu i usunięcie jeśli takie wystąpiły
    - generowanie opłat za konferencje, warsztaty dla klientów indywidualnych i zbiorowych

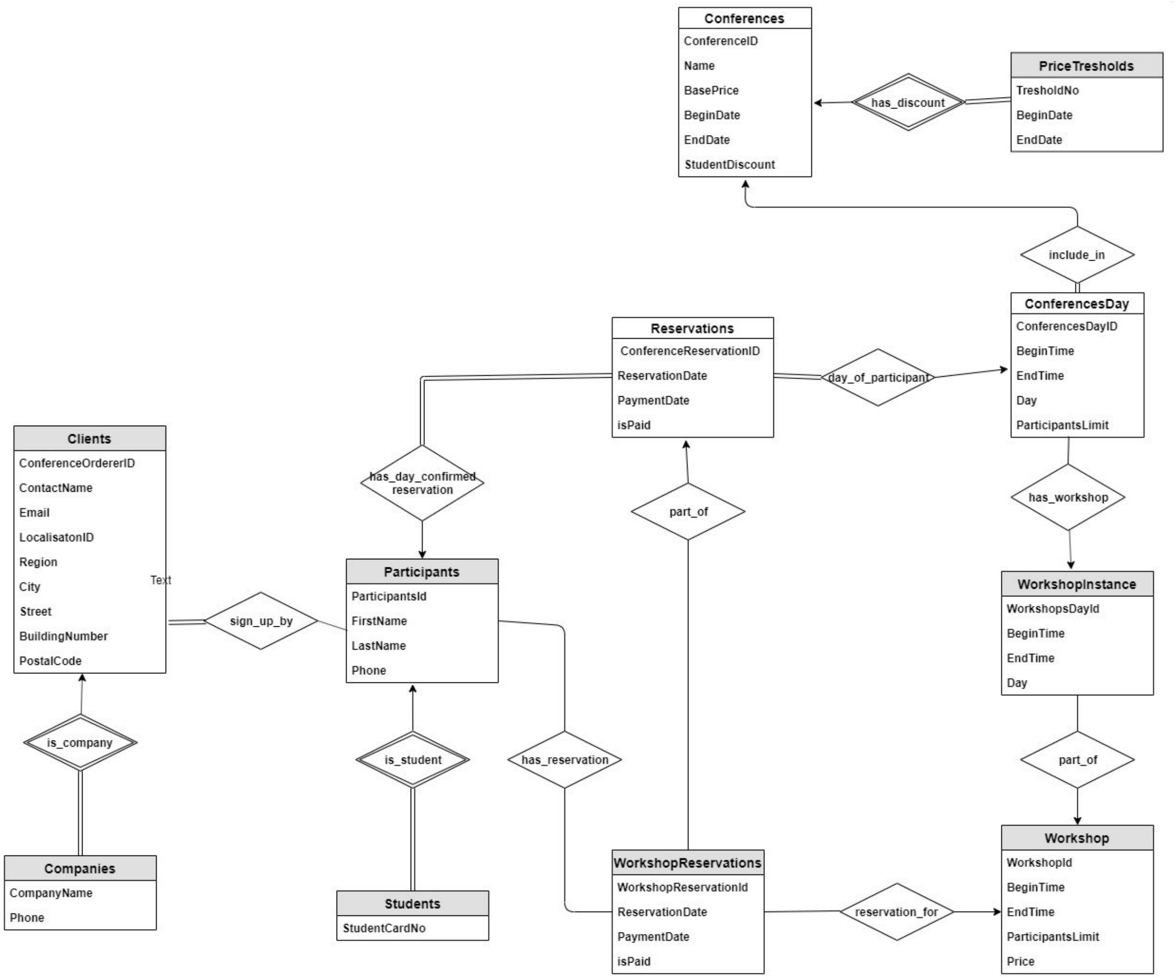
- Organizator
    - Stworzenie konferencji w systemie i podanie wszystkich danych (czas rozpoczęcia/zakończenia, nazwa konferencji, cena, zniżki studenckie, limity uczestników, miejsce)
    - dodanie progów zniżek dla konferencji
    - Stworzenie warsztatów i podanie odpowiednich danych (czas trwania warsztatu, nazwa warsztatów, cena, limit miejsc)
    - Dodanie warsztatów do konferencji
    - Wyświetlanie listy uczestników konkretnej konferencji/warsztatu
    - Wyświetlanie statystyk wygenerowanych przez system, odnośnie uczestników biorących udział w przeszłości w konferencjach danego organizatora (który uczestnik ile razy brał udział w konferencji organizatora)

- Klient indywidualny
    - Rejestracja w systemie (jeśli jest studentem musi podać nr legitymacji)
    - Sprawdzenie czy są dostępne wolne miejsca na konferencje lub warsztat,
    - Sprawdzenie ceny warsztatu, konferencji
    - Zapis na konferencję, warsztaty(jeśli zapisał się wcześniej na konferencję w dniu warsztatu i warsztaty nie zachodzą na siebie)
    - Sprawdzenie własnych rezerwacji, opłat, czasu do kiedy klient musi zapłacić za swoje rezerwacje

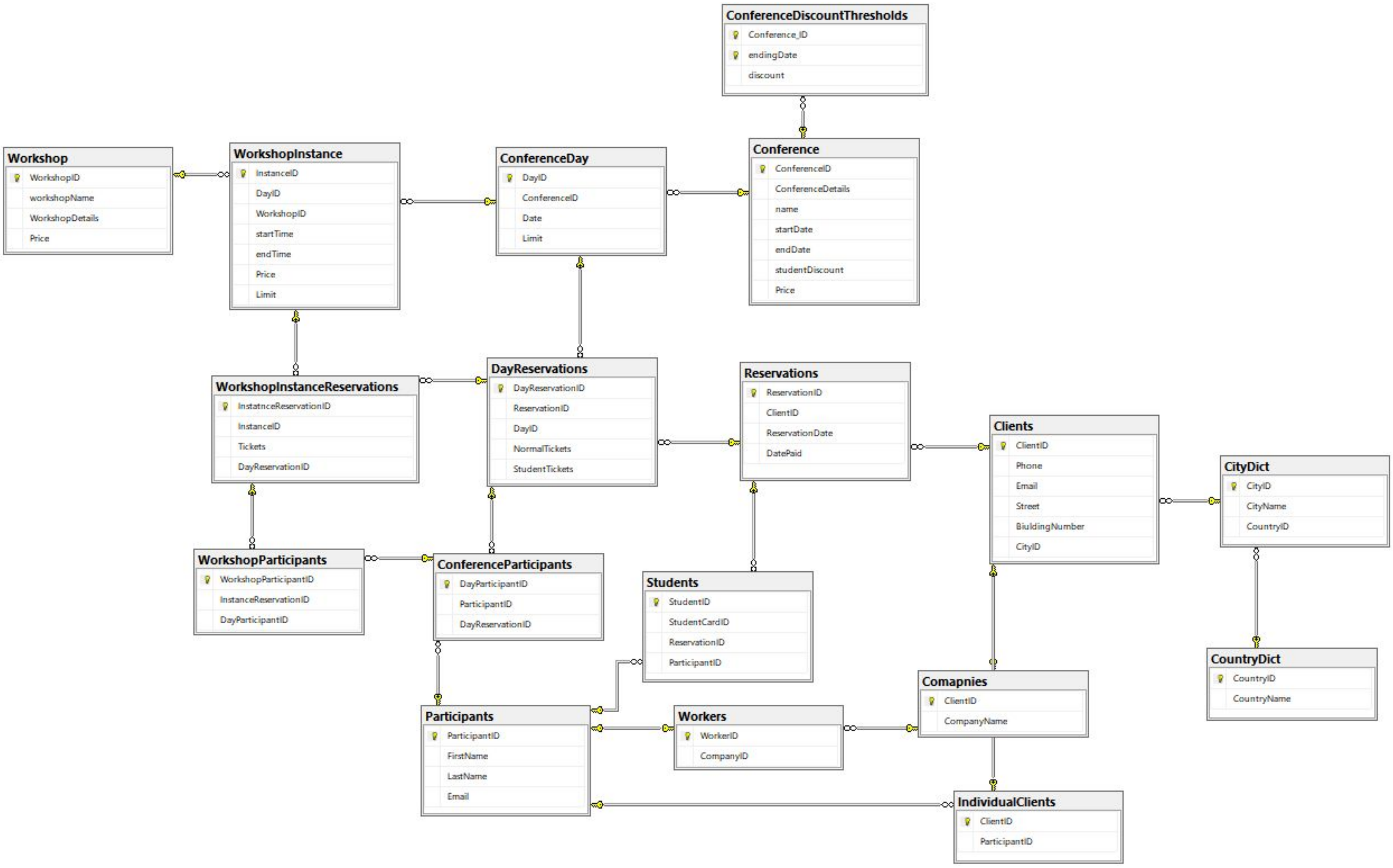
- Klient zbiorowy
    - Rejestracja w systemie klienta zbiorowego
    - Znajdowanie konferencji i warsztatów wraz z ilością wolnych miejsc, ceną, zniżkami

- ○ Rezerwacja odpowiedniej ilości miejsc na konferencję, warsztaty, generowanie kosztu rezerwacji uczestników
- ○ Wprowadzenie danych uczestników najpóźniej 2 tygodnie przed rozpoczęciem konferencji
- ○ Przegląd własnych rezerwacji i obowiązków z nich wynikających (wprowadzenia danych, uiszczenia opłat)

Schemat ER



Schemat bazy danych



Tabele z indexami

1. CityDict - słownik miast

```
CREATE TABLE [dbo].[CityDict](
	[CityID] [INT] IDENTITY(1,1) NOT NULL,
	[CityName] [VARCHAR](20) NOT NULL,
	[CountryID] [INT] NOT NULL,
 CONSTRAINT [CityDict_pk] PRIMARY KEY CLUSTERED ( [CityID] ASC ) WITH
(PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

2. Clients - zawiera informacje o klientach

Warunki integralnościowe:
-Numer telefonu musi być uniwersalny i musi się składać z cyfr
-Email musi być unikalny i musi zawierać “@” i “.”

```
CREATE TABLE Clients (
   ClientID int  NOT NULL IDENTITY,
   Phone char(9)  NOT NULL UNIQUE,
         CHECK ((ISNUMERIC(Phone)=(1))),
   Email varchar(20)  NOT NULL UNIQUE,
         check (Email like '%@%.%') ,
   Street varchar(20)  NOT NULL,
   BiuldingNumber int  NOT NULL,
   CityID int  NOT NULL,
   CONSTRAINT Clients_pk PRIMARY KEY CLUSTERED  (ClientID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

3. Companies - przechowuje dane o firmach

```
CREATE TABLE Comapnies (
   ClientID int  NOT NULL,
   CompanyName varchar(20)  NOT NULL,
   CONSTRAINT Comapnies_pk PRIMARY KEY CLUSTERED  (ClientID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

4. Conference - zawiera informacje ogólne o konferencji

Warunki integralnościowe:
-Konferencja musi się kończyć później niż zaczynać
-Zniżka studencka domyślnie przyjmuje wartość 0 i musi zawierać się w przedziale [0,1]

```
CREATE TABLE Conference (
   ConferenceID int  NOT NULL IDENTITY,
   ConferenceDetails varchar(255)  NOT NULL,
   name varchar(20)  NOT NULL,
   startDate date  NOT NULL,
   endDate date  NOT NULL,
         check (endDate>=startDate),
   studentDiscount int  NULL,
         default 0 for studentDiscount,
         check (studentDiscount<=1 and studentDiscount >=0),
   Price money  NOT NULL,
   CONSTRAINT ID PRIMARY KEY CLUSTERED (ConferenceID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

5. ConferenceDay - zawiera informacje o dniu konferencji

Warunki integralnościowe:
-Limit miejsc musi być większy od 0

```
CREATE TABLE ConferenceDay (
   DayID int  NOT NULL IDENTITY,
   ConferenceID int  NOT NULL,
   Date date  NOT NULL,
   Limit int  NOT NULL,
         check (Limit>0),
   CONSTRAINT ConferenceDay_pk PRIMARY KEY CLUSTERED  (DayID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

6. ConferenceDiscountThresholds - progi cenowe konferencji

Warunki integralnościowe:
-zniżka musi się zawierać w przedziale [0,1]

```
CREATE TABLE ConferenceDiscountThresholds (
   Conference_ID int  NOT NULL,
   endingDate date  NOT NULL,
   discount real  NOT NULL,
         check([Discount] > 0 and [Discount] < 1),
   CONSTRAINT ConferenceDiscountThresholds_pk PRIMARY KEY
(Conference_ID,endingDate)
);
```

7. ConferenceParticipants - zawiera informacje o uczestnikach danego dnia konferencji

```
CREATE TABLE ConferenceParticipants (
   DayParticipantID int  NOT NULL IDENTITY,
   ParticipantID int  NOT NULL,
   DayReservationID int  NOT NULL,
   CONSTRAINT ConferenceParticipants_pk PRIMARY KEY CLUSTERED
(DayParticipantID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

8. CountryDict - słownik krajów

```
CREATE TABLE CountryDict (
   CountryID int  NOT NULL IDENTITY,
   CountryName varchar(20)  NOT NULL,
   CONSTRAINT CountryDict_pk PRIMARY KEY CLUSTERED  (CountryID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

9. DayReservations - dane rezerwacji o dniach konferencji

Warunki integralnościowe:
-liczba biletów normalnych i ulgowych musi być nieujemna, a ich suma dodatnia

```
CREATE TABLE DayReservations (
   DayReservationID int  NOT NULL IDENTITY,
   ReservationID int  NOT NULL,
   DayID int  NOT NULL,
   NormalTickets int  NOT NULL,
         check (normaltickets>=0),
   StudentTickets int  NOT NULL,
         check (studenttickets>=0),
         check (studenttickets+normaltickets>0),
   CONSTRAINT DayReservations_pk PRIMARY KEY CLUSTERED (DayReservationID
ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

10. IndividualClients - przechowuje informacje który klient jest klienten indywidualnym

```
CREATE TABLE IndividualClients (
   ClientID int  NOT NULL,
   ParticipantID int  NOT NULL,
   CONSTRAINT IndividualClients_pk PRIMARY KEY CLUSTERED  (ClientID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

11. Participants - zawiera dane o wszystkich uczestnikach konferencji

Warunki integralnościowe:
-Email musi zawierać ‘@’ i ‘.’

```
CREATE TABLE Participants (
   ParticipantID int  NOT NULL IDENTITY,
   FirstName varchar(10)  NOT NULL,
   LastName varchar(20)  NOT NULL,
   Email varchar(30)  NOT NULL,
         check (Email like '%@%.%') ,

   CONSTRAINT Participants_pk PRIMARY KEY CLUSTERED (ParticipantID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

12. Reservations - zawiera informacje o rezerwacji

Warunki integralnościowe:
-Data rezerwacji domyślnie przyjmuje wartość dzisiejszej daty

```
CREATE TABLE Reservations (
   ReservationID int  NOT NULL IDENTITY,
   ClientID int  NOT NULL,
   ReservationDate date  NULL DEFAULT getdate(),
   DatePaid date  NULL,
   CONSTRAINT Reservations_pk PRIMARY KEY CLUSTERED (ReservationID)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

13. Students - zawiera numery legitymacji studenckich

```
CREATE TABLE Students (
   StudentID int  NOT NULL,
   StudentCardID int  NOT NULL,
   ReservationID int  NOT NULL,
   ParticipantID int  NOT NULL,
   CONSTRAINT Students_pk PRIMARY KEY CLUSTERED (StudentID)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

14. Workers - zawiera informację o tym który uczestnik jest pracownikiem której firmy

```
CREATE TABLE Workers (
   WorkerID int  NOT NULL,
   CompanyID int  NOT NULL,
   CONSTRAINT Workers_pk PRIMARY KEY CLUSTERED (WorkerID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

15. Workshop - słownik warsztatów

```
CREATE TABLE Workshop (
   WorkshopID int  NOT NULL IDENTITY,
   workshopName varchar(20)  NOT NULL,
   WorkshopDetails varchar(255)  NOT NULL,
   Price money  NULL DEFAULT 0,
   CONSTRAINT Workshop_pk PRIMARY KEY CLUSTERED (WorkshopID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

16. WorkshopInstance - Instancja warsztatu

Warunki integralnościowe:

-Cena warsztatu domyślnie jest równa 0

```
CREATE TABLE WorkshopInstance (
   InstanceID int  NOT NULL IDENTITY,
   DayID int  NOT NULL,
   WorkshopID int  NOT NULL,
   startTime time  NOT NULL,
   endTime time  NOT NULL,
   Price money  NULL DEFAULT 0,
   Limit int  NOT NULL,
   CONSTRAINT WorkshopInstance_pk PRIMARY KEY CLUSTERED (InstanceID ASC)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

17. WorkshopInstanceReservations - rezerwacje na warsztaty

Warunki integralnościowe:

-Liczba biletów musi być większa od zera

```
CREATE TABLE WorkshopInstanceReservations (
   InstatnceReservationID int  NOT NULL IDENTITY,
   InstanceID int  NOT NULL,
   Tickets int  NOT NULL,
         check (tickets>0),
   DayReservationID int  NOT NULL,
   CONSTRAINT WorkshopInstanceReservations_pk PRIMARY KEY CLUSTERED
(InstatnceReservationID)
```

18. WorkshopParticipants - zawiera informacje o uczestnikach warsztatu

```
CREATE TABLE WorkshopParticipants (
   WorkshopParticipantID int  NOT NULL IDENTITY,
   InstanceReservationID int  NOT NULL,
   DayParticipantID int  NOT NULL,
   CONSTRAINT WorkshopParticipants_pk PRIMARY KEY CLUSTERED
(WorkshopParticipantID)
WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

Klucze obce

```
-- Reference: CityDict_CountryDict (table: CityDict)
ALTER TABLE CityDict ADD CONSTRAINT CityDict_CountryDict
   FOREIGN KEY (CountryID)
   REFERENCES CountryDict (CountryID);

-- Reference: Client_IndividualClient (table: IndividualClients)
ALTER TABLE IndividualClients ADD CONSTRAINT Client_IndividualClient
   FOREIGN KEY (ClientID)
   REFERENCES Clients (ClientID);

-- Reference: Clients_CityDict (table: Clients)
ALTER TABLE Clients ADD CONSTRAINT Clients_CityDict
   FOREIGN KEY (CityID)
   REFERENCES CityDict (CityID);

-- Reference: Comapnies_Client (table: Comapnies)
ALTER TABLE Comapnies ADD CONSTRAINT Comapnies_Client
   FOREIGN KEY (ClientID)
   REFERENCES Clients (ClientID);

-- Reference: ConferenceDiscountThresholds_Conference (table:
ConferenceDiscountThresholds)
ALTER TABLE ConferenceDiscountThresholds ADD CONSTRAINT
ConferenceDiscountThresholds_Conference
   FOREIGN KEY (Conference_ID)
   REFERENCES Conference (ConferenceID);

-- Reference: ConferenceParticipants_DayReservations (table: ConferenceParticipants)
ALTER TABLE ConferenceParticipants ADD CONSTRAINT
ConferenceParticipants_DayReservations
   FOREIGN KEY (DayReservationID)
   REFERENCES DayReservations (DayReservationID);

-- Reference: ConferenceParticipants_Participants (table: ConferenceParticipants)
ALTER TABLE ConferenceParticipants ADD CONSTRAINT
ConferenceParticipants_Participants
   FOREIGN KEY (ParticipantID)
   REFERENCES Participants (ParticipantID);

-- Reference: Conference_ConferenceDay (table: ConferenceDay)
ALTER TABLE ConferenceDay ADD CONSTRAINT Conference_ConferenceDay
   FOREIGN KEY (ConferenceID)
   REFERENCES Conference (ConferenceID);
```

```
-- Reference: DayReservation_ConferenceDay (table: DayReservations)
ALTER TABLE DayReservations ADD CONSTRAINT DayReservation_ConferenceDay
   FOREIGN KEY (DayID)
   REFERENCES ConferenceDay (DayID);

-- Reference: DayReservation_Revesrvation (table: DayReservations)
ALTER TABLE DayReservations ADD CONSTRAINT DayReservation_Revesrvation
   FOREIGN KEY (ReservationID)
   REFERENCES Reservations (ReservationID);

-- Reference: Participants_IndividualClient (table: IndividualClients)
ALTER TABLE IndividualClients ADD CONSTRAINT Participants_IndividualClient
   FOREIGN KEY (ParticipantID)
   REFERENCES Participants (ParticipantID);

-- Reference: Participants_Workers (table: Workers)
ALTER TABLE Workers ADD CONSTRAINT Participants_Workers
   FOREIGN KEY (WorkerID)
   REFERENCES Participants (ParticipantID);

-- Reference: Revesrvation_Client (table: Reservations)
ALTER TABLE Reservations ADD CONSTRAINT Revesrvation_Client
   FOREIGN KEY (ClientID)
   REFERENCES Clients (ClientID);

-- Reference: StudentCardIDs_Participants (table: Students)
ALTER TABLE Students ADD CONSTRAINT StudentCardIDs_Participants
   FOREIGN KEY (ParticipantID)
   REFERENCES Participants (ParticipantID);

-- Reference: StudentCardIDs_Reservations (table: Students)
ALTER TABLE Students ADD CONSTRAINT StudentCardIDs_Reservations
   FOREIGN KEY (ReservationID)
   REFERENCES Reservations (ReservationID);

-- Reference: Workers_Comapnies (table: Workers)
ALTER TABLE Workers ADD CONSTRAINT Workers_Comapnies
   FOREIGN KEY (CompanyID)
   REFERENCES Comapnies (ClientID);

-- Reference: WorkshopInstanceReservation_WorkshopInstance (table:
WorkshopInstanceReservations)
ALTER TABLE WorkshopInstanceReservations ADD CONSTRAINT
WorkshopInstanceReservation_WorkshopInstance
   FOREIGN KEY (InstanceID)
   REFERENCES WorkshopInstance (InstanceID);

-- Reference: WorkshopInstanceReservations_DayReservations (table:
WorkshopInstanceReservations)
ALTER TABLE WorkshopInstanceReservations ADD CONSTRAINT
WorkshopInstanceReservations_DayReservations
   FOREIGN KEY (DayReservationID)
   REFERENCES DayReservations (DayReservationID);

-- Reference: WorkshopInstance_ConferenceDay (table: WorkshopInstance)
ALTER TABLE WorkshopInstance ADD CONSTRAINT WorkshopInstance_ConferenceDay
   FOREIGN KEY (DayID)
   REFERENCES ConferenceDay (DayID);

-- Reference: WorkshopInstance_Workshop (table: WorkshopInstance)
ALTER TABLE WorkshopInstance ADD CONSTRAINT WorkshopInstance_Workshop
   FOREIGN KEY (WorkshopID)
   REFERENCES Workshop (WorkshopID);

-- Reference: WorkshopParticipants_ConferenceParticipants (table: WorkshopParticipants)
ALTER TABLE WorkshopParticipants ADD CONSTRAINT
WorkshopParticipants_ConferenceParticipants
   FOREIGN KEY (DayParticipantID)
   REFERENCES ConferenceParticipants (DayParticipantID);

-- Reference: WorkshopParticipants_WorkshopInstanceReservations (table:
WorkshopParticipants)
ALTER TABLE WorkshopParticipants ADD CONSTRAINT
WorkshopParticipants_WorkshopInstanceReservations
   FOREIGN KEY (InstanceReservationID)
   REFERENCES WorkshopInstanceReservations (InstatnceReservationID);
```

Widoki

1.CompaniesStats - wyświetla firmy uszeregowane od największej liczby zamówień

```
CREATE VIEW [dbo].[CompaniesStats]

AS
```

```
SELECT TOP (100) PERCENT dbo.Clients.ClientID, dbo.Comapnies.CompanyName,
                    (SELECT COUNT(ReservationID) AS Expr1
                     FROM      dbo.Reservations
                     WHERE   (ClientID = dbo.Clients.ClientID) AND (DatePaid IS NOT NULL)) AS
SumOfReservation
FROM     dbo.Clients INNER JOIN
                dbo.Comapnies ON dbo.Clients.ClientID = dbo.Comapnies.ClientID
GROUP BY dbo.Clients.ClientID, dbo.Comapnies.CompanyName
ORDER BY SumOfReservation DESC
GO
```

2.ConferenceOccupy- zwraca liczbę miejsc wolnych miejsc na konferencję

```
CREATE VIEW [dbo].[ConferenceOccupy]
AS
SELECT dbo.Conference.ConferenceID, dbo.Conference.name, dbo.ConferenceDay.Limit -
                    (SELECT SUM(NormalTickets) AS Expr1
                     FROM      dbo.DayReservations
                     WHERE   (dbo.ConferenceDay.DayID = DayID)) -
                    (SELECT SUM(StudentTickets) AS Expr1
                     FROM      dbo.DayReservations AS DayReservations_2
                     WHERE   (dbo.ConferenceDay.DayID = DayID)) AS EmptySeats
FROM     dbo.Conference INNER JOIN
                dbo.ConferenceDay ON dbo.Conference.ConferenceID =
dbo.ConferenceDay.ConferenceID INNER JOIN
```

```
            dbo.DayReservations AS DayReservations_1 ON dbo.ConferenceDay.DayID = DayReservations_1.DayID

GO
```

3.IndClientsStats-zwraca liczbę rezerwacji dla każdego klienta indywidualnego posortowane od największej liczby rezerwacji

```
CREATE VIEW [dbo].[IndClientsStats]

AS

SELECT TOP (100) PERCENT dbo.Clients.ClientID, dbo.Participants.FirstName, dbo.Participants.LastName,

                (SELECT COUNT(ReservationID) AS S

                 FROM      dbo.Reservations

                 WHERE   (ClientID = dbo.Clients.ClientID) AND (DatePaid IS NOT NULL)) AS SumOfReservation

FROM     dbo.Participants INNER JOIN

            dbo.IndividualClients ON dbo.Participants.ParticipantID = dbo.IndividualClients.ParticipantID INNER JOIN

            dbo.Clients ON dbo.IndividualClients.ClientID = dbo.Clients.ClientID

GROUP BY dbo.Clients.ClientID, dbo.Participants.FirstName, dbo.Participants.LastName

ORDER BY SumOfReservation DESC

GO
```

4. PopularityRateOfConference -Zwraca konferencje uszeregowane od największej liczby zakupionych biletów

```
CREATE VIEW [dbo].[PopularityRateOfConference]

AS

SELECT TOP (100) PERCENT dbo.Conference.ConferenceID,
dbo.DayReservations.DayID, SUM(dbo.DayReservations.NormalTickets) +
SUM(dbo.DayReservations.StudentTickets) AS NumberOfTIckets

FROM     dbo.DayReservations INNER JOIN

                dbo.ConferenceDay ON dbo.DayReservations.DayID =
dbo.ConferenceDay.DayID INNER JOIN

                dbo.Conference ON dbo.ConferenceDay.ConferenceID =
dbo.Conference.ConferenceID

GROUP BY dbo.Conference.ConferenceID, dbo.DayReservations.DayID

ORDER BY NumberOfTIckets DESC

GO
```

5.PopularityRateOfWorkshops-zwraca warsztaty posortowane od tego na który zakupiono największą ilość biletów

```
CREATE VIEW [dbo].[PopularityRateOfWorkshops]

AS

SELECT TOP (100) PERCENT dbo.Workshop.workshopName,
SUM(dbo.WorkshopInstanceReservations.Tickets) AS NumberOfTickets

FROM     dbo.WorkshopInstanceReservations INNER JOIN

                 dbo.DayReservations ON dbo.WorkshopInstanceReservations.DayReservationID
= dbo.DayReservations.DayReservationID INNER JOIN

                 dbo.WorkshopInstance ON dbo.WorkshopInstanceReservations.InstanceID =
dbo.WorkshopInstance.InstanceID INNER JOIN

                 dbo.Workshop ON dbo.WorkshopInstance.WorkshopID =
dbo.Workshop.WorkshopID

GROUP BY dbo.Workshop.workshopName

ORDER BY NumberOfTickets DESC

GO
```

6.UnpaidCompanyReservations - zwraca nie zapłacone rezerwacje firmowe

```
CREATE VIEW [dbo].[UnpaidCompanyReservations]

AS

SELECT dbo.Reservations.ReservationDate, dbo.Comapnies.CompanyName, dbo.Clients.Phone, dbo.Conference.startDate, dbo.Conference.endDate

FROM     dbo.Reservations INNER JOIN

               dbo.Clients ON dbo.Reservations.ClientID = dbo.Clients.ClientID INNER JOIN

               dbo.Comapnies ON dbo.Clients.ClientID = dbo.Comapnies.ClientID INNER JOIN

               dbo.DayReservations ON dbo.Reservations.ReservationID = dbo.DayReservations.ReservationID INNER JOIN

               dbo.ConferenceDay ON dbo.DayReservations.DayID = dbo.ConferenceDay.DayID INNER JOIN

               dbo.Conference ON dbo.ConferenceDay.ConferenceID = dbo.Conference.ConferenceID

WHERE  (dbo.Reservations.DatePaid IS NULL) AND (DATEDIFF(day, dbo.Reservations.ReservationDate, GETDATE()) > 0)

GO
```

7.UnpaidIndividualReservations - zwraca niezapłącone rezerwacje indywidualne

```
CREATE VIEW [dbo].[UnpaidIndividualReservations]
AS
SELECT dbo.Reservations.ReservationDate, dbo.Participants.FirstName,
dbo.Participants.LastName, dbo.Clients.Phone, dbo.Conference.startDate,
dbo.Conference.endDate
FROM     dbo.Conference INNER JOIN
              dbo.ConferenceDay ON dbo.Conference.ConferenceID =
dbo.ConferenceDay.ConferenceID INNER JOIN
              dbo.DayReservations ON dbo.ConferenceDay.DayID =
dbo.DayReservations.DayID INNER JOIN
              dbo.Reservations ON dbo.DayReservations.ReservationID =
dbo.Reservations.ReservationID INNER JOIN
              dbo.Clients ON dbo.Reservations.ClientID = dbo.Clients.ClientID INNER JOIN
              dbo.IndividualClients ON dbo.Clients.ClientID = dbo.IndividualClients.ClientID
INNER JOIN
              dbo.Participants ON dbo.IndividualClients.ParticipantID =
dbo.Participants.ParticipantID
WHERE  (dbo.Reservations.DatePaid IS NULL) AND (DATEDIFF(day,
dbo.Reservations.ReservationDate, GETDATE()) > 0)
GO
```

Funkcje

1. function_GetConferenceDayTakenPlaces - zwraca zajęte miejsca na dzień konferencji

```
CREATE FUNCTION function_GetConferenceDayTakenPlaces (@DayID int)
RETURNS int
AS
BEGIN
RETURN ISNULL((SELECT SUM(NormalTickets) + SUM(StudentTickets)
FROM DayReservations
Where DayID = @DayID), 0)
END
```

2. function_GetConferenceDayFreePlaces - zwraca wolne miejsca na dzień konferencji

```
CREATE function [dbo].[function_GetConferenceDayFreePlaces] (@DayID int)
RETURNS int
AS
BEGIN
RETURN (SELECT limit FROM ConferenceDay where @DayID = DayID) -
dbo.FUNCTION_GetConferenceDayTakenPlaces(@DayID)
END
```

3. function_GetWorkshopInstanceFreePlaces - zwraca wolne miejsca na warsztat

```
create function function_GetWorkshopInstanceFreePlaces (@InstanceID int)
returns int
as
begin
```

```
return ISNULL(
(select limit from WorkshopInstance where InstanceID=@InstanceID)-
(select sum(tickets) from WorkshopInstanceReservations where InstanceID=@InstanceID)
, 0)
END
```

4. function_GetDayParticipantsList - zwraca uczestników danego dnia konferencji

```
CREATE FUNCTION function_GetDayParticipantsList (@DayID int)
RETURNS @DayParticipantsListTable TABLE
(participantID int, name varchar(50), surname varchar(50))
AS
BEGIN
INSERT @DayParticipantsListTable
SELECT DISTINCT cp.participantID, p.FirstName, p.LastName
FROM DayReservations as dr
JOIN ConferenceParticipants as cp ON cp.DayReservationID = dr.DayReservationID
JOIN Participants as p ON p.participantID = cp.participantID
WHERE dr.dayID = @dayID
RETURN
END
GO
```

5. function_GetConferenceDayID - zwraca ID dnia konferencji

```
CREATE FUNCTION function_GetConferenceDayID
(@ConferenceID int, @Date date)
RETURNS int
AS
BEGIN
RETURN (Select DayID
From ConferenceDay
WHERE ConferenceID = @ConferenceID AND Date = @Date)
END
```

6.function_GetWorkshopInstanceParticipantsList - zwraca listę uczestników warsztatu

```
CREATE FUNCTION function_GetWorkshopInstanceParticipantsList (@InstanceID int)
RETURNS @WorkshopInstanceParticipantsListTable TABLE
```

```
(participantID int, name varchar(50), surname varchar(50))
AS
BEGIN
INSERT @WorkshopInstanceParticipantsListTable
        SELECT DISTINCT cp.participantID, p.FirstName, p.LastName
        FROM WorkshopInstanceReservations as wir
        JOIN WorkshopParticipants as wp ON wp.InstanceReservationID =
wir.InstatnceReservationID
        JOIN ConferenceParticipants as cp ON wp.DayParticipantID=cp.DayParticipantID
        JOIN Participants as p ON p.participantID = cp.participantID
        WHERE wir.InstanceID=@InstanceID
RETURN
END
```

7. function_GetConferenceDayLimitID - zwraca limit dnia konferencji

```
CREATE FUNCTION function_GetConferenceDayLimitID (@DayID int)
RETURNS int
AS
BEGIN
RETURN (Select Limit
From ConferenceDay
WHERE DayID = @DayID)
END
```

8. function_GetThresholdOnDate - Zwraca zniżkę na rezerwację konferencji danego dnia

```
CREATE function [dbo].[function_GetThresholdOnDate] (@conferenceID int, @date date)
RETURNS real
AS
BEGIN
RETURN isnull((SELECT top 1 discount
FROM ConferenceDiscountThresholds
where @date < endingDate AND
@conferenceID = Conference_ID
order by endingDate asc) , 0)
END
```

9. function_GetReservationValue - zwraca wartość zamówienia

```
ALTER function [dbo].[function_getReservationValue](@reservationID int)
  returns money
AS
BEGIN
  RETURN (SELECT sum(price *
           (1 - dbo.function_GetThresholdOnDate(conference.conferenceID,
ReservationDate)) *
           (1 - studentDiscount) * studentTickets
    + price * (1 - dbo.function_GetThresholdOnDate(conference.conferenceID,
ReservationDate)) *
     normalTickets)
      FROM dbo.reservations
          inner join dbo.dayReservations on reservations.reservationID =
dayReservations.reservationID
          inner join dbo.ConferenceDay on conferenceday.DayID =
DayReservations.dayID
          inner join dbo.Conference on conference.conferenceID =
conferenceDay.conferenceID
      where reservations.reservationID = @reservationID
     ) + (SELECT sum(price * tickets)
        from dbo.Reservations
            inner join dbo.DayReservations on Reservations.ReservationID =
DayReservations.ReservationID
            inner join dbo.WorkshopInstanceReservations
                  on DayReservations.DayReservationID =
WorkshopInstanceReservations.DayReservationID
            inner join dbo.WorkshopInstance
                  on WorkshopInstanceReservations.InstanceID =
WorkshopInstance.InstanceID
        where Reservations.ReservationID = @reservationID)
END
```

10. Split - dzieli string wejściowy danych rezerwacji

```
CREATE FUNCTION [dbo].[Split] (@sep char(1), @list varchar(3000))
RETURNS table
AS
RETURN (
      WITH Pieces(first, start, stop) AS (
       SELECT 1, 1, CHARINDEX(@sep, @list)
       UNION ALL
       SELECT first + 1, stop + 1, CHARINDEX(@sep, @list, stop + 1)
       FROM Pieces
```

```
          WHERE stop > 0
         )
         SELECT first,
          SUBSTRING(@list, start, CASE WHEN stop > 0 THEN stop-start ELSE 5000 END)
AS second
         FROM Pieces
         )
```

11. function_getInvoice - zwraca fakturę do rezerwacji

```
ALTER FUNCTION [dbo].[function_GetInvoice] (@ReservationID INT)
RETURNS @Invoice TABLE
(Description VARCHAR(300))
AS
BEGIN
INSERT @Invoice
         SELECT CONCAT('Conference: ',name,' on ', Date,' ', NormalTickets, ' normal
ticket(s)', ' Price: ', Price,
         ' discount: ', dbo.function_GetThresholdOnDate(Conference.ConferenceID,
ReservationDate), ' sum: ', Price * NormalTickets * (1- dbo.function_GetThresholdOnDate (
Conference.ConferenceID, ReservationDate )))  FROM dbo.Reservations
         JOIN dbo.DayReservations ON DayReservations.ReservationID =
Reservations.ReservationID
         JOIN dbo.ConferenceDay ON ConferenceDay.DayID = DayReservations.DayID
         JOIN dbo.Conference ON Conference.ConferenceID =
ConferenceDay.ConferenceID
         WHERE NormalTickets > 0 AND Reservations.ReservationID=@ReservationID
INSERT @Invoice
         SELECT CONCAT('Conference: ',name,' on ', Date,' ', StudentTickets, ' Student
ticket(s)', ' Price: ', Price,
         ' total discount: ',
1-((1-dbo.function_GetThresholdOnDate(Conference.ConferenceID,
ReservationDate))*(1-studentDiscount)), ' sum: ',

Price*StudentTickets*(1-dbo.function_GetThresholdOnDate(Conference.ConferenceID,
ReservationDate))*(1-studentDiscount))  FROM dbo.Reservations
         JOIN dbo.DayReservations ON DayReservations.ReservationID =
Reservations.ReservationID
         JOIN dbo.ConferenceDay ON ConferenceDay.DayID = DayReservations.DayID
         JOIN dbo.Conference ON Conference.ConferenceID =
ConferenceDay.ConferenceID
         WHERE StudentTickets > 0 AND Reservations.ReservationID=@ReservationID
INSERT @Invoice
```

```
	SELECT CONCAT('Workshop: ',workshopName, ' price: ', WorkshopInstance.Price, ' 
Tickets: ', Tickets, ' sum: ',
	WorkshopInstance.Price*Tickets)  FROM dbo.Reservations
	JOIN dbo.DayReservations ON DayReservations.ReservationID = 
Reservations.ReservationID
	JOIN dbo.WorkshopInstanceReservations ON 
WorkshopInstanceReservations.DayReservationID = DayReservations.DayReservationID
	JOIN dbo.WorkshopInstance ON WorkshopInstance.InstanceID = 
WorkshopInstanceReservations.InstanceID
	JOIN dbo.Workshop ON Workshop.WorkshopID = WorkshopInstance.WorkshopID
	WHERE Reservations.ReservationID=@ReservationID
INSERT @Invoice
	SELECT CONCAT('Total:    ', dbo.function_getReservationValue(@ReservationID))
RETURN
END
```

Procedury

dodawanie danych pracownika

1.procedure_addClientCompany-dodaje klienta firmowego

```
CREATE PROCEDURE [dbo].[procedure_addClientCompany]
    @companyName varchar(20),
    @phone char(9),
    @email varchar(20),
    @street varchar(20) = NULL,
    @biuldingNumber int ,
    @cityName varchar(20) = NULL,
    @countryName varchar(20) = NULL


AS
    BEGIN
        SET NOCOUNT ON;
            BEGIN TRY
                BEGIN TRAN ADD_ClientCompany
                DECLARE @cityId int
                EXEC @cityId =  procedure_findCity @cityName,
@CountryName, @cityId=0
                 INSERT INTO Clients(Phone,Email, Street, BiuldingNumber,
CityID)
                 VALUES (
                    @phone,
                    @email,
                    @street,
                    @biuldingNumber,
                    @cityId)
                    DECLARE @clientId INT = @@IDENTITY

                    INSERT INTO Comapnies(ClientID, CompanyName)
                    VALUES(@clientId,
                    @companyName);

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

2.procedure_addClientIndividual-dodaje klienta indywidualnego

```
CREATE PROCEDURE [dbo].[procedure_addClientIndividual]
    @firstname varchar(255),
    @lastname varchar(255),
    @email varchar(255),
    @phone char(9),
    @street varchar(255) = NULL,
    @biuldingNumber int = NULL,
    @cityName varchar(255) = NULL,
    @countryName varchar(255) = NULL
    AS
    BEGIN
        SET NOCOUNT ON;
            BEGIN TRY
                BEGIN TRAN ADD_ClientIndividual

                DECLARE @cityId int
                EXEC @cityId =  procedure_findCity @cityName,
@CountryName, @cityId=0
                    INSERT INTO
Clients(Phone,Email,Street,BiuldingNumber,CityID)
                    VALUES(
                    @phone,
                    @email,
                    @street,
                    @biuldingNumber,
                    @cityId)
                    DECLARE @clientId INT = @@IDENTITY

                INSERT INTO Participants(FirstName,LastName,Email)
                VALUES(@firstname,
                    @lastname,
                    @email)
                    DECLARE @participantId INT = @@IDENTITY
```

```
                        INSERT INTO IndividualClients(ClientID, ParticipantID)
                        VALUES(@clientId, @participantId)


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

3.procedure_addConference-dodaje konferencje

```
CREATE PROCEDURE [dbo].[procedure_addConference]
        @Name varchar(20),
        @StartDate date,
        @EndDate date,
        @StudentDiscount real,
        @price money,
        @ConferenceDetails varchar(255),
        @Limit INT,
        @ConferenceId int OUTPUT
        AS
        BEGIN
                SET NOCOUNT ON;
                IF(@StartDate > @EndDate)
                BEGIN;
                        THROW 52000, 'EndDate should not be earlier than StartDate.',1
                END
                IF(@StartDate < GETDATE() )
                BEGIN;
                        THROW 52000, 'Cant add past conference.',1
                END
                IF(@StudentDiscount < 0 OR @StudentDiscount > 1)
                BEGIN
                        ;THROW 52000, 'The discount must be between 0 and 1.',1
        END
        INSERT INTO Conference(name, startDate, endDate, studentDiscounT,
Price, ConferenceDetails)
        VALUES(@Name, @StartDate, @EndDate, @StudentDiscount, @price,
@ConferenceDetails)
        SET @conferenceId = @@IDENTITY

                DECLARE @i date = @StartDate
        WHILE @i <= @EndDate
        BEGIN
            INSERT INTO ConferenceDay(
            ConferenceID, Date,Limit)
            VALUES(@conferenceID, @i, @Limit)
            SET @i = DATEADD(d,1,@i)
        END

    END
```

4.procedure_addReservation-dodaje rezerwacje na konferencje

```
CREATE PROCEDURE [dbo].[procedure_addReservation]
    @clientID int,
    @reservationID int out
    AS
    BEGIN
        SET NOCOUNT ON;
            BEGIN TRY
                BEGIN TRAN ADD_Reservation

                    INSERT INTO Reservations(ClientID, ReservationDate,
DatePaid)
                    VALUES( @clientID, GETDATE(),null)
                    SET @reservationID = @@IDENTITY
            COMMIT TRAN ADD_Reservation
        END TRY
        BEGIN CATCH
            ROLLBACK TRAN ADD_Reservation
            DECLARE @msg NVARCHAR(2048) =
            'Bład dodania rezerwacji:' +
                        CHAR(13) + CHAR(10) + ERROR_MESSAGE();
                        THROW 52000,@msg, 1;
                END CATCH
        END
```

5.procedure_addReservationDayCompany4-dodaje rezerwacje na dzień konferencji dla firmy, jako argument otrzymuje listę składającą się z DayID i nazwisk oddzielonych przecinkami, a poszczególnych dni średnikami

```
procedure [dbo].[procedure_addReservationDayCompany4]
@list varchar(4000),
@clientID int
as
begin
        declare @inTable table(ID int identity (1,1), Val varchar(3000));
                insert into @inTable(Val)
                select second from Split ( ';',@list)



        declare @detailTable table(ID int identity (1,1), detailVal varchar(3000));
        declare @iterator2 int = 1
        declare @card2 int
        declare @index2 int
        declare @iterator int = 1
        Declare @sumsize int =1
        Declare @size int
        declare @oldsize int=0
        declare @oldsizecopy int =0
        declare @startindex int =1;
        DECLARE @iterator3 INT =0
        while @iterator <= (select count(id) from @intable)
                Begin
                Declare @oneday varchar(3000) = (select val from @inTable where
id=@iterator)
```

```
        insert into @detailTable(detailVal)
                select second from Split (',',@oneday)-- 1 string
                set @size  =(select count(id) from  @detailTable)





            set @oldsizecopy=@oldsizecopy +1
            Declare         @dayID int = (select  detailVal from @detailTable
where id=@oldsizecopy)
            set @oldsizecopy=@oldsizecopy +1
            declare         @normalTickets int = (select  detailVal from
@detailTable where id=@oldsizecopy)

            if( ((select limit from ConferenceDay where DayID=@dayID)-(
                select sum(normalTickets)+sum(studenttickets) from
DayReservations where DayID=@dayid)) < @normalTickets+ @size-@oldsize-2)
                begin
                DECLARE @msg NVARCHAR(2048) =
                'Brak wystarczającej ilości miejsc:' + CHAR(13)+CHAR(10) +
ERROR_MESSAGE();
                print @msg
                end
            ELSE

            begin

            insert into Reservations(ClientID,ReservationDate,DatePaid)
            values(@clientID,GETDATE(),null)
            DECLARE @ReservationID int = @@IDENTITY

                insert into DayReservations(ReservationID, DayID,
NormalTickets, StudentTickets)
                values(@reservationID, @dayID, @normalTickets,
@size-@oldsize-2);
                Declare @DayReservationID int =@@IDENTITY

        SELECT @clientid
            SET @iterator3=@normalTickets
            WHILE @iterator3 >0
            BEGIN
                insert into Participants default values
                    Declare @ParticipantID int =@@IDENTITY
                    INSERT INTO dbo.ConferenceParticipants
                    (ParticipantID,
                      DayReservationID)
```

```
                                        VALUES
                                        ( @ParticipantID,
                                          @DayReservationID
                                           )
                                                Declare @DayParticipantID int =@@IDENTITY

                                INSERT INTO Workers(WorkerId,CompanyID)
VALUES(@ParticipantID, @clientid)



                        SET @iterator3 = @iterator3 - 1
        end

                                while @oldsizecopy<@size
                                begin
                                set @oldsizecopy =@oldsizecopy +1

                                        set @card2=(select detailval from @detailTable where
id=@oldsizecopy)
                                        insert into Participants default values
                                        set @participantID   =@@IDENTITY
                                        insert into
Students(StudentCardID,ReservationID,ParticipantID)
                                        values(@card2,@reservationID,@participantid)
                                        Declare @StudentID int =@@IDENTITY
                                                INSERT INTO dbo.ConferenceParticipants
                                        (ParticipantID,
                                           DayReservationID)
                                        VALUES
                                        ( @ParticipantID,
                                          @DayReservationID
                                           )
                                                set @DayParticipantID  =@@IDENTITY

                                        INSERT INTO Workers(WorkerId,CompanyID)
VALUES(@ParticipantID, @clientid)
                                end


                                set @iterator=@iterator+1

                                        set @oldsize=@size
                                        set @oldsizecopy=@oldsize
                        end
```

```
		end
	end
```

6.procedure_addReservationDayIndividual-Dodaje rezerwacje na dzień konferencji dla klienta indywidualnego

```
CREATEPROCEDURE [dbo].[procedure_addReservationDayIndividual]
	@reservationID int,
	@dayID int,
	@studentCardID int,
	@firstName varchar(10),
	@lastName varchar(10),
	@email varchar(30)
	AS
	BEGIN
		BEGIN TRY
			BEGIN TRAN ADD_ReservationDayIndividual
				DECLARE @participantID int = NULL
				SET @participantID = (SELECT top 1 P.ParticipantID
				FROM Reservations as R
				JOIN Clients as C on C.ClientId=R.ClientID
				JOIN IndividualClients as IC on C.ClientID=IC.ClientID
				JOIN Participants as P ON IC.ParticipantID= P.ParticipantID
				)

					 INSERT INTO
Participants(FirstName,LastName,Email)
					VALUES(
					@firstname,
					@lastname,
					@email
					);
					SET @participantId = @@IDENTITY

					IF(@studentCardID is not null)
					BEGIN
						DECLARE @normal int = 0
						DECLARE @student int = 1
```

```
                        DECLARE @studentCardIDSearched INT
=NULL
                        SET @studentCardIDSearched = (SELECT
StudentID FROM dbo.Students WHERE ParticipantID=@ParticipantID)
                        IF(@studentCardIDSearched is NULL)
                        BEGIN
                            INSERT  INTO Students(StudentCardId,
ReservationID,
                            ParticipantId)
                            VALUES
(@studentCardID,@reservationID,
                            @participantId);
                            DECLARE @StudentId int =
@@IDENTITY;

                        END

                    END
                    else
                    BEGIN
                        SET @normal = 1
                        SET @student = 0
                    END


                        INSERT INTO DayReservations(ReservationID,
                    dayID,
                    NormalTickets,
                    StudentTickets)
                    VALUES(@reservationID,
                    @dayID,
                    @normal,@student)
                    DECLARE @DayReservationID int = @@IDENTITY

                    INSERT INTO dbo.ConferenceParticipants
                    (
                        ParticipantID,
                        DayReservationID
                    )
                    VALUES
                    (   @ParticipantID,
                        @DayReservationID
                        )
                    DECLARE @DayParticipantID int = @@IDENTITY
```

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

7.procedure_addReservationWorkshop-dodaje rezerwacje na wrsztat dla firmy

```
CREATE PROCEDURE [dbo].[procedure_addReservationWorkshop]
	@dayReservationID int,
	@instanceID int,
	@tickets int,

	@workshopReservationID int out
	AS
	BEGIN
		SET NOCOUNT ON;
			BEGIN TRY
				BEGIN TRAN ADD_WorkshopReservation
					IF(@tickets = 0)
					BEGIN
						;THROW 52000,
						'Trzeba rezerwowac przynajmniej jedno
miejsce', 1;
					END
					IF((SELECT R.DatePaid
					FROM Reservations as R
```

```
                                        JOIN DayReservations as DR
                                        ON DR.ReservationID = R.ReservationID
                                        WHERE DR.DayReservationID = @dayReservationID)
is not null)
                                        BEGIN
                                              ;THROW 52000,
                                              'Rezerwacja została już opłacona', 1;
                                        END
                                        IF((SELECT count(DayReservationID)
                                        FROM WorkshopInstanceReservations
                                        WHERE DayReservationID = @dayReservationID
                                        and @instanceID = instanceID)> 0)
                                        BEGIN
                                              ;THROW 52000,
                                              'Klient posiada już rezerwacje na dany
warsztat', 1;
                                        END
                                        IF((SELECT DayID
                                        FROM WorkshopInstance
                                        WHERE instanceID = @instanceID) <>
                                        (SELECT DayID
                                        FROM DayReservations
                                        WHERE dayReservationID = @dayReservationID))
                                        BEGIN
                                              ;THROW 52000,
                                              'Rezerwacja i warsztat odwołują sie do innego
dnia konferencji', 1;
                                        END


IF(dbo.function_GetWorkshopInstanceFreePlaces(@instanceID)
                                        < @tickets)
                                        BEGIN
                                              ;THROW 52000,
                                              'Niestety nie ma wystarczajacej ilosci wolnych
miejsc', 1;
                                        END

                                        INSERT INTO
WorkshopInstanceReservations(InstanceID,
                                        Tickets,
                                        DayReservationID)
                                        VALUES(@instanceID,
                                        @tickets,
                                        @dayReservationID)
```

```
                    DECLARE @instanceReservationID int =
@@IDENTITY

                    DECLARE @dayParticipantId int =
                    (SELECT TOP 1 DayparticipantId FROM
dbo.ConferenceParticipants AS CP
                    JOIN dbo.DayReservations AS DR ON
DR.DayReservationID=CP.DayReservationID
                    WHERE CP.DayReservationID=@dayReservationID)


                        DECLARE @WorkshopParticipantId int =
@@IDENTITY

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

8.procedure_addTresholds - dodaje próg cenowy konferencji, który musi być mniejszy niż wcześniejsze i większy niż późniejsze

```
CREATE PROCEDURE [dbo].[procedure_addTresholds]
    @conferenceID int,
    @endingDate date,
    @discount real
        AS
        BEGIN
            SET NOCOUNT ON;
                BEGIN TRY
                    BEGIN TRAN Add_Tresholds
                        INSERT INTO
ConferenceDiscountThresholds(Conference_ID, endingDate, discount)
                        VALUES(@conferenceID,
                        @endingDate,
                        @discount)
                    COMMIT TRAN Add_Tresholds
                END TRY
                BEGIN CATCH
                ROLLBACK TRAN Add_Tresholds
                DECLARE @msg NVARCHAR(2048) =
                'Bład dodania progu cenowego do konferencji:' +
                CHAR(13) + CHAR(10) + ERROR_MESSAGE();
                THROW 52000,@msg, 1;
                END CATCH
        END
```

9.procedure_addWorkshop - dodaje dane warsztatu

```
CREATE PROCEDURE [dbo].[procedure_addWorkshop]
    @workshopName varchar(20),
    @workshopDetails varchar(255),
    @price money,
    @workshopId int
    AS
    BEGIN
        SET NOCOUNT ON;
        INSERT INTO Workshop(
        WorkshopName,
        WorkshopDetails,
        Price)
        VALUES(@workshopName,
        @workshopDetails,
        @price)
        SET @workshopId = @@IDENTITY
    END
```

10.procedure_addWorkshopInstance-dodaje instancje warsztatu

```
CREATE PROCEDURE [dbo].[procedure_addWorkshopInstance]
    @workshopID int,
    @conferenceID int,
    @date date,
    @startTime time(7),
    @endTime time(7),
    @limit int,
    @price money = 0,
    @workshopInstanceID int out
    AS
        BEGIN
            SET NOCOUNT ON;
                BEGIN TRY
                BEGIN TRAN Add_WorkshopInstance
                    IF(@date < GETDATE())
                    BEGIN
                    ;THROW 52000,'Nie można tworzyć warsztatów w
przeszłosci', 1;
                    END
                    IF((select Limit from ConferenceDay where
ConferenceID=@conferenceID and
dayid=dbo.GetConferenceDayID(@conferenceID,@date)) < @limit)
                    BEGIN
                    ;THROW 52000,'Limit miejsc nie może być
                    wieksza od liczby miejsc na konferencje', 1;
                    END
                    DECLARE @conferenceDayID int =
dbo.GetConferenceDayID(@conferenceID,@date)
                    IF(@conferenceDayID is null)
                    BEGIN
                    ;THROW 52000,'Konferencja nie odbywa sie
                    danego dnia', 1;
                    END
                    INSERT INTO WorkshopInstance
                    (DayID,
                    WorkshopID,
                    StartTime,
                    EndTime,
                    Price,
                    Limit)
                    VALUES(@conferenceDayID,
                    @workshopID,
                    @startTime,
                    @endTime,
                    @price,
                    @limit)
                    SET @workshopInstanceID = @@IDENTITY
                COMMIT TRAN Add_WorkshopInstance
            END TRY
                BEGIN CATCH
                ROLLBACK TRAN Add_WorkshopInstance
```

```
                    DECLARE @msg NVARCHAR(2048) =
                    'Bład dodania warsztatu do konferencji:' +
                    CHAR(13) + CHAR(10) + ERROR_MESSAGE();
                    THROW 52000,@msg, 1;
                END CATCH
    END
```

11.procedure_findCity-znajduje i zwraca miasto, jeśli nie znajdzie miasta i kraju, to dodaje miasto i państwo odpowiednio do słownika miast i państw

```
CREATE PROCEDURE[dbo].[procedure_findCity]
    @cityName varchar(255),
    @countryName varchar(255),
    @cityId int OUTPUT
    AS
    BEGIN
        SET NOCOUNT ON;
        BEGIN TRY
            --BEGIN TRAN FIND_CITY
                DECLARE @countryID int
                SET @cityID  = null
                IF((@cityName is not null and @countryName is null) OR
(@cityName is null and @countryName is not null))
                BEGIN
                    ;THROW 52000,
                    'Nalezy podac nazwe miasta i nazwe kraju
                    albo zadne z nich', 1;
                END

                IF(@cityName is not null and @countryName is not null)
                BEGIN
                    EXEC procedure_findCountry @countryName =
@countryName, @countryId=@countryId out
                END
                IF(@cityName is not null)
                BEGIN

                    SET @countryID = (select CountryID from CountryDict
where countryName=@countryName)
                    SET @cityID  = (Select top 1 CityID
                    From CityDict
                    Where CityName = @cityName)
                    IF(@cityID is null)
                    BEGIN

                        INSERT INTO CityDict(CityName,CountryID)
                        VALUES (@cityName,@countryID);
                        SET @cityID = @@IDENTITY;

                    END
                END
                RETURN @cityId
            --COMMIT TRAN FIND_CITY
        END TRY
        BEGIN CATCH
            --ROLLBACK TRAN FIND_CITY
            DECLARE @msg NVARCHAR(2048) =
            'Bład wyszukiwania miasta:' + CHAR(13)+CHAR(10) +
ERROR_MESSAGE();
            THROW 52000,@msg, 1;
        END CATCH
    END
```

12.procedure_findCountry-znajduje kraj, a jeśli go nie znajdzie to dodaje go do słownika państw

```
CREATE PROCEDURE [dbo].[procedure_findCountry]
@countryName varchar(255),
@countryID int OUT

AS
BEGIN
```

```
    SET NOCOUNT ON;
        BEGIN TRY
            BEGIN TRAN FIND_COUNTRY
                 SET @countryID  = (Select CountryID
                                    From CountryDict
                                    Where CountryName =
@countryName)
                IF(@countryID is null)
                BEGIN
                                INSERT INTO
CountryDict(CountryName)
                                VALUES (@countryName);
                SET @countryID = @@IDENTITY;
            END
            COMMIT TRAN FIND_COUNTRY
        END TRY
    BEGIN CATCH
        ROLLBACK TRAN FIND_COUNTRY
        DECLARE @msg NVARCHAR(2048) =
        'Bład wyszukiwania kraju:' + CHAR(13)+CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

13.procedure_insertClient-wstawia dane klienta do tablicy client

```
CREATE PROCEDURE [dbo].[procedure_insertClient]
    @phone char(9) = NULL,
    @email varchar(20),
    @street varchar(20) = NULL,
    @BiuldingNumber int,
    @cityName varchar(20) = NULL,
    @countryName varchar(20) = NULL,
    @clientId int
    AS
        SET NOCOUNT ON;
    BEGIN
            BEGIN TRY
                DECLARE @cityId int
                EXEC procedure_findCity @cityName=@cityName,
@countryName=@countryName, @cityId=@cityId out
                SET @cityID  = (select top 1 CityID from CityDict where
CityName=@cityName)
                IF(@cityID is not null)
                INSERT INTO
Clients(Phone,Email,Street,BiuldingNumber,CityID)

VALUES(@phone,@email,@street,@BiuldingNumber,@cityID);
                SET @clientID = @@IDENTITY
                RETURN @clientID
            END TRY
        BEGIN CATCH
            DECLARE @msg NVARCHAR(2048) =
            'Bład dodania klienta:' + CHAR(13)+CHAR(10) +
ERROR_MESSAGE();
            THROW 52000,@msg, 1;
        END CATCH

    END
```

14.procedure_insertParticipant-wstawia dane uczestnika do tablicy Participants

```
CREATE PROCEDURE [dbo].[procedure_insertParticipant]
    @firstname varchar(10),
    @lastname varchar(20),
    @email varchar(30),
    @participantId int

        AS
        BEGIN
            SET NOCOUNT ON;
            BEGIN TRY
                INSERT INTO Participants(FirstName,LastName,Email)
                VALUES(
                @firstname,
                @lastname,
                @email
                );

            SET @participantId = @@IDENTITY;

            END TRY
            BEGIN CATCH
                DECLARE @msg NVARCHAR(2048) =
                'Bład dodania osoby:' + CHAR(13)+CHAR(10) +
ERROR_MESSAGE();
                THROW 52000,@msg, 1;
        END CATCH
        return @participantId
    END
```

15.procedure_payReservation- umożliwia zapłatę za rezerwacje

```
CREATE PROCEDURE [dbo].[procedure_payReservation]
@reservationID int
    AS
    BEGIN
        BEGIN TRY
            BEGIN TRAN PayReservation
                IF((SELECT DatePaid
                FROM Reservations
                WHERE ReservationID = @reservationID) is not null)
                BEGIN
                ;THROW 52000,'Rezerwacja jest oplacona',1;
                END
                UPDATE Reservations
                SET DatePaid = GETDATE()
                WHERE ReservationID = @reservationID
            COMMIT TRAN PayReservation
```

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

16.procedure_removeOldReservations- Usuwa rezerwacje, których termin zapłaty już upłynął (ma więcej niż 7 dni)

```
CREATE PROCEDURE [dbo].[procedure_removeOldReservations]
AS
        BEGIN
                BEGIN TRY
                        BEGIN TRAN RemoveOldReservations
                                DELETE FROM Reservations
                                WHERE DatePaid is null and DATEDIFF(d, ReservationDate, 
GETDATE()) >= 7
                        COMMIT TRAN RemoveOldReservations
                END TRY
                BEGIN CATCH
                        ROLLBACK TRAN RemoveOldReservations
                        DECLARE @msg NVARCHAR(2048) = 'Bład usuniecia rezerwacji:'
                        + CHAR(13)+CHAR(10) + ERROR_MESSAGE();
                        THROW 52000,@msg, 1;
        END CATCH
END
```

17.procedure_addPeople-dodaje listę imion, nazwisk i email osób zarejestrowanych na konferencję

```
CREATE  procedure [dbo].[procedure_addPeople]
@list3 VARCHAR(4000)
as
```

```
begin
    declare @inTable2 table(ID int identity (1,1), Val varchar(3000));
        insert into @inTable2(Val)
        select second from Split (';',@list3)



    declare @detailTable table(ID int identity (1,1), detailVal varchar(3000));
    declare @iterator2 int = 1
    declare @card2 int
    declare @index2 int
    declare @iterator int = 1
    Declare @sumsize int =1
    Declare @size int
    declare @oldsize int=0
    declare @oldsizecopy int =0
    declare @startindex int =1
    DECLARE @oldsizecopyplus1 INT
    DECLARE @oldsizecopyplus2 INT
    DECLARE @oldsizecopyplus3 INT
    while @iterator <= (select count(id) from @inTable2)
        Begin
        Declare @oneday varchar(3000) = (select val from @inTable2 where
id=@iterator)

        insert into @detailTable(detailVal)
                select second from Split (',',@oneday)-- 1 string
                set @size  =(select count(id) from  @detailTable)





            set @oldsizecopy=@oldsizecopy +1
            Declare     @DayReservationID int = (select  detailVal from
@detailTable where id=@oldsizecopy)
            set @oldsizecopy=@oldsizecopy +1
            Declare     @dayID int = (select  detailVal from @detailTable
where id=@oldsizecopy)




                        DECLARE @participantidtable TABLE (
                        idx INT IDENTITY,
                participant_id INT
            );

                    INSERT into @participantidtable(participant_id)
SELECT p.ParticipantID FROM Participants p
            JOIN conferenceParticipants as cp ON cp.participantid=p.participantid
            JOIN dayreservations as dr ON
dr.dayreservationid=cp.dayreservationId
            WHERE cp.DayReservationID=@dayreservationID AND
dr.DayID=@dayID



                while @oldsizecopy<@size
                begin
                set @oldsizecopy =@oldsizecopy +1

                    declare @firstname VARCHAR(10) =(select detailval
from @detailTable where id=@oldsizecopy)
                    set @oldsizecopy =@oldsizecopy +1
                    Declare @lastname VARCHAR(20) =(select detailval
from @detailTable where id=@oldsizecopy)
                    set @oldsizecopy =@oldsizecopy +1
                    Declare @email VARCHAR(30) =(select detailval from
@detailTable where id=@oldsizecopy)
                    DECLARE @participantid INT = (SELECT
participant_id FROM @participantidtable WHERE idx=@startindex)


                    UPDATE Participants SET
Firstname=@firstname,LastName=@lastname, email=@email WHERE
Participantid=@participantid




                SET @startindex = @startindex + 1
                end

            set @iterator=@iterator+1

                    set @oldsize=@size
                    set @oldsizecopy=@oldsize
            end
        end
```

18.procedure_showConferenceDetails-pokazuje szczegółowe informacje o konferencji i warsztatach podczas niej się odbywających

```
CREATE PROCEDURE [dbo].[procedure_showConferenceDetails]
@ConferenceId int
        AS
        BEGIN
                BEGIN TRY
                        SELECT C.ConferenceID, Name, ConferenceDetails,
                        startDate as 'Start Conference', endDate as 'End Conference',
studentDiscount, C.Price as 'Conference price', CD.Limit as 'Conference Limit',
                        W.WorkshopID,W.workshopName,W.WorkshopDetails,W.Price as
'Workshop price',
                        WI.startTime as 'Start Workshop', WI.endtime as 'End WorkShop',
WI.Price as 'Workshop price', WI.Limit as 'Workshop limit'
                        FROM Conference AS C

                        JOIN ConferenceDay CD on C.ConferenceID=CD.ConferenceID
                        JOIN WorkshopInstance WI on WI.DayID=CD.DayID
                        LEFT OUTER JOIN Workshop W on W.WorkshopID=WI.WorkshopID
                        WHERE C.ConferenceID=@ConferenceId

                END TRY
                BEGIN CATCH

                        DECLARE @msg NVARCHAR(2048) =
                        'Brak konferencji o danym ID:' +
                        CHAR(13) + CHAR(10) + ERROR_MESSAGE();
                        THROW 52000,@msg, 1;
                END CATCH
        END
```

19.procedure_addWorkshopReservationCompany-jako argument przyjmuje listę składającą nazwisk i InstanceID warsztatów, na które dany uczestnik ma zostać zarejestrowany

```
CREATE procedure [dbo].[procedure_addWorkshopReservationCompany]
@list varchar(4000)

as
begin
```

```
    declare @inTable table(ID int identity (1,1), Val varchar(3000));
        insert into @inTable(Val)
        select second from Split ( ';',@list)



    declare @detailTable table(ID int identity (1,1), detailVal varchar(3000));
    declare @iterator2 int = 1
    declare @card2 int
    declare @index2 int
    declare @iterator int = 1
    Declare @sumsize int =1
    Declare @size int
    declare @oldsize int=0
    declare @oldsizecopy int =0
    declare @startindex int =1
    DECLARE @iterator3 INT =0
    DECLARE @instanceReservationID INT
    DECLARE @dayParticipantID INT
    while @iterator <= (select count(id) from @intable)
        Begin
        Declare @oneday varchar(3000) = (select val from @inTable where id=@iterator)

        insert into @detailTable(detailVal)
                select second from Split (',',@oneday)-- 1 string
                set @size  =(select count(id) from  @detailTable)




            set @oldsizecopy=@oldsizecopy +1
            Declare         @firstname varchar(10) = (select  detailVal from @detailTable where id=@oldsizecopy)
            set @oldsizecopy=@oldsizecopy +1
            Declare         @lastname varchar(20) = (select  detailVal from @detailTable where id=@oldsizecopy)
            set @oldsizecopy=@oldsizecopy +1
            declare         @email varchar(30) = (select  detailVal from @detailTable where id=@oldsizecopy)




            DECLARE @ParticipantId INT =
            (SELECT ParticipantID FROM Participants
                        WHERE FirstName=@firstname AND LastName=@lastname AND
Email=@email)







                                while @oldsizecopy<@size
                                begin
                                set @oldsizecopy =@oldsizecopy +1

                                        set @card2=(select detailval from @detailTable where
id=@oldsizecopy)

                                        SET @instanceReservationID = (SELECT TOP (1)
InstatnceReservationID FROM dbo.ConferenceParticipants
                                        JOIN dbo.DayReservations ON
DayReservations.DayReservationID = ConferenceParticipants.DayReservationID
                                        JOIN dbo.WorkshopInstanceReservations ON
WorkshopInstanceReservations.DayReservationID = DayReservations.DayReservationID
                                        WHERE InstanceID = @card2 AND ParticipantID =
@ParticipantId)

                                        SET @dayParticipantID = (SELECT TOP (1)
DayParticipantID FROM dbo.ConferenceParticipants
                                        JOIN dbo.DayReservations ON
DayReservations.DayReservationID = ConferenceParticipants.DayReservationID
                                        JOIN dbo.WorkshopInstanceReservations ON
WorkshopInstanceReservations.DayReservationID = DayReservations.DayReservationID
                                        WHERE InstanceID = @card2 AND ParticipantID =
@ParticipantId)
                                        INSERT INTO dbo.WorkshopParticipants
                                        (
                                           InstanceReservationID,
                                           DayParticipantID
                                        )
                                        VALUES
                                        (   @instanceReservationID, -- InstanceReservationID -
int

                                           @dayParticipantID  -- DayParticipantID - int
                                           )


                                end
```

```
                                        set @iterator=@iterator+1

                                                set @oldsize=@size
                                                set @oldsizecopy=@oldsize
                                end
                        end
```

Triggery

1. trigger_conferenceDayReservationExists - uniemożliwia zarezerwowanie tego samego dnia konferencji dwukrotnie

```
create trigger trigger_conferenceDayReservationExists
  on dbo.DayReservations
  after insert
as
begin
  set nocount on
  declare @reservationID int = (select ReservationID from inserted)
  declare @clientID int = (select ClientID from inserted as i
     inner join Reservations as R on R.ReservationID = i.ReservationID)
  if exists(
        select * from DayReservations
                   JOIN dbo.Reservations ON Reservations.ReservationID =
DayReservations.ReservationID
        where ClientID = @clientID and dbo.Reservations.ReservationID != @reservationID
     )
     begin
        ;throw 50001, 'User has already booked this Conference Day', 1
     end
end
go
```

2. trigger_participantInOverlapingWorkshopInstances - sprawdza czy uczestnik nie jest zapisany na nachodzące na siebie warsztaty

```
create trigger trigger_participantInOverlapingWorkshopInstances
  on dbo.WorkshopParticipants
  after insert
as
  begin
     set nocount on
     declare @InstanceReservationID int = (select InstanceReservationID from inserted)
     declare @DayParticipantID int = (select DayParticipantID from inserted)
     declare @InstanceID int = (
        select dbo.WorkshopInstanceReservations.InstanceID from
dbo.WorkshopInstanceReservations
           inner join dbo.WorkshopInstance on
dbo.WorkshopInstanceReservations.InstanceID = dbo.WorkshopInstance.InstanceID
           where dbo.WorkshopInstanceReservations.InstatnceReservationID =
@InstanceReservationID
     )
     declare @StartTime time = (select StartTime from dbo.WorkshopInstance where
InstanceID = @InstanceID)
     declare @EndTime time = (select EndTime from dbo.WorkshopInstance where
InstanceID = @InstanceID)
     if exists(
        select * from dbo.ConferenceParticipants
           inner join WorkshopParticipants on ConferenceParticipants.DayParticipantID =
WorkshopParticipants.DayParticipantID
           inner join WorkshopInstanceReservations on
WorkshopParticipants.InstanceReservationID =
WorkshopInstanceReservations.InstatnceReservationID
           inner join dbo.WorkshopInstance on
dbo.WorkshopInstanceReservations.InstanceID = dbo.WorkshopInstance.InstanceID
           where dbo.ConferenceParticipants.DayParticipantID = @DayParticipantID and
WorkshopInstance.InstanceID != @InstanceID
              and (
                 (WorkshopInstance.StartTime < @StartTime and
WorkshopInstance.EndTime > @StartTime )
              or ( WorkshopInstance.StartTime > @StartTime and
WorkshopInstance.StartTime < @EndTime)
              or (WorkshopInstance.StartTime > @StartTime and
WorkshopInstance.EndTime < @EndTime)
              or (WorkshopInstance.StartTime < @StartTime and WorkshopInstance.EndTime
> @EndTime)
              )
        )
        begin
            ;throw 50001, 'Workshops are overlapping', 1
        end
    end
go
```

Generator

Do wygenerowania danych użyliśmy SQL Data Generetor 4 firmy RedGate. Wygenerowaliśmy za jego pomocą dane do większości tabel jednak do niektórych konieczne było użycie skryptu. Tabele ConferenceDay, ConferenceParticipants, DayReservation, WorkshopInstanceReservation, WorkshopInstance i WorkshopParticipants wygenerowaliśmy dodatkowymi sktyptami uruchamianymi po generatorze.

Wygenerowaliśmy:

- 72 konferencje
- 840 warsztatów (300 w słowniku)
- 2000 klientów
- 3000 rezerwacji
- 300 miast
- 100 krajów

Link do strony skąd pobraliśmy generator:

(https://www.red-gate.com/products/sql-development/sql-data-generator/?gclid=Cj0KCQiAvJXxBRCeARIsAMSkApqkxV9XbSYAdBwoYXQqH0tZVdIvCZe5jCjUpD6APix7YSoxsLeNAqIaApp0EALw_wcB&gclsrc=aw.ds)