Baza danych “Konferencje”

Dokumentacja

Grzegorz K, Szymon K

Spis treści

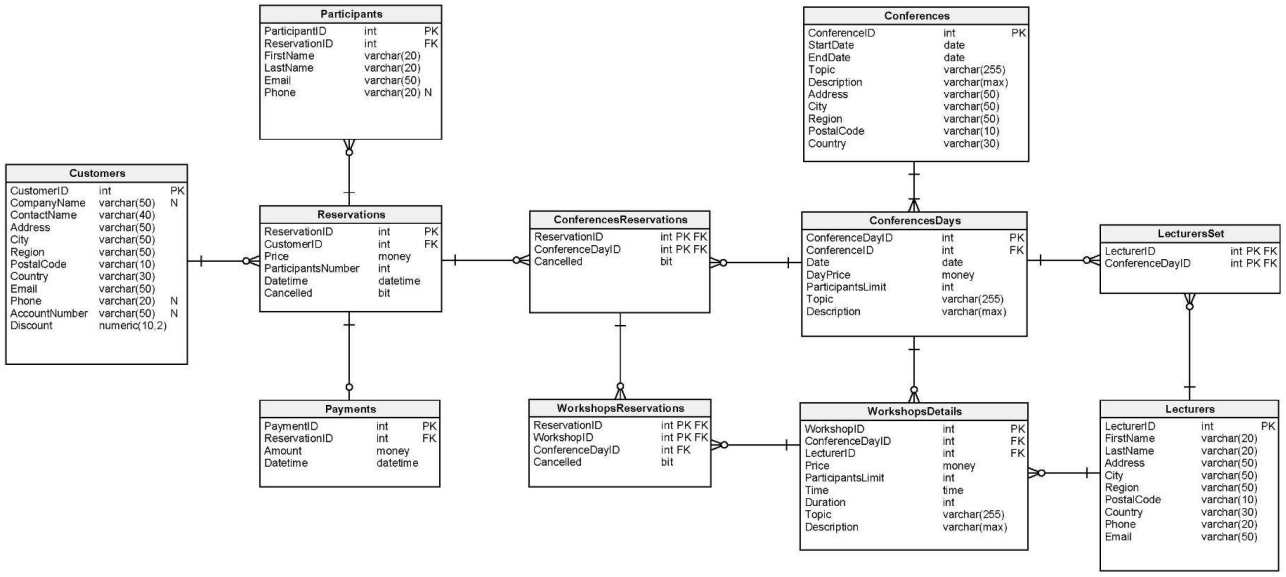
1. Opis bazy danych

Nazwa modelu: Konferencje

Wersja: 1.0

Baza danych: Microsoft SQL Server

Opis:
Baza danych wspomagająca system zapisów na konferencję. Firmy zgłaszają pracowników na konferencje i warsztaty w ich ramach. Klient dokonuje rezerwacji miejsca dla określonej liczby osób, jeżeli jest to możliwe - czyli jeżeli miejsce na dany dzień konferencji/warsztat nie skończyło się. Po rezerwacji należy dokonać płatności na 7 dni przed terminem wydarzenia oraz podać listę uczestników na 14 dni przed terminem wydarzenia. Można zakupić bilet na każdy dzień konferencji oraz opcjonalnie na warsztaty w ich ramach. Nie można kupić biletu tylko na warsztaty, ale można tylko na konferencję.



Schemat bazy danych, dostępny również w załączonym do projektu pliku PDF

2. Tabele

2.1. Tabela Customers

Opis: Tabela z klientami, którzy dokonują rezerwacji miejsca na wydarzeniach.

2.1.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
CustomerID	int	PK	Unikalny identyfikator klienta, generowany automatycznie
CompanyName	varchar(50)	null	Nazwa firmy (może mieć wartość NULL jeżeli klientem jest osoba prywatna)
ContactName	varchar(40)		Imię i nazwisko reprezentanta firmy (lub osoby prywatnej jeżeli nie jest to firma)
Address	varchar(50)		Adres
City	varchar(50)		Miasto
Region	varchar(50)		Region/województwo itp.
PostalCode	varchar(10)		Kod pocztowy
Country	varchar(30)		Kraj
Email	varchar(50)		Adres e mail do kontaktu w razie problemów z rejestracją
Phone	varchar(20)	null	Opcjonalnie numer telefonu do kontaktu
AccountNumber	varchar(50)	null	Numer konta (nie musi być podany) w przypadku jakichkolwiek zwrotów - na przykład z powodu anulowania przez niepodanie listy uczestników
Discount	numeric(10,2)	default = 0	Zniżka dla stałych klientów

2.1.2. DDL

```
CREATE TABLE Customers (
   CustomerID int  NOT NULL IDENTITY(1, 1),
   CompanyName varchar(50)  NULL,
   ContactName varchar(40)  NOT NULL,
   Address varchar(50)  NOT NULL,
   City varchar(50)  NOT NULL,
   Region varchar(50)  NOT NULL,
   PostalCode varchar(10)  NOT NULL,
   Country varchar(30)  NOT NULL,
   Phone varchar(20)  NULL,
   Email varchar(50) NOT NULL CHECK (Email LIKE '%@%.%'),
   AccountNumber varchar(50)  NULL,
   Discount numeric(10,2)  NOT NULL DEFAULT 0 CHECK (Discount >=
0 AND Discount <= 0.25),
   CONSTRAINT Customers_pk PRIMARY KEY  (CustomerID)
)
```

2.1.3. Warunki integralnościowe

1. Zniżka nie może być większa od 0.25, ani mniejsza od 0.
2. Email musi być postaci [...]@[...].[...]

2.2. Tabela Participants

Opis: Tabela z uczestnikami konferencji i warsztatów. Każdy uczestnik ma unikalny numer

2.2.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
ParticipantID	int	PK	Unikalny numer uczestnika. Jeżeli dany uczestnik będzie uczestniczył w dwóch konferencjach, pojawi się w tabeli dwa razy, ale z innym ParticipantID
ReservationID	int		Numer rezerwacji, w której pracownik został wyszczególniony
FirstName	varchar(20)		Imię
LastName	varchar(20)		Nazwisko
Email	varchar(50)		Adres e-mail służący do kontaktu i przesłania biletu na wydarzenie.
Phone	varchar(20)	null	Opcjonalny numer telefonu do uczestnika

2.2.2. DDL

```
CREATE TABLE Participants (
   ParticipantID int  NOT NULL IDENTITY(1, 1),
   ReservationID int  NOT NULL,
   FirstName varchar(20)  NOT NULL,
   LastName varchar(20)  NOT NULL,
   Email varchar(50)  NOT NULL CHECK (Email LIKE '%@%.%'),
   Phone varchar(20)  NULL,
   CONSTRAINT Participants_pk PRIMARY KEY  (ParticipantID),
   FOREIGN KEY (ReservationID) REFERENCES Reservations
(ReservationID)
)
```

2.2.3. Warunki integralnościowe

1. Email musi być postaci [...]@[...].[...]

2.3. Tabela Reservations

Opis: Tabela rezerwacji miejsca przez klientów.

2.3.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
ReservationID	int	PK	Numer rezerwacji
CustomerID	int		ID klienta
Price	money		Cena wyliczona na podstawie liczby uczestników oraz ceny za poszczególne dni konferencji i warsztaty w ich ramach.
ParticipantsNumber	int		Liczba zgłoszonych uczestników
Datetime	datetime	default = GETDATE()	Data złożenia rezerwacji
Cancelled	bit		Atrybut informujący czy rezerwacja została anulowana

2.3.2. DDL

```
CREATE TABLE Reservations (
   ReservationID int  NOT NULL IDENTITY(1, 1),
   CustomerID int  NOT NULL,
   Price money  NOT NULL DEFAULT 0 CHECK (Price >= 0),
   ParticipantsNumber int  NOT NULL CHECK (ParticipantsNumber >
0),
   Datetime datetime DEFAULT GETDATE() NOT NULL,
   Cancelled bit  NOT NULL DEFAULT 0,
   CONSTRAINT Reservations_pk PRIMARY KEY  (ReservationID),
   FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID)
)
```

2.3.3. Warunki integralnościowe

1. Cena większa, bądź równa 0
2. Liczba zadeklarowanych uczestników większa od 0

2.4. Tabela Payments

Opis: Tabela z płatnościami. Płatność za rezerwację musi zostać dokonana w jednym przelewie.

2.4.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
PaymentID	int	PK	ID płatności
ReservationID	int		Numer rezerwacji
Amount	money		Kwota przelewu
Datetime	datetime		Dokładny czas przelewu

2.4.2. DDL

```
CREATE TABLE Payments (
   PaymentID int  NOT NULL IDENTITY(1, 1),
   ReservationID int  NOT NULL,
   Amount money  NOT NULL CHECK (Amount > 0 ),
   Datetime datetime DEFAULT GETDATE() NOT NULL,
   CONSTRAINT Payments_pk PRIMARY KEY  (PaymentID),
   FOREIGN KEY (ReservationID) REFERENCES Reservations
(ReservationID)
)
```

2.4.3. Warunki integralnościowe

1. Wartość przelewu większa od 0

2.5. Tabela Conferences

Opis: Tabela z konferencjami (wielodniowymi). Każda konferencja ma unikalny numer ConferenceID. W tej tabeli przechowywane są informacje o każdej planowanej, odbywającej się oraz przeszłej konferencji. Do każdej konferencji przypisane są dane adresowe, czyli miejsce, gdzie konferencja się odbywa.

2.5.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
ConferenceID	int	PK	Unikalny numer konferencji
StartDate	date		Data rozpoczęcia wydarzenia
EndDate	date		Data zakończenia wydarzenia
Topic	varchar(255)		Ogólny temat konferencji
Opis	varchar(max)		Ogólny opis konferencji
Address	varchar(50)		Adres, pod którym odbywa się konferencja
City	varchar(50)		Miasto
Region	varchar(50)		Region/województwo
PostalCode	varchar(10)		Kod pocztowy
Country	varchar(30)		Państwo

2.5.2. DDL

```
CREATE TABLE Conferences (
   ConferenceID int  NOT NULL IDENTITY(1, 1),
   StartDate date  NOT NULL,
   EndDate date  NOT NULL,
   Topic varchar(255)  NOT NULL,
   Description varchar(max)  NOT NULL,
   Address varchar(50)  NOT NULL,
   City varchar(50)  NOT NULL,
   Region varchar(50)  NOT NULL,
   PostalCode varchar(10)  NOT NULL,
   Country varchar(30)  NOT NULL,
   CONSTRAINT StartDateBeforeEndDate CHECK (StartDate <= EndDate),
   CONSTRAINT Conferences_pk PRIMARY KEY  (ConferenceID))
```

2.5.3. Warunki integralnościowe

1. Data rozpoczęcia mniejsza bądź równa dacie zakończenia konferencji

2.6. Tabela ConferencesDays

Opis: Tabela zawierająca poszczególne dni konferencji i ich szczegóły. Jeżeli konferencja trwa trzy dni, to w tej tabeli pojawią się trzy rekordy.

2.6.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
ConferenceDayID	int	PK	Nowy klucz, który jednoznacznie wskazuje na konferencję i jej dzień.
ConferenceID	int		Zaimportowany klucz obcy, informujący której konferencji dotyczy ten rekord w bazie.
Date	date		Dzień konferencji (data)
DayPrice	money		Cena za ten dzień konferencji.
ParticipantsLimit	int		Limit miejsca na wybranym dniu konferencji. Ile maksymalnie osób może się na nią zapisać.
Topic	varchar(255)		Temat wybranego dnia konferencji. Jest to już bardziej szczegółowy temat, niż ten, który był w tabeli Conferences.
Opis	varchar(max)		Szczegółowy opis konkretnego dnia konferencji.

2.6.2. DDL

```
CREATE TABLE ConferencesDays (
   ConferenceDayID int  NOT NULL IDENTITY(1, 1),
   ConferenceID int  NOT NULL,
   Date date  NOT NULL,
   DayPrice money  NOT NULL CHECK (DayPrice >= 0),
   ParticipantsLimit int  NOT NULL CHECK (ParticipantsLimit > 0),
   Topic varchar(255)  NOT NULL,
   Description varchar(max)  NOT NULL,
   CONSTRAINT ConferencesDetails_pk PRIMARY KEY  (ConferenceDayID),
   FOREIGN KEY (ConferenceID) REFERENCES Conferences (ConferenceID)
)
```

2.6.3. Warunki integralnościowe

1. Cena za dzień konferencji większa bądź równa 0
2. Limit uczestników większy od 0

2.7. Tabela Lecturers

Opis: Tabela z danymi prowadzących konferencje i warsztaty

2.7.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
LecturerID	int	PK	Numer identyfikujący prowadzącego
FirstName	varchar(20)		Imię
LastName	varchar(20)		Nazwisko
Address	varchar(50)		Adres
City	varchar(50)		Miasto
Region	varchar(50)		Region/województwo
PostalCode	varchar(10)		Kod pocztowy
Country	varchar(30)		Państwo
Phone	varchar(20)		Numer telefonu
Email	varchar(50)		Email

2.7.2. DDL

```
CREATE TABLE Lecturers (
   LecturerID int  NOT NULL IDENTITY(1, 1),
   FirstName varchar(20)  NOT NULL,
   LastName varchar(20)  NOT NULL,
   Address varchar(50)  NOT NULL,
   City varchar(50)  NOT NULL,
   Region varchar(50)  NOT NULL,
   PostalCode varchar(10)  NOT NULL,
   Country varchar(30)  NOT NULL,
   Phone varchar(20)  NOT NULL,
   Email varchar(50)  NOT NULL CHECK (Email LIKE '%@%.%'),
   CONSTRAINT Lecturers_pk PRIMARY KEY  (LecturerID)
)
```

2.7.3. Warunki integralnościowe

1. Email musi być postaci [...]@[...].[...]

2.8. Tabela WorkshopsDetails

Opis: Warsztaty na dany dzień konferencji.

2.8.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
WorkshopID	int	PK	Numer identyfikujący warsztat
LecturerID	int		Osoba prowadząca warsztat
Price	money	default = 0	Cena za warsztat.
ParticipantsLimit	int		Limit uczestników, którzy mogą brać udział w warsztacie
Time	time		Godzina rozpoczęcia warsztatu
Duration	int		Czas trwania warsztatu (w minutach)
Topic	varchar(255)		Temat warsztatu
Opis	varchar(max)		Opis warsztatu

2.8.2. DDL

```
CREATE TABLE WorkshopsDetails (
   WorkshopID int  NOT NULL IDENTITY(1, 1),
   ConferenceDayID int  NOT NULL,
   LecturerID int  NOT NULL,
   Price money  NOT NULL DEFAULT 0 CHECK (Price >= 0),
   ParticipantsLimit int  NOT NULL CHECK (ParticipantsLimit > 0),
   Time time  NOT NULL,
   Duration int  NOT NULL CHECK (Duration <= 1440 AND Duration > 0),
   Topic varchar(255)  NOT NULL,
   Description varchar(max)  NOT NULL,
   CONSTRAINT WorkshopsDetails_pk PRIMARY KEY  (WorkshopID),
   FOREIGN KEY (ConferenceDayID) REFERENCES ConferencesDays
(ConferenceDayID),
   FOREIGN KEY (LecturerID) REFERENCES Lecturers (LecturerID)
)
```

2.8.3. Warunki integralnościowe

1. Cena za warsztat większa bądź równa 0
2. Limit uczestników większy od 0
3. Czas trwania (w minutach) większy od 0 i mniejszy od 1440 (24h)

2.9. Tabela ConferencesReservations

Opis: Tabela ze szczegółami rezerwacji na dany dzień konferencji. Dwa klucze główne jednoznacznie wskazują na wybrany dzień konferencji i rezerwację. Atrybut Cancelled informuje nas, czy rezerwacja na dany dzień konferencji została anulowana czy nie.

2.9.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
ReservationID	int	PK	Numer rezerwacji
ConferenceDayID	int	PK	ID dnia konferencji
Cancelled	bit	default = 0	Informacja, czy rezerwacja została anulowana

2.9.2. Indeksy

Nazwa indeksu	Kolumny	Opis
ConferencesReservations_idx_1	ReservationID (ASC), ConferenceDayID (ASC)	Numer rezerwacji i numer identyfikujący dzień konferencji jednoznacznie identyfikują rekord w tabeli I taki zestaw nie może się powtórzyć (nie można zarezerwować dwa razy tego samego w ramach jednej rezerwacji)

2.9.3. DDL

```
CREATE TABLE ConferencesReservations (
   ReservationID int  NOT NULL,
   ConferenceDayID int  NOT NULL,
   Cancelled bit  NOT NULL DEFAULT 0,
   CONSTRAINT ConferencesReservations_pk PRIMARY KEY
(ReservationID,ConferenceDayID),
   FOREIGN KEY (ReservationID) REFERENCES Reservations (ReservationID),
   FOREIGN KEY (ConferenceDayID) REFERENCES ConferencesDays
(ConferenceDayID),
   CONSTRAINT ParticipantsNumber CHECK
(dbo.ParticipantsNumberPossible(ReservationID, ConferenceDayID) >= 0));
CREATE UNIQUE INDEX ConferencesReservations_idx_1 on
ConferencesReservations (ReservationID ASC,ConferenceDayID ASC);
```

2.9.4. Warunki integralnościowe

1. Możliwe dodanie rezerwacji z taką liczbą uczestników (funkcja ParticipantsNumberPossible)

2.10. Tabela WorkshopsReservations

Opis: Tabela ze szczegółami rezerwacji miejsca na warsztaty. Trzy klucze główne jednoznacznie wskazują na dzień konferencji, warsztat i numer rezerwacji. Jeżeli rezerwacja została anulowana, atrybut Cancelled przyjmie wartość 1.

2.10.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
ReservationID	int	PK	Numer rezerwacji
WorkshopID	int	PK	ID warsztatu
ConferenceDayID	int		ID dnia konferencji
Cancelled	bit	default = 0	Informacja czy rezerwacja została anulowana

2.9.2. Indeksy

Nazwa indeksu	Kolumny	Opis
WorkshopsReservations_idx_1	ReservationID (ASC), WorkshopID(ASC)	Numer rezerwacji i numer warsztatu jednoznacznie identyfikują rekord w tabeli I taki zestaw nie może się powtórzyć (nie można zarezerwować dwa razy tego samego w ramach jednej rezerwacji)

2.10.3. DDL

```
CREATE TABLE WorkshopsReservations (
   ReservationID int  NOT NULL,
   WorkshopID int  NOT NULL,
   ConferenceDayID int  NOT NULL,
   Cancelled bit  NOT NULL DEFAULT 0,
   CONSTRAINT WorkshopsReservations_pk PRIMARY KEY  (ReservationID,
WorkshopID),
   FOREIGN KEY (ReservationID,ConferenceDayID) REFERENCES
ConferencesReservations (ReservationID, ConferenceDayID),
   FOREIGN KEY (WorkshopID) REFERENCES WorkshopsDetails
(WorkshopID),
   CONSTRAINT NoReservationOnThisDay CHECK
(dbo.ReservationDayExists(ReservationID,
       ConferenceDayID) = 1),
   CONSTRAINT WorkShopPartLimitExceeded CHECK
```

```
(dbo.PartNumWorkshopPossible(ReservationID,
       ConferenceDayID, WorkshopID) >= 0)
);
CREATE UNIQUE INDEX WorkshopsReservations_idx_1 on
WorkshopsReservations (ReservationID ASC,WorkshopID ASC);
```

2.10.3. Warunki integralnościowe

1. Rezerwacja musi istnieć w tabeli ConferencesReservations (trzeba być zarejestrowanym na dany dzień konferencji, żeby móc uczestniczyć w warsztacie) - funkcja ReservationDayExists
2. Możliwe jest dodanie rezerwacji z taką liczbą uczestników - funkcja PartNumWorkshopPossible

2.11. Tabela LecturersSet

Opis: Tabela przechodnia między ConferencesDays i Lecturers, aby umożliwić sytuację, że wiele prowadzących prowadzi wiele konferencji w danym dniu.

2.11.1. Kolumny

Nazwa kolumny	Typ danych	Właściwości	Opis
LecturerID	int	PK	ID prelegenta
ConferenceDayID	int	PK	ID dnia konferencji

2.11.2. Indeksy

Nazwa indeksu	Kolumny	Opis
LecturersSet_idx_1	LecturerID (ASC), ConferenceDayID (ASC)	Prowadzący może prowadzić konferencję i taki zestaw nie może się powtórzyć, jeżeli już jest w tabeli

2.11.3. DDL

```
CREATE TABLE LecturersSet (
   LecturerID int  NOT NULL,
   ConferenceDayID int  NOT NULL,
   CONSTRAINT LecturersSet_pk PRIMARY KEY
(LecturerID,ConferenceDayID),
   FOREIGN KEY (ConferenceDayID) REFERENCES ConferencesDays
(ConferenceDayID),
   FOREIGN KEY (LecturerID) REFERENCES Lecturers (LecturerID)
);
CREATE UNIQUE INDEX LecturersSet_idx_1 on LecturersSet (LecturerID
ASC,ConferenceDayID ASC);
```

3. Relacje

3.1. Relacja Reservations_Customers

Opis: Klient może złożyć wiele rezerwacji.

Customers	0..*	Reservations
CustomerID	<->	CustomerID

3.2. Relacja ConferencesDetails_Conferences

Opis: Konferencja może trwać wiele dni, ale przynajmniej jeden

Conferences	1..*	ConferencesDays
ConferenceID	<->	ConferenceID

3.3. Relacja WorkshopsDetails_Lecturers

Opis: Prowadzący może prowadzić wiele warsztatów, ale konkretny warsztat jest prowadzony przez jednego prowadzącego.

Lecturers	0..*	WorkshopsDetails
LecturerID	<->	LecturerID

3.4. Relacja Payments_Reservations

Opis: Rezerwacja musi zostać opłacona w jednym przelewie, ale nie musi być to zrobione w momencie rezerwowania, stąd relacja 0 do 1

Reservations	0..1	Payments
ReservationID	<->	ReservationID

3.5. Relacja ConferencesReservations_Reservations

Opis: Rezerwacja może dotyczyć wielu dni konferencji.

Reservations	0..*	ConferencesReservations
ReservationID	<->	ReservationID

3.6. Relacja Participants_Reservations

Opis: Wielu pracowników może być przypisanych do tej samej rezerwacji.

Reservations	**0..***	**Participants**
ReservationID	<->	ReservationID

3.7. Relacja ConferencesReservations_ConferencesDetails

Opis: Dzień konferencji może być przedmiotem wielu, różnych rezerwacji.

ConferencesDays	**0..***	**ConferencesReservations**
ConferenceDayID	<->	ConferenceDayID

3.8. Relacja WorkshopsDetails_ConferencesDetails

Opis: W konkretnym dniu konferencji może być wiele warsztatów, ale może też nie być żadnego.

ConferencesDays	**0..***	**WorkshopsDetails**
ConferenceDayID	<->	ConferenceDayID

3.9. Relacja WorkshopsReservations_ConferencesReservations

Opis: W ciągu jednego dnia konferencji uczestnicy mogą brać udział w wielu warsztatach (w ramach rezerwacji)

ConferencesReservations	**0..***	**WorkshopsReservations**
ReservationID	<->	ReservationID
ConferenceDayID	<->	ConferenceDayID

3.10. Relacja WorkshopsReservations_WorkshopsDetails

Opis: Konkretny warsztat może być wyszczególniony w wielu rezerwacjach.

WorkshopsDetails	**0..***	**WorkshopsReservations**
WorkshopID	<->	WorkshopID

3.11. Relacja LecturersSet_Lecturers

Opis: Prowadzący może prowadzić wiele konferencji

Lecturers	**0..***	**LecturersSet**
LecturerID	<->	LecturerID

3.12. Relacja LecturersSet_ConferencesDays

Opis: Konferencję może prowadzić wiele prowadzących

ConferencesDays	**0..***	**LecturersSet**
ConferenceDayID	<->	ConferenceDayID

4. Funkcje

4.1. ParticipantsNumberPossible

Opis: Funkcja sprawdza czy jest możliwe zarejestrowanie wskazanej w rezerwacji liczby uczestników. Oblicza liczbę wolnych miejsc po potencjalnym zarezerwowaniu, jeżeli ta liczba jest mniejsza od 0, to nie da się zarejestrować tylu uczestników, w przeciwnym wypadku da się.

Argument	Opis
@ResID	ID rezerwacji
@ConfDayID	ID dnia konferencji

Miejsce wykorzystania: Warunek integralnościowy w tabeli ConferencesReservations. Podczas rezerwacji funkcja sprawdza czy jest możliwe zarejestrowanie podanej liczby uczestników.

DDL:

```
CREATE FUNCTION dbo.ParticipantsNumberPossible(@ResID INT,
@ConfDayID INT)
RETURNS INT
   BEGIN
       DECLARE @registered INT
       DECLARE @partlimit INT
       DECLARE @declared INT
       SET @declared = (
           SELECT ParticipantsNumber
           FROM Reservations
           WHERE ReservationID = @ResID
           )
       SET @partlimit = (
           SELECT ParticipantsLimit
           FROM ConferencesDays
           WHERE ConferenceDayID = @ConfDayID
           )
       SET @registered = (
           SELECT ISNULL(SUM(ParticipantsNumber), 0)
           FROM Reservations
           JOIN ConferencesReservations CR on
Reservations.ReservationID = CR.ReservationID
           WHERE ConferenceDayID = @ConfDayID AND CR.Cancelled =
0
           )
       RETURN @partlimit - (@registered + @declared)
   end
GO
```

4.2. ReservationDayExists

Opis: Funkcja sprawdzająca czy w ramach podanej rezerwacji jest podany dzień rejestracji. Funkcja zwraca wartość prawda lub fałsz.

Argument	Opis
@ResID	ID rezerwacji
@ConfDayID	ID dnia konferencji

Miejsce wykorzystania: Warunek integralnościowy w tabeli WorkshopsReservations. Podczas dodawania rezerwacji na warsztat funkcja sprawdza czy rezerwacja obejmuje dzień, w którym ten warsztat się odbywa, ponieważ jest to jedno z założeń systemu.

DDL:

```
CREATE FUNCTION dbo.ReservationDayExists (@ResID INT, @ConfDayID
INT)
RETURNS BIT
   BEGIN
       DECLARE @exists BIT
       SET @exists = (
           SELECT 1
           FROM ConferencesReservations
           WHERE ConferenceDayID = @ConfDayID AND ReservationID =
@ResID
           )
       IF @exists IS NOT NULL
           RETURN 1
       RETURN 0
   end
GO
```

4.3. PartNumWorkshopPossible

Opis: Funkcja sprawdza czy jest możliwe zarejestrowanie wskazanej w rezerwacji liczby uczestników na warsztat. Oblicza liczbę wolnych miejsc po potencjalnym zarezerwowaniu, jeżeli ta liczba jest mniejsza od 0, to nie da się zarejestrować tylu uczestników, w przeciwnym wypadku da się.

Argument	Opis
@ResID	ID rezerwacji
@ConfDayID	ID dnia konferencji
@WSID	ID warsztatu

Miejsce wykorzystania: Warunek integralnościowy w tabeli WorkshopsReservations. Podczas rezerwacji funkcja sprawdza czy jest możliwe zarejestrowanie podanej liczby uczestników na wybrany warsztat.

DDL:

```
CREATE FUNCTION dbo.PartNumWorkshopPossible (@ResID INT,
@ConfDayID INT, @WSID INT)
RETURNS INT
   BEGIN
        DECLARE @registered INT
        DECLARE @partlimit INT
        DECLARE @declared INT
        SET @declared = (
            SELECT ParticipantsNumber
            FROM Reservations
            WHERE ReservationID = @ResID
            )
        SET @partlimit = (
            SELECT ParticipantsLimit
            FROM WorkshopsDetails
            WHERE ConferenceDayID = @ConfDayID AND WorkshopID =
@WSID
            )
        SET @registered = (
            SELECT ISNULL(SUM(ParticipantsNumber), 0)
            FROM WorkshopsReservations AS WR
            JOIN ConferencesReservations CR on WR.ReservationID =
CR.ReservationID
            JOIN Reservations R2 on R2.ReservationID =
CR.ReservationID
            WHERE CR.ConferenceDayID = @ConfDayID AND WorkshopID =
@WSID AND WR.Cancelled = 0
            )
        RETURN @partlimit - (@registered + @declared)
   end
GO
```

4.4. CustomerPaid

Opis: Funkcja sprawdzająca czy wskazana rezerwacja została opłacona przez klienta. Wykorzystywany jest w tej funkcji widok PaymentsStatus. Wpłacona musi zostać dokładnie ta kwota, która została obliczona i wpisana do tabeli Reservations.

Argument	Opis
@ResID	ID rezerwacji

Miejsce wykorzystania: Widok ReservationSummary, procedury generujące listy uczestników itp. Wszędzie tam gdzie jest potrzebna informacja czy klient zapłacił, wykorzystuje się tę funkcję.

DDL:

```
CREATE FUNCTION dbo.CustomerPaid (@ResID INT)
RETURNS BIT
   BEGIN
       DECLARE @paid INT
       SET @paid = (
           SELECT (PS.Price - PS.Paid)
           FROM PaymentsStatus AS PS
           WHERE ReservationID = @ResID
           )
       DECLARE @paymentexists INT
       SET @paymentexists = (
           SELECT PaymentStatus
           FROM PaymentsStatus AS PS
           WHERE ReservationID = @ResID
           )
       IF @paymentexists IS NULL RETURN 0
       IF @paid > 0 RETURN 0
       RETURN 1
   end
GO
```

4.5. CustomerSentParticipants

Opis: Funkcja sprawdzająca czy klient podał listę uczestników w ramach podanej rezerwacji. Wykorzystywany jest w tej funkcji widok ParticipantsSent. Podana musi zostać dokładnie taka liczba uczestników jaką zadeklarował klient w rezerwacji.

Argument	Opis
@ResID	ID rezerwacji

Miejsce wykorzystania: Widok ReservationSummary, procedury generujące listy uczestników itp. Wszędzie tam gdzie jest potrzebna informacja czy klient podał listę uczestników, wykorzystuje się tę funkcję.

DDL:

```
CREATE FUNCTION dbo.CustomerSentParticipants (@ResID INT)
RETURNS BIT
   BEGIN
       DECLARE @left INT
       SET @left = (
           SELECT (PS.ParticipantsDeclared - PS.ParticipantsSent)
           FROM ParticipantsSent AS PS
           WHERE ReservationID = @ResID
           )
       IF @left = 0 RETURN 1
       RETURN 0
   end
GO
```

4.6. CountPrice

Opis: Funkcja obliczająca łączną cenę rezerwacji dla podanej liczby uczestników, uwzględniając zniżkę dla stałego klienta.

Argument	Opis
@ResID	ID rezerwacji

Miejsce wykorzystania: Tabela Reservations, po zapisaniu się na wybrane dni konferencji i warsztaty, funkcja jest wywoływana a zwracana wartość jest umieszczana w Price.

DDL:

```
CREATE FUNCTION dbo.CountPrice (@ResID INT)
RETURNS INT
   BEGIN
       DECLARE @discount INT = (
           SELECT DISTINCT Discount FROM Customers
           JOIN Reservations R2 on Customers.CustomerID =
R2.CustomerID
           WHERE ReservationID = @ResID
           )
       DECLARE @partnum INT = (SELECT ParticipantsNumber FROM
Reservations WHERE ReservationID = @ResID)
       DECLARE @price INT
       SET @price = (
           SELECT ISNULL(SUM(PRICE), 0)
           FROM (
               SELECT ReservationID, DayPrice AS PRICE
               FROM ConferencesReservations AS CR
               JOIN ConferencesDays CD on CR.ConferenceDayID =
CD.ConferenceDayID
               WHERE CR.Cancelled = 0
               UNION
               SELECT ReservationID, Price AS PRICE
               FROM WorkshopsReservations AS WR
               JOIN WorkshopsDetails WD on WR.WorkshopID =
WD.WorkshopID and WR.ConferenceDayID = WD.ConferenceDayID
               WHERE WR.Cancelled = 0
                    ) AS PRICES
           WHERE PRICES.ReservationID = @ResID
           GROUP BY ReservationID
       )
       RETURN @price * @partnum * (1 - @discount)
   end
GO
```

5. Procedury

5.1. ParticipantsConfList

Opis: Procedura wyświetlająca listę uczestników na dany dzień konferencji. Argumentem jaki należy przekazać do procedury jest ID dnia konferencji.

Argument	Opis
@ConfDayID	ID dnia konferencji

DDL:

```
CREATE PROCEDURE dbo.ParticipantsConfList @ConfDayID INT
AS
    SELECT FirstName, LastName, Email, CustomerID AS Customer
    FROM ConferencesReservations
    JOIN Reservations R2 on ConferencesReservations.ReservationID
= R2.ReservationID
    JOIN Participants P on R2.ReservationID = P.ReservationID
    WHERE ConferenceDayID = @ConfDayID AND
dbo.CustomerSentParticipants(P.ReservationID) = 1
    AND dbo.CustomerPaid(P.ReservationID) = 1
GO
```

5.2. ParticipantsWorkshopList

Opis: Procedura wyświetlająca listę uczestników na warsztat o podanym ID.

Argument	Opis
@WSID	Workshop ID

DDL:

```
CREATE PROCEDURE dbo.ParticipantsWorkShopList @WSID INT
AS
    SELECT FirstName, LastName, Email, CustomerID AS Customer
    FROM WorkshopsReservations AS WR
    JOIN ConferencesReservations CR on WR.ReservationID =
CR.ReservationID and WR.ConferenceDayID = CR.ConferenceDayID
    JOIN Reservations R2 on CR.ReservationID = R2.ReservationID
    JOIN Participants P on R2.ReservationID = P.ReservationID
    WHERE WR.WorkshopID = @WSID
    AND dbo.CustomerSentParticipants(P.ReservationID) = 1 AND
dbo.CustomerPaid(P.ReservationID) = 1
GO
```

6. Widoki

6.1. TopCustomers

Opis: Widok zwraca listę 5 uczestników, którzy sumarycznie wydali najwięcej na konferencje. W przypadku, gdy wiele elementów z tej listy ma tą samą wartość, zwracane są dodatkowo wszystkie elementy o wartościach takich jak element ostatni.

DDL:

```
CREATE VIEW TopCustomers AS
SELECT TOP 5 WITH TIES ISNULL(CompanyName, ContactName) AS
Customer, SumPrice
FROM Customers AS C
JOIN (
   SELECT CustomerID, SUM(Price) AS SumPrice
   FROM Reservations
   WHERE Cancelled = 0
   GROUP BY CustomerID
   ) AS S ON C.CustomerID = S.CustomerID
ORDER BY SumPrice DESC
go
```

6.2. TopCancelled

Opis: Widok zwraca listę 5 uczestników, którzy najczęściej rezygnowali z konferencji. W przypadku, gdy wiele elementów z tej listy ma tą samą wartość, zwracane są dodatkowo wszystkie elementy o wartościach takich jak element ostatni.

DDL:

```
CREATE VIEW TopCancelled AS
SELECT TOP 5 WITH TIES ISNULL(CompanyName, ContactName) AS
Customer, CancelledNumber AS Cancelled
FROM Customers AS C
JOIN (
   SELECT CustomerID, COUNT(*) AS CancelledNumber
   FROM Reservations
   WHERE Cancelled = 1
   GROUP BY CustomerID
   ) AS S ON C.CustomerID = S.CustomerID
ORDER BY CancelledNumber DESC
go
```

6.3. TopParticipants

Opis: Widok zwraca listę 5 uczestników z największą ilością zapisanych uczestników. W przypadku, gdy wiele elementów z tej listy ma tą samą wartość, zwracane są dodatkowo wszystkie elementy o wartościach takich jak element ostatni.

DDL:

```
CREATE VIEW TopParticipants AS
SELECT TOP 5 WITH TIES ISNULL(CompanyName, ContactName) AS
Customer, AllParticipants
FROM Customers AS C
JOIN (
   SELECT CustomerID, SUM(ParticipantsNumber) AS AllParticipants
   FROM Reservations
   WHERE Cancelled = 0
   GROUP BY CustomerID
   ) AS S ON C.CustomerID = S.CustomerID
ORDER BY AllParticipants DESC
go
```

6.4. TopReservations

Opis: Widok zwraca listę 5 uczestników z największą ilością zarezerwowanych konferencji. W przypadku, gdy wiele elementów z tej listy ma tą samą wartość, zwracane są dodatkowo wszystkie elementy o wartościach takich jak element ostatni.

DDL:

```
CREATE VIEW TopReservations AS
SELECT TOP 5 WITH TIES ISNULL(CompanyName, ContactName) AS
Customer, ReservationsNumber
FROM Customers AS C
JOIN (
   SELECT CustomerID, COUNT(*) AS ReservationsNumber
   FROM Reservations
   WHERE Cancelled = 0
   GROUP BY CustomerID
   ) AS S ON C.CustomerID = S.CustomerID
ORDER BY ReservationsNumber DESC
go
```

6.5. PaymentsStatus

Opis: Widok zwraca listę płatności na każdą konferencję. Null występujący w kolumnie PaymentStatus oznacza, że płatność nie została dokonana.

DDL:

```
CREATE VIEW PaymentsStatus AS
SELECT ISNULL(CompanyName, ContactName) AS Customer,
C.CustomerID, Reservations.ReservationID, Price,
      PaymentID AS PaymentStatus, ISNULL(Amount, 0) AS Paid
FROM Reservations
LEFT JOIN Payments P on Reservations.ReservationID =
P.ReservationID
JOIN Customers C on Reservations.CustomerID = C.CustomerID
WHERE Cancelled = 0
go
```

6.6. ParticipantsSent

Opis: Widok zwraca listę zgłoszonych oraz wysłanych uczestników konferencji dla każdej firmy, która ich zgłosiła

DDL:

```
CREATE VIEW ParticipantsSent AS
SELECT Reservations.ReservationID, ISNULL(CompanyName,
ContactName) AS Customer, C.CustomerID,
      PSENT.ParticipantsSent, ParticipantsNumber AS
ParticipantsDeclared
FROM Reservations
JOIN (
   SELECT Reservations.ReservationID, COUNT(ParticipantID) AS
ParticipantsSent
   FROM Reservations
   LEFT JOIN Participants P on Reservations.ReservationID =
P.ReservationID
   GROUP BY Reservations.ReservationID
   ) AS PSENT ON PSENT.ReservationID = Reservations.ReservationID
JOIN Customers C on Reservations.CustomerID = C.CustomerID
go
```

6.7. ReservationSummary

Opis: Widok przedstawia informację o rezerwacjach, gdzie:

Kolumna	Typ danych	Opis
PartSent	bit	Wysłano listę uczestników
Paid	bit	Opłacono
DaysToConference	int	Pozostało dni do konferencji

DDL:

```
CREATE VIEW ReservationSummary AS
   SELECT R.ReservationID, R.CustomerID,
dbo.CustomerSentParticipants(R.ReservationID) AS PartSent,
      dbo.CustomerPaid(R.ReservationID) AS Paid, DATEDIFF(d,
GETDATE(), StartDate) AS DaysToConference
   FROM Reservations AS R
   JOIN (
        SELECT DISTINCT ReservationID, C.ConferenceID, StartDate
        FROM ConferencesReservations AS CR
        JOIN ConferencesDays AS CD ON CR.ConferenceDayID =
CD.ConferenceDayID
        JOIN Conferences C on CD.ConferenceID = C.ConferenceID
        ) AS CONF ON R.ReservationID = CONF.ReservationID
   JOIN Customers ON Customers.CustomerID = R.CustomerID
   WHERE R.Cancelled = 0 AND DATEDIFF(d, GETDATE(), StartDate)>=0
go
```

7. Triggery

7.1. CancelledSync

Opis: Trigger aktualizuje status rezerwacji konferencji w tabeli ConferencesReservations, oraz warsztatów w tabeli WorkshopReservations, w zależności od statusu w tabeli Reservations

DDL:

```
CREATE TRIGGER CANCELLEDSYNC
ON Reservations AFTER UPDATE AS
   UPDATE ConferencesReservations
   SET Cancelled = 1
   WHERE ReservationID IN (
        SELECT Reservations.ReservationID FROM Reservations
        JOIN ConferencesReservations CR on
Reservations.ReservationID = CR.ReservationID
        WHERE Reservations.Cancelled = 1
        )
   UPDATE WorkshopsReservations
   SET Cancelled = 1
   WHERE ReservationID IN (
        SELECT Reservations.ReservationID FROM Reservations
        JOIN ConferencesReservations CR on
Reservations.ReservationID = CR.ReservationID
        JOIN WorkshopsReservations WR on CR.ReservationID =
WR.ReservationID and CR.ConferenceDayID = WR.ConferenceDayID
        WHERE Reservations.Cancelled = 1
        )
   UPDATE ConferencesReservations
   SET Cancelled = 0
   WHERE ReservationID IN (
        SELECT Reservations.ReservationID FROM Reservations
        JOIN ConferencesReservations CR on
Reservations.ReservationID = CR.ReservationID
        WHERE Reservations.Cancelled = 0
        )
   UPDATE WorkshopsReservations
   SET Cancelled = 0
   WHERE ReservationID IN (
        SELECT Reservations.ReservationID FROM Reservations
        JOIN ConferencesReservations CR on
Reservations.ReservationID = CR.ReservationID
        JOIN WorkshopsReservations WR on CR.ReservationID =
WR.ReservationID and CR.ConferenceDayID = WR.ConferenceDayID
        WHERE Reservations.Cancelled = 0
        )
GO
```

8. Role

Administrator - posiada dostęp do wszystkich tabel, widoków oraz procedur. Może dodawać, usuwać i modyfikować rekordy we wszystkich tabelach. Jako jedyny posiada uprawnienia do usuwania i dodawania tabel do bazy danych. Nadaje uprawnienia innym użytkownikom bazy danych.

Organizator - posiada dostęp do tabel: Conferences, ConferencesDays, WorkshopsDetails, Lecturers, LecturersSet, do widoków oraz procedur. Nie posiada dostępu do innych tabel. W wymienionych tabelach może dodawać, usuwać i modyfikować rekordy.

Pracownik - posiada dostęp do wszystkich tabel, widoków i procedur. Może dodawać, usuwać i modyfikować rekordy, ale nie może modyfikować struktury bazy danych.

Księgowy - posiada dostęp do tabel: Reservations, ConferencesReservations, WorkshopReservations, Payments, Participants oraz Customers oraz do wszystkich widoków. Nie może modyfikować żadnych tabel z wyłączeniem Payments, może wykonywać widoki i polecenia, która nie wpływają na zawartość tabel (SELECT). Może dodawać, usuwać i modyfikować rekordy w tablicy Payments.

Klient - posiada dostęp do tabeli Customers oraz Participants z uprawnieniami INSERT oraz do tabel Conferences, ConferencesDays, WorkshopDetails, Reservations z uprawnieniami SELECT. Fizyczny klient nie ma bezpośredniego dostępu do bazy danych, a przez interfejs klienta w aplikacji lub na stronie internetowej. Klient nie może zobaczyć danych innych klientów oraz pracowników zgłoszonych przez innych klientów, jak również nie swoich rezerwacji.

9. Generator

Do generowania danych wykorzystaliśmy własny skrypt, który łączy się z bazą danych i wypełnia ją podaną ilością rekordów (ile konferencji, klientów itp.). Duża część skryptu opiera się na losowości, żeby nie trzeba było za każdym razem wprowadzać danych np. ile dni ma trwać konferencja. Rekordy są tworzone losowo w oparciu o plik ***data.py***. Generator znajduje się w pliku **generator.py.**

data.py:

```
from random import randint
import datetime

numer_tel = ''.join([str(randint(0, 9)) for _ in range(9)])
numer_konta = ''.join([str(randint(0, 9)) for _ in range(26)])


def randelem(T):
   return T[randint(0, len(T) - 1)]


def random_date(start, end):
   return start + datetime.timedelta(
       seconds=randint(0, int((end - start).total_seconds())),
   )


def gen_customer():
   ad = ADDRESS[randint(0, len(ADDRESS) - 1)]
   address = ad[0]
   city = ad[1]
   region = ad[2]
   postal = ad[3]
   customer = {
       "company": COMPANY[randint(0, len(COMPANY) - 1)],
       "contact_name": FIRSTNAME[randint(0, len(FIRSTNAME) - 1)]
+ " " + LASTNAME[randint(0, len(LASTNAME) - 1)],
       "address": address,
       "city": city,
       "region": region,
       "postal": str(postal),
       "country": COUNTRIES[randint(0, len(COUNTRIES) - 1)],
       "phone": numer_tel,
       "email": randelem(EMAIL_COMPANY),
       "account": numer_konta,
       "discount": str(0)
   }
   return customer
```

```
# tablice z losowymi wartościami

COMPANY = []
EMAIL_PERSONAL = []
EMAIL_COMPANY = []
LASTNAME = []
FIRSTNAME = []
ADDRESS = []
COUNTRIES = []
TOPICS = []
DESCRIPTIONS = []
```

generator.py:

```
import pyodbc
import datetime
from data import *
from random import randint, shuffle

# DEFINIOWANIE PARAMETRÓW POŁĄCZENIA Z SERWEREM

server = [nazwa serwera]
database = [nazwa_bazy danych]
username = [login]
password = [hasło]
cnxn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL
Server};SERVER=' + server + ';DATABASE=' +
                database + ';UID='+ username +';PWD='+
password))
cursor = cnxn.cursor()


def generate_customers(n):
   table = "Customers"
   for i in range(n):

       c = gen_customer()
       company_name = c['company'].replace("'", "''")
       contact_name = c['contact_name'].replace("'", "''")
       address = c['address'].replace("'", "''")
       city = c['city'].replace("'", "''")
       region = c['region'].replace("'", "''")
       postal = c['postal']
       country = c['country'].replace("'", "''")
       phone = c['phone']
       account = c['account']
       email = c['email']
       sql = f'''
           INSERT {database}.dbo.{table} (CompanyName,
ContactName,
           Address, City, Region, PostalCode, Country, Phone,
Email, AccountNumber)
```

```
            VALUES ('{company_name}', '{contact_name}',
'{address}', '{city}', '{region}', '{postal}', '{country}',
            '{phone}', '{email}', '{account}')
            '''
        cursor.execute(sql)
        cursor.commit()
        cursor.execute('SELECT SCOPE_IDENTITY()')
        CID = int(cursor.fetchall()[0][0])
        for _ in range(randint(1, 2)):
            generate_reservation(CID)
    print(f"Successfully generated {n} records in {table}")


def generate_lecturers(n):
    table = "Lecturers"
    for i in range(n):
        firstname = FIRSTNAME[randint(0, len(FIRSTNAME) - 1)]
        lastname = LASTNAME[randint(0, len(LASTNAME) - 1)]
        c = gen_customer()
        address = c['address'].replace("'", "''")
        city = c['city'].replace("'", "''")
        region = c['region'].replace("'", "''")
        postal = c['postal']
        country = c['country'].replace("'", "''")
        phone = c['phone']
        email = EMAIL_PERSONAL[randint(0, len(EMAIL_PERSONAL) -
1)]
        sql = f'''
                INSERT {database}.dbo.{table} (FirstName,
LastName,
                Address, City, Region, PostalCode, Country, Phone,
Email)
                VALUES ('{firstname}', '{lastname}', '{address}',
'{city}', '{region}', '{postal}', '{country}',
                '{phone}', '{email}')
                '''
        cursor.execute(sql)
        cursor.commit()
    print(f"Successfully generated {n} records in {table}")


def generate_participants(n, RID):
    table = "Participants"
    for i in range(n):
        firstname = randelem(FIRSTNAME)
        lastname = randelem(LASTNAME)
        email = randelem(EMAIL_PERSONAL)
        phone = numer_tel
        sql = f'''
                INSERT {database}.dbo.{table} (ReservationID,
FirstName, LastName, Email, Phone)
                VALUES ({RID}, '{firstname}', '{lastname}',
'{email}', '{phone}')
                '''
        cursor.execute(sql)
```

```
        cursor.commit()
    print(f"Successfully generated {n} records in {table}")


def generate_conferences(n):
    table = "Conferences"
    # zakres dat
    start = datetime.date(2021, 6, 1)
    end = datetime.date(2025, 12, 31)
    for i in range(n):
        startdate = random_date(start, end)
        length = randint(0, 2)
        enddate = startdate + datetime.timedelta(days=i)
        topic = TOPICS[randint(0, len(TOPICS) - 1)].replace("'",
"''")
        desc = DESCRIPTIONS[randint(0, len(DESCRIPTIONS) -
1)].replace("'", "''")
        ad = ADDRESS[randint(0, len(ADDRESS) - 1)]
        address = ad[0].replace("'", "''")
        city = ad[1].replace("'", "''")
        region = ad[2].replace("'", "''")
        postal = ad[3]
        country = COUNTRIES[randint(0, len(COUNTRIES) -
1)].replace("'", "''")
        sql = f'''
                INSERT {database}.dbo.{table} (StartDate, EndDate,
Topic, Description, Address, City,
                Region, PostalCode, Country)
                VALUES ('{str(startdate)}', '{str(enddate)}',
'{topic}', '{desc}', '{address}', '{city}', '{region}',
                '{postal}', '{country}')
                '''
        cursor.execute(sql)
        cursor.commit()
        cursor.execute('SELECT SCOPE_IDENTITY()')
        CID = int(cursor.fetchall()[0][0])
        generate_confday(CID, startdate, length)
    print(f"Successfully generated {n} records in {table}")


def generate_confday(CID, startdate, length):
    table = "ConferencesDays"
    for i in range(length + 1):
        date = startdate + datetime.timedelta(days=i)
        dayprice = randint(300, 800)
        limit = randelem([500, 700, 800, 900, 1000, 1200, 1500,
100, 200, 300, 400, 1750, 2000])
        topic = randelem(TOPICS).replace("'", "''")
        description = randelem(DESCRIPTIONS).replace("'", "''")
        sql = f'''
                INSERT {database}.dbo.{table} (ConferenceID, Date,
DayPrice, ParticipantsLimit, Topic, Description)
                VALUES ({CID}, '{str(date)}', '{dayprice}',
'{limit}', '{topic}', '{description}')
                '''
```

```
        cursor.execute(sql)
        cursor.commit()
        cursor.execute('SELECT SCOPE_IDENTITY()')
        CDID = int(cursor.fetchall()[0][0])
        generate_workshops(CDID, randint(2, 6))
        generate_lectset(CDID, randint(1, 5))
    print(f"Successfully generated {length + 1} records in
{table}")


def generate_workshops(CDID, n):
    table = "WorkshopsDetails"
    for i in range(n):
        price = randint(300, 800)
        limit = randelem([10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
120, 140, 160, 150, 170, 180, 200])
        topic = randelem(TOPICS).replace("'", "''")
        description = randelem(DESCRIPTIONS).replace("'", "''")
        cursor.execute('SELECT LecturerID FROM Lecturers')
        lecturers = [int(x[0]) for x in cursor.fetchall()]
        LID = randelem(lecturers)
        time = str(randint(8, 19)) + ':' + '00:00'
        duration = randelem([10, 15, 20, 25, 30, 35, 40, 45, 50,
60, 90, 120, 100, 180, 150, 240, 300])
        sql = f'''
                INSERT {database}.dbo.{table} (ConferenceDayID,
LecturerID, Price, ParticipantsLimit, Time,
                Duration, Topic, Description)
                VALUES ({CDID}, {LID}, '{price}', '{limit}',
'{time}', {duration}, '{topic}', '{description}')
                '''
        cursor.execute(sql)
        cursor.commit()
    print(f"Successfully generated {n} records in {table}")


def generate_payment(RID, amount):
    table = "Payments"
    sql = f'''
            INSERT {database}.dbo.{table} (ReservationID, Amount)
            VALUES ({RID}, {amount})
            '''
    cursor.execute(sql)
    cursor.commit()
    print(f"Successfully generated payment record")


def generate_lectset(CDID, n):
    table = "LecturersSet"
    cursor.execute('SELECT LecturerID FROM Lecturers')
    lecturers = [int(x[0]) for x in cursor.fetchall()]
    shuffle(lecturers)
    for i in range(n):
        LID = lecturers[i]
        sql = f'''
```

```
                INSERT {database}.dbo.{table} (LecturerID,
ConferenceDayID)
                VALUES ({LID}, {CDID})
                '''
        cursor.execute(sql)
        cursor.commit()
    print(f"Successfully generated {n} records in {table}")


def generate_reservation(CID):
    table = "Reservations"
    participants = randint(5, 100)
    sql = f'''
            INSERT {database}.dbo.{table} (CustomerID,
ParticipantsNumber)
            VALUES ({CID}, {participants})
            '''
    cursor.execute(sql)
    cursor.commit()
    print(f"Successfully generated record in {table}")
    cursor.execute('SELECT SCOPE_IDENTITY()')
    RID = int(cursor.fetchall()[0][0])
    try:
        # BŁĄD MOŻE WYSTĄPIĆ, JEŻELI NIE BĘDZIE MIEJSCA NA DANEJ
KONFERENCJI JUŻ
        generate_confdayres(RID)
        cursor.execute(f'SELECT dbo.CountPrice({RID})')
        price = cursor.fetchall()[0][0]
        cursor.execute(f'UPDATE {database}.dbo.{table} SET Price =
{price} WHERE ReservationID = {RID};')
        cursor.commit()
        # tutaj się obliczy jaka jest cena przez trigger, więc
trzeba pobrać i wygenerować płatność lub nie (na próbę)
        # trzeba też wygenerować uczestników lub nie
        decide = randint(0, 10)
        if decide < 8:
            generate_participants(participants, RID)
        decide = randint(0, 10)
        if decide < 8:
            cursor.execute(f'SELECT Price FROM Reservations WHERE
ReservationID = {RID}')
            amount = cursor.fetchall()[0][0]
            generate_payment(RID, amount)
    except Exception:
        cursor.execute(f'UPDATE Reservations SET Cancelled = 1
WHERE ReservationID = {RID}')


def generate_confdayres(RID):
    table = "ConferencesReservations"
    cursor.execute('SELECT ConferenceID FROM Conferences')
    conferences = [int(x[0]) for x in cursor.fetchall()]
    CID = randelem(conferences)
    cursor.execute(f'SELECT ConferenceDayID FROM ConferencesDays
AS CD WHERE CD.ConferenceID = {CID}')
```

```
    confday = [int(x[0]) for x in cursor.fetchall()]
    shuffle(confday)
    howmany = randint(1, len(confday))
    for i in range(howmany):
        sql = f'''
                INSERT {database}.dbo.{table} (ReservationID,
ConferenceDayID)
                VALUES ({RID}, {confday[i]})
                '''
        try:
            cursor.execute(sql)
            cursor.commit()
            generate_workshopres(RID, confday[i])
        except Exception:
            pass
    print(f"Successfully generated {howmany} records in {table}")


def generate_workshopres(RID, CDID):
    table = "WorkshopsReservations"
    cursor.execute(f'SELECT WorkshopID FROM WorkshopsDetails AS WD
WHERE WD.ConferenceDayID = {CDID}')
    workshops = [int(x[0]) for x in cursor.fetchall()]
    shuffle(workshops)
    howmany = randint(0, len(workshops))
    for i in range(howmany):
        sql = f'''
                    INSERT {database}.dbo.{table} (ReservationID,
WorkshopID, ConferenceDayID)
                    VALUES ({RID}, {workshops[i]}, {CDID})
                    '''
        try:
            cursor.execute(sql)
            cursor.commit()
        except Exception:
            pass
    print(f"Successfully generated {howmany} records in {table}")


def main():
    # na początku generujemy prowadzących
    generate_lecturers(int(input("Liczba prowadzących: ")))
    # potem konferencje, dni konferencji, warsztaty i połączenia
między prowadzącymi a konferencjami
    generate_conferences(int(input("Liczba konferencji: ")))
    # na końcu klientów, rezerwację (1 lub 2 na klienta) i w ramach
rezerwacji rezerwujemy dni konferencji i
    # warsztaty, potem generujemy płatność i pracowników (20%
rezerwacji nie ma płatności i listy uczestników)
    generate_customers(int(input("Liczba klientów: ")))


if __name__ == "__main__":
    main()
```

10. Inne

Dodatkowo utworzyliśmy skrypt, który powinien zostać wywoływany codziennie na serwerze z bazą danych. Jego zadaniem jest wysyłanie maili z przypomnieniami lub anulowanie rezerwacji, jeżeli klient nie spełnił wymogów rezerwacji (płatność 14 dni przed i podanie listy uczestników 7 dni przed wydarzeniem). Skrypt może zostać wskazany do uruchomienia na przykład przez demona zegarowego Cron w systemie linux codziennie.

daily_updater.py:

```
import pyodbc

server = [nazwa serwera]
database = [nazwa_bazy danych]
username = [login]
password = [hasło]
cnxn = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL
Server};SERVER=' + server + ';DATABASE=' +
                  database + ';UID='+ username +';PWD='+ password))
cursor = cnxn.cursor()


def update_cancelled():

   cursor.execute(
       f'''
       SELECT R.ReservationID, R.CustomerID, Email,
dbo.CustomerSentParticipants(R.ReservationID) AS PartSent,
                dbo.CustomerPaid(R.ReservationID) AS Paid, DATEDIFF(d,
GETDATE(), StartDate) AS DaysLeft
       FROM Reservations AS R
       JOIN (
           SELECT DISTINCT ReservationID, C.ConferenceID, StartDate
           FROM ConferencesReservations AS CR
           JOIN ConferencesDays AS CD ON CR.ConferenceDayID =
CD.ConferenceDayID
           JOIN Conferences C on CD.ConferenceID = C.ConferenceID
           ) AS CONF ON R.ReservationID = CONF.ReservationID
       JOIN Customers ON Customers.CustomerID = R.CustomerID
       WHERE R.Cancelled = 0 AND DATEDIFF(d, GETDATE(), StartDate) >= 0
       '''
   )
   reservations = cursor.fetchall()
   count = 0
   reminders = 0
   for res in reservations:
       days = res[5]
       paid = res[4]
       partsend = res[3]
       email = res[2]
```

```
        CID = res[1]
        RID = res[0]
        if days == 14:
            if not paid:
                print(f"Sending email to: {email}\nSubject: You have
unpaid meeting starting in 14 days")
                print(f"ReservationID: {RID}")
                reminders += 1
            if not partsend:
                print(f"Sending email to: {email}\nSubject: You still
have not sent us participants list")
                print(f"ReservationID: {RID}. Events starts in 14 days")
                reminders += 1
        if days <= 7:
            print(res)
            if not paid:
                cursor.execute(
                    f'''
                    UPDATE Reservations SET Cancelled = 1 WHERE
ReservationID = {RID}
                    '''
                )
                cursor.commit()
                print(f"Sending email to: {email}\nSubject: Your
reservation has been cancelled")
                print(f"ReservationID: {RID}. Your reservation has been
cancelled because we did not receive"
                      f" payment from you")
                count += 1
            elif not partsend:
                cursor.execute(
                    f'''
                    UPDATE Reservations SET Cancelled = 1 WHERE
ReservationID = {RID}
                    '''
                )
                cursor.commit()
                print(f"Sending email to: {email}\nSubject: Your
reservation has been cancelled")
                print(f"ReservationID: {RID}. Your reservation has been
cancelled because we did not receive"
                      f" a participants list from you. If you have already
paid, we will return you your payment")
                count += 1

    print(f"Reminders sent: {reminders}\nReservations cancelled:
{count}")


def main():
    update_cancelled()


if __name__ == "__main__":
    main()
```