# Podstawy baz danych

## Projekt systemu zarządzania konferencjami

Tomasz Kozyra, Uladzislau Tumilovich

## Spis treści

# 1. Wprowadzanie

Przedmiotem projektu jest stworzenie systemu bazy danych dla firmy organizującej konferencje. Projektowany system ma na celu wspomaganie działania firmy i umożliwienie efektywnego zarządzania organizowanymi wydarzeniami.

# 2. Analiza wymagań

Projektowany system przechowuje dane o organizowanych konferencjach i towarzyszącym im warsztatach, oraz o uczestnikach konferencji i ich płatnościach.

## 2.1. Użytkownicy systemu i ich wymagania

Projektowanym systemem zarządza organizator. Klientami konferencji mogą być zarówno indywidualne osoby (klienci indywidualni) jak i firmy (klienci biznesowi).

### 2.1.1. Organizator

- Możliwość dodawania do systemu danych dotyczących nowych konferencji wraz z towarzyszącymi im warsztatami
    - Możliwość ustalania liczby dni trwania konferencji
    - Możliwość ustalania limitu uczestników
    - Możliwość ustalania wysokości opłat za konferencje, oraz definiowania progów cenowych
- Możliwość odwoływania konferencji
- Dostęp do list uczestników konferencji
- Dostęp do informacji o płatnościach klientów
- Dostęp do informacji o klientach, którzy najczęściej korzystają z jego usług

### 2.1.2. Klient indywidualny

- Dostęp do informacji o dostępnych terminach konferencji
- Dostęp do informacji o liczbie miejsc dostępnych na daną konferencję i towarzyszące jej warsztaty
- Dostęp do informacji o cenie udziału w konferencji, zależnej od zarezerwowanych usług oraz terminu złożenia rezerwacji
- Możliwość rejestracji na dowolnie wybrane dni w przypadku konferencji kilkudniowych
- Możliwość anulowania rezerwacji miejsc

- Dostęp do historii płatności
- Dostęp do informacji o zarezerwowanych terminach

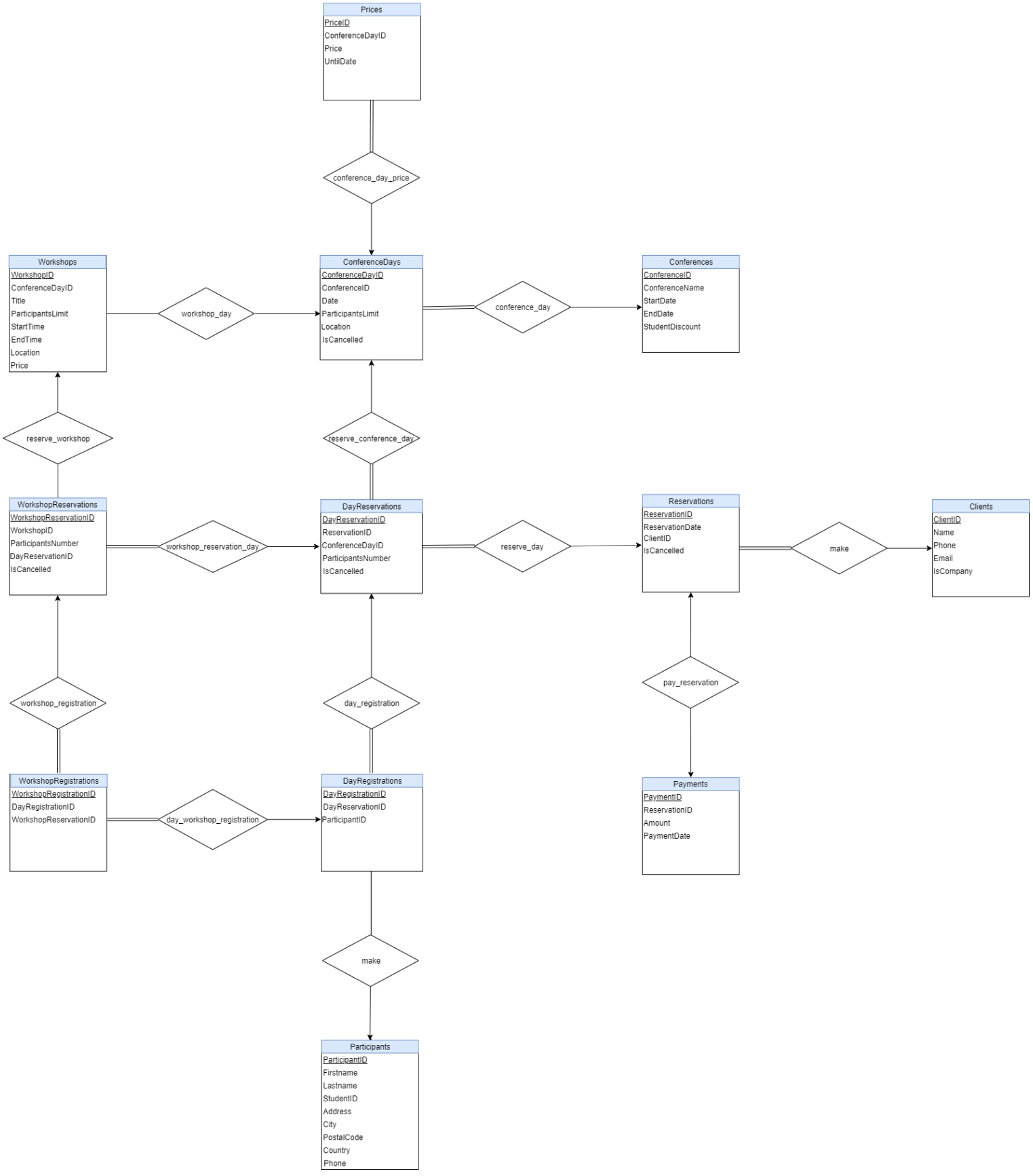
### 2.1.3. Klient biznesowy

Wymagania klienta biznesowego obejmują również wszystkie wymagania klienta indywidualnego.

Dodatkowe wymagania klienta biznesowego:

- Możliwość rezerwacji wybranej liczby miejsc na konferencje i towarzyszące im warsztaty
- Możliwość udostępnienia listy zgłoszonych uczestników z opóźnieniem (w terminie późniejszym niż termin rezerwacji liczby miejsc)

# 3. Diagram ER


Prices
PriceID
ConferenceDayID
Price
UntilDate
conference_day_price
Workshops
WorkshopID
ConferenceDayID
Title
ParticipantsLimit
StartTime
EndTime
Location
Price
workshop_day
ConferenceDays
ConferenceDayID
ConferenceID
Date
ParticipantsLimit
Location
IsCancelled
conference_day
Conferences
ConferenceID
ConferenceName
StartDate
EndDate
StudentDiscount
reserve_workshop
reserve_conference_day
WorkshopReservations
WorkshopReservationID
WorkshopID
ParticipantsNumber
DayReservationID
IsCancelled
workshop_reservation_day
DayReservations
DayReservationID
ReservationID
ConferenceDayID
ParticipantsNumber
IsCancelled
reserve_day
Reservations
ReservationID
ReservationDate
ClientID
IsCancelled
make
Clients
ClientID
Name
Phone
Email
IsCompany
workshop_registration
day_registration
pay_reservation
WorkshopRegistrations
WorkshopRegistrationID
DayRegistrationID
WorkshopReservationID
day_workshop_registration
DayRegistrations
DayRegistrationID
DayReservationID
ParticipantID
Payments
PaymentID
ReservationID
Amount
PaymentDate
make
Participants
ParticipantID
Firstname
Lastname
StudentID
Address
City
PostalCode
Country
Phone

# 4. Schemat bazy danych

# 5. Tabele

## 5.1. Tabela Clients

Tabela Clients zawiera informacje o klientach firmy organizującej konferencje. Pole IsCompany może przyjmować wartość true lub false, zależnie od tego czy klient jest firmą czy klientem indywidualnym.

```
-- Table: Clients
CREATE TABLE Clients (
    ClientID int identity(1,1) NOT NULL,
    Name varchar(40) NOT NULL,
    Phone varchar(20) NOT NULL,
    Email varchar(40) NOT NULL,
    IsCompany bit NOT NULL,
    CONSTRAINT Clients_pk PRIMARY KEY (ClientID)
);
```

## 5.2. Tabela Conferences

Jeden rekord tabeli Conferences odpowiada jednej konferencji. Tabela zawiera podstawowe informacje o danej konferencji. Pole StudentDiscount oznacza procentową zniżkę studencką przypisaną do danej konferencji.

```
-- Table: Conferences
CREATE TABLE Conferences (
    ConferenceID int identity(1,1) NOT NULL,
    ConferenceName nvarchar(40) NOT NULL,
    StartDate datetime NOT NULL,
    EndDate datetime NOT NULL,
    StudentDiscount real NOT NULL,
    CONSTRAINT Conferences_pk PRIMARY KEY (ConferenceID)
);

ALTER TABLE Conferences
WITH CHECK ADD CONSTRAINT StudentDiscountBetweenZeroAndOne
    CHECK (StudentDiscount >= 0 AND StudentDiscount <= 1)
```

## 5.3. Tabela ConferenceDays

Jeden rekord tabeli ConferenceDays odpowiada jednemu dniu konferencji. Tabela zawiera podstawowe informacje o dniu danej konferencji. Pole ParticipantsLimit oznacza limit uczestników danego dnia konferencji. Pole isCancelled zawiera dane typu ‘true/false’ i informuje o tym czy dany dzień konferencji jest odwołany.

```
-- Table: ConferenceDays
CREATE TABLE ConferenceDays (
    ConferenceDayID int identity(1,1) NOT NULL,
    ConferenceID int NOT NULL,
    Date datetime NOT NULL,
    ParticipantsLimit int NOT NULL,
    Location nvarchar(60) NOT NULL,
    IsCancelled bit NOT NULL DEFAULT 0,
    CONSTRAINT ConferenceDays_pk PRIMARY KEY (ConferenceDayID)
);

ALTER TABLE ConferenceDays
WITH CHECK ADD CONSTRAINT ParticipantsLimitGreaterThanZero
    CHECK (ParticipantsLimit > 0)

ALTER TABLE ConferenceDays ADD CONSTRAINT ConferenceDay_Conference_fk
    FOREIGN KEY (ConferenceID)
    REFERENCES Conferences (ConferenceID);
```

## 5.4. Tabela Reservations

Jeden rekord tabeli Reservations odpowiada pojedynczej rezerwacji miejsc. Pole ReservationDate oznacza datę złożenia rezerwacji. Pole isCancelled zawiera dane typu ‘true/false’ i informuje o tym czy dana rezerwacja została anulowana.

```
-- Table: Reservations
CREATE TABLE Reservations (
    ReservationID int identity(1,1)  NOT NULL,
    ReservationDate datetime NOT NULL,
    ClientID int NOT NULL,
    IsCancelled bit NOT NULL DEFAULT 0,
    CONSTRAINT Reservations_pk PRIMARY KEY (ReservationID)
);

ALTER TABLE Reservations ADD CONSTRAINT Reservation_Client_fk
    FOREIGN KEY (ClientID)
    REFERENCES Clients (ClientID);
```

## 5.5. Tabela DayReservations

Jeden rekord tabeli DayReservations odpowiada pojedynczej rezerwacji miejsc na podany dzień. Tabela zawiera podstawowe informacje o dokonywanej rezerwacji. Pole ConferenceDayID oznacza identyfikator dnia konferencji na który dokonywana jest rezerwacja. Pole isCancelled zawiera dane typu ‘true/false’ i informuje o tym czy dana rezerwacja została anulowana. StudentsNumber oznacza liczbę zarejestrowanych studentów, a ParticipantsNumber sumaryczną liczbę wszystkich zarejestrowanych uczestników.

```
-- Table: DayReservations
CREATE TABLE DayReservations (
    DayReservationID int identity(1,1) NOT NULL,
    ReservationID int NOT NULL,
    ConferenceDayID int NOT NULL,
    StudentsNumber int NOT NULL,
    ParticipantsNumber int NOT NULL,
    IsCancelled bit NOT NULL DEFAULT 0,
    CONSTRAINT DayReservations_pk PRIMARY KEY (DayReservationID)
);

ALTER TABLE DayReservations
WITH CHECK ADD CONSTRAINT StudentsNumberGreaterEqualsZero
    CHECK (StudentsNumber >= 0)

ALTER TABLE DayReservations
WITH CHECK ADD CONSTRAINT ParticipantsNumberGreaterThanZero
    CHECK (ParticipantsNumber > 0)

ALTER TABLE DayReservations ADD CONSTRAINT DayReservation_Reservation_fk
    FOREIGN KEY (ReservationID)
    REFERENCES Reservations (ReservationID);

ALTER TABLE DayReservations ADD CONSTRAINT DayReservation_ConferenceDay_fk
    FOREIGN KEY (ConferenceDayID)
    REFERENCES ConferenceDays (ConferenceDayID);
```

## 5.6. Tabela Workshops

Tabela Workshops zawiera listę wszystkich warsztatów. Jeden rekord tabeli Workshops odpowiada jednemu warsztatowi. Pole Title oznacza nazwe warsztatu. Pole ConferenceDayID oznacza identyfikator dnia konferencji w którym odbywa się dany warsztat. Pole ParticipantsLimit oznacza limit uczestników danego warsztatu. Pole Price oznacza cenę warsztatu, ponieważ część warsztatów może być płatna, a część darmowa.

```sql
-- Table: Workshops
CREATE TABLE Workshops (
    WorkshopID int identity(1,1) NOT NULL,
    ConferenceDayID int NOT NULL,
    Title nvarchar(40) NOT NULL,
    ParticipantsLimit int NOT NULL,
    StartTime time NOT NULL,
    EndTime time NOT NULL,
    Location nvarchar(60) NOT NULL,
    Price money NOT NULL,
    CONSTRAINT Workshops_pk PRIMARY KEY (WorkshopID)
);

ALTER TABLE Workshops WITH CHECK ADD CONSTRAINT Workshop_WorkshopConferenceDay_fk
FOREIGN KEY(ConferenceDayID) REFERENCES ConferenceDays (ConferenceDayID)

ALTER TABLE Workshops WITH CHECK ADD CONSTRAINT
    WorkshopsPriceGreaterEqualsZero
    CHECK ((Price >= (0)));

ALTER TABLE Workshops WITH CHECK ADD CONSTRAINT
    WorkshopsParticipantsLimitGreaterEqualsZero
    CHECK ((ParticipantsLimit >= (0)));

ALTER TABLE Workshops WITH CHECK ADD CONSTRAINT
    WorkshopsStartTimeLesserEqualsEndTime
    CHECK ((StartTime <= EndTime));
```

## 5.7. Tabela WorkshopReservations

Jeden rekord tabeli WorkshopReservations odpowiada pojedynczej rezerwacji miejsc na podany warsztat. Tabela zawiera podstawowe informacje o rezerwacji warsztatu. Pole WorkshopID oznacza identyfikator warsztatu, na który dokonywana jest rezerwacja. Pole DayReservationID informuje o tym, z którą rezerwacją dnia konferencji powiązana jest rezerwacja warsztatu. Pole IsCancelled zawiera dane typu ‘true/false’ i pokazują czy rezerwacja na dany warsztat została anulowana.

```sql
-- Table: WorkshopReservations
CREATE TABLE WorkshopReservations (
    WorkshopReservationID int identity(1,1) NOT NULL,
    WorkshopID int NOT NULL,
    DayReservationID int NOT NULL,
    ParticipantsNumber int NOT NULL,
    IsCancelled bit NOT NULL DEFAULT 0,
    CONSTRAINT WorkshopReservations_pk PRIMARY KEY (WorkshopReservationID)
);

ALTER TABLE WorkshopReservations ADD CONSTRAINT
    WorkshopReservation_DayReservation_fk FOREIGN KEY(DayReservationID)
    REFERENCES DayReservations (DayReservationID)

ALTER TABLE WorkshopReservations ADD CONSTRAINT
    WorkshopReservation_Workshop_fk FOREIGN KEY(WorkshopID)
    REFERENCES Workshops (WorkshopID)

ALTER TABLE WorkshopReservations WITH CHECK ADD CONSTRAINT
    WorkshopReservationsParticipantsNumberGreaterThanZero
    CHECK ((ParticipantsNumber > (0)));
```

## 5.8. Tabela DayRegistrations

Tabela DayRegistrations zawiera dane o rejestracji uczestników na dni konferencji. Jeden rekord tabeli DayRegistrations odpowiada rejestracji jednego uczestnika na jeden dzień konferencji. Pole DayReservationID informuje o tym której rezerwacji dnia konferencji dotyczy rejestracja.

```sql
-- Table: DayRegistrations
CREATE TABLE DayRegistrations (
    DayRegistrationID int identity(1,1) NOT NULL,
    DayReservationID int NOT NULL,
    ParticipantID int NOT NULL,
    IsStudent bit NOT NULL,
    CONSTRAINT DayRegistrations_pk PRIMARY KEY (DayRegistrationID)
);

ALTER TABLE DayRegistrations ADD CONSTRAINT
    DayRegistration_Participant_fk FOREIGN KEY (ParticipantID)
    REFERENCES Participants (ParticipantID);

ALTER TABLE DayRegistrations ADD CONSTRAINT
    DayRegistration_DayReservation_fk FOREIGN KEY (DayReservationID)
    REFERENCES DayReservations (DayReservationID);
```

## 5.9. Tabela WorkshopRegistrations

Tabela WorkshopRegistrations zawiera dane o rejestracji uczestników na warsztaty. Pole DayRegistrationID informuje o identyfikatorze rezerwacji na dzień konferencji, ponieważ uczestnik musi być zarejstrowany w danym dniu na konferencje, aby móc brać udział w warsztatach. Pole WorkshopReservationID informuje o tym której rezerwacji warsztatu dotyczy rejestracja.

```
-- Table: WorkshopRegistrations
CREATE TABLE WorkshopRegistrations (
    WorkshopRegistrationID int identity(1,1) NOT NULL,
    DayRegistrationID int NOT NULL,
    WorkshopReservationID int NOT NULL,
    CONSTRAINT WorkshopRegistrations_pk PRIMARY KEY (WorkshopRegistrationID)
);

ALTER TABLE WorkshopRegistrations ADD CONSTRAINT
    WorkshopRegistration_DayRegistration_fk FOREIGN KEY(DayRegistrationID)
    REFERENCES DayRegistrations (DayRegistrationID)

ALTER TABLE WorkshopRegistrations ADD CONSTRAINT
    WorkshopRegistration_WorkshopReservation_fk FOREIGN KEY(WorkshopReservationID)
    REFERENCES WorkshopReservations (WorkshopReservationID)
```

## 5.10. Tabela Participants

Tabela Participants zawiera informacje o uczestnikach konferencji. Pole StudentID oznacza numer legitymacji studenckiej uczestnika, na którego podstawie udzielana jest zniżka studencka. Pole StudentID przyjmuje wartość null kiedy uczestnik nie jest studentem.

```
-- Table: Participants
CREATE TABLE Participants (
    ParticipantID int identity(1,1) NOT NULL,
    Firstname nvarchar(40) NOT NULL,
    Lastname nvarchar(40) NOT NULL,
    Address nvarchar(40) NOT NULL,
    City nvarchar(20) NOT NULL,
    PostalCode nvarchar(10) NOT NULL,
    Country nvarchar(20) NOT NULL,
    Phone nvarchar(20) NOT NULL,
    CONSTRAINT Participants_pk PRIMARY KEY (ParticipantID)
);
```

## 5.11. Tabela Payments

Tabela Payments zawiera dane o płatnościach za rezerwacje. Pole ReservationID informuje o tym której rezerwacji dotyczy opłata.

```sql
-- Table: Payments
CREATE TABLE Payments (
    PaymentID int identity(1,1) NOT NULL,
    ReservationID int NOT NULL,
    Amount money NOT NULL,
    PaymentDate datetime NOT NULL,
    CONSTRAINT Payments_pk PRIMARY KEY (PaymentID)
);

ALTER TABLE Payments ADD CONSTRAINT
    Payment_Reservation_fk FOREIGN KEY(ReservationID)
    REFERENCES Reservations (ReservationID)

ALTER TABLE Payments WITH CHECK ADD CONSTRAINT
    PaymentAmountNotZero
    CHECK ((Amount <> (0)));
```

## 5.12. Tabela Prices

Tabela Prices zawiera informacje o cenach dni konferencji. Pole ConferenceDayID informuje o tym którego dnia konferencji dotyczy cena. Pole UntilDate oznacza datę, do której dana cena obowiązuje - związane jest to z tym, że dany dzień konferencji może mieć różne progi ceny zależnie od tego, kiedy zostanie dokonana rezerwacja.

```sql
-- Table: Prices
CREATE TABLE Prices (
    PriceID int identity(1,1) NOT NULL,
    ConferenceDayID int NOT NULL,
    Price money NOT NULL,
    UntilDate datetime NOT NULL,
    CONSTRAINT Prices_pk PRIMARY KEY (PriceID)
);

ALTER TABLE Prices ADD CONSTRAINT
    Price_ConferenceDay_fk FOREIGN KEY(ConferenceDayID)
    REFERENCES ConferenceDays (ConferenceDayID)

ALTER TABLE Prices WITH CHECK ADD CONSTRAINT
    PricesPriceGreaterEqualsZero
    CHECK ((Price >= (0)));
```

# 6. Widoki

```
-- widok paymentsBalance
-- wyświetla klientów i ich bilans wpłat
CREATE VIEW VIEW_paymentsBalance AS
    SELECT C.ClientID, C.Name, C.Phone, C.Email,C.IsCompany,
           ISNULL(ROUND(SUM(Total)
            - SUM(dbo.FUNCTION_conferenceReservationPrice(R.ReservationID))
               ,2), 0)
            AS Balance
    FROM Reservations AS R
    JOIN (
        SELECT ReservationID, sum(Amount) AS Total
        from Payments
        GROUP BY ReservationID
    ) AS P
    ON P.ReservationID = R.ReservationID
    RIGHT JOIN Clients AS C
    ON C.ClientID = R.ClientID
    GROUP BY C.ClientID, C.Name, C.Phone, C.Email, C.IsCompany
```

---

```
-- widok clientsWithDebt
-- wyświetla klientów którzy zalegają z płatnościami za rezerwacje
-- korzysta z widoku paymentsBalance
CREATE VIEW VIEW_clientsWithDebt AS
    SELECT *
    FROM VIEW_paymentsBalance
    WHERE Balance < 0
```

---

```
-- widok clientsWithOverpayment
-- wyświetla klientów którzy zapłacili zbyt dużo za rezerwacje
-- korzysta z widoku paymentsBalance
CREATE VIEW VIEW_clientsWithOverpayment AS
    SELECT *
    FROM VIEW_paymentsBalance
    WHERE Balance > 0
```

---

```sql
-- widok participantsStats
-- wyświetla statystyki uczestników konferencji i warsztatów
-- (liczbę dni i warsztatów w których brali udział)
CREATE VIEW VIEW_participantsStats AS
    SELECT P.ParticipantID,
           COUNT(DR.DayRegistrationID) AS 'NumberOfRegistartions',
           'DAYS' AS ActivityType
    FROM Participants AS P
    LEFT OUTER JOIN DayRegistrations AS DR
        ON DR.ParticipantID = P.ParticipantID
    LEFT OUTER JOIN DayReservations AS DR2
        ON DR.DayReservationID = DR2.DayReservationID
               AND DR2.isCancelled = '0'
    GROUP BY P.ParticipantID

    UNION

    SELECT P.ParticipantID,
           COUNT(WR.WorkshopRegistrationID) AS 'NumberOfRegistrations',
           'WORKSHOPS' AS ActivityType
    FROM Participants AS P
    LEFT OUTER JOIN DayRegistrations AS DR
        ON DR.ParticipantID = P.ParticipantID
    LEFT OUTER JOIN WorkshopRegistrations AS WR
        ON DR.DayRegistrationID = WR.DayRegistrationID
    LEFT OUTER JOIN WorkshopReservations WR2
        ON WR.WorkshopReservationID = WR2.DayReservationID
               AND WR2.IsCancelled = '0'
    LEFT OUTER JOIN DayReservations AS DR2
        ON DR.DayReservationID = DR2.DayReservationID
               AND DR2.isCancelled = '0'
    GROUP BY P.ParticipantID

-----------------------------------------------------------------------------------------------------------

-- Widok ClientsStats
-- wyświetla ilość rezerwacji
-- (które nie zostały anulowane)
-- dokonanych przez każdego klienta
CREATE VIEW VIEW_ClientsStats AS
    SELECT  c.ClientID, COUNT(ReservationID) as NumberOfReservations
    FROM Clients AS c
    JOIN Reservations AS r ON c.ClientID = r.ClientID AND r.isCancelled = 0
    GROUP BY c.ClientID

-----------------------------------------------------------------------------------------------------------

-- conferenceReservationsNotPaid
-- wyświetla rezerwacje na konferencje, które nie zostały opłacone
-- w terminie 7 dni od rezerwacji i powinny zostać anulowane
CREATE VIEW VIEW_conferenceReservationsNotPaid AS
    SELECT ReservationID, ClientID,
           dbo.FUNCTION_conferenceReservationBalance(ReservationID)
               AS debt FROM Reservations
    WHERE  DATEDIFF(day, ReservationDate, GETDATE()) > 7
    GROUP BY ReservationID, ClientID
    HAVING dbo.FUNCTION_conferenceReservationBalance(ReservationID) < 0
```

```sql
-- Widok clientsWithFreePlacesOnReservations
-- wyświetla klientów, którzy nie podali
-- jeszcze wszystkich uczestników warsztatów lub dni konferencji,
-- a konferencja zaczyna się za co najwyżej 14 dni,
-- wraz z informacją o pozostałej liczbie miesc dla danej rezerwacji
CREATE VIEW VIEW_clientsWithFreePlacesOnReservations AS

    SELECT  c.ClientID,  c.name,  c.email,  c.phone, r.ReservationID,
        'DAY' AS ReservationType, dr.DayReservationID,
        dbo.FUNCTION_freePlacesForConferenceDayReservation(dr.DayReservationID) AS
freePlaces
    FROM Reservations AS r
    JOIN DayReservations AS dr ON dr.ReservationID = r.ReservationID AND dr.isCan-
celled = '0'
    JOIN ConferenceDays AS cd ON dr.ConferenceDayID = cd.ConferenceDayID
    JOIN Conferences AS conf ON conf.ConferenceID = cd.ConferenceID
    JOIN Clients AS c ON c.ClientID = r.ClientID
    WHERE dbo.FUNCTION_freePlacesForConferenceDayReservation(dr.DayReservationID)
> 0
      AND DATEDIFF(day, GETDATE(), conf.startDate) BETWEEN 0 AND 14
      AND r.isCancelled = '0'

    UNION

    SELECT  c.ClientID,  c.name,  c.email,  c.phone, r.ReservationID,
        'WORKSHOP' AS ReservationType, wr.WorkshopReservationID,
        dbo.FUNCTION_freePlacesForWorkshopReservation(wr.WorkshopReservationID) AS
freePlaces
    FROM Reservations AS r
    JOIN DayReservations AS dr ON dr.ReservationID = r.ReservationID AND dr.IsCan-
celled = '0'
    JOIN ConferenceDays as cd ON cd.ConferenceDayID = dr.ConferenceDayID
    JOIN Conferences AS conf ON conf.ConferenceID = cd.conferenceID
    JOIN WorkshopReservations AS wr ON wr.DayReservationID = dr.DayReservationID
AND wr.isCancelled = '0'
    JOIN Clients AS c ON c.ClientID = r.ClientID
    WHERE dbo.FUNCTION_freePlacesForWorkshopReservation(wr.WorkshopReservationID)
> 0
      AND DATEDIFF(day, GETDATE(), conf.StartDate) BETWEEN 0 AND 14
    AND r.isCancelled = '0'

-----------------------------------------------------------------------------------------------------

-- widok conferenceDaysPlaces
-- wyświetla dni konferencji wraz z informacją
-- o ilości wolnych i zajętych miejsc
CREATE VIEW VIEW_conferenceDayPlaces AS
    SELECT ConferenceDayID,
           dbo.FUNCTION_freePlacesForConferenceDay(ConferenceDayID) as FreePlaces,
           (ParticipantsLimit - dbo.FUNCTION_freePlacesForConferenceDay(Conferen-
ceDayID)) as ReservedPlaces
    from ConferenceDays

-----------------------------------------------------------------------------------------------------
```

```sql
-- widok conferenceDaysWithFreePlaces
-- wyświetla dni konferencji na które
-- są jeszcze miejsca i podaje ich liczbę
CREATE VIEW VIEW_conferenceDaysWithFreePlaces AS
    SELECT * from VIEW_conferenceDayPlaces
    where FreePlaces > 0

---------------------------------------------------------------------------------------------------------

-- widok workshopsPlaces
-- wyświetla warsztaty wraz z informacją o ilości wolnych i zajętych miejsc
CREATE VIEW VIEW_workshopsPlaces AS
    SELECT WorkshopID,
           dbo.FUNCTION_freePlacesForWorkshop(WorkshopID) as FreePlaces,
           (ParticipantsLimit - dbo.FUNCTION_freePlacesForWorkshop(WorkshopID)) as
ReservedPlaces
    from Workshops

---------------------------------------------------------------------------------------------------------

-- widok workshopsWithFreePlaces
-- wyświetla dni konferencji na które są jeszcze miejsca
-- i podaje ich liczbę
CREATE VIEW VIEW_workshopsWithFreePlaces AS
    SELECT * from VIEW_workshopsPlaces
    where FreePlaces > 0

---------------------------------------------------------------------------------------------------------
```

# 7. Funkcje

```
-- Funkcja freePlacesForConferenceDay
-- zwraca ilość wolnych miejsc na dany dzień konferencji.
CREATE FUNCTION FUNCTION_freePlacesForConferenceDay
    (@ConferenceDayID int) RETURNS int
AS BEGIN

    DECLARE @totalPlaces int;
    SET @totalPlaces = (SELECT ParticipantsLimit
                        FROM ConferenceDays
                        WHERE ConferenceDayID = @ConferenceDayID)

    DECLARE @takenPlaces int;
    SET @takenPlaces = (SELECT SUM(ParticipantsNumber)
                        FROM DayReservations
                        WHERE ConferenceDayID = @ConferenceDayID
                        AND IsCancelled = 0)

    RETURN (@totalPlaces - @takenPlaces);

END

-----------------------------------------------------------------------------------------------------------

-- Funkcja freePlacesForConferenceDayReservation
-- zwraca ilość wolnych miejsc dla uczestników na daną
-- rezerwację dnia konferencji
CREATE FUNCTION FUNCTION_freePlacesForConferenceDayReservation
    (@DayReservationID int) RETURNS int
AS BEGIN

    DECLARE @totalPlaces int;
    SET @totalPlaces = (SELECT ParticipantsNumber
                        FROM DayReservations
                        WHERE DayReservationID = @DayReservationID)

    DECLARE @takenPlaces int;
    SET @takenPlaces = (SELECT COUNT(*)
                        FROM DayRegistrations
                        WHERE DayReservationID = @DayReservationID)

    RETURN (@totalPlaces - @takenPlaces);
END
```

```sql
-- Funkcja freePlacesForWorkshop
-- zwraca ilość wolnych miejsc na dany warsztat
CREATE FUNCTION FUNCTION_freePlacesForWorkshop
    (@WorkshopID int) RETURNS int
AS BEGIN

    DECLARE @totalPlaces int;
    SET @totalPlaces = (SELECT ParticipantsLimit
                        FROM Workshops
                        WHERE WorkshopID = @WorkshopID)

    DECLARE @takenPlaces int;
    SET @takenPlaces = (SELECT SUM(ParticipantsNumber)
                            FROM WorkshopReservations
                            WHERE WorkshopID = @WorkshopID
                            AND IsCancelled = 0)

    RETURN (@totalPlaces - @takenPlaces);
END

-----------------------------------------------------------------------------------------------------

-- Funkcja freePlacesForWorkshopReservation
-- zwraca ilość wolnych miejsc dla uczestników na daną rezerwację warsztatu
CREATE FUNCTION FUNCTION_freePlacesForWorkshopReservation
    (@WorkshopReservationID int) RETURNS int
AS BEGIN

    DECLARE @totalPlaces int;
    SET @totalPlaces = (SELECT ParticipantsNumber
                        FROM WorkshopReservations
                        WHERE WorkshopReservationID = @WorkshopReservationID)

    DECLARE @takenPlaces int;
    SET @takenPlaces = (SELECT COUNT(*)
                        FROM WorkshopRegistrations
                        WHERE WorkshopReservationID = @WorkshopReservationID)

    RETURN (@totalPlaces - @takenPlaces);
END

-----------------------------------------------------------------------------------------------------
```

```sql
-- Funkcja conferenceDayReservationPrice
-- zwraca koszt danej rezerwacji dnia konferencji
CREATE FUNCTION FUNCTION_conferenceDayReservationPrice
    (@DayReservationID int) RETURNS money
AS BEGIN

    DECLARE @reservationDate datetime;
    SET @reservationDate = (SELECT ReservationDate
                            FROM Reservations as r
                            JOIN DayReservations d
                                on r.ReservationID = d.ReservationID
                            WHERE d.DayReservationID = @DayReservationID)

    DECLARE @studentDiscount real;
    SET @studentDiscount = (SELECT StudentDiscount
                            FROM Conferences as c
                            JOIN ConferenceDays as cd
                                on c.ConferenceID = cd.ConferenceID
                            JOIN DayReservations dr
                                on dr.ConferenceDayID = cd.ConferenceDayID
                            WHERE dr.DayReservationID = @DayReservationID)

    RETURN(
        SELECT SUM(p.Price*StudentsNumber*@studentDiscount)
                    + SUM(p.Price*(ParticipantsNumber-StudentsNumber))
        FROM DayReservations as dr
        JOIN Prices as p ON p.ConferenceDayID = dr.ConferenceDayID
                        AND p.untilDate = (
                            SELECT MIN(p2.untilDate) as soonestDate
                            FROM Prices as p2
                            WHERE p2.untilDate >= @reservationDate
                              AND p2.ConferenceDayID = dr.ConferenceDayID
                        )
        WHERE dr.DayReservationID = @DayReservationID
        )
END

-----------------------------------------------------------------------------------------------------

-- Funkcja reservedConferenceDaysPrice
-- zwraca koszt rezerwacji wszystkich dni
-- konferencji dla danej rezerwacji
CREATE FUNCTION FUNCTION_reservedConferenceDaysPrice
    (@ReservationID int) RETURNS money
AS BEGIN
    RETURN(
        SELECT SUM(dbo.FUNCTION_conferenceDayReservationPrice(DayReservationID))
        FROM DayReservations as dr
        WHERE dr.ReservationID = @ReservationID
    )
END

-----------------------------------------------------------------------------------------------------
```

```sql
-- Funkcja workshopReservationPrice
-- zwraca koszt danej rezerwacji warsztatu
CREATE FUNCTION FUNCTION_workshopReservationPrice
    (@WorkshopReservationID int) RETURNS money
AS BEGIN
    RETURN(
        SELECT (w.Price*wr.ParticipantsNumber)
        FROM WorkshopReservations as wr
        JOIN Workshops w on wr.WorkshopID = w.WorkshopID
        WHERE wr.WorkshopReservationID = @WorkshopReservationID
    )
END

-----------------------------------------------------------------------------------------------------

-- Funkcja reservedWorkshopsPrice
-- zwraca koszt wszystkich rezerwacji
-- warsztatów dla danej rezerwacji
CREATE FUNCTION FUNCTION_reservedWorkshopsPrice
    (@ReservationID int) RETURNS money
AS BEGIN
    RETURN(
        SELECT SUM(dbo.FUNCTION_workshopReservationPrice(WorkshopReservationID))
        FROM WorkshopReservations as wr
        JOIN DayReservations dr on dr.DayReservationID = wr.DayReservationID
        JOIN Reservations as r on dr.ReservationID = r.ReservationID
        WHERE r.ReservationID = @ReservationID
    )
END

-----------------------------------------------------------------------------------------------------

-- Funkcja conferenceReservationPrice
-- zwraca sumaryczny koszt danej rezerwacji
-- (Czyli sumę kosztów rezerwacji wszystkich dni konferencji i warsztatów)
CREATE FUNCTION FUNCTION_conferenceReservationPrice
    (@ReservationID int) RETURNS money
AS BEGIN
    RETURN(dbo.FUNCTION_reservedConferenceDaysPrice(@ReservationID)
            + dbo.FUNCTION_reservedWorkshopsPrice(@ReservationID));
END

-----------------------------------------------------------------------------------------------------

-- Funkcja conferenceReservationBalance
-- zwraca bilans wpłat za daną rezerwację
-- (Czyli sumę wpłat minus koszt rezerwacji).
CREATE FUNCTION FUNCTION_conferenceReservationBalance
        (@ReservationID int) RETURNS money
AS BEGIN
    RETURN(
        ISNULL(ROUND((SELECT SUM(Amount)
                      FROM Payments
                      WHERE ReservationID = @ReservationID)
                - dbo.FUNCTION_conferenceReservationPrice(@ReservationID), 2), 0)
    );
END
```

```sql
-- Funkcja conferenceDayParticipantList
-- zwraca liste wszystkich użytkowników
-- zarejstrowanych na podany dzień konferencji
CREATE FUNCTION FUNCTION_conferenceDayParticipantsList
   (@ConferenceDayID int) RETURNS @DayParticipantsListTable TABLE
   (
      ParticipantID int,
      Firstname varchar(50),
      Lastname varchar(50)
   )
AS BEGIN

   INSERT @DayParticipantsListTable
      SELECT DISTINCT P.ParticipantID, P.Firstname, P.Lastname
      FROM DayReservations AS DRES
      JOIN DayRegistrations AS DREG
          ON DREG.DayReservationID = DRES.DayReservationID
      JOIN Participants AS P
          ON P.ParticipantID = DREG.ParticipantID
      WHERE DRES.ConferenceDayID = @ConferenceDayID AND DRES.IsCancelled = 0
      ORDER BY P.ParticipantID

   RETURN
END
```

---

```sql
-- Funkcja workshopParticipantList
-- zwraca liste wszystkich użytkowników zarejstrowanych na podany warsztat
CREATE FUNCTION FUNCTION_workshopParticipantsList
   (@WorkshopID int) RETURNS @WorkshopParticipantListTable TABLE
   (
      ParticipantID int,
      Firstname varchar(50),
      Lastname varchar(50)
   )
AS BEGIN

   INSERT @WorkshopParticipantListTable
      SELECT DISTINCT P.ParticipantID, P.Firstname, P.Lastname
      FROM WorkshopReservations AS WRES
      JOIN WorkshopRegistrations AS WREG
          ON WREG.WorkshopReservationID = WRES.WorkshopReservationID
      JOIN DayRegistrations AS DREG
          ON DREG.DayRegistrationID = WREG.DayRegistrationID
      JOIN Participants AS P ON P.ParticipantID = DREG.ParticipantID
      WHERE WRES.WorkshopID = @WorkshopID AND WRES.isCancelled = 0
      ORDER BY P.ParticipantID

   RETURN
END
```

```sql
-- Funkcja participantConferenceDayList
-- zwraca liste wszystkich dni konferencji
-- na które jest zapisany podany użytkownik
CREATE FUNCTION FUNCTION_participantConferenceDayList
   (@ParticipantID int) RETURNS @ParticipantConferenceDayListTable TABLE
   (ConferenceDayID int)
AS BEGIN

   INSERT @ParticipantConferenceDayListTable
      SELECT DISTINCT DRES.ConferenceDayID
      FROM Participants AS P
      JOIN DayRegistrations AS DREG ON
          DREG.ParticipantID = P.ParticipantID
      JOIN DayReservations AS DRES ON
          DRES.DayReservationID = DREG.DayRegistrationID
      WHERE P.ParticipantID = @ParticipantID
      ORDER BY DRES.ConferenceDayID

   RETURN
END
```

---

```sql
-- Funkcja participantWorkshopList
-- zwraca liste wszystkich warsztatów na które jest zapisany podany użytkownik
CREATE FUNCTION FUNCTION_participantWorkshopList
   (@ParticipantID int) RETURNS @ParticipantWorkshopListTable TABLE
   (WorkshopID int)
AS BEGIN

   INSERT @ParticipantWorkshopListTable
      SELECT DISTINCT WRES.WorkshopID
      FROM Participants AS P
      JOIN DayRegistrations AS DREG ON DREG.ParticipantID = P.ParticipantID
      JOIN WorkshopRegistrations AS WREG
          ON WREG.DayRegistrationID = DREG.DayRegistrationID
      JOIN WorkshopReservations AS WRES
          ON WRES.WorkshopReservationID = WREG.WorkshopReservationID
      WHERE P.ParticipantID = @ParticipantID
      ORDER BY WRES.WorkshopID

   RETURN
END
```

```sql
-- Funkcja workshopCollision
-- jeżeli dwa podane warsztaty przecinają się
-- to funkcja zwróci wartość logiczną true
-- Ta funkcja jest potrzebna przy zapisywaniu uczestnika na warsztat,
-- żeby uniemożliwić rejestrację na warsztaty odbywające się w tym samym czasie
CREATE FUNCTION FUNCTION_workshopCollision
   (@WorkshopID1 int, @WorkshopID2 int) RETURNS bit
AS BEGIN

   DECLARE @StartTime1 TIME(7);
   DECLARE @EndTime1 TIME(7);
   DECLARE @Date1 DATE;

   DECLARE @StartTime2 TIME(7);
   DECLARE @EndTime2 TIME(7);
   DECLARE @Date2 DATE;

   SET @StartTime1 = (SELECT StartTime
                      FROM Workshops
                      WHERE WorkshopID = @WorkshopID1)
   SET @EndTime1 = (SELECT EndTime
                      FROM Workshops
                      WHERE WorkshopID = @WorkshopID1)
   SET @Date1 = (SELECT D.Date
                FROM ConferenceDays AS D
                JOIN Workshops AS W
                    ON W.ConferenceDayID = D.ConferenceDayID
                WHERE W.WorkshopID = @WorkshopID1)

   SET @StartTime2 = (SELECT StartTime
                      FROM Workshops
                      WHERE WorkshopID = @WorkshopID2)
   SET @EndTime2 = (SELECT EndTime
                    FROM Workshops
                    WHERE WorkshopID = @WorkshopID2)
   SET @Date2 = (SELECT D.Date
                FROM ConferenceDays AS D
                JOIN Workshops AS W
                    ON W.ConferenceDayID = D.ConferenceDayID
                WHERE W.WorkshopID = @WorkshopID2)

   DECLARE @Collision BIT;
   IF(@StartTime1 > @EndTime2 OR @StartTime2 > @EndTime1 OR (@Date1 <> @Date2))
      SET @Collision = 0
   ELSE
      SET @Collision = 1

      RETURN @Collision

END
```

# 8. Procedury

```
-- Procedura addConference
-- dodawanie konferencji
CREATE PROCEDURE PROCEDURE_addConference
    @ConferenceName nvarchar(40), @StartDate datetime,
    @EndDate datetime, @StudentDiscount real AS
BEGIN
    SET NOCOUNT ON;

    IF(@StartDate > @EndDate) BEGIN
        THROW 51000, 'EndDate should not be earlier than StartDate.', 1
    END

    IF(@StudentDiscount < 0 OR @StudentDiscount > 1) BEGIN
        THROW 51000, 'The discount must be between 0 and  1.', 1
    END

    INSERT INTO Conferences(ConferenceName, StartDate, EndDate, studentDiscount)
    VALUES(@ConferenceName, @StartDate, @EndDate, @StudentDiscount)

END
```

---

```
-- procedura addWorkshopRegistration
-- dodawanie rejestracji na warsztat
CREATE PROCEDURE PROCEDURE_addWorkshopRegistration
@dayRegistrationID int, @workshopReservationID int
AS
BEGIN
SET NOCOUNT ON;
    DECLARE @dayRegistration int = (
        SELECT DayRegistrationID
        FROM DayRegistrations
        WHERE DayRegistrationID = @dayRegistrationID
    )
    IF (@dayRegistration IS NULL) BEGIN
        THROW 51000, 'There is no such DayRegistration.', 1
    END

    DECLARE @workshopReservation int = (
        SELECT WorkshopReservationID
        FROM WorkshopReservations
        WHERE WorkshopReservationID = @workshopReservationID
    )
    IF (@workshopReservation IS NULL) BEGIN
        THROW 51000, 'There is no such WorkshopReservation.', 1
    END

    INSERT INTO WorkshopRegistrations(DayRegistrationID, WorkshopReservationID)
    VALUES (@dayRegistrationID, @workshopReservationID)
END
```

---

```sql
-- Procedura addConferenceDay
-- dodawanie dnia konferencji
CREATE PROCEDURE PROCEDURE_addConferenceDay
    @ConferenceID int, @Date datetime,
    @ParticipantsLimit int, @Location nvarchar(60) AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Conference int = (
        SELECT ConferenceID FROM Conferences
        WHERE ConferenceID = @ConferenceID
    )
    IF(@Conference IS NULL) BEGIN
        THROW 51000, 'Conference does not exist.', 1
    END

    DECLARE @SecondDay int = (
        SELECT ConferenceID FROM ConferenceDays
        WHERE ConferenceID = @ConferenceID AND Date = @Date
    )
    IF(@SecondDay IS NOT NULL) BEGIN
        THROW 51000, 'This day of conference already exists.', 1
    END

    IF((@Date < (SELECT StartDate
                 FROM Conferences
                 WHERE ConferenceID = @ConferenceID)) or
       (@Date > (SELECT EndDate
                 FROM Conferences
                 WHERE ConferenceID = @ConferenceID))) BEGIN
        THROW 51000, 'Invalid ConferenceDay date, selected conference starts ear-
lier or ends later.', 1
    END

    INSERT INTO ConferenceDays(ConferenceID, Date,
            ParticipantsLimit, Location, IsCancelled)
    VALUES(@ConferenceID, @Date, @ParticipantsLimit, @Location, 0)

END

-----------------------------------------------------------------------------------------------------

-- Procedura addParticipant
-- dodawanie uczestnika konferencji
CREATE PROCEDURE PROCEDURE_addParticipant
    @Firstname nvarchar(40), @Lastname varchar(40), @Address nvarchar(40),
    @City varchar(20), @PostalCode nvarchar(10), @Country varchar(20), @Phone
nvarchar(20)
AS BEGIN
SET NOCOUNT ON;

    INSERT INTO Participants(Firstname, Lastname, Address,
            City, PostalCode, Country, Phone)
    VALUES(@Firstname, @Lastname, @Address,
            @City,@PostalCode, @Country, @Phone)

END

-----------------------------------------------------------------------------------------------------
```

```sql
-- Procedura addClient
-- dodawanie klienta
CREATE PROCEDURE PROCEDURE_addClient
    @Name varchar(40), @Phone varchar(20),
    @Email varchar(40), @IsCompany bit AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO Clients(Name, Phone, Email, IsCompany)
    VALUES(@Name, @Phone, @Email, @IsCompany)

END
```

---

```sql
-- Procedura addPayment
-- dodawanie płatności
CREATE PROCEDURE PROCEDURE_addPayment
    @reservationID int, @amount money, @paymentDate datetime
AS
BEGIN
SET NOCOUNT ON;
    IF (@amount = 0) BEGIN
        THROW 51000, 'The amount cannot be equal to 0.00.', 1
    END

    DECLARE @reservations int = (
        SELECT ReservationID
        FROM Reservations
        WHERE ReservationID = @reservationID
    )
    IF(@reservations IS NULL) BEGIN
        THROW 51000, 'There is no such a Reservation.', 1
    END

    INSERT INTO Payments(ReservationID, Amount, PaymentDate)
    VALUES (@reservationID, @amount, @paymentDate)
END
```

```sql
-- Procedura addDayRegistration
-- dodawanie rejestracji na dzień konferencji
CREATE PROCEDURE PROCEDURE_addDayRegistration
    @DayReservationID int, @ParticipantID int, @IsStudent bit AS
BEGIN
SET NOCOUNT ON;

    DECLARE @Participant int = (
        SELECT participantID FROM Participants
        WHERE participantID = @ParticipantID
    )
    IF(@Participant IS NULL) BEGIN
        THROW 51000, 'Selected participant does not exist.', 1
    END

    DECLARE @DayReservation int = (
        SELECT @DayReservationID FROM DayReservations
        WHERE DayReservationID = @DayReservationID
    )
    IF(@DayReservation IS NULL) BEGIN
        THROW 51000, 'There is no such reservation.', 1
    END

    DECLARE @IsCancelled int = (
        SELECT isCancelled FROM DayReservations
        WHERE DayReservationID = @DayReservationID
    )
    IF(@IsCancelled = 1) BEGIN
        THROW 51000, 'This conference reservation was cancelled.', 1
    END

    DECLARE @ReservedPlacesNumber int = (
        SELECT ParticipantsNumber FROM DayReservations
        WHERE @DayReservationID = @DayReservationID
    )
    DECLARE @RegisteredPlacesNumber int = (
        SELECT  COUNT(*)
        FROM DayRegistrations
        WHERE DayReservationID = @DayReservationID
    )
    IF(@ReservedPlacesNumber = @RegisteredPlacesNumber ) BEGIN
        THROW  51000, 'You can not register more participants. All reserved places
are already taken.', 1
    END

    DECLARE @ReservedPlacesForStudentsNumber int = (
        SELECT StudentsNumber FROM DayReservations
        WHERE @DayReservationID = @DayReservationID
    )
    DECLARE @RegisteredPlacesForStudentsNumber int = (
        SELECT  COUNT(*)
        FROM DayRegistrations
        WHERE DayReservationID = @DayReservationID and IsStudent = 1
    )
    IF(@ReservedPlacesForStudentsNumber = @RegisteredPlacesForStudentsNumber)
    BEGIN
        THROW  51000, 'You can not register more student participants. All places
reserved for students are already taken.', 1
    END
```

```
    DECLARE @SecondParticipant int = (
        SELECT @ParticipantID FROM DayRegistrations
        WHERE ParticipantID = @ParticipantID AND DayReservationID =
@DayReservationID
        GROUP BY participantID
    )
    IF(@SecondParticipant IS NOT NULL) BEGIN
        THROW  51000, 'This participant has been already registered.', 1
    END

    INSERT INTO DayRegistrations(DayReservationID, ParticipantID, isStudent)
    VALUES(@DayReservationID, @ParticipantID, @IsStudent)

END
```

---

```
-- Procedura addWorkshopReservation
-- dodawanie rezerwacji warsztatu
CREATE PROCEDURE PROCEDURE_addWorkshopReservation
    @workshopID int, @dayReservationID int, @participantsNumber int
AS
BEGIN
SET NOCOUNT ON;

    DECLARE @workshop int = (
        SELECT WorkshopID
        FROM Workshops
        WHERE WorkshopID = @workshopID
    )
    IF (@workshop IS NULL) BEGIN
        THROW 51000, 'There is no such Workshop.', 1
    END

    DECLARE @dayReservation int = (
        SELECT DayReservationID
        FROM DayReservations
        WHERE DayReservationID = @dayReservationID
    )
    IF (@dayReservation IS NULL) BEGIN
        THROW 51000, 'There is no such DayReservation.',1
    END

    IF (@participantsNumber <= 0) BEGIN
        THROW 51000, 'The ParticipantsNumber cannot be lesser than 1.', 1
    END

    INSERT INTO WorkshopReservations(WorkshopID, DayReservationID,
                                     ParticipantsNumber, IsCancelled)
    VALUES (@workshopID, @dayReservationID,@participantsNumber, 0)
END
```

---

```sql
-- procedura addReservation
-- dodawanie rezerwacji
CREATE PROCEDURE PROCEDURE_addReservation
    @reservationDate datetime, @clientID int
AS
BEGIN
SET NOCOUNT ON;
    DECLARE @clients int = (
        SELECT ClientID
        FROM Clients
        WHERE ClientID = @clientID
    )
    IF (@clients IS NULL) BEGIN
        THROW 51000, 'There is no such a Client.',1
    END

    INSERT INTO Reservations(ReservationDate, ClientID, IsCancelled)
    VALUES (@reservationDate, @clientID, 0)
END
```

---

```sql
-- Procedura addPrice
-- dodawanie ceny
CREATE PROCEDURE PROCEDURE_addPrice
    @conferenceDayID int, @price money, @untilDate datetime
AS
BEGIN
    SET NOCOUNT ON;

    IF (@price < 0) BEGIN
        THROW 51000, 'The price cannot be lesser than 0.', 1
    END

    DECLARE @conferenceDays int = (
        SELECT ConferenceDayID
        FROM ConferenceDays
        WHERE ConferenceDayID = @conferenceDayID
    )
    IF (@conferenceDays IS NULL) BEGIN
        THROW 51000, 'There is no such a ConferenceDay.', 1
    END

    DECLARE @beginDate datetime = (
            SELECT DATEADD(day, CD.Date - 1, C.StartDate)
            FROM ConferenceDays AS CD
            JOIN Conferences C ON C.ConferenceID = CD.ConferenceID
    )
    IF (@untilDate >= @beginDate) BEGIN
        THROW 51000, 'The UntilDate is too late.', 1
    END
    IF (@untilDate < GETDATE()) BEGIN
        THROW 51000, 'The UntilDate is too early.', 1
    END

    DECLARE @anotherPrice int = (
        SELECT ConferenceDayID
        FROM Prices
        WHERE ConferenceDayID = @conferenceDayID AND UntilDate = @untilDate
    )
    IF (@anotherPrice IS NOT NULL) BEGIN
       THROW 51000, 'The price already exists.', 1
    END

    INSERT INTO Prices(ConferenceDayID, Price, UntilDate)
    VALUES (@conferenceDayID, @price, @untilDate)
END
```

---

```sql
-- Procedura addDayReservation
-- dodawanie rezerwacji na dzień konferencji
CREATE PROCEDURE PROCEDURE_addDayReservation
    @ReservationID int, @ConferenceDayID int, @StudentsNumber int,
    @ParticipantsNumber int AS
BEGIN
SET NOCOUNT ON;

    IF(@ParticipantsNumber <= 0) BEGIN
        THROW 51000, 'The number of participants must be greater than 0.', 1
    END
    IF(@StudentsNumber < 0) BEGIN
        THROW 51000, 'The number of students cannot be less than 0.', 1
    END

    DECLARE @ConferenceReservation int = (
        SELECT ReservationID FROM Reservations
        WHERE ReservationID = @ReservationID
    )
    IF(@ConferenceReservation IS NULL) BEGIN
        THROW 51000, 'You cannot reserve conference day, if you did not make rese-
rvation for conference.', 1
    END

    DECLARE @ConferenceDayIsCancelled int = (
        SELECT isCancelled FROM ConferenceDays
        WHERE ConferenceDayID = @ConferenceDayID
    )
    IF(@ConferenceDayIsCancelled = 1) BEGIN
        THROW 51000, 'This conference day was cancelled.', 1
    END

    DECLARE @ConferenceReservationIsCancelled int = (
        SELECT isCancelled FROM Reservations
        WHERE ReservationID = @ReservationID
    )
    IF(@ConferenceReservationIsCancelled = 1) BEGIN
        THROW 51000, 'Reservation for this conference was cancelled.', 1
    END

    DECLARE @ConferenceDay int = (
        SELECT @ConferenceDayID
        FROM ConferenceDays
        WHERE ConferenceDayID = @ConferenceDayID
    )
    IF(@ConferenceDay IS NULL) BEGIN
        THROW 51000, 'There is no such conference day.', 1
    END

    DECLARE @FreePlacesNumber int = (
        SELECT dbo.FUNCTION_freePlacesForConferenceDay(@ConferenceDayID)
    )
    IF(@FreePlacesNumber = 0) BEGIN
        THROW 51000, 'All places for this day have been already taken.', 1
    END
```

```
    IF(@FreePlacesNumber < @ParticipantsNumber) BEGIN
        DECLARE @errorAlert NVARCHAR(2048)
        SET @errorAlert = 'Not enough places. This day has only '
                    + CAST(@FreePlacesNumber AS varchar)
                    + ' free places.';
        THROW 51000, @errorAlert, 1
    END

    INSERT INTO DayReservations(ReservationID, ConferenceDayID,
                                StudentsNumber, ParticipantsNumber, IsCancelled)
    VALUES (@ReservationID, @ConferenceDayID,
                                @StudentsNumber, @ParticipantsNumber, 0)
END
```

---

```
-- procedura addWorkshop
-- dodawanie warsztatu
CREATE PROCEDURE PROCEDURE_addWorkshop
    @conferenceDayID int, @title nvarchar(40), @participantsLimit int,
    @startTime time, @endTime time, @location nvarchar(60), @price money
AS
BEGIN
SET NOCOUNT ON;

    DECLARE @conferenceDay int = (
        SELECT ConferenceDayID
        FROM ConferenceDays
        WHERE ConferenceDayID = @conferenceDayID
    )
    IF (@conferenceDay IS NULL) BEGIN
        THROW 51000, 'There is no such ConferenceDay.',1
    END

    IF (@participantsLimit < 0) BEGIN
        THROW 51000, 'The ParticipantsLimit cannot be lesser than 0.',1
    END

    IF (@startTime >= @endTime) BEGIN
        THROW 51000, 'The StartTime cannot be after the EndTime.',1
    END

    IF (@price < 0) BEGIN
        THROW 51000, 'The price cannot be lesser than 0.',1
    END

    INSERT INTO Workshops(ConferenceDayID, Title, ParticipantsLimit, StartTime,
                          EndTime, Location, Price)
    VALUES (@conferenceDayID, @title, @participantsLimit, @startTime,
                            @endTime, @location, @price)
END
```

```sql
-- procedura changeParticipantsLimitForConferenceDay
-- służy do zmiany limitu uczestników dnia konferencji
-- Nie pozwala zmniejszyć limitu uczestników poniżej liczby miejsc
-- która została już zarezerwowana
CREATE PROCEDURE PROCEDURE_changeParticipantsLimitForConferenceDay
    @ConferenceDayID int, @NewParticipantsLimit int
AS
BEGIN
SET NOCOUNT ON;

    IF(@NewParticipantsLimit <= 0) BEGIN
        THROW 51000, 'ParticipantsLimit must be greater than zero.', 1
    END

    DECLARE @OldParticipantsLimit int = (
       SELECT ParticipantsLimit
       FROM ConferenceDays
       WHERE ConferenceDayID = @ConferenceDayID
    )
    IF(@OldParticipantsLimit IS NULL) BEGIN
        THROW 51000, 'There is no ConferenceDay with inserted ID', 1
    END

    DECLARE @FreePlacesForConferenceDay int =
        dbo.FUNCTION_freePlacesForConferenceDay(@ConferenceDayID);
    DECLARE @OccupiedPlacesForConferenceDay int =
        @OldParticipantsLimit - @FreePlacesForConferenceDay;

    IF(@NewParticipantsLimit < @OccupiedPlacesForConferenceDay) BEGIN
        DECLARE @errorAlert NVARCHAR(2048)
        SET @errorAlert = 'You cannot add ParticipantsLimit equal'
                        + @NewParticipantsLimit
                        + ', because '
                        + @OccupiedPlacesForConferenceDay
                        + ' places have been already reserved.';
        THROW 51000, @errorAlert, 1
    END
END
```

```sql
-- Procedura changeParticipantsLimitForWorkshop
-- służy do zmiany limitu uczestników na warsztat
-- Nie pozwala zmniejszyć limitu uczestników poniżej liczby miejsc
-- która została już zarezerwowana
CREATE PROCEDURE PROCEDURE_changeParticipantsLimitForWorkshop
    @WorkshopID int, @NewParticipantsLimit int
AS
BEGIN
SET NOCOUNT ON;

    IF(@NewParticipantsLimit <= 0) BEGIN
        THROW 51000, 'ParticipantsLimit must be greater than zero.', 1
    END

    DECLARE @OldParticipantsLimit int = (
       SELECT ParticipantsLimit
       FROM Workshops
       WHERE WorkshopID = @WorkshopID
    )
    IF(@OldParticipantsLimit IS NULL) BEGIN
        THROW 51000, 'There is no Workshop with inserted ID', 1
    END

    DECLARE @FreePlacesForWorkshop int =
        dbo.FUNCTION_freePlacesForWorkshop(@WorkshopID);

    DECLARE @OccupiedPlacesForWorkshop int =
        @OldParticipantsLimit - @FreePlacesForWorkshop;

    IF(@NewParticipantsLimit < @OccupiedPlacesForWorkshop) BEGIN
        DECLARE @errorAlert NVARCHAR(2048)
        SET @errorAlert = 'You cannot add ParticipantsLimit equal'
                        + @NewParticipantsLimit
                        + ', because '
                        + @OccupiedPlacesForWorkshop
                        + ' places have been already reserved.';
        THROW 51000, @errorAlert ,1
    END
END

-----------------------------------------------------------------------------------------------------------

-- procedura cancelReservation
-- anulowanie rezerwacji
CREATE PROCEDURE cancelReservation
@reservationID int
AS
BEGIN
SET NOCOUNT ON;
    UPDATE Reservations
    SET IsCancelled = '1'
    WHERE ReservationID = @reservationID
END

-----------------------------------------------------------------------------------------------------------
```

```sql
-- procedura mostActiveClients
-- zwraca klientów uporządkowanych malejąco ze względu na
-- ilość dokonanych rezerwacji
CREATE PROCEDURE mostActiveClients
AS
BEGIN
    SELECT C.ClientID, C.Name, CS.Number
    FROM ClientsStats AS CS
    JOIN Clients AS C
        ON C.ClientID = CS.ClientID
    ORDER BY CS.Number DESC
END

-----------------------------------------------------------------------------------------------------

-- procedura mostActiveDayParticipants
-- zwraca uczestników uporządkowanych malejąco ze względu na
-- liczbe dni konferencji w których brali udział
CREATE PROCEDURE PROCEDURE_mostActiveDayParticipants
AS
BEGIN
    SELECT P.ParticipantID, P.Firstname, P.Lastname, PS.Number
    FROM ParticipantsStats AS PS
    JOIN Participants AS P
        ON P.ParticipantID = PS.ParticipantID
    WHERE ActivityType = 'DAYS'
    ORDER BY Number DESC
END

-----------------------------------------------------------------------------------------------------

-- procedura mostActiveWorkshopParticipants
-- zwraca uczestników uporządkowanych malejąco ze względu na
-- liczbe warsztatów w których brali udział
CREATE PROCEDURE PROCEDURE_mostActiveWorkshopParticipants
AS
BEGIN
    SELECT P.ParticipantID, P.Firstname, P.Lastname, PS.Number
    FROM ParticipantsStats AS PS
    JOIN Participants AS P
        ON P.ParticipantID = PS.ParticipantID
    WHERE ActivityType = 'WORKSHOPS'
    ORDER BY Number DESC
END
```

# 9. Triggery

```
-- trigger conferenceDayParticipantsLimitExceeded
-- sprawdzenie czy ilość miejsc na dany dzień konferencji
-- nie jest mniejsza od rezerwowanej ilości miejsc
CREATE TRIGGER TRIGGER_conferenceDayParticipantsLimitExceeded ON DayReservations
AFTER INSERT, UPDATE
AS BEGIN
    SET NOCOUNT ON

    IF((SELECT COUNT(*) from INSERTED) > 1)
            OR ((SELECT COUNT(*) from DELETED) > 1) BEGIN
        RAISERROR('You cannot insert or update more than one DayReservation at on-
ce', 1, 1)
        ROLLBACK TRANSACTION
    END

    DECLARE @placesWantedToReserve int;
    SET @placesWantedToReserve = (SELECT ParticipantsNumber FROM INSERTED);

    DECLARE @freePlaces int;
    SET @freePlaces = dbo.FUNCTION_freePlacesForConferenceDay(
                        (SELECT ConferenceDayID FROM INSERTED)
                      );

    IF(@freePlaces < @placesWantedToReserve) BEGIN
        RAISERROR('You tried to reserve %d places, but only %d are available',
            1, 1, @placesWantedToReserve, @freePlaces)
        ROLLBACK TRANSACTION
    END

END

-----------------------------------------------------------------------------------------------------

-- Trigger cancelConferenceReservation
-- przy anulowaniu rezerwacji na konferencję automatycznie
-- anuluje wszystkie przypisane do niej rezerwacje na dni konferencji.
CREATE TRIGGER TRIGGER_cancelConferenceReservation ON Reservations AFTER UPDATE
AS BEGIN
    SET NOCOUNT ON;
    IF((SELECT COUNT(*) from INSERTED) > 1)
            OR ((SELECT COUNT(*) from DELETED) > 1) BEGIN
        RAISERROR('You cannot update more than one Reservation at once', 1, 1)
        ROLLBACK TRANSACTION
    END

    UPDATE DayReservations
    SET isCancelled = 1
    WHERE ReservationID IN(
        SELECT I.ReservationID FROM INSERTED as I, DELETED as D
        WHERE I.isCancelled = 1 AND D.isCancelled = 0
    )
END

-----------------------------------------------------------------------------------------------------
```

```
-- Trigger workshopParticipantsLimitExceeded
-- sprawdza, czy ilość uczestników podanych w rezerwacji na
-- warsztat nie przekracza ilości miejsc dostępnych na dany warsztat,
-- a także czy nie przekracza ilości miejsc zadeklarowanej w DayReservations.
CREATE TRIGGER TRIGGER_workshopParticipantsLimitExceeded ON WorkshopReservations
AFTER INSERT, UPDATE
AS BEGIN

    IF((SELECT COUNT(*) from INSERTED) > 1)
            OR ((SELECT COUNT(*) from DELETED) > 1) BEGIN
        RAISERROR('You cannot insert or update more than one WorkshopReservation
at once', 1, 1)
        ROLLBACK TRANSACTION
    END

    DECLARE @workshopPlacesWantedToReserve int;
    DECLARE @freePlacesForWorkshop int;

    SET @workshopPlacesWantedToReserve = (SELECT ParticipantsNumber
                                          FROM INSERTED);
    SET @freePlacesForWorkshop = dbo.FUNCTION_freePlacesForWorkshop(
                                    (SELECT WorkshopID FROM INSERTED)
                                )
                                    + @workshopPlacesWantedToReserve;

    IF(@freePlacesForWorkshop < @workshopPlacesWantedToReserve) BEGIN
        RAISERROR('You tried to reserve %d places for workshop, but only %d are
available',
            1, 1, @workshopPlacesWantedToReserve, @freePlacesForWorkshop)
        ROLLBACK TRANSACTION
    END

    DECLARE @placesReservedForDay int;
    SET @placesReservedForDay = (SELECT participantsNumber
                                    FROM DayReservations
                                    WHERE DayReservations.DayReservationID =
                                          (SELECT DayReservationID
                                           FROM INSERTED));

    IF(@placesReservedForDay < @workshopPlacesWantedToReserve) BEGIN
        RAISERROR('You tried to reserve %d places for workshop, but only %d places
are reserved for this conference day!',
            1, 1, @workshopPlacesWantedToReserve, @placesReservedForDay)
        ROLLBACK TRANSACTION
    END
END

-----------------------------------------------------------------------------------------------
```

```sql
-- trigger cancelDayReservation
-- przy anulowaniu rezerwacji na dany dzien
-- anuluje rezerwacje na warsztaty
-- powiazane z tą rezerwacją dnia konferencji
CREATE TRIGGER TRIGGER_cancelDayReservation On DayReservations
AFTER UPDATE
AS
BEGIN
    IF((SELECT COUNT(*) from INSERTED) > 1)
            OR ((SELECT COUNT(*) from DELETED) > 1) BEGIN
        RAISERROR('You cannot update more than one DayReservation at once', 1, 1)
        ROLLBACK TRANSACTION
    END

    UPDATE WorkshopReservations SET IsCancelled = 1
    WHERE DayReservationID IN (
        SELECT I.DayReservationID
        FROM INSERTED AS I, DELETED AS D
        WHERE I.isCancelled = 1 AND D.isCancelled = 0
    )
END
```

---

```sql
-- trigger createWorkshop
-- sprawdza czy przy tworzeniu warsztatu podano limit uczestników
-- mniejszy lub rowny limitowi uczestników na dany dzien
CREATE TRIGGER TRIGGER_createWorkshop ON Workshops
AFTER INSERT
AS
BEGIN
    DECLARE @workshopLimit int;
    DECLARE @dayLimit int;
    SET @workshopLimit = (SELECT ParticipantsLimit FROM INSERTED);
    SET @dayLimit = (SELECT ParticipantsLimit
                     FROM ConferenceDays
                     WHERE ConferenceDayID = (SELECT ConferenceDayID
                                              FROM INSERTED));

    IF(@dayLimit < @workshopLimit)
        BEGIN
            RAISERROR('You tried to create a workshop with participants limit of
%d places, but %d is the limit of places for this day!', 1, 1,
                      @workshopLimit, @dayLimit)
            ROLLBACK TRANSACTION
        END
END
```

---

```sql
-- trigger registerForConferenceDay
-- uruchamia się kiedy próbujemy zarejestrować osobę na rezerwację dnia
-- Konferencji w której wykorzystano już wszystkie miejsca
CREATE TRIGGER TRIGGER_registerForConferenceDay On DayRegistrations
AFTER INSERT
AS
BEGIN
    IF(dbo.FUNCTION_freePlacesForConferenceDayReservation((SELECT DayReservationID
                                                           FROM INSERTED)) = 0)
        BEGIN
            RAISERROR('All places for this reservation are already taken', 1, 1)
            ROLLBACK TRANSACTION
        END
END

-- trigger registerForWorkshop
-- uruchamia się kiedy próbujemy zarejestrować osobę na rezerwację warsztatu
-- w której wykorzystano już wszystkie miejsca
CREATE TRIGGER TRIGGER_registerForWorkshop On WorkshopRegistrations
AFTER INSERT
AS
BEGIN
    IF(dbo.FUNCTION_freePlacesForWorkshopReservation((SELECT WorkshopReservationID
                                                      FROM INSERTED)) = 0)
        BEGIN
            RAISERROR('All places for this reservation are already taken', 1, 1)
            ROLLBACK TRANSACTION
        END
END

-- Trigger reservationWorkshopReservationsColliding
-- sprawdza, czy warsztat na który użytkownik rezerwuję miejsce
-- nie przecina się z innymi warsztatami na które ten użytkownik
-- jest już zarejestrowany
CREATE TRIGGER TRIGGER_workshopReservationsColliding
   ON WorkshopRegistrations
   AFTER INSERT
AS
BEGIN
   SET NOCOUNT ON;
   IF EXISTS
   (
      SELECT * FROM INSERTED AS wr
      JOIN DayRegistrations AS dr ON dr.DayRegistrationID = wr.DayRegistrationID
      CROSS APPLY dbo.FUNCTION_participantWorkshopList(dr.ParticipantID) AS w1
      JOIN WorkshopReservations wres
          on wr.WorkshopReservationID = wres.WorkshopReservationID
      JOIN Workshops AS w ON w.WorkshopID = wres.WorkshopID
      WHERE dbo.FUNCTION_workshopCollision(w1.WorkshopID, w.WorkshopID) = 1
         AND w1.WorkshopID <> w.WorkshopID
   )
   BEGIN
      THROW 51000, 'You cannot register participant for this workshop,
                    because selected participant is already registered
                    for workshop on this time', 1
      ROLLBACK TRANSACTION;
   END
END
```

# 10. Indeksy

```
-- Indeks Payments służący do usprawnienia obliczania bilansu wpłat klientów
CREATE NONCLUSTERED INDEX INDEX_Payments ON Payments (
    ReservationID ASC
)
WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    DROP_EXISTING = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON
)


-- Indeks DayRegistration służący do usprawnienia pozyskiwania
-- list osób przypisanych do rezerwacji na dni konferencji
CREATE NONCLUSTERED INDEX INDEX_DayRegistration ON DayRegistrations (
    DayReservationID ASC
)
WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    DROP_EXISTING = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON
)


-- Indeks WorkshopRegistration służący do usprawnienia pozyskiwania
-- list osób przypisanych do rezerwacji na warsztaty
CREATE NONCLUSTERED INDEX INDEX_WorkshopRegistration ON WorkshopRegistrations (
    WorkshopReservationID ASC
)
WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    DROP_EXISTING = OFF,
    ONLINE  =  OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON
)
```

# 11. Role użytkowników

Proponujemy utworzenie następujących ról dla użytkowników systemu: klient, administrator, pracownik firmy

## a. Administrator

Posiada dostęp do wszystkich funkcji, procedur i widoków w systemie

## b. Pracownik firmy

Posiada dostęp do następujących funkcji, procedur i widoków:

- Widoku klientów i bilansu ich wpłat (widok paymentsBalance)
- Widoku statystyk uczestników konferencji i warsztatów (widok participantsStats)
- Widoku statystyk klientów konferencji i warsztatów (widok clientsStats)
- Widoku rezerwacji które nie zostały opłacone w terminie do 7 dni od ich złożenia i powinny zostać anulowane (conferenceReservationsNotPaid)
- Widoku klientów którzy nie podali jeszcze wszystkich uczestników dla swoich rezerwacji (clientsWithFreePlacesOnReservations)
- Widoku dni konferencji z informacją o ilości wolnych i zajętych miejsc na dany dzień konferencji (conferenceDaysPlaces)
- Widoku warsztatów z informacją o ilości wolnych i zajętych miejsc na dany warsztat (workshopsPlaces)
- Funkcji zwracającej listę uczestników zarejestrowanych na dany dzień konferencji (conferenceDayParticipantsList)
- Funkcji zwracającej listę uczestników zarejestrowanych na dany warsztat (workshopParticipantsList)
- Procedury dodającej konferencje, dni konferencji, warsztaty (addConference, addConferenceDay, addWorkshop)
- Procedury służącej do wprowadzania płatności (addPayment)
- Procedury służącej do dodawania nowych cen (addPrice)
- Procedury służącej do edytowania limitów uczestników na dni konferencji i warsztaty (changeParticipantsLimitForConferenceDay, changeParticipantsLimitForWorkshop)
- Procedury służącej do wyświetlania najbardziej aktywnych klientów (mostActiveClients)
- Procedury służącej do wyświetlania najbardziej aktywnych uczestników dni konferencji i warsztatów (mostActiveDayParticipants, mostActiveWorkshopParticipants)

### c. Klient

Posiada dostęp do następujących funkcji, procedur i widoków:

- Widoku dni konferencji z informacją o ilości wolnych i zajętych miejsc na dany dzień konferencji (conferenceDaysPlaces)
- Widoku warsztatów z informacją o ilości wolnych i zajętych miejsc na dany warsztat (workshopsPlaces)
- Funkcji zwracającej koszt danej rezerwacji dnia konferencji (conferenceDayReservationPrice)
- Funkcji zwracającej koszt danej rezerwacji warsztatu (workshopReservationPrice)
- Funkcji zwracającej sumaryczny koszt danej rezerwacji (conferenceReservationPrice)
- Funkcji zwracającej listę dni konferencji na które zarejestrowany jest dany uczestnik konferencji (participantConferenceDayList)
- Funkcji zwracającej listę warsztatów na które zarejestrowany jest dany uczestnik konferencji (participantWorkshopList)
- Procedury dodającej rezerwację na konferencję, dzień konferencji i warsztat (addReservation, addDayReservation, addWorkshopReservation)
- Procedury dodającej rejestracje na warsztat lub dzień konferencji (addWorkshopRegistration, addDayRegistration)
- Procedury dodającej rezerwację na konferencję, dzień konferencji i warsztat (addReservation, addDayReservation, addWorkshopReservation)
- Procedury służącej do dodawania uczestników konferencji (addParticipant)
- Procedury służącej do rejestracji (addClient)
- Procedury służącej do wprowadzania płatności (addPayment)
- Procedury służącej do anulowania rezerwacji (cancelReservation)

# 12. Generator

Do wygenerowania danych w projekcie wykorzystaliśmy SQL Data Generator firmy RedGate. Program jest prosty w użyciu i za jego pomocą mieliśmy możliwość wygenerowania bardzo dobrych danych stałych. W niektórych sytuacji było jednak ciężko lub nawet niemożliwie wygenerować poprawne dane, więc musieliśmy napisać własne podzapytania.

Baza została wypełniona danymi symulującymi 3-letnią działalność firmy, w czasie której firma zorganizowała **80 konferencji** na które złożyło się **239 dni konferencji**. Każda konferencja trwała średnio około 3 dni. Każdy dzień konferencji mógł zawierać kilka warsztatów, w sumie odbyło się **899 warsztatów**. Na wszystkie konferencje były dokonywane rejestracje na dni konferencji i warsztaty. Suma wynosi około **30000 rejestracji**. Firma obsłużyła w tym czasie **250 klientów** i **10000 uczestników. W** bazie znajduje się też **1500 płatności i 3989 cen.**

Funkcje które napisaliśmy ręcznie:

```
-- Poprawienie adresów email klientów na bardziej rzeczywiste
-- Przypisuje nowe adresy sugerując się imieniem i nazwiskiem klienta
SELECT CONCAT(LOWER(REPLACE(SUBSTRING(Name, 0, LEN(Name)), ' ','')),
       '@contact.com') AS EmailName, ClientID
INTO #TableA
FROM Clients

UPDATE Clients
SET
       Email = A.EmailName
FROM #TableA A
WHERE Clients.ClientID = A.ClientID
```

```
-- Poprawienie dat rezerwacji klientów zgodnie z datami konferencji
-- Przypisuje nowe daty uwzględniając daty konferencji
SELECT * INTO #TableA
FROM
	(
		SELECT C.StartDate, R.ReservationID
		FROM Conferences AS C
		JOIN ConferenceDays AS CD
			ON CD.ConferenceID = C.ConferenceID
		JOIN DayReservations AS DRES
			ON DRES.ConferenceDayID = CD.ConferenceDayID
		JOIN Reservations AS R
			ON R.ReservationID = DRES.ReservationID
	) T

UPDATE Reservations
SET
	ReservationDate = DATEADD(DAY, -ABS(CHECKSUM(NEWID())) % 30,
A.StartDate)
FROM #TableA AS A
WHERE Reservations.ReservationID = A.ReservationID



--------------------------------------------------------------------------


-- Poprawienie dat płatności klientów zgodnie z datami rezerwacji
-- Przypisuje nowe daty uwzględniając daty rezerwacji klienta
SELECT * INTO #TableA
FROM
	(
		SELECT C.StartDate, R.ReservationID, R.ReservationDate
		FROM Conferences AS C
		JOIN ConferenceDays AS CD
			ON CD.ConferenceID = C.ConferenceID
		JOIN DayReservations AS DRES
			ON DRES.ConferenceDayID = CD.ConferenceDayID
		JOIN Reservations AS R
			ON R.ReservationID = DRES.ReservationID
	) T

UPDATE Payments
SET
	PaymentDate = DATEADD(DAY, ABS(CHECKSUM(NEWID())) % (13 - 1)
			+ 1, ReservationDate)
FROM #TableA AS A
WHERE Payments.ReservationID = A.ReservationID
```

```
-- Usunięcie progów cenowych ustawionych na datę po rozpoczęciu
-- konferencji
-- Usuwa złe dane wprowadzone generatorem danych
DELETE  TARGET
FROM    Prices AS TARGET
JOIN    ConferenceDays AS CD
ON              CD.ConferenceDayID = TARGET.ConferenceDayID
WHERE   TARGET.UntilDate >= CD.Date

-- Tworzenie tabel conferenceDays według ilości dni przepisanych w
tabeli Conferences
DECLARE @i INT = 0
DECLARE @j INT = 0
WHILE @i < (SELECT COUNT (*) FROM Conferences)
BEGIN
    SET @i = @i + 1
       SET @j = 0
    WHILE @j < (SELECT DATEDIFF(DAY, (SELECT StartDate FROM Conferences
WHERE ConferenceID=@i), (SELECT EndDate FROM Conferences WHERE
ConferenceID=@i)) AS datediff)
  BEGIN
    INSERT INTO ConferenceDays(ConferenceID, Date, ParticipantsLimit,
Location, IsCancelled)
    VALUES (@i, DATEADD(DAY, @j, (SELECT StartDate FROM Conferences
WHERE ConferenceID = @i)), FLOOR(RAND()*(100)+1), 'Somewhere'
,FLOOR(RAND()*(2 - 1) + 1))
    SET @j = @j + 1
  END
END
```