Projekt „Konferencje”

Dokumentacja

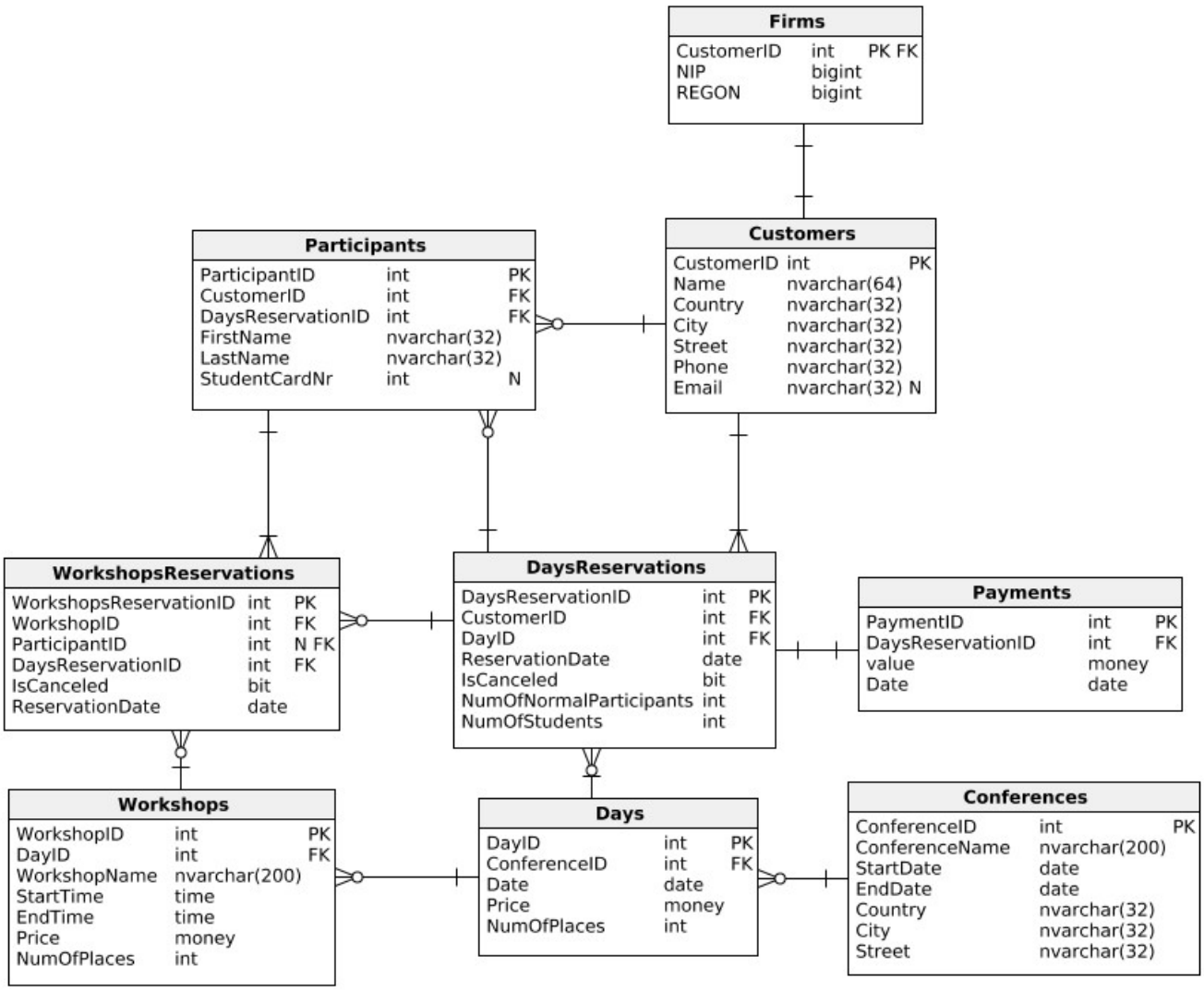
Krzysztof Olewiński Andrzej Kołodziej

22.01.2019

1. Opis bazy danych

Baza danych służy do obsługi konferencji. Przechowuje informacje na temat konferencji, dni konferencji, warsztatów, rezerwacji, uczestników, płatności oraz klientów. Klientami mogą być firmy a także osoby prywatne.

2. Schemat bazy danych



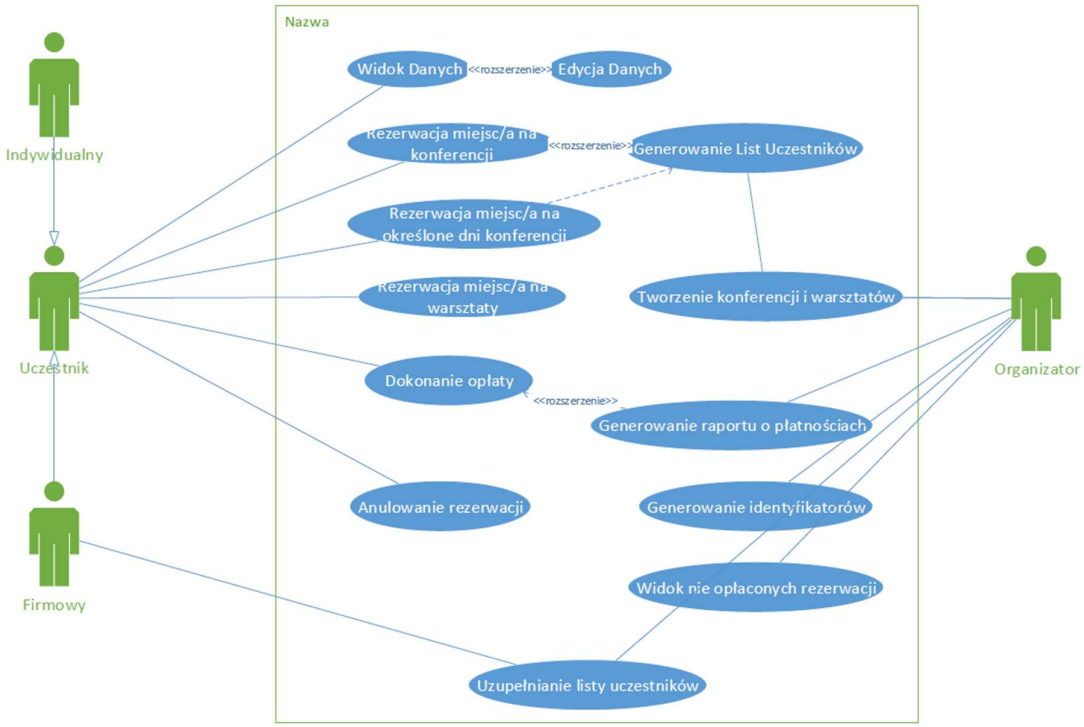
3. Lista aktorów

1. Klient indywidualny
    - Widok swoich danych
    - Edycja swoich danych
    - Rezerwacja miejsca na konferencje (o ile są wolne miejsca)
    - Rezerwacja miejsca na określone dni konferencji (o ile są wolne miejsca)
    - Rezerwacja miejsca na określone warsztaty (o ile są wolne miejsca, oraz klient posiada rezerwacje na dzień konferencji w którym dane warsztaty mają miejsce)
    - Anulowanie rezerwacji miejsca na warsztaty
    - Anulowanie rezerwacji miejsca na konferencje
    - Dokonanie opłaty za udział w konferencji (w ciągu tygodnia od złożenia rezerwacji miejsc)

2. Firma
    - Widok swoich danych
    - Edycja swoich danych
    - Rezerwacja miejsc na konferencje (o ile są wolne miejsca)
    - Rezerwacja miejsc na określone dni konferencji (o ile są wolne miejsca)
    - Rezerwacja miejsc na określone warsztaty (o ile są wolne miejsca, oraz klient posiada rezerwacje na dzień konferencji w którym dane warsztaty mają miejsce)
    - Uzupełnienie listy uczestników
    - Anulowanie rezerwacji miejsc na warsztaty
    - Anulowanie rezerwacji miejsc na konferencje
    - Dokonanie opłaty za udział w konferencji (w ciągu tygodnia od złożenia rezerwacji miejsc)

3. Uczestnik
    - Widok swoich danych
    - Edycja swoich danych
    - Rezerwacja miejsca na określone warsztaty (o ile są wolne miejsca, oraz klient posiada rezerwacje na dzień konferencji w którym dane warsztaty mają miejsce)
    - Anulowanie rezerwacji miejsca na warsztaty

4. Organizator
    - Generowanie listy uczestników na dzień konferencji
    - Generowanie listy uczestników na dane warsztaty
    - Generowanie raportu o płatnościach
    - Generowanie statystyk
    - Tworzenie konferencji i warsztatów
    - Generowanie identyfikatorów uczestników
    - Widok nieopłaconych rezerwacji
    - Raportowanie o braków danych uczestników



4. Tabele

3.1 **Conferences** - tabela przechowuje informacje o konferencjach. Zawiera klucz główny (ConferenceID), nazwę konferencji (ConferenceName), datę rozpoczęcia (StartDate) i jej datę zakończenia (EndDate), oraz adres zawierający kraj (Country), miasto (City) oraz ulicę (Street). Nazwa konferencji wraz z datą jej rozpoczęcia muszą tworzyć unikalną parę. Ponadto żadne pole w tabeli nie może przyjmować wartości NULL oraz data zakończenia musi być datą późniejszą w stosunku do daty rozpoczęcia. Nazwa konferencji może składać się maksymalnie z 200 znaków, a dane adresowe maksymalnie z 32. Kod tworzący tabele:

```
CREATE TABLE Conferences (
   ConferenceID int IDENTITY (1, 1) NOT NULL,
   ConferenceName nvarchar(200) NOT NULL,
   StartDate date NOT NULL,
   EndDate date NOT NULL,
   Country nvarchar(32)  NOT NULL,
   City nvarchar(32)  NOT NULL,
   Street nvarchar(32)  NOT NULL,
   PRIMARY KEY (ConferenceID),
   CONSTRAINT good_dates check (StartDate <= EndDate),
   CONSTRAINT name_and_date unique (ConferenceName, StartDate)
);
```

3.2 **Days** - tabela przechowuje informacje o dniach konferencji. Zawiera klucz główny (DayID), datę (Date), cenę (Price) oraz liczbę miejsc na dzień konferencji (NumOfPlaces). Zawiera również ID konferencji (ConferenceID), do której ten dzień należy (klucz obcy). ID konferencji wraz z datą muszą tworzyć unikalną parę. Ponadto żadne pole w tabeli nie może przyjmować wartości NULL, liczba miejsc musi być liczbą dodatnią, a cena musi być większa bądź równa zero. Wartością domyślną dla liczby miejsc jest 200. Kod tworzący tabele:

```
CREATE TABLE Days (
   DayID int IDENTITY (1, 1) NOT NULL,
   ConferenceID int NOT NULL,
   Date date NOT NULL,
   Price money NOT NULL,
   NumOfPlaces int default (200) NOT NULL,
   PRIMARY KEY (DayID),
   CONSTRAINT num_of_places check (NumOfPlaces > 0),
   CONSTRAINT price check (Price >= 0),
   CONSTRAINT conference_day UNIQUE (ConferenceID, date)
);

ALTER TABLE Days ADD CONSTRAINT Days_Conferences
   FOREIGN KEY (ConferenceID)
   REFERENCES Conferences (ConferenceID);
```

3.3 **Workshops -** tabela przechowuje informacje o warsztatach odbywających się w trakcie dni konferencji. Zawiera klucz główny (WorkshopID), nazwę warsztatu(WorkshopName), godzinę rozpoczęcia(StartTime) i zakończenia(EndTime), cenę(Price) oraz liczbę miejsc na warsztat(NumOfPlaces). Zawiera również ID Dnia(DayID), do którego ten warsztat należy (klucz obcy). Nazwa warsztatu wraz z ID Dnia i godziną rozpoczęcia muszą tworzyć unikalną trójkę. Ponadto żadne pole w tabeli nie może przyjmować wartości NULL, liczba miejsc musi być liczbą dodatnią, cena musi być większa bądź równa zero, a godzina rozpoczęcia warsztatu musi być wcześniejsza w stosunku do godziny jego zakończenia. Wartością domyślną dla liczby miejsc jest 200. Nazwa warsztatu może mieć maksymalnie 200 znaków. Kod tworzący tabele:

```
CREATE TABLE Workshops (
    WorkshopID int IDENTITY (1, 1) NOT NULL,
    DayID int NOT NULL,
    WorkshopName nvarchar(200) NOT NULL,
    StartTime time NOT NULL,
    EndTime time NOT NULL,
    Price money NOT NULL,
    NumOfPlaces int NOT NULL,
    PRIMARY KEY (WrkshopID),
    CONSTRAINT good_times check (StartTime < EndTime),
    CONSTRAINT num_of_places_on_Workshop check (NumOfPlaces > 0),
    CONSTRAINT price_of_workshop check (Price >= 0),
    CONSTRAINT workshop_and_day unique (StartTime,WorkshopName,DayID)
);
ALTER TABLE Workshops ADD CONSTRAINT Workshops_Days
    FOREIGN KEY (DayID)
    REFERENCES Days (DayID);
```

3.4 **Customers** – tabela przechowuje informacje o klientach. Zawiera klucz główny (CustomerID), nazwę klienta(Name), w przypadku klienta indywidualnego jest to jego imię i nazwisko, a w przypadku firmy nazwa firmy, adres zawierający kraj(Country), miasto(City) oraz ulicę(Street), numer telefonu(Phone), oraz adres email(Email). Nazwa klienta musi być unikalna. Jeżeli dwóch klientów będzie się tak samo nazywać należy uwzględnić w polu Name drugie imię lub numer PESEL. Ponadto żadne pole w tabeli nie może przyjmować wartości NULL z wyjątkiem pola z adresem email, numer telefonu musi składać się przynajmniej z 9 cyfr, a adres email musi pasować do wzoru (zawierać znak @ oraz ‘.’). Wszystkie pola w tej tabeli mogą zawierać maksymalnie 32 znaki. Kod tworzący tabele:

```
CREATE TABLE Customers (
    CustomerID int IDENTITY (1, 1) NOT NULL,
    Name nvarchar(32) NOT NULL,
    Country nvarchar(32) NOT NULL,
    City nvarchar(32) NOT NULL,
    Street nvarchar(32) NOT NULL,
    Phone nvarchar(32) NOT NULL,
    Email nvarchar(32),
    PRIMARY KEY  (CustomerID),
    CONSTRAINT phone check (Phone like '%[0-9]{9}'),
```

```
    CONSTRAINT email check (email like '%@%.%'),
    CONSTRAINT name UNIQUE (Name)
);
```

3.5 **Firms** - tabela przechowuje informacje o firmach klientów. Zawiera klucz główny (CustomerID), numer NIP (NIP) i REGON (REGON) firmy. Zawiera również ID Klienta (klucz obcy). Żadne pole w tabeli nie może przyjmować wartości NULL, NIP musi składać się z 10 cyfr. Kod tworzący tabele:

```
CREATE TABLE Firms (
    CustomerID int IDENTITY (1, 1) NOT NULL,
    NIP bigint NOT NULL,
    REGON bigint NOT NULL,
    PRIMARY KEY (CustomerID),
    CONSTRAINT nip check (NIP like '[0-9]{10}')
);

ALTER TABLE Firms ADD CONSTRAINT Customers_Firms
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

3.6 **DaysReservation** - tabela przechowuje informacje o rezerwacjach na dni konferencji. Zawiera klucz główny (DaysReservationID), datę złożenia rezerwacji(ReservationDate), informację czy została anulowana(IsCanceled), liczbę studentów(NumOfStudents) oraz liczbę zwykłych osób(NumOfNormalPsrticipants), na którą została rezerwacja złożona. Zawiera również ID Dnia(DayID), na który została ta rezerwacją złożona (klucz obcy) oraz ID Klienta(CustomerID), który tą rezerwację złożył (klucz obcy). Żadne pole w tabeli nie może przyjmować wartości NULL, a liczba studentów i liczba normalnych uczestników nie mogą jednocześnie być zerami i obie muszą być większe równe zero. Wartością domyślną dla pola informującego o tym czy rezerwacja została anulowana jest fałsz, dla liczby normalnych uczestników jest 1, a dla liczby studentów jest 0. Kod tworzący tabele:

```
CREATE TABLE DaysReservations (
    DaysReservationID int IDENTITY (1, 1) NOT NULL,
    CustomerID int NOT NULL,
    DayID int NOT NULL,
    ReservationDate date NOT NULL,
    IsCanceled bit default (0) NOT NULL,
    NumOfNormalParticipants int default (1) NOT NULL,
    NumOfStudents int default 0 NOT NULL,
    PRIMARY KEY (DaysReservationID),
    CONSTRAINT num_of_participants check (NumOfNormalParticipants > 0 or NumOfStudents > 0),
    CONSTRAINT num_of_normal_participants check (NumOfNormalParticipants >= 0),
    CONSTRAINT num_of_student_participants check (NumOfStudents >= 0),
);
ALTER TABLE DaysReservations ADD CONSTRAINT DaysReservations_Customers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);

ALTER TABLE DaysReservations ADD CONSTRAINT DaysReservations_Days
    FOREIGN KEY (DayID)
    REFERENCES Days (DayID);
```

3.7 **Payments** - tabela przechowuje informacje o płatnościach rezerwacji na dni konferencji. Zawiera klucz główny (PaymentID), datę płatności(Date) i kwotę(Value). Zawiera również ID Rezerwacji Dnia(DaysReservationID), którego ta płatność dotyczy (klucz obcy). Żadne pole w tabeli nie może przyjmować wartości NULL, a kwota płatności musi być większa od zera. Kod tworzący tabele:

```
CREATE TABLE Payments (
	PaymentID int IDENTITY (1, 1) NOT NULL,
	DaysReservationID int NOT NULL,
	Value money NOT NULL,
	Date date NOT NULL,
	PRIMARY KEY (PaymentID),
	CONSTRAINT value check (value > 0)
);

ALTER TABLE Payments ADD CONSTRAINT Payments_DaysReservations
    FOREIGN KEY (DaysReservationID)
    REFERENCES DaysReservations (DaysReservationID);
```

3.8 **Participants** - tabela przechowuje informacje uczestnikach dni konferencji. Zawiera klucz główny (ParticipantID), imię(FirstName) i nazwisko(LastName) oraz numer legitymacji studenckiej(StudentCardNr). Zawiera również ID Klienta(CustomerID) (klucz obcy) oraz ID Rezerwacji Dnia(DaysReservationID) (klucz obcy), na który ten uczestnik jest zapisany. Żadne pole w tabeli nie może przyjmować wartości NULL z wyjątkiem pola z numerem legitymacji studenckiej, który musi składać się z sześciu cyfr. Kod tworzący tabele:

```
CREATE TABLE Participants (
	ParticipantID int IDENTITY (1, 1) NOT NULL,
	CustomerID int NOT NULL,
	DaysReservationID int NOT NULL,
	FirstName nvarchar(32) NOT NULL,
	LastName nvarchar(32) NOT NULL,
	StudentCardNr int NULL,
	PRIMARY KEY (ParticipantID),
	CONSTRAINT student_card_nr check (StudentCardNr like '[0-9]{6}')
);
ALTER TABLE Participants ADD CONSTRAINT Participants_DaysReservation
    FOREIGN KEY (DaysReservationID)
    REFERENCES DaysReservations (DaysReservationID);

ALTER TABLE Participants ADD CONSTRAINT Participants_Customers
    FOREIGN KEY (CustomerID)
    REFERENCES Customers (CustomerID);
```

3.9 **WorkshopReservations** - tabela przechowuje informacje o rezerwacjach na warsztaty odbywające się w trakcie dni konferencji. Zawiera klucz główny (WorkshopsReservationID), datę złożenia rezerwacji(ReservationDate) oraz informację czy została anulowana(IsCanceled). Zawiera również ID Warsztatu(WorkshopID), na który została złożona rezerwacja (klucz obcy), ID Rezerwacji Dnia(DaysReservationID), do której ta rezerwacja należy (klucz obcy) oraz ID Uczestnika(CustomerID) który w tych warsztatach weźmie udział (klucz obcy). Żadne pole w tabeli nie może przyjmować wartości NULL z wyjątkiem pola z ID uczestnika, ponieważ taka informacja nie musi być podana w momencie składania rezerwacji. Kod tworzący tabele:

```
CREATE TABLE WorkshopsReservations (
    WorkshopsReservationID int IDENTITY (1, 1) NOT NULL,
    WorkshopID int NOT NULL,
    ParticipantID int,
    DaysReservationID int NOT NULL,
    IsCanceled bit default (0) NOT NULL,
    ReservationDate date NOT NULL,
    PRIMARY KEY (WorkshopsReservationID),
);

ALTER TABLE WorkshopsReservations ADD CONSTRAINT WorkshopsReservations_DaysReservations
    FOREIGN KEY (DaysReservationID)
    REFERENCES DaysReservations (DaysReservationID);

ALTER TABLE WorkshopsReservations ADD CONSTRAINT WorkshopsReservations_Workshops
    FOREIGN KEY (WorkshopID)
    REFERENCES Workshops (WorkshopID);

ALTER TABLE WorkshopsReservations ADD CONSTRAINT WorkshopsReservations_Participants
    FOREIGN KEY (ParticipantID)
    REFERENCES Participants (ParticipantID);
```

5. Funkcje

Funkcja zwracająca liczbę wolnych miejsc na podany dzień konferencji. Jako argument przyjmuje ID Dnia i zwraca liczbę wolnych miejsc (int). Kod tworzący funkcję:

```
CREATE FUNCTION NumberOfFreePlaces_Day (@DayID int)
      RETURNS INT
      AS
      BEGIN
            DECLARE @Places int
            SET @Places = (select NumOfPlaces from Days where DayID = @DayID)
            DECLARE @Occupied int
            SET @Occupied = ( select count(NumOfNormalParticipants + NumOfStudents) from
            DaysReservations where DayID = @DayID and IsCanceled = 0)
            IF @Occupied is null
            BEGIN
                  SET @Occupied = 0
            END
            RETURN (@Places - @Occupied)
```

```
	END
GO
```

Funkcja zwracająca liczbę wolnych miejsc na podany warsztat. Jako argument przyjmuje ID Warsztatu i zwraca liczbę wolnych miejsc (int). Kod tworzący funkcję:

```
CREATE FUNCTION NumberOfFreePlaces_Workshop (@WorkshopID int)
	RETURNS INT
	AS
	BEGIN
		DECLARE @Places int
		SET @Places = (select NumOfPlaces from Workshops where WorkshopID = @WorkshopID)
		DECLARE @Occupied int
		SET @Occupied = ( select count(*) from WorkshopsReservations where WorkshopID =
		@WorkshopID and IsCanceled = 0)
		IF @Occupied is null
		BEGIN
			SET @Occupied = 0
		END
		RETURN (@Places - @Occupied)
	END
GO
```

Funkcja sprawdzająca czy dwa warsztaty są w tym samym czasie. Jako argumenty przyjmuje dwa ID Warsztatu i zwraca 1, jeżeli warsztaty odbywają się w tym samym czasie, 0 w przeciwnym razie. Kod tworzący funkcję:

```
CREATE FUNCTION isTheSameTimeOfWorkshops (@WorkshopID1 int, @WorkshopID2 int)
	RETURNS bit
	AS
	BEGIN
		DECLARE @StartTime1 time
		SET @StartTime1 = (select StartTime from Workshops where WorkshopID =
		@WorkshopID1)
		DECLARE @StartTime2 time
		SET @StartTime2 = (select StartTime from Workshops where WorkshopID =
		@WorkshopID2)
		DECLARE @EndTime1 time
		SET @EndTime1 = (select EndTime from Workshops where WorkshopID = @WorkshopID1)
		DECLARE @EndTime2 time
		SET @EndTime2 = (select EndTime from Workshops where WorkshopID = @WorkshopID2)
		DECLARE @Date1 date
		SET @Date1 = ( select Date from Days where DayID = ( select DayID from Workshops
		where WorkshopID = @WorkshopID1))
		DECLARE @Date2 date
		SET @Date2 = ( select Date from Days where DayID = ( select DayID from Workshops
		where WorkshopID = @WorkshopID2))
		IF @Date1 <> @Date2
		BEGIN
			RETURN 0
		END
		IF (@StartTime1 <= @StartTime2 and @StartTime2 <= @EndTime1) or (@StartTime2 <=
		@StartTime1 and @StartTime1 <= @EndTime2)
		BEGIN
			RETURN 1
		END
		RETURN 0
	END
GO
```

Funkcja zwracająca ID Konferencji pasującej do podanych argumentów. Jako argumenty przyjmuje nazwę i datę rozpoczęcia konferencji i zwraca znalezione ID (int). Kod tworzący funkcję:

```
CREATE FUNCTION idOfConferenceIncludingDate (@ConferenceName nvarchar(200), @Date date)
	RETURNS int
	AS
	BEGIN
		RETURN (
			select ConferenceID from Conferences
			where ConferenceName = @ConferenceName and StartDate <= @Date and EndDate
			>= @Date
		)
	END
GO
```

Funkcja zwracająca ID Klienta pasującego do podanego argumentu. Jako argument przyjmuje nazwę klienta i zwraca znalezione ID (int). Kod tworzący funkcję:

```
CREATE FUNCTION idOfCustomer (@CustomerName nvarchar(64))
	RETURNS int
	AS
	BEGIN
		Return (
			select CustomerID from Customers
			where Name = @CustomerName )
	END
GO
```

Funkcja zwracająca ID Dnia pasującego do podanych argumentów. Jako argumenty przyjmuje nazwę Konferencji i datę szukanego dnia i zwraca znalezione ID (int). Kod tworzący funkcję:

```
CREATE FUNCTION idOfDay (@ConferenceName nvarchar(200), @Date date)
	RETURNS int
	AS
	BEGIN
		Return (
			select DayID from Days
			where ConferenceID = dbo.idOfConferenceIncludingDate(@ConferenceName,@Date)
			and Date = @Date )
	END
GO
```

Funkcja zwracająca ID Warsztatu pasującego do podanych argumentów. Jako argumenty przyjmuje nazwę warsztatu, datę oraz godzinę jego rozpoczęcia i zwraca znalezione ID (int). Kod tworzący funkcję:

```
CREATE FUNCTION idOfWorkshop (@WorkshopName nvarchar(200), @SatrtTime time, @DayID int)
	RETURNS int
	AS
	BEGIN
		Return (
			select WorkshopID from Workshops
			where WorkshopName = @WorkshopName and StartTime = @SatrtTime and DayID =
			@DayID)
	END
GO
```

Funkcja zwracająca ID Rezerwacji na dzień pasującej do podanych argumentów. Jako argumenty przyjmuje ID Klienta oraz ID Dnia i zwraca znalezione ID (int). Kod tworzący funkcję:

```
CREATE FUNCTION idOfDaysReservation (@CustomerID int, @DayID int)
	RETURNS int
	AS
	BEGIN
		Return (
			select DaysReservationID from DaysReservations
			where CustomerID = @CustomerID and DayID = @DayID and IsCanceled = 0)
	END
GO
```

Funkcja zwracająca ID Uczestnika pasującego do podanych argumentów. Jako argumenty przyjmuje ID Klienta, ID Rezerwacji na dzień oraz imię i nazwisko szukanego uczestnika i zwraca znalezione ID (int). Kod tworzący funkcję:

```
CREATE FUNCTION idOfParticipant (@CustomerID int, @FirstName nvarchar(32), @LastName
nvarchar(32), @DaysReservationID int)
	RETURNS int
	AS
	BEGIN
		Return (
			select ParticipantID from Participants
			where CustomerID = @CustomerID and FirstName = @FirstName and LastName =
			@LastName and DaysReservationID = @DaysReservationID)
	END
GO
```

Funkcja zwracająca obliczoną cenę rezerwacji na dzień konferencji uwzględniającą koszt zarezerwowanych w tym dniu warsztatów, zniżki studenckie oraz zniżki z tytułu wcześniejszej rezerwacji. Jako argument funkcja przyjmuje ID Rezerwacji na dzień i zwraca obliczoną cenę (int). Kod tworzący funkcję:

```
CREATE FUNCTION countPriceOfDayResrvation (@DaysReservationID int)
	RETURNS money
	AS
	BEGIN
		DECLARE @WorkshopsPrice money
		SET @WorkshopsPrice = (
			select sum(Price) from Workshops
			where WorkshopID in ( select WorkshopID from WorkshopsReservations where
			DaysReservationID = @DaysReservationID and IsCanceled = 0 )
		)
		DECLARE @DayPrice money
		SET @DayPrice = (
			select Price from Days
			where DayID = (
				select DayID from DaysReservations
				where DaysReservationID = @DaysReservationID and IsCanceled = 0
			)
		)
		DECLARE @NormalParticipantsPrice money
		SET @NormalParticipantsPrice = @DayPrice * (
			select NumOfNormalParticipants from DaysReservations
			where DaysReservationID = @DaysReservationID
		)
		DECLARE @StudentParticipantsPrice money
		SET @StudentParticipantsPrice = @DayPrice * (
			select NumOfStudents from DaysReservations
			where DaysReservationID = @DaysReservationID
		) * 0.7
		SET @DayPrice = @StudentParticipantsPrice + @NormalParticipantsPrice
		DECLARE @ConferenceStartDay date
		SET @ConferenceStartDay = (
			select Date from Days
			where DayID in (
				select DayID from DaysReservations
				where DaysReservationID = @DaysReservationID
			)
		)
		DECLARE @DaysToConference int
		SET @DaysToConference = datediff(day,getdate(),@ConferenceStartDay)
		IF @DaysToConference > 30
		BEGIN
			SET @DayPrice = @DayPrice * 0.7
		END
		ELSE IF @DaysToConference > 14
		BEGIN
			SET @DayPrice = @DayPrice * 0.85
		END
		IF @WorkshopsPrice is null
		BEGIN
			SET @WorkshopsPrice = 0
		END
		return @WorkshopsPrice + @DayPrice
	END
GO
```

6. Triggery

Trigger anulujący rezerwacje na warsztaty wtedy, gdy zostaje anulowana rezerwacja na dzień konferencji. Kod tworzący trigger:

```
CREATE TRIGGER CancelWorkshopsReservations
	ON DaysReservations
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(IsCanceled)
		BEGIN
			DECLARE @DaysReservationID AS int
			SET @DaysReservationID = (SELECT DaysReservationID FROM INSERTED)

			UPDATE WorkshopsReservations
			SET IsCanceled = 1, ParticipantID = null
			WHERE DaysReservationID = @DaysReservationID and IsCanceled = 0
		END
	END
GO
```

Trigger usuwający uczestników rezerwacji na dzień, gdy ta zostaje anulowana. Kod tworzący trigger:

```
CREATE TRIGGER DeleteParticipantsWithoutActualReservation
	ON DaysReservations
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(IsCanceled)
		BEGIN
			DECLARE @DaysReservationID AS int
			SET @DaysReservationID = (SELECT DaysReservationID FROM INSERTED)
			DELETE Participants
			WHERE DaysReservationID = @DaysReservationID
		END
	END
GO
```

Trigger sprawdzający czy data rozpoczęcia konferencji nie jest z przyszłości. Kod tworzący trigger:

```
CREATE TRIGGER checkDate
	ON Conferences
	AFTER INSERT,UPDATE
	AS
	BEGIN
		SET NOCOUNT ON;
		DECLARE @Date date = (SELECT StartDate FROM inserted)
		IF((DATEDIFF(day,GETDATE(),@Date) <= 0))
		BEGIN ;
			THROW 52000, 'Podana data jest z przeszłości', 2
		END
	END
GO
```

Trigger sprawdzający czy istnieją już takie same warsztaty o tej samej godzinie. Kod tworzący trigger:

```
CREATE TRIGGER WorkshopsInTheSameTime
	ON Workshops
	AFTER INSERT
	AS
	BEGIN
		SET NOCOUNT ON;
		DECLARE @WorkshopID int
		SET @WorkshopID= ( select WorkshopID from INSERTED)
		DECLARE @WorkshopName nvarchar(200)
		SET @WorkshopName = ( select WorkshopName from INSERTED)
		IF exists (
			select * from Workshops as w
			where WorkshopName = @WorkshopName and WorkshopID <> @WorkshopID and
			dbo.isTheSameTimeOfWorkshops(WorkshopID,@WorkshopID) = 1
		)
		BEGIN;
			THROW 52000, 'Istnieją już takie warsztaty w podanym czasie', 2
		END
	END
GO
```

Trigger sprawdzający czy anulowana rezerwacja na dzień nie została już opłacona. Kod tworzący trigger:

```
CREATE TRIGGER CancelPaidDayReservation
	ON DaysReservations
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(IsCanceled)
		BEGIN
			DECLARE @DaysReservationID AS int
			SET @DaysReservationID = (SELECT DaysReservationID FROM INSERTED)
			IF exists (
				select * from Payments
				where DaysReservationID = @DaysReservationID
			)
			BEGIN;
				THROW 52000, 'Rezerwacja została już opłacona', 2
			END
		END
	END
GO
```

Trigger sprawdzający czy anulowana rezerwacja na dzień nie została już opłacona. Kod tworzący trigger:

```
CREATE TRIGGER CancelPaidWorkshopReservation
	ON WorkshopsReservations
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(IsCanceled)
		BEGIN
			DECLARE @DaysReservationID AS int
			SET @DaysReservationID = (SELECT top 1 DaysReservationID FROM INSERTED)
			IF exists (
				select * from Payments
				where DaysReservationID = @DaysReservationID
			)
			BEGIN;
				THROW 52000, 'Rezerwacja została już opłacona', 2
			END
		END
	END
GO
```

Trigger wywoływany przy zmianie ilości miejsc na dzień konferencji, sprawdzający czy istnieje warsztat odbywający się w dniu konferencji, który ma więcej miejsc niż nowa liczba. Kod tworzący trigger:

```
IF OBJECT_ID('WorkshopWithBiggerNumberOfPlaces') is not null
DROP TRIGGER WorkshopWithBiggerNumberOfPlaces
GO
CREATE TRIGGER WorkshopWithBiggerNumberOfPlaces
	ON Days
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(NumOfPlaces)
		BEGIN
			DECLARE @DayID int
			SET @DayID = ( select DayID from inserted )
			DECLARE @NewNumOfPlaces int
			SET @NewNumOfPlaces = ( select NumOfPlaces from inserted )
			DECLARE @BiggestNumOfPlaces int
			SET @BiggestNumOfPlaces = (
				select max(NumOfPlaces) from Workshops
				where DayID = @DayID
			)
			IF @NewNumOfPlaces < @BiggestNumOfPlaces
			BEGIN;
				THROW 52000, 'Dzień zawiera warsztaty z większą ilością miejsc niż
				podana liczba', 2
			END
		END
	END
GO
```

Trigger wywoływany przy zmianie ilości miejsc na warsztat, sprawdzający czy na dzień, w którym się odbywa nie ma mniejszej liczby mniejsc niż podana nowa liczba. Kod tworzący trigger:

```
IF OBJECT_ID('DayWithSmallerNumberOfPlaces') is not null
DROP TRIGGER DayWithSmallerNumberOfPlaces
GO
CREATE TRIGGER DayWithSmallerNumberOfPlaces
	ON Workshops
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(NumOfPlaces)
		BEGIN
			DECLARE @DayID int
			SET @DayID = ( select DayID from inserted )
			DECLARE @NewNumOfPlaces int
			SET @NewNumOfPlaces = ( select NumOfPlaces from inserted )
			IF @NewNumOfPlaces > ( select NumOfPlaces from Days where DayID = @DayID )
			BEGIN;
				THROW 52000, 'Dzień ma mniejsza ilość miejsc niz podana liczba', 2
			END
		END
	END
GO
```

Trigger wywoływany przy zmianie ilości miejsc na dzień konferencji, sprawdzający czy na ten dzień nie ma więcej rezerwacji niż podana liczba miejsc. Kod tworzący trigger:

```
CREATE TRIGGER MoreDayReservationsThanNewNumOfPlaces
	ON Days
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(NumOfPlaces)
		BEGIN
			DECLARE @DayID int
			SET @DayID = ( select DayID from inserted )
			DECLARE @NewNumOfPlaces int
			SET @NewNumOfPlaces = ( select NumOfPlaces from inserted )
			IF @NewNumOfPlaces < (select SUM(NumOfNormalParticipants + NumOfStudents)
			from DaysReservations where DayID = @DayID and IsCanceled = 0)
			BEGIN;
				THROW 52000, 'Na ten dzień jest wiecej złożonych rezerwacji niż
				podana liczba miejsc', 2
			END
		END
	END
GO
```

Trigger wywoływany przy zmianie ilości miejsc na warsztat, sprawdzający czy na ten warsztat nie ma więcej rezerwacji niż podana liczba miejsc. Kod tworzący trigger:

```
CREATE TRIGGER MoreWorkshopReservationsThanNewNumOfPlaces
	ON Workshops
	FOR UPDATE
	AS
	BEGIN
		IF UPDATE(NumOfPlaces)
		BEGIN
```

```
                    DECLARE @WorkshopID int
                    SET @WorkshopID = ( select WorkshopID from inserted )
                    DECLARE @NewNumOfPlaces int
                    SET @NewNumOfPlaces = ( select NumOfPlaces from inserted )
                    IF @NewNumOfPlaces < ( select COUNT(*) from WorkshopsReservations where
                    WorkshopID = @WorkshopID and IsCanceled = 0 )
                    BEGIN;
                          THROW 52000, 'Na ten warsztat jest wiecej złożonych rezerwacji niż
                          podana liczba miejsc', 2
                    END
              END
        END
GO
```

7. Procedury

Procedura dodająca dzień konferencji. Jako argumenty przyjmuje nazwę konferencji, datę dnia, cenę i liczbę miejsc. Kod tworzący procedurę:

```
CREATE PROCEDURE AddDay
	@ConferenceName nvarchar(200),
	@Date date,
	@Price money,
	@NumOfPlaces int
	AS
	BEGIN
		SET NOCOUNT ON;
		DECLARE @ConferenceID int
		SET @ConferenceID = dbo.idOfConferenceIncludingDate(@ConferenceName,@Date)
		IF @ConferenceID is not null
		BEGIN
			INSERT INTO Days(ConferenceID, Date, Price, NumOfPlaces)
			VALUES (@ConferenceID, @Date, @Price, @NumOfPlaces)
		END
	END
GO
```

Procedura dodająca konferencje. Jako argumenty przyjmuje nazwę konferencji, datę początku i datę końca, adres w postaci kraju, miasta i ulicę, cenę i liczbę miejsc. Kod tworzący procedurę:

```
CREATE PROCEDURE AddConference
	@ConferenceName nvarchar(200),
	@StartDate date,
	@EndDate date,
	@Country nvarchar(32),
	@City nvarchar(32),
	@Street nvarchar(32),
	@Price money,
	@NumOfPlaces int
	AS
	BEGIN
		BEGIN TRANSACTION
		SET NOCOUNT ON;
		INSERT INTO Conferences(ConferenceName, Country, City, Street, StartDate, EndDate)
		VALUES (@ConferenceName,@Country,@City,@Street,@StartDate,@EndDate)
		DECLARE @Date date
		SET @Date = @StartDate
		WHILE (@Date <= @EndDate)
		BEGIN
			exec AddDay @ConferenceName, @Date, @Price, @NumOfPlaces
			SET @Date = dateadd(day,1,@Date)
		END
		COMMIT TRANSACTION
	END
GO
```

Procedura dodająca warsztat do dnia konferencji. Jako argumenty przyjmuje nazwę konferencji, datę, nazwę warsztatu, godzinę rozpoczęcia i zakończenia warsztatu, cenę i liczbę miejsc. W przypadku nieznalezienia konferencji lub dnia konferencji procedura rzuca odpowiedni wyjątek. Kod tworzący procedurę:

```
CREATE PROCEDURE AddWorkshop
	@ConferenceName nvarchar(200),
	@Date date,
	@WorkshopName nvarchar(200),
	@StartTime time,
	@EndTime time,
	@Price money,
	@NumOfPlaces int
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @NumOfPlaces > (
			select NumOfPlaces from Days
			where DayID = @DayID
		)
		BEGIN
			RAISERROR('Za duza liczba miejsc',16,2)
		END
		INSERT INTO Workshops(DayID, WorkshopName, StartTime, EndTime, Price, NumOfPlaces)
		VALUES (@DayID,@WorkshopName,@StartTime,@EndTime,@Price,@NumOfPlaces)
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot add Workshop. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura dodająca rezerwację na warsztat. Jako argumenty przyjmuje nazwę klienta, nazwę konferencji, datę, nazwę warsztatu i godzinę rozpoczęcia warsztatu. W przypadku nieznalezienia klienta, konferencji, dnia konferencji, rezerwacji na dzień konferencji, warsztatu lub braku wolnych miejsc procedura rzuca odpowiedni wyjątek. Nie można również dodać rezerwacji warsztatu do opłaconej już rezerwacji na dzień. Kod tworzący procedurę:

```
CREATE PROCEDURE AddWorkshopReservation
	@CustomerName nvarchar(64),
	@ConferenceName nvarchar(200),
	@Date date,
	@WorkshopName nvarchar(200),
	@StartTime time
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		DECLARE @DaysReservationID int
		SET @DaysReservationID = dbo.idOfDaysReservation(@CustomerID,@DayID)
		IF @DaysReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany dzień konferencji',16,6) END

		DECLARE @WorkshopID int
		SET @WorkshopID = dbo.idOfWorkshop(@WorkshopName,@StartTime,@DayID)
		IF @WorkshopID is null
		BEGIN RAISERROR('Nie znaleziono warsztatu',16,5) END

		DECLARE @FreePlaces int
		SET @FreePlaces = dbo.NumberOfFreePlaces_Workshop(@WorkshopID)
		IF ( @FreePlaces <= 0 )
		BEGIN RAISERROR('Brak wolnych miejsc na podany warsztatu',16,5) END

		IF exists (
			select * from Payments
			where DaysReservationID = @DaysReservationID
		)
		BEGIN RAISERROR('Nie można dodać warsztatu do opłaconej już rezerwacji',16,5) END

		INSERT INTO WorkshopsReservations(WorkshopID, DaysReservationID, ReservationDate)
		VALUES (@WorkshopID, @DaysReservationID, getdate())
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot add WorkshopReservation. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura anulująca rezerwację na warsztat. Jako argumenty przyjmuje nazwę klienta, nazwę konferencji, datę, nazwę warsztatu, godzinę rozpoczęcia warsztatu oraz imię i nazwisko uczestnika. Dane uczestnika mogą przyjmować wartość NULL. W przypadku nieznalezienia klienta, dnia konferencji, rezerwacji na dzień konferencji, warsztatu procedura rzuca odpowiedni wyjątek. Nie można również anulować rezerwacji warsztatu z opłaconej już rezerwacji na dzień. Kod tworzący procedurę:

```
CREATE PROCEDURE CancelWorkshopReservation
	@CustomerName nvarchar(64),
	@ConferenceName nvarchar(200),
	@Date date,
	@WorkshopName nvarchar(200),
	@StartTime time,
	@Firstname nvarchar(32),
	@Lastname nvarchar(32)
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		DECLARE @DaysReservationID int
		SET @DaysReservationID = dbo.idOfDaysReservation(@CustomerID,@DayID)
		IF @DaysReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany dzień konferencji',16,5) END

		DECLARE @WorkshopID int
		SET @WorkshopID = dbo.idOfWorkshop(@WorkshopName,@StartTime,@DayID)
		IF @WorkshopID is null
		BEGIN RAISERROR('Nie znaleziono warsztatu',16,5) END

		DECLARE @ParticipantID int
		SET @ParticipantID =
		dbo.idOfParticipant(@CustomerID,@Firstname,@Lastname,@DaysReservationID)
		DECLARE @WorkshopReservatioID int
		IF @ParticipantID is not null
		BEGIN
			SET @WorkshopReservatioID = (
				select top 1 WorkshopsReservationID from WorkshopsReservations
				where WorkshopID = @WorkshopID and DaysReservationID =
				@DaysReservationID and ParticipantID = @ParticipantID and IsCanceled
				= 0
			)
		END
		ELSE
		BEGIN
			SET @WorkshopReservatioID = (
				select top 1 WorkshopsReservationID from WorkshopsReservations
				where WorkshopID = @WorkshopID and DaysReservationID =
				@DaysReservationID and ParticipantID is null and IsCanceled = 0
			)
		END

		UPDATE WorkshopsReservations
```

```
            SET IsCanceled = 1 where WorkshopsReservationID = @WorkshopReservatioID
            END TRY
            BEGIN CATCH
            DECLARE @ErrorMsg nvarchar (2048)
                  = 'Cannot cancel WorkshopReservation. Error message : '
                        + ERROR_MESSAGE () ;
            ; THROW 50003 , @ErrorMsg ,1
            END CATCH
      END
GO
```

Procedura dodająca rezerwację na dzień konferencji. Jako argumenty przyjmuje nazwę klienta, nazwę konferencji, datę, liczbę normalnych uczestników konferencji i liczbę studentów. W przypadku nieznalezienia klienta, konferencji, dnia konferencji lub braku wolnych miejsc procedura rzuca odpowiedni wyjątek. Nie można również dodać rezerwacji, jeżeli posiada się już inną rezerwację na ten dzień. Kod tworzący procedurę:

```
CREATE PROCEDURE AddDaysReservation
      @CustomerName nvarchar(64),
      @ConferenceName nvarchar(200),
      @Date date,
      @NumOfPlaces int,
      @NumOfStudents int
      AS
      BEGIN
            SET NOCOUNT ON;
            BEGIN TRY
            DECLARE @CustomerID int
            SET @CustomerID = dbo.idOfCustomer(@CustomerName)
            IF @CustomerID is null
            BEGIN RAISERROR('Nie znaleziono klienta',16,4) END
            DECLARE @DayID int
            SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
            IF @DayID is null
            BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

            if ( (select count(*) from DaysReservations where CustomerID = @CustomerID and
            DayID = @DayID and IsCanceled = 0) > 0 )
            BEGIN RAISERROR('Podany użytkownik posiada już rezerwacje na podany dzień',16,5)
            END
            DECLARE @FreePlaces int
            SET @FreePlaces = dbo.NumberOfFreePlaces_Day(@DayID)
            IF ( @FreePlaces < (@NumOfPlaces + @NumOfStudents))
            BEGIN RAISERROR('Brak podanej ilości miejsc na podany dzień konferencji',16,5) END

            INSERT INTO DaysReservations(CustomerID, DayID,
            ReservationDate,NumOfNormalParticipants,NumOfStudents)
            VALUES (@CustomerID, @DayID, getdate(),@NumOfPlaces,@NumOfStudents)
            END TRY
            BEGIN CATCH
            DECLARE @ErrorMsg nvarchar (2048)
                  = 'Cannot add DaysReservation. Error message : '
                        + ERROR_MESSAGE () ;
            ; THROW 50003 , @ErrorMsg ,1
            END CATCH
      END
GO
```

Procedura anulująca rezerwację na warsztat. Jako argumenty przyjmuje nazwę klienta, nazwę konferencji i datę. W przypadku nieznalezienia klienta, dnia konferencji, rezerwacji na dzień konferencji procedura rzuca odpowiedni wyjątek. Nie można również anulować opłaconej już rezerwacji na dzień. Kod tworzący procedurę:

```
CREATE PROCEDURE CancelDaysReservation
	@CustomerName nvarchar(64),
	@ConferenceName nvarchar(200),
	@Date date
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		DECLARE @DaysReservationID int
		SET @DaysReservationID = dbo.idOfDaysReservation(@CustomerID,@DayID)
		IF @DaysReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany dzień konferencji',16,5) END

		UPDATE DaysReservations
		SET IsCanceled = 1 where DaysReservationID = @DaysReservationID
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot cancel DaysReservation. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura dodająca uczestnika do rezerwacji na warsztat. Jako argumenty przyjmuje nazwę klienta, imię i nazwisko uczestnika, nazwę konferencji i datę. W przypadku nieznalezienia klienta, dnia konferencji, rezerwacji na dzień konferencji, warsztatu, rezerwacji na warsztat lub uczestnika procedura rzuca odpowiedni wyjątek. Nie można również dodać uczestnika, który jest już zapisany na inne warsztaty odbywające się w tym samym czasie. Kod tworzący procedurę:

```
CREATE PROCEDURE AddParticipantToWorkshopsReservation
	@CustomerName nvarchar(64),
	@Firstname nvarchar(32),
	@Lastname nvarchar(32),
	@ConferenceName nvarchar(200),
	@Date date,
	@WorkshopName nvarchar(200),
	@StartTime time
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
```

```
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		DECLARE @DaysReservationID int
		SET @DaysReservationID = dbo.idOfDaysReservation(@CustomerID,@DayID)
		IF @DaysReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany dzień konferencji',16,6) END

		DECLARE @ParticipantID int
		SET @ParticipantID =
		dbo.idOfParticipant(@CustomerID,@Firstname,@Lastname,@DaysReservationID)
		IF @ParticipantID is null
		BEGIN RAISERROR('Nie znaleziono uczestnika',16,7) END

		DECLARE @WorkshopID int
		SET @WorkshopID = dbo.idOfWorkshop(@WorkshopName,@StartTime,@DayID)
		IF @WorkshopID is null
		BEGIN RAISERROR('Nie znaleziono warsztatu',16,5) END

		DECLARE @WorkshopReservationID int
		SET @WorkshopReservationID = (
			select top 1 WorkshopsReservationID from WorkshopsReservations
			where WorkshopID = @WorkshopID and DaysReservationID = @DaysReservationID
			and ParticipantID is null and IsCanceled = 0
		)
		IF @WorkshopReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany warsztat',16,5) END

		IF exists (
			select * from WorkshopsReservations
			where ParticipantID = @ParticipantID and
			dbo.isTheSameTimeOfWorkshops(WorkshopID,(select WorkshopID from
			WorkshopsReservations where WorkshopsReservationID =
			@WorkshopReservationID)) = 1
		)
		BEGIN RAISERROR('Uczestnik jest juz zapisany na inny warsztat w tym samym
		czasie',16,5) END

		UPDATE WorkshopsReservations
		SET ParticipantID = @ParticipantID where WorkshopsReservationID =
		@WorkshopReservationID
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot add ParticipantToWorkshopsReservation. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura usuwająca uczestnika z rezerwacji na warsztat. Jako argumenty przyjmuje nazwę klienta, imię i nazwisko uczestnika, nazwę konferencji i datę. W przypadku nieznalezienia klienta, dnia konferencji, rezerwacji na dzień konferencji, warsztatu, rezerwacji na warsztat lub uczestnika procedura rzuca odpowiedni wyjątek. Kod tworzący procedurę:

```
CREATE PROCEDURE DeleteParticipantFromWorkshopsReservation
	@CustomerName nvarchar(64),
	@Firstname nvarchar(32),
	@Lastname nvarchar(32),
	@ConferenceName nvarchar(200),
	@Date date,
	@WorkshopName nvarchar(200),
	@StartTime time
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		DECLARE @DaysReservationID int
		SET @DaysReservationID = dbo.idOfDaysReservation(@CustomerID,@DayID)
		IF @DaysReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany dzień konferencji',16,6) END

		DECLARE @ParticipantID int
		SET @ParticipantID =
		dbo.idOfParticipant(@CustomerID,@Firstname,@Lastname,@DaysReservationID)
		IF @ParticipantID is null
		BEGIN RAISERROR('Nie znaleziono uczestnika',16,7) END

		DECLARE @WorkshopID int
		SET @WorkshopID = dbo.idOfWorkshop(@WorkshopName,@StartTime,@DayID)
		IF @WorkshopID is null
		BEGIN RAISERROR('Nie znaleziono warsztatu',16,5) END

		DECLARE @WorkshopReservationID int
		SET @WorkshopReservationID = (
			select top 1 WorkshopsReservationID from WorkshopsReservations
			where WorkshopID = @WorkshopID and DaysReservationID = @DaysReservationID
		and ParticipantID is null and IsCanceled = 0
		)
		IF @WorkshopReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany warsztat',16,5) END

		UPDATE WorkshopsReservations
		SET ParticipantID = null where WorkshopsReservationID = @WorkshopReservationID
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot add ParticipantToWorkshopsReservation. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura dodająca nowego uczestnika dnia konferencji. Jako argumenty przyjmuje nazwę klienta, imię i nazwisko uczestnika, nazwę konferencji i datę oraz opcjonalny numer albumu studenta. W przypadku nieznalezienia klienta, dnia konferencji, rezerwacji na dzień konferencji procedura rzuca odpowiedni wyjątek. Nie można również dodać uczestnika do rezerwacji, na dzień która ma już kompletną listę uczestników oraz nie można dodać do jednej rezerwacji na dzień dwóch uczestników o tych samych danych. Kod tworzący procedurę:

```
CREATE PROCEDURE AddParticipant
	@CustomerName nvarchar(64),
	@ConferenceName nvarchar(200),
	@Date date,
	@FirstName nvarchar(32),
	@LastName nvarchar(32),
	@StudentCardNr int
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		DECLARE @DaysReservationID int
		SET @DaysReservationID = dbo.idOfDaysReservation(@CustomerID,@DayID)
		IF @DaysReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany dzień konferencji',16,5) END

		DECLARE @NumOfParticipants int
		IF @StudentCardNr is null
		BEGIN
			SET @NumOfParticipants = (
				select NumOfNormalParticipants from DaysReservations
				where DaysReservationID = @DaysReservationID
			) - (
				select count(*) from Participants
				where DaysReservationID = @DaysReservationID and StudentCardNr is
				null
			)
			IF @NumOfParticipants < 1
			BEGIN RAISERROR('Rezerwacja zawiera już pełną listę uczestników',16,8) END
		END
		ELSE
		BEGIN
			SET @NumOfParticipants = (
				select NumOfStudents from DaysReservations
				where DaysReservationID = @DaysReservationID
			) - (
				select count(*) from Participants
				where DaysReservationID = @DaysReservationID and StudentCardNr is not
				null
			)
			IF @NumOfParticipants < 1
			BEGIN RAISERROR('Rezerwacja zawiera już pełną listę uczestników',16,8) END
		END
```

```
            IF exists ( select * from Participants where FirstName = @FirstName and LastName =
            @LastName and CustomerID = @CustomerID and DaysReservationID = @DaysReservationID
            )
            BEGIN RAISERROR('Rezerwacja zawiera już uczestnika o podanych danych',16,8) END

            INSERT INTO Participants(CustomerID, FirstName, LastName, StudentCardNr,
            DaysReservationID)
            VALUES (@CustomerID,@FirstName,@LastName,@StudentCardNr,@DaysReservationID)
            END TRY
            BEGIN CATCH
            DECLARE @ErrorMsg nvarchar (2048)
                  = 'Cannot add Participant. Error message : '
                        + ERROR_MESSAGE () ;
            ; THROW 50003 , @ErrorMsg ,1
            END CATCH
      END
GO
```

Procedura dodająca nowego klienta. Jako argumenty przyjmuje nazwę klienta, dane adresowe w postaci kraju, miasta i ulice oraz numer telefonu i adres email, który jest opcjonalny. Kod tworzący procedurę:

```
CREATE PROCEDURE AddCustomer
      @Name nvarchar(64),
      @Country nvarchar(32),
      @City nvarchar(32),
      @Street nvarchar(32),
      @Phone nvarchar(32),
      @Email nvarchar(32)
      AS
      BEGIN
            SET NOCOUNT ON;
            INSERT INTO Customers(Name,Country,City,Street,Phone,Email)
            VALUES (@Name,@Country,@City,@Street,@Phone,@Email)
      END
GO
```

Procedura anulujące wszystkie rezerwację, które nie zostały opłacone w ciągu tygodnia. Kod tworzący procedurę:

```
CREATE PROCEDURE CancelUnpaidReservation
      AS
      BEGIN
            SET NOCOUNT ON;
            UPDATE DaysReservations
            SET IsCanceled = 1 where IsCanceled = 0 and
            datediff(day,ReservationDate,getdate()) >= 7
      END
GO
```

Procedura dodająca firmę. Jako argumenty przyjmuje nazwę klienta, numer NIP ora REGON. W przypadku nieznalezienia klienta procedura rzuca odpowiedni wyjątek. Kod tworzący procedurę:

```
CREATE PROCEDURE AddFirm
	@CustomerName nvarchar(64),
	@NIP bigint,
	@Regon bigint
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		INSERT INTO Firms(CustomerID,NIP,REGON)
		VALUES (@CustomerID,@NIP,@Regon)
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot add Firm. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura umożliwiająca zmianę liczby miejsc na dzień konferencji. Jako argumenty przyjmuje nazwę konferencji, datę dnia i nową liczbę miejsc. W przypadku nieznalezienia konferencji procedura rzuca odpowiedni wyjątek. Nie można również zmienić liczby miejsc w przypadku, gdy w tym dniu istnieją warsztaty z większą możliwą ilością uczestników niż podana liczba. Kod tworzący procedurę:

```
CREATE PROCEDURE ChangeNumberOfDayPlaces
	@ConferenceName nvarchar(200),
	@Date date,
	@NewNumOfPlaces int
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		UPDATE Days
		SET NumOfPlaces = @NewNumOfPlaces where DayID = @DayID
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot change NumberOfDayPlaces. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura umożliwiająca zmianę liczby miejsc na warsztat. Jako argumenty przyjmuje nazwę konferencji, datę dnia, nazwę warsztatu i godzinę jego rozpoczęcia oraz nową liczbę miejsc. W przypadku nieznalezienia dnia konferencji procedura rzuca odpowiedni wyjątek. Nie można również zmienić liczby miejsc w przypadku, gdy dzień konferencji ma mniejsza liczbę możliwych uczestników niż podana liczba. Kod tworzący procedurę:

```
CREATE PROCEDURE ChangeNumberOfWorkshopsPlaces
	@ConferenceName nvarchar(200),
	@Date date,
	@WorkshopName nvarchar(200),
	@StartTime time,
	@NewNumOfPlaces int
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @ConferenceID int
		SET @ConferenceID = dbo.idOfConferenceIncludingDate(@ConferenceName,@Date)

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		UPDATE Workshops
		SET NumOfPlaces = @NewNumOfPlaces where DayID = @DayID and WorkshopName =
		@WorkshopName and StartTime = @StartTime
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot change NumberOfWorkshopsPlaces. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura dodająca płatność na podany dzień konferencji. Jako argumenty przyjmuje nazwę klienta, nazwę konferencji i datę dnia. W przypadku nieznalezienia klienta, konferencji, dnia konferencji lub rezerwacji na dzień procedura rzuca odpowiedni wyjątek. Nie można również dodać płatności, gdy taka płatność już istnieje. Kwota płatności obliczana jest przez funkcje osobną. Kod tworzący procedurę:

```
CREATE PROCEDURE AddPaymentForDay
	@CustomerName nvarchar(64),
	@ConferenceName nvarchar(200),
	@Date date
	AS
	BEGIN
		SET NOCOUNT ON;
		BEGIN TRY
		DECLARE @CustomerID int
		SET @CustomerID = dbo.idOfCustomer(@CustomerName)
		IF @CustomerID is null
		BEGIN RAISERROR('Nie znaleziono klienta',16,4) END

		DECLARE @ConferenceID int
		SET @ConferenceID = dbo.idOfConferenceIncludingDate(@ConferenceName,@Date)
		IF @ConferenceID is null
		BEGIN RAISERROR('Nie znaleziono Konferencji',16,9) END

		DECLARE @DayID int
		SET @DayID = dbo.idOfDay(@ConferenceName,@Date)
		IF @DayID is null
		BEGIN RAISERROR('Nie znaleziono dnia konferencji',16,5) END

		DECLARE @DaysReservationID int
		SET @DaysReservationID = dbo.idOfDaysReservation(@CustomerID,@DayID)
		IF @DaysReservationID is null
		BEGIN RAISERROR('Nie znaleziono rezerwacji na podany dzień konferencji',16,5) END

		IF exists ( select * from Payments where DaysReservationID = @DaysReservationID )
		BEGIN RAISERROR('Płatność na podany dzień konferencji już istneieje',16,5) END

		DECLARE @Price money
		SET @Price = dbo.countPriceOfDayResrvation(@DaysReservationID)

		INSERT INTO Payments(DaysReservationID,value,Date)
		VALUES (@DaysReservationID,@Price,getdate())
		END TRY
		BEGIN CATCH
		DECLARE @ErrorMsg nvarchar (2048)
			= 'Cannot add PaymentForDay. Error message : '
				+ ERROR_MESSAGE () ;
		; THROW 50003 , @ErrorMsg ,1
		END CATCH
	END
GO
```

Procedura dodająca płatności za konferencji. Jako argumenty przyjmuje nazwę klienta, nazwę konferencji i datę jej rozpoczęcia. W przypadku nieznalezienia klienta, konferencji, dnia konferencji procedura rzuca odpowiedni wyjątek. Funkcja umożliwia klientowi opłacenie wszystkich dni danej konferencji, na które posiada rezerwację, również w przypadku gdy są to wybrane dni konferencji. Kod tworzący procedurę:

```
CREATE PROCEDURE AddPaymentForConference
	@CustomerName nvarchar(64),
	@ConferenceName nvarchar(200),
	@StartDate date
	AS
	BEGIN
		SET NOCOUNT ON;
		DECLARE @EndDate date
		SET @EndDate = ( select EndDate from Conferences where ConferenceID =
		dbo.idOfConferenceIncludingDate(@ConferenceName,@StartDate))
		DECLARE @Date date
		SET @Date = @StartDate
		BEGIN TRANSACTION
			WHILE (@Date <= @EndDate)
			BEGIN
				BEGIN TRY
					exec AddPaymentForDay @CustomerName, @ConferenceName, @Date
					SET @Date = dateadd(day,1,@Date)
				END TRY
				BEGIN CATCH
					IF ERROR_MESSAGE() <> 'Cannot add PaymentForDay. Error message
					: Nie znaleziono rezerwacji na podany dzień konferencji'
					BEGIN
						DECLARE @ErrorMsg nvarchar (2048)
						= 'Cannot add PaymentForConference. Error message : '
						+ ERROR_MESSAGE () ;
						ROLLBACK TRANSACTION;
						THROW 50003 , @ErrorMsg ,1
					END
					ELSE
					BEGIN
						SET @Date = dateadd(day,1,@Date)
					END
				END CATCH
			END
		COMMIT TRANSACTION
	END
GO
```

8. Widoki

Lista uczestników warsztatów - *workshops_participants*
Widok zwraca następujące pola: ID_Warsztatu *(WorkshopID),* Nazwę Warsztatu *(WorkshopName),* Datę Warsztatu *(Date),* Imię (*FirstName),* Nazwisko(*Las Name)* klienta.
Kod tworzący widok:

```
CREATE VIEW workshops_participants AS
SELECT wr.WorkshopID, WorkshopName, Date, FirstName, LastName,p.ParticipantID
FROM WorkshopsReservations AS wr
JOIN Workshops AS w ON wr.WorkshopID = w.WorkshopID
JOIN Days AS d ON d.DayID=w.DayID
JOIN Participants AS p ON p.ParticipantID = wr.ParticipantID
GO
```

Lista uczestników konferencji - *Conferences_participants*
Widok zwraca następujące pola: Nazwa konferencji *(ConferenceName),* Datę rozpoczęcia *(StartDate),* Datę Zakończenia *(EndDate),* Imię (*FirstName),* Nazwisko(*LastName)* klienta.
Kod tworzący widok:

```
CREATE VIEW Conferences_participants AS
SELECT DISTINCT ConferenceName, StartDate, EndDate, FirstName, LastName
FROM DaysReservations as dr
JOIN Days AS d on d.DayID=dr.DayID
JOIN Conferences as c on d.ConferenceID = c.ConferenceID
JOIN Participants as p on p.DaysReservationID=dr.DaysReservationID
GO
```

Lista rezerwacji dokonana przez klienta - *Count_Reservation*
Widok zawiera następujące pola: ID Klienta *(CustomerID)*, Nazwa klienta (*Name)*, Liczba zarezerwowanych miejsc (*How),* gdzie liczba rezerwacji jest sumaryczną liczbą wszystkich zarezerwowanych miejsc. Kod tworzący widok:

```
CREATE VIEW Count_Reservation AS
select c.CustomerID, Name,      SUM (dr.NumOfNormalParticipants+NumOfStudents) as How
FROM Customers as c
JOIN DaysReservations as dr on dr.CustomerID=c.CustomerID
GROUP BY c.CustomerID, Name
GO
```

Lista nieopłaconych rezerwacji – *not_pay*
Widok zawiera następujące pola: Numer Klienta *(CustomerID)* Nazwa klienta (*Name)* klientów, którzy nie dokonali opłaty za rezerwacje. Kod tworzący widok:

```
CREATE VIEW not_pay AS
select c.CustomerID, Name
FROM Customers as c
JOIN DaysReservations as dr on dr.CustomerID=c.CustomerID
where not DaysReservationID in (select DaysReservationID from Payments)
GO
```

Historia Transakcji - *Payments_hist*
Widok zawiera następujące pola: Kolejny numer transakcji (PaymentID), Nazwę klienta*(Name)*, nazwę konferencji*(ConferenceName)*, datę dnia konferencji*(d.Date),* Kwota *(value),* Data *(p.Date)*. Kod tworzący widok:

```
CREATE VIEW Payments_hist AS
SELECT PaymentID, c.Name, ConferenceName, d.Date, value, p.Date as 'Date of payment'
FROM Payments as p
join DaysReservations as dr on p.DaysReservationID = dr.DaysReservationID
join Customers as c on c.CustomerID = dr.CustomerID
join Days as d on d.DayID = dr.DayID
join Conferences as con on d.ConferenceID = con.ConferenceID
GO
```

Lista warsztatów z podziałem na dni - *Workshops_list*
Widok zawiera następujące pola: Data *(Date),* Nazwa Konferencji (*ConferenceName*), Nazwa warsztatu *(WorkshopName),* godzina rozpoczęcia *(StartTime),* godzina zakończenia *(EndTime),* cena warsztatu *(Price),* liczba miejsc *(NumOfPlaces).* Kod tworzący widok:

```
CREATE VIEW Workshops_list AS
SELECT Date, ConferenceName, WorkshopName, StartTime, EndTime, w.Price, w.NumOfPlaces
FROM Workshops as w
JOIN Days as d ON d.DayID=w.DayID
JOIN Conferences as c ON c.ConferenceID=d.ConferenceID
GO
```

Dane klienta - *Customers_view*
Widok zawiera następujące pola: Nazwę klienta (*Name),* Kraj *(Country*), Miasto (*City*), Ulicę (*Street),* Numer kontaktowy (*Phone*), Email, NIP, REGON. Kod tworzący widok:

```
CREATE VIEW Customers_view AS
SELECT Name,Country, City, Street, Phone, Email, NIP, REGON
FROM Customers as c
LEFT JOIN Firms as f on c.CustomerID=f.CustomerID
GO
```

Dane uczestnika - *Participants_view*
Widok zawiera następujące pola: *Zarejestrowany przez*, Imię (*FirstName*), Nazwisko (*LastName*), Nr legitymacji (*StudentCardName)*. Kod tworzący widok:

```
CREATE VIEW Participants_view AS
SELECT distinct Name as 'Zarejestrowany przez', FirstName,LastName, StudentCardNr
FROM Participants as p
JOIN Customers as c ON c.CustomerID = p.CustomerID
GO
```

Najczęściej wybierane warsztaty - *Workshops_top*
Widok wyświetla 20 najczęściej wybieranych warsztatów wraz z liczbą uczestników (sumuje wszystkie wystąpienia danego warsztatu) tzn., jeżeli warsztat jest cykliczny zostanie wyświetlony tylko raz. Kod tworzący widok:

```
CREATE VIEW Workshops_top AS
SELECT top 20 WorkShopName, count (WorkShopsReservationID) as ile
FROM Workshops as w
```

```
JOIN WorkshopsReservations as wr on wr.WorkshopID=w.WorkshopID
GROUP BY WorkshopName
ORDER BY ile DESC
GO
```

Najczęściej wybierane warsztaty - *Workshops_top*

Widok wyświetla 20 konferencji o największej liczbie uczestników. Podaje Nazwę konferencji, Datę rozpoczęcia, ilość uczestników. Kod tworzący widok:

```
CREATE VIEW Conference_top AS
SELECT top 20 ConferenceName, StartDate, count (DaysReservationID) as ile
FROM Conferences as c
JOIN Days as d on d.ConferenceID=c.ConferenceID
JOIN DaysReservations as dr on d.DayID=dr.DayID

GROUP BY StartDate,ConferenceName
ORDER BY ile DESC
GO
```

Widok wyświetla informacje, o warsztatach które jeszcze się nie odbyły, a na które jeszcze są wolne miejsca. Kod tworzący widok:

```
CREATE VIEW Available_workshops AS
select WorkshopName, w.StartTime, d.Date, w.EndTime, w.Price, c.ConferenceName
from Workshops as w
join Days as d on d.DayID = w.DayID
join Conferences as c on c.ConferenceID = d.DayID
where w.NumOfPlaces > ( select count(*) from WorkshopsReservations where WorkshopID =
w.WorkshopID) and d.Date > getdate()
GO
```

9. Role i uprawnienia użytkowników

Propozycja uprawnień i dostępu dla grup użytkowników:

9.1 **Organizator** – zajmuje się obsługą rezerwacji i płatności, dodawaniem nowych konferencji i warsztatów. Jest również odpowiedzialna za kontakt z klientem np. w celu uzupełnienia listy uczestników dnia konferencji. Posiada dostęp do wszystkich widoków, procedur i funkcji bazy danych.

9.2 **Klient indywidualny/Firma** – może składać i anulować rezerwacje na dni konferencji oraz na warsztaty i dokonywać płatności za nie. Może również uzupełniać listę uczestników oraz przydzielać ich do warsztatów. Posiada dostęp do:

procedur:

- AddWorkshopReservation
- CancelWorkshopReservation
- AddDaysReservation
- CancelDayReservation
- AddParticipantToWorkshopReservation

- DeleteParticipantFromWorkshopReservation
- AddParticipant
- AddPaymentForDay
- AddPaymentForConference

funkcji:

- countPriceForDayReservation

widoków:

- Available_workshops

9.3 **Uczestnik** – może składać i anulować rezerwacje na warsztaty. Może również zapisać się na warsztaty Posiada dostęp do:

procedur:

- AddWorkshopReservation
- CancelWorkshopReservation
- AddParticipantToWorkshopReservation
- DeleteParticipantFromWorkshopReservation

widoków:

- Available_workshops

10. **Generator Danych**

Generator działa na zasadzie tworzenia wirtualnej struktury bazy danych. Tabele są uzupełniane w następującej kolejności:

1. Customers
2. Firms
3. Confereces
4. Days
5. Workshops
6. DaysReservation
7. Participants

Generator tworzy losowe zmienne bazując na plikach tekstowych zawierających dane wejściowe (lista: imion - 290, nazwisk - 14500, miast - 1700, ulic - 680, dat - 157, tematów - 35, końcówek przyrostków warsztatów - 7) Tworząc i zapisując do pliku skrypt SQL pozwalający w odpowiedniej kolejności wypełnić tabelę. Niestety ze względu na zmiany

struktury bazy danych, na kilka dni przed terminem oddania generatora nie udało się dokończyć, generując 7 tabelę pojawia się błąd wykonywania programu.