AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Podstawy baz danych

Dokumentacja projektu “Konferencje”

Szymon Wieczorek
Mateusz Monasterski

1. Wprowadzenie

Celem zadania projektowego jest poznanie podstaw projektowania i implementowania systemu bazodanowego. Tworzony system ma wspomagać funkcjonowanie firmy organizującej konferencje. Są to konferencje płatne, jedno lub kilkudniowe. Ich cena jest zależna od zarezerwowanych usług, oraz od terminu rezerwacji. Studentom przysługuje zniżka. Klientami dokonującymi rezerwacji mogą być firmy lub osoby indywidualne, lecz uczestnikami konferencji są osoby. W czasie konferencji odbywają się również warsztaty, w których mogą uczestniczyć osoby, które są zarejestrowane na konferencję, która odbywa się danego dnia. Warsztaty mogą być płatne lub darmowe.

2. Założenia

Przystępując do wykonania zadania projektowego, na początek wyszczególniamy następujące założenia:

- Klienci firmy korzystającej z systemu bazodanowego są uczestnikami konferencji, nie są ich organizatorami
- Klienci mogą rejestrować się na konferencje za pomocą systemu www.
  W przypadku, gdy klientem jest firma, może ona rezerwować określoną ilość miejsc na konkretne dni, lecz musi na dwa tygodnie przed rozpoczęciem uzupełnić dane osobowe uczestników. Jeżeli tego nie zrobi, rezerwacja przepada.
- Konferencje mogą być kilkudniowe, jednakże zakłada się ciągłość dni.
- Kilka warsztatów może trwać równocześnie, jednakże uczestnik może się zarejestrować tylko na jeden warsztat, który trwa w danym czasie.
- Opłaty za udział w konferencji zależą od tego, ile dni pozostaje do rozpoczęcia konferencji. Jest kilka progów ceny (im bliżej konferencji tym cena wyższa)
- Cena za udział w warsztatach jest stała (brak progów ceny)
- Dla studentów przewidziana jest zniżka (jako wartość procentowa ceny regularnej)
- Zniżka studencka dotyczy opłaty za udział w konferencji, cena warsztatów płatnych jest jedna dla każdego uczestnika
- Klienci mają tydzień od rezerwacji na zapłatę należności za zamówione usługi. Jeżeli tego terminu nie dotrzymają, to rezerwacja jest anulowana.
- Konferencja kilkudniowa ma ustaloną cenę za jeden dzień konferencji (nie ma odmiennych cen dla dwóch różnych dni konferencji)
- Zarówno konferencja, jak i warsztat posiada minimalną liczbę uczestników. Jeżeli nie zapisze się odpowiednia ilość osób, to można zmienić termin konferencji, lub też anulować konferencję/warsztat
- Nie ma możliwości anulowania poszczególnych dni konferencji (anuluje się całą konferencję)

- Jeżeli firma organizująca konferencję anuluje konferencję, to pieniądze są zwracane
- Klienci mogą anulować rezerwację zarówno na konferencję, jak i na warsztat
- Można anulować rezerwację tylko na całą konferencję, nie na poszczególne dni
- Jeżeli klient anuluje rezerwację na konferencję maksymalnie na trzy tygodnie przed rozpoczęciem konferencji, to pieniądze są mu zwracane w wysokości 90% wpłaty
- W przypadku anulowania rezerwacji na warsztat, pieniądze nie są klientowi zwracane
- Klient ma obowiązek zapłacenia całej ceny za rezerwację konferencji w jednej płatności

3. Analiza wymagań

3.1 Konferencje i warsztaty

3.1.1 Z punktu widzenia firmy

- Firma organizująca konferencje może dodawać nową konferencję podając jej:
    - Nazwę
    - Datę rozpoczęcia
    - Datę zakończenia
    - Opis
- Istnieje możliwość ustalania ceny za dzień konferencji, oraz zniżek procentowych w zależności od liczby dni do rozpoczęcia konferencji.
- W połączeniu z ceną za dzień konferencji, ustala się również zniżkę procentową dla studentów.
- Firma organizująca może ustalać dla każdego z dni konferencji:
    - Miejsce, w którym konferencja się będzie w danym dniu odbywała
    - Limit uczestników
- Do każdego dnia, może być dodany warsztat (lub kilka warsztatów). Konieczne jest dodanie:
    - Nazwy warsztatu
    - Czas rozpoczęcia i zakończenia
    - Cenę (może być 0)
    - Limit uczestników
    - Opis
- Firma organizująca konferencje może anulować konferencję lub warsztat, jeżeli istnieje taka potrzeba (brak minimalnej liczby uczestników, inne przyczyny) lub

zmienić termin konferencji. Płatności dokonane za anulowaną konferencję lub warsztat są klientowi zwracane.

3.1.2 Z punktu widzenia klientów

- Klienci rejestrują się jako klienci indywidualni bądź instytucjonalni, podając:
    - Klienci indywidualni
        - Imię
        - Nazwisko
        - Pełny adres
        - e-mail
        - telefon
    - Firmy
        - Nazwę
        - NIP
        - REGON
        - adres siedziby
        - e-mail
        - telefon
- Klienci dokonują rezerwacji na konferencję, wyszczególniając rezerwacje na konkretne dni. Firmy mogą podawać ilość uczestników, których dane szczegółowe mogą być dodane najpóźniej na dwa tygodnie przed konferencją.
- Dane szczegółowe uczestników:
    - Imię i nazwisko
    - Numer legitymacji studenckiej (w przypadku studentów)
    - Adres
    - e-mail
    - nr telefonu
- Klienci mogą anulować rezerwację zarówno na warsztat jak i na konferencję

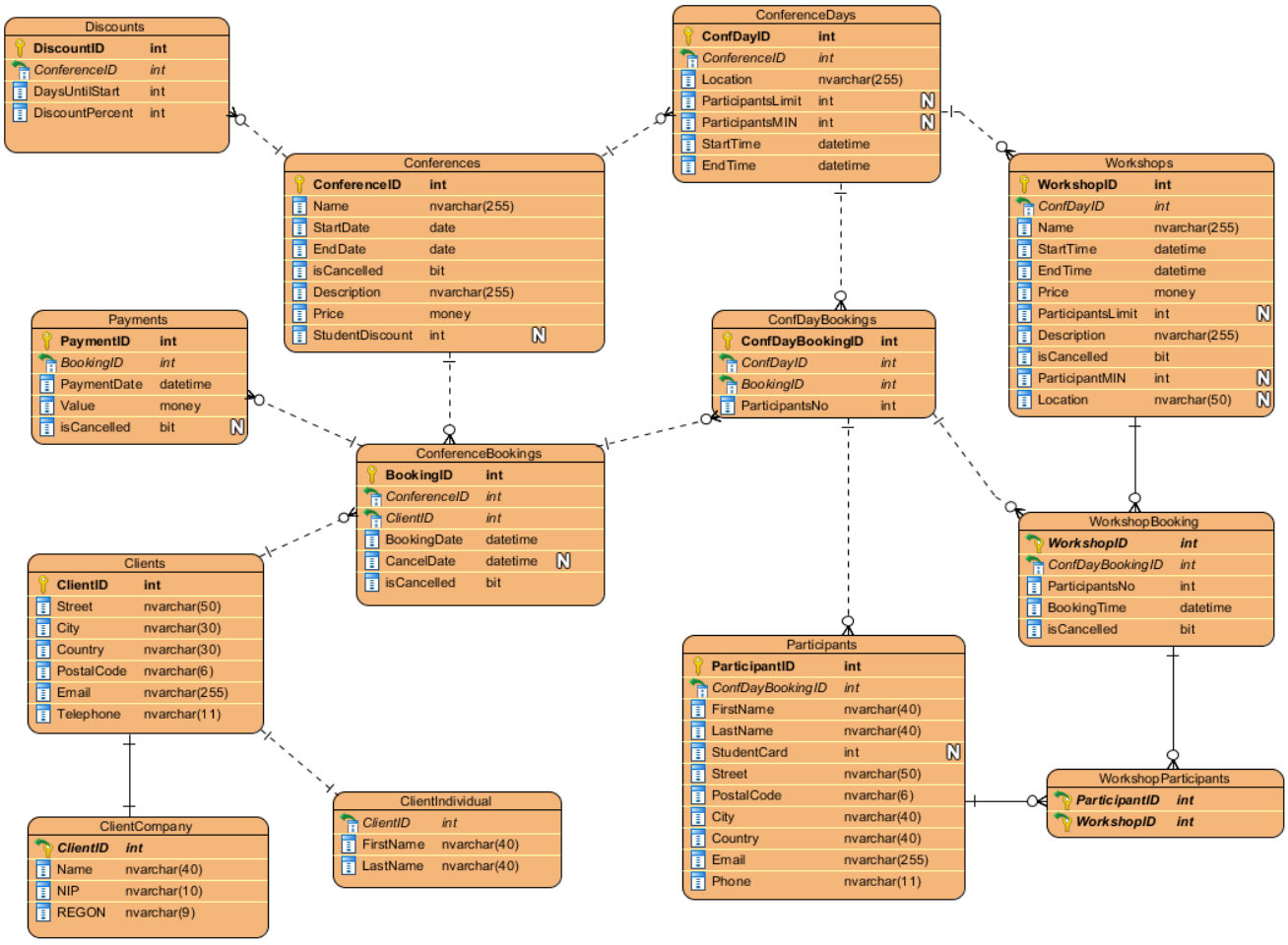
3.2 Raporty

Ze względu na potrzeby firmy organizującej konferencje, system musi spełniać następujące wymagania:

- Możliwość wygenerowania listy klientów firmowych i indywidualnych wraz z informacjami o płatnościach i uczestnictwie w konferencjach
- Możliwość wygenerowania list osobowych uczestników:
    - Na konferencję
    - Na dzień konferencji

    - Na warsztat
- Możliwość wygenerowania listy najczęstszych klientów, oraz najpopularniejszych konferencji
- Możliwość wygenerowania faktur

4. Schemat bazy danych



5. Opis tabel

5.1 CONFERENCES

W tabeli przechowywane są podstawowe informacje o danej konferencji.

- ConferenceID - numer identyfikacyjny
- Name - nazwa konferencji
- StartDate - data rozpoczęcia konferencji
- EndDate - data zakończenia konferencji
- isCancelled - flaga informująca czy dana konferencja została anulowana
- Description - opis konferencji
- Value - cena za całą konferencję
- StudentDiscount - wartość zniżki studenckiej

```
CREATE TABLE Conferences
(
 ConferenceID    INT IDENTITY
   PRIMARY KEY,
 Name            NVARCHAR(255) NOT NULL,
 StartDate       DATE          NOT NULL,
 EndDate         DATE          NOT NULL,
 isCancelled     BIT DEFAULT 0 NOT NULL,
 Description     NVARCHAR(255) NOT NULL,
 Price           MONEY         NOT NULL
   CONSTRAINT CHK_priceC
   CHECK ([price] >= 0),
 StudentDiscount INT,
 CONSTRAINT CHK_dates
 CHECK ([StartDate] <= [EndDate])
)
GO
```

5.2 DISCOUNTS

Przechowywanie informacji o zniżkach w zależności od daty opłaty z wyprzedzeniem.

- DiscountID - numer identyfikacyjny
- ConferenceID - numer konferencji
- DaysUntilStart - liczba dni do rozpoczęcia konferencji, do której obowiązuje dana zniżka
- DiscountPercent - wartość procentowa zniżki

```
CREATE TABLE Discounts
(
 DiscountID      INT IDENTITY
   PRIMARY KEY,
 ConferenceID    INT NOT NULL
   CONSTRAINT FKDiscounts292995
   REFERENCES Conferences,
 DaysUntilStart  INT NOT NULL
   CONSTRAINT CHK_daysUntilStart
   CHECK ([DaysUntilStart] >= 0),
 DiscountPercent INT NOT NULL
   CONSTRAINT CHK_discPercent
   CHECK ([DiscountPercent] > 0 AND [DiscountPercent] <= 100)
)
GO
```

5.3 PAYMENTS

Przechowywanie informacji o dokonanych płatnościach dla danej rezerwacji.

- PaymentID - numer identyfikacyjny
- BookingID - numer rezerwacji
- PaymentDate - data dokonania zapłaty
- Value - wartość dokonanej płatności
- isCancelled - flaga informująca czy płatność została anulowana

```
CREATE TABLE Payments
(
 PaymentID   INT IDENTITY
   PRIMARY KEY,
 BookingID   INT      NOT NULL
   CONSTRAINT FKPayments904947
   REFERENCES [Conference Bookings],
 PaymentDate DATETIME NOT NULL,
 Value       MONEY    NOT NULL
   CONSTRAINT CHK_value
   CHECK ([Value] > 0),
 isCancelled BIT DEFAULT 0
)
GO
```

5.4 CLIENTS

Przechowywanie ogólnych informacji o klientach.

- ClientID - numer identyfikacyjny
- Street - ulica pod którą mieszka klient / na której mieści się firma
- City - miasto, j.w.
- Country - kraj, j.w.
- PostalCode - kod pocztowy
- Email - adres email klienta
- Telephone - numer telefonu klienta

```
CREATE TABLE Clients
(
 ClientID   INT IDENTITY
   PRIMARY KEY,
 Street     NVARCHAR(50) NOT NULL,
 City       NVARCHAR(30) NOT NULL,
 Country    NVARCHAR(30) NOT NULL,
 PostalCode NVARCHAR(6) NOT NULL,
 Email      NVARCHAR(255) NOT NULL
   CONSTRAINT CHK_EmailC
   CHECK ([Clients].[Email] LIKE '%@%.%'),
 Telephone  NVARCHAR(11) NOT NULL
)
GO
```

5.5 CLIENT COMPANY

Dane klientów firmowych.

- ClientID - numer klienta
- Name - nazwa firmy
- NIP - numer identyfikacji podatkowej
- REGON - numer REGON

```
CREATE TABLE [Client Company]
(
 ClientID INT           NOT NULL
```

```
   PRIMARY KEY
   CONSTRAINT [FKClient Com783847]
   REFERENCES Clients,
 Name      NVARCHAR(40) NOT NULL,
 NIP       NVARCHAR(10) NOT NULL
   CONSTRAINT CHK_NIP
   CHECK ([Client Company].[NIP] LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
 REGON     NVARCHAR(9) NOT NULL
   CONSTRAINT CHK_REGON
   CHECK ([Client Company].[REGON] LIKE
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
GO
```

5.6 CLIENT INDIVIDUAL

Dane klientów indywidualnych.

- ClientID - numer klienta
- FirstName - imię
- LastName - nazwisko

```
CREATE TABLE [Client Individual]
(
 ClientID  INT           NOT NULL
   PRIMARY KEY
   CONSTRAINT [FKClient Ind474200]
   REFERENCES Clients,
 FirstName NVARCHAR(40) NOT NULL,
 LastName  NVARCHAR(40) NOT NULL
)
GO
```

5.7 CONFERENCE BOOKINGS

Tabela przechowująca dane rezerwacji na konferencje.

- BookingID - numer identyfikacyjny rezerwacji

- ConferenceID - numer rezerwowanej konferencji
- ClientID - numer rezerwującego klienta
- BookingDate - data rezerwacji
- isCancelled - flaga mówiąca czy rezerwacja została anulowana
- CancelDate - data anulowania rezerwacji

```
CREATE TABLE [Conference Bookings]
(
 BookingID    INT      IDENTITY
   PRIMARY KEY,
 ConferenceID INT          NOT NULL
   CONSTRAINT FKConference776144
   REFERENCES Conferences,
 ClientID     INT          NOT NULL
   CONSTRAINT FKConference217598
   REFERENCES Clients,
 isCancelled  BIT DEFAULT 0 NOT NULL,
 CancelDate   DATETIME DEFAULT NULL,
 BookingDate  DATETIME     NOT NULL,
 CONSTRAINT CHK_datesCB
 CHECK ([BookingDate] < [CancelDate])
)
GO
```

5.8 CONF DAY BOOKINGS

Dane rezerwacji rozłożone na poszczególne dni konferencji.

- ConfDayBookingID - numer identyfikacyjny
- ConfDayID - numer dnia konferencji
- BookingID - numer rezerwacji na konferencję
- ParticipantsNo - liczba zgłoszonych przez rezerwującego uczestników

```
CREATE TABLE [Conf Day Bookings]
(
 ConfDayBookingID INT IDENTITY
   PRIMARY KEY,
 ConfDayID        INT NOT NULL
```

```
   CONSTRAINT [FKConf Day B365845]
   REFERENCES [Conference Days],
 BookingID         INT NOT NULL
   CONSTRAINT [FKConf Day B621235]
   REFERENCES [Conference Bookings],
 ParticipantsNo    INT NOT NULL
   CONSTRAINT CHK_paricipNo
   CHECK ([ParticipantsNo] > 0)
)
GO
```

5.9 CONFERENCE DAYS

Dane poszczególnych dni konferencji.

- ConfDayID - numer identyfikacyjny
- ConferenceID - numer konferencji
- Location - miejsce wydarzenia
- ParticipantsLimit - limit uczestników na dany dzień
- Date - data
- ParticipantsMIN - minimalna liczba uczestników, aby konferencja się odbyła w danym dniu.

```
§CREATE TABLE [Conference Days]
(
 ConfDayID          INT IDENTITY
   PRIMARY KEY,
 ConferenceID       INT            NOT NULL
   CONSTRAINT FKConference111540
   REFERENCES Conferences,
 Location           NVARCHAR(255) NOT NULL,
 ParticipantsLimit INT,
 ParticipantsMIN    INT DEFAULT 0,
 StartTime          DATETIME       NOT NULL,
 EndTime            DATETIME       NOT NULL,
 Date               DATE           NOT NULL,
 CONSTRAINT CHK_ParticipantsNo
 CHECK ([ParticipantsLimit] >= [ParticipantsMIN]),
 CONSTRAINT CHK_timeCD
```

```
 CHECK ([StartTime] < [EndTime])
)
GO
```

5.10 PARTICIPANTS

Dane uczestników.

- ParticipantID - numer identyfikacyjny
- ConfDayBookingID - numer dnia konferencji w której uczestnik bierze udział
- FirstName - imię
- LastName - nazwisko
- StudentCard - numer legitymacji studenckiej
- Street - ulica zamieszkania
- PostalCode - kod pocztowy
- City - miasto zamieszkania
- Country - kraj zamieszkania
- Email - adres email
- Phone - numer telefonu kontaktowego

```
CREATE TABLE Participants
(
 ParticipantID    INT IDENTITY
   PRIMARY KEY,
 ConfDayBookingID INT           NOT NULL
   CONSTRAINT FKParticipan542399
   REFERENCES [Conf Day Bookings],
 FirstName        NVARCHAR(40) NOT NULL,
 LastName         NVARCHAR(40) NOT NULL,
 StudentCard      INT,
 Street           NVARCHAR(50) NOT NULL,
 PostalCode       NVARCHAR(6) NOT NULL,
 City             NVARCHAR(40) NOT NULL,
 Country          NVARCHAR(40) NOT NULL,
 Email            NVARCHAR(255) NOT NULL
   CONSTRAINT CHK_EmailP
   CHECK ([Participants].[Email] LIKE '%@%.%'),
 Phone            NVARCHAR(11) NOT NULL
)
GO
```

5.11 WORKSHOP PARTICIPANTS

Tabela łącznikowa, przechowująca dane uczestniczących w warsztatach.

- ParticipantID - numer uczestnika
- WorkshopID - numer warsztatu

```
CREATE TABLE [Workshop Participants]
(
 ParticipantID INT NOT NULL
   CONSTRAINT [FKWorkshop P901540]
   REFERENCES Participants
   CONSTRAINT FKWorkshopPa872329
   REFERENCES Participants,
 WorkshopID    INT NOT NULL
   CONSTRAINT [FKWorkshop P471517]
   REFERENCES [Workshop Booking]
   CONSTRAINT FKWorkshopPa726202
   REFERENCES [Workshop Booking],
 PRIMARY KEY (ParticipantID, WorkshopID),
 CONSTRAINT Unique_Pair_Constraint
 UNIQUE (ParticipantID, WorkshopID)
)
GO
```

5.12 WORKSHOP BOOKING

Rezerwacje na dane warsztaty.

- WorkshopID - numer identyfikacyjny warszatu
- ConfDayBookingID - numer rezerwacji danego dnia konferencji
- ParticipantsNo - liczba zarezerwowanych przez klienta miejsc dla uczestników
- BookingTime - data rezerwacji
- isCancelled - flaga, która mówi czy rezerwacja została anulowana

```
CREATE TABLE [Workshop Booking]
(
 WorkshopID       INT           NOT NULL
```

```
   PRIMARY KEY
   CONSTRAINT [FKWorkshop B623904]
   REFERENCES Workshops,
 ConfDayBookingID INT           NOT NULL
   CONSTRAINT [FKWorkshop B445833]
   REFERENCES [Conf Day Bookings],
 ParticipantsNo   INT           NOT NULL
   CONSTRAINT CHK_participNo
   CHECK ([ParticipantsNo] > 0),
 BookingDate      DATETIME      NOT NULL,
 isCancelled      BIT DEFAULT 0 NOT NULL
)
GO
```

5.13 WORKSHOPS

Dane poszczególnych warsztatów.

- WorkshopID - numer identyfikacyjny
- ConfDayID - numer dnia konferencji w którym odbywa się warszat
- StartTime - czas rozpoczęcia
- EndTime - czas zakończenia
- Price - cena za warszat
- ParticipantsLimit - limit uczestników
- Description - opis warszatu
- isCancelled - czy został anulowany przez organizatora
- ParticipantsMIN - minimalna liczba uczestników aby warsztat mógł się odbyć

```
CREATE TABLE Workshops
(
 WorkshopID        INT IDENTITY
   PRIMARY KEY,
 ConfDayID         INT           NOT NULL
   CONSTRAINT FKWorkshops200651
   REFERENCES [Conference Days],
 Name              NVARCHAR(255) NOT NULL,
 StartTime         DATETIME      NOT NULL,
 EndTime           DATETIME      NOT NULL,
 Price             MONEY         NOT NULL
   CONSTRAINT CHK_price
   CHECK ([Price] >= 0),
 ParticipantsLimit INT,
```

```
 Description        NVARCHAR(255) NOT NULL,
 isCancelled        BIT DEFAULT 0 NOT NULL,
 ParticipantsMIN    INT DEFAULT 0,
 Location           NVARCHAR(50),
 CONSTRAINT CHK_time
 CHECK ([StartTime] < [EndTime]),
 CONSTRAINT CHK_participants
 CHECK ([ParticipantsMIN] <= [ParticipantsLimit] AND
[ParticipantsMIN] >= 0)
)
GO
```

6. Widoki

6.1 Konferencje

Widok **conference_days_available** pokazuje dni konferencji, w których nie został wyczerpany limit rezerwacji. Dla każdego dnia konferencji podaje limit miejsc oraz ilość wolnych miejsc.

```
CREATE VIEW conference_days_available AS
SELECT
   conf_D.ConferenceID,
   conf.[Name],
   conf_D.StartTime,
   conf_D.EndTime,
   conf_D.ParticipantsLimit,
   conf_D.ParticipantsLimit - sum(conf_D_B.ParticipantsNo) as
nr_of_available
FROM Conferences as conf
JOIN [Conference Days] as conf_D ON conf.ConferenceID =
conf_D.ConferenceID AND conf.isCancelled = 0
JOIN [Conf Day Bookings] as conf_D_B ON conf_D_B.ConfDayID =
conf_D.ConfDayID
JOIN [Conference Bookings] as conf_B ON conf_D_B.BookingID =
conf_B.BookingID AND conf_B.isCancelled = 0
GROUP BY conf_D.ConferenceID, conf.[Name],
conf_D.StartTime,conf_D.EndTime, conf_D.ParticipantsLimit
```

```
HAVING sum(conf_D_B.ParticipantsNo) < conf_D.ParticipantsLimit
```

Widok **days_popularity** pokazuje poszczególne dni konferencji wraz z liczbą rezerwacji.

```
CREATE VIEW days_popularity AS
SELECT
    conf.ConferenceID,
    conf.[Name],
    conf_D.[Date],
    sum(conf_D_B.ParticipantsNo) AS reserved
FROM Conferences as conf
JOIN [Conference Days] as conf_D ON conf_D.ConferenceID =
conf.ConferenceID
JOIN [Conf Day Bookings] as conf_D_B ON conf_D_B.ConfDayID =
conf_D.ConfDayID
JOIN [Conference Bookings] as conf_B ON conf_D_B.BookingID =
conf_B.BookingID AND conf_B.isCancelled = 0
GROUP BY conf.ConferenceID, conf.[Name], conf_D.[Date]
```

Widok **cancelled_bookings** wyświetla rezerwacje, które zostały anulowane.

```
CREATE VIEW cancelled_bookings AS
SELECT
   cb.BookingID,
   c.ClientID,
   c.Email,
   c.Telephone,
   cb.ConferenceID,
   conf.[Name]
FROM
Clients as c
JOIN [Conference Bookings] as cb ON  cb.ClientID = c.ClientID AND
cb.isCancelled = 1
JOIN Conferences as conf ON cb.ConferenceID = conf.ConferenceID
```

Widok **incomplete_bookings** pokazuje listę klientów, którzy dokonali rezerwacji pewnej liczby miejsc na konkretne dni konferencji, ale pomimo iż pozostaje mniej niż dwa tygodnie do rozpoczęcia konferencji, to wciąż nie uzupełnili listy uczestników.

```
CREATE VIEW incomplete_bookings AS
SELECT
   c.ClientID,
 cb.BookingID,
 cdb.ConfDayBookingID,
 cdb.ParticipantsNo,
 count(p.ParticipantID) as nr_of_participants,
 cdb.ParticipantsNo - count(p.ParticipantID) as participants_left

FROM
 Clients as c
 JOIN [Conference Bookings] as cb ON c.ClientID = cb.ClientID AND
cb.isCancelled = 0
 JOIN Conferences as conf ON conf.ConferenceID = cb.ConferenceID
AND DATEDIFF(day, GETDATE(), conf.StartDate) < 14
 JOIN [Conf Day Bookings] as cdb ON cb.BookingID = cdb.BookingID
 JOIN Participants as p ON p.ConfDayBookingID =
cdb.ConfDayBookingID

GROUP BY c.ClientID,
 cb.BookingID,
 cdb.ConfDayBookingID,
 cdb.ParticipantsNo

HAVING cdb.ParticipantsNo - count(p.ParticipantID) > 0
```

Widok **unpaid_bookings** wyświetla numer klienta, numer rezerwacji, oraz numer i nazwę konferencji, dla klientów którzy dokonali rezerwacji, a nie zapłacili w ciągu 7 dni.

```
CREATE VIEW unpaid_bookings AS
SELECT
   c.ClientID,
 cb.BookingID,
 conf.ConferenceID,
 conf.Name

FROM
 Clients as c
```

```
 JOIN [Conference Bookings] as cb ON c.ClientID = cb.ClientID AND
cb.isCancelled = 0
 JOIN Conferences as conf ON conf.ConferenceID = cb.ConferenceID
AND DATEDIFF(day, cb.BookingDate, GETDATE()) > 7
 JOIN Payments as p ON p.BookingID = cb.BookingID
GROUP BY c.ClientID,
 cb.BookingID,
 conf.ConferenceID,
 conf.Name
HAVING count(p.PaymentID) = 0
```

6.2 Warsztaty

Widok **workshops_available** pokazuje warsztaty, których limit uczestników nie jest wyczerpany.

```
CREATE VIEW workshops_available AS
SELECT
   W.WorkshopID,
   W.[Name],
   W.ParticipantsLimit,
   W.ParticipantsLimit - sum(WB.ParticipantsNo) as places_available
FROM Workshops as W
JOIN [Workshop Booking] as WB ON W.WorkshopID = WB.WorkshopID AND
W.isCancelled = 0
GROUP BY W.WorkshopID, W.[Name], W.ParticipantsLimit
HAVING sum(WB.ParticipantsNo) < W.ParticipantsLimit
```

6.3 Klienci

Widok **client_reservations** pokazuje klientów z ich ilością rezerwacji na konferencje.

```
CREATE VIEW client_reservations AS
SELECT
   C.ClientID, count(*) as nr_of_bookings
FROM
Clients as C
```

```
JOIN [Conference Bookings] as CB ON CB.ClientID = C.ClientID
AND CB.isCancelled = 0
GROUP BY C.ClientID
```

Widok **client_payments** pokazuje klientów, oraz kwotę, którą oni zapłacili za konferencje łącznie.

```
CREATE VIEW client_payments AS
SELECT
   C.ClientID, sum(P.Value) as total_paid
FROM
Clients as C
JOIN [Conference Bookings] as CB ON CB.ClientID = C.ClientID
AND CB.isCancelled = 0
JOIN Payments as P ON P.BookingID = CB.BookingID AND P.isCancelled = 0
GROUP BY C.ClientID
```

7. Procedury

AddNewConference

Procedura dodająca nową konferencję do tabeli ‘Conferences’

```
CREATE PROCEDURE AddNewConference
@Name nvarchar(50),
@StartDate date,
@EndDate date,
@Description nvarchar(255),
@Price MONEY,
@StudDisc INT
 AS
BEGIN

BEGIN TRANSACTION
 BEGIN TRY
 SET NOCOUNT ON;
 INSERT INTO Conferences
(StartDate,EndDate,Name,Description,Price,StudentDiscount)
 VALUES (@StartDate,@EndDate,@Name,@Description,@Price,@StudDisc)
   END TRY
```

```
   BEGIN CATCH
     SELECT
   ERROR_NUMBER() AS ErrorNumber
   ,ERROR_SEVERITY() AS ErrorSeverity
   ,ERROR_STATE() AS ErrorState
   ,ERROR_PROCEDURE() AS ErrorProcedure
   ,ERROR_LINE() AS ErrorLine
   ,ERROR_MESSAGE() AS ErrorMessage;

     IF @@TRANCOUNT >0
       ROLLBACK TRANSACTION
   END CATCH
   IF @@TRANCOUNT >0
     COMMIT TRANSACTION
END
GO
```

AddNewConfDay

Procedura dodająca nowy dzień danej konferencji. Sprawdza czy zgadzają się daty z przedziałem czasowym konferencji oraz czy nie ma już dodanego dnia konferencji w tym samym dniu.

```
CREATE PROCEDURE AddNewConfDay
@ConferenceID int,
@Location nvarchar(255),
@ParticipantsLimit int,
@Start DATETIME,
@End DATETIME,
@ParticipantsMIN INT
AS
 BEGIN
 BEGIN TRANSACTION
   BEGIN TRY
     SET NOCOUNT ON
  DECLARE @ConfStart AS DATE
  DECLARE @ConfEnd AS DATE
  SET @ConfStart = (SELECT StartDate
                    FROM Conferences
                    WHERE ConferenceID = @ConferenceID)
  SET @ConfEnd = (SELECT EndDate
                  FROM Conferences
                  WHERE ConferenceID = @ConferenceID)
  DECLARE @isEmpty AS INT
```

```
  SET @isEmpty = ISNULL((SELECT count(ConfDayID)
                         FROM [Conference Days]
                         WHERE ConferenceID = @ConferenceID AND
convert(DATE, StartTime) = convert(DATE, @Start)
                         GROUP BY ConferenceID), 0)
  IF (convert(DATE, @Start) >= @ConfStart AND CONVERT(DATE, @End) <=
@ConfEnd AND @isEmpty = 0)
    BEGIN
      INSERT INTO [Conference Days] (ConferenceID, Location,
ParticipantsLimit, StartTime, EndTime, ParticipantsMIN)
      VALUES (@ConferenceID, @Location, @ParticipantsLimit, @Start,
@End, @ParticipantsMIN)
    END
  ELSE
    BEGIN
      RAISERROR ('Wrong date given', -1, -1)
    END
   END TRY
   BEGIN CATCH
     SELECT
       ERROR_NUMBER() AS ErrorNumber
       ,ERROR_SEVERITY() AS ErrorSeverity
       ,ERROR_STATE() AS ErrorState
       ,ERROR_PROCEDURE() AS ErrorProcedure
       ,ERROR_LINE() AS ErrorLine
       ,ERROR_MESSAGE() AS ErrorMessage;

     if @@trancount > 0
       ROLLBACK TRANSACTION;
   END CATCH;
   if @@trancount > 0
     COMMIT TRANSACTION;
 END
GO
```

AddNewParticipant

Procedura dodająca nowego uczestnika konferencji.

```
CREATE PROCEDURE AddNewParticipant
 @ConfDayBookingID int,
 @FirstName NVARCHAR(40),
 @LastName NVARCHAR(40),
 @StudentCardNo int,
 @Street NVARCHAR(50),
```

```
 @PostalCode NVARCHAR(10),
 @City NVARCHAR(40),
 @Country NVARCHAR(40),
 @Email NVARCHAR(255),
 @Phone NVARCHAR(11)
AS
BEGIN
 IF exists(SELECT ConfDayBookingID FROM [Conf Day Bookings] WHERE
ConfDayBookingID=@ConfDayBookingID)
   BEGIN
     SET NOCOUNT ON
     INSERT INTO Participants (ConfDayBookingID, FirstName, LastName,
                               StudentCard, Street, PostalCode, City,
Country, Email, Phone)
     VALUES (@ConfDayBookingID,@FirstName,@LastName,@StudentCardNo,
             @Street,@PostalCode,@City,@Country,@Email,@Phone)
   END
 ELSE PRINT 'Nie istnieje podana rezerwacja dnia konferencji'
END
go
```

AddNewWorkshop

Procedura dodająca nowy warsztat

```
CREATE PROCEDURE AddNewWorkshop
@ConfDayID int,
@Name NVARCHAR(255),
@StartTime DATETIME,
@EndTime DATETIME,
@Location NVARCHAR(50),
@Price money,
@ParticipantsLimit int = NULL ,
@Description NVARCHAR(255),
@ParticipantsMIN int = 0
AS
BEGIN
       IF exists(SELECT ConfDayID FROM [Conference Days] WHERE
ConfDayID=@ConfDayID) BEGIN
SET NOCOUNT ON
INSERT INTO Workshops (ConfDayID, Name, StartTime, EndTime, Location,
Price,
ParticipantsLimit, Description,ParticipantsMIN)
```

```
VALUES (@ConfDayID, @Name, @StartTime, @EndTime, @Location, @Price,
@ParticipantsLimit,@Description,
       @ParticipantsMIN)
END
ELSE PRINT 'Nie istnieje podany dzień konferencji'
END
```

AddWorkshopParticipant

Procedura sprawdzająca, czy istnieje podana para uczestnik-konferencja, a następnie ją dodaje do tabeli Workshop Participants

```
CREATE PROCEDURE AddWorkshopParticipant
 @ParticipantID int,
 @WorkshopID int

AS
BEGIN
IF (
 EXISTS
 (
 select ParticipantID from Participants
 where ParticipantID = @ParticipantID
 )
 AND EXISTS(
 select WorkshopID from Workshops
 where WorkshopID = @WorkshopID
 )
)
BEGIN
 SET NOCOUNT ON
 INSERT INTO [Workshop Participants] (ParticipantID, WorkshopID)
 VALUES (@ParticipantID, @WorkshopID)
 END
 ELSE
 PRINT 'Warsztat badz uczestnik nie istnieje'
END
GO
```

AddNewClientCompany

Procedura dodająca nowego klienta

```
CREATE PROCEDURE AddNewClientCompany
@Street NVARCHAR(50),
@City NVARCHAR(30),
@Country NVARCHAR(30),
@PostalCode NVARCHAR(6),
@Email NVARCHAR(255),
@Telephone NVARCHAR(11),
  @name NVARCHAR(40),
  @nip NVARCHAR(10),
  @regon NVARCHAR(9)
AS
BEGIN
SET NOCOUNT ON
INSERT INTO Clients (Street, City, Country, PostalCode, Email,
Telephone)
VALUES (@Street, @City, @Country, @PostalCode, @Email, @Telephone)
INSERT INTO [Client Company] (Name,NIP,REGON,ClientID)
  VALUES (@name,@nip,@regon,scope_identity())
END
GO
```

AddNewClientIndividual

Procedura dodająca nowego klienta indywidualnego.

```
CREATE PROCEDURE AddNewClientIndividual
@Street NVARCHAR(50),
@City NVARCHAR(30),
@Country NVARCHAR(30),
@PostalCode NVARCHAR(6),
@Email NVARCHAR(255),
@Telephone NVARCHAR(11),
@fName NVARCHAR(40),
@lName NVARCHAR(40)
AS
```

```
BEGIN
SET NOCOUNT ON
INSERT INTO Clients (Street, City, Country, PostalCode, Email,
Telephone)
VALUES (@Street, @City, @Country, @PostalCode, @Email, @Telephone)
INSERT INTO [Client Individual](FirstName,LastName,ClientID)
  VALUES (@fName,@lName,scope_identity())
END
GO
```

AddPayment

Dodaje nową płatność za konferencję pod warunkiem, że istnieje podana rezerwacja konferencji.

```
CREATE PROCEDURE AddPayment
@BookingID INT,
  @paymentDate DATETIME,
  @value MONEY
AS
BEGIN
  IF exists(SELECT BookingID FROM [Conference Bookings] WHERE
BookingID=@BookingID)
    BEGIN
      SET NOCOUNT ON
      INSERT INTO Payments (BookingID, PaymentDate, Value)
        VALUES (@BookingID,@paymentDate,@value)
    END
  ELSE PRINT 'Podana rezerwacja nie istnieje'
END
go
```

AddWorkshopBooking

Dodaje rezerwację warsztatu.

```
CREATE PROCEDURE AddWorkshopBooking
 @workshopID INT,
 @ConferenceDayBookingID INT,
 @ParticipantsNo INT,
```

```
 @bookingDate DATETIME
AS
BEGIN
 if (exists(SELECT WorkshopID FROM Workshops WHERE
WorkshopID=@workshopID)
   AND exists(SELECT ConfDayBookingID FROM [Conf Day Bookings]
WHERE @ConferenceDayBookingID=ConfDayBookingID))
     BEGIN
     SET NOCOUNT ON
     INSERT INTO [Workshop Booking] (WorkshopID, ConfDayBookingID,
ParticipantsNo, BookingDate)
     VALUES
(@workshopID,@ConferenceDayBookingID,@ParticipantsNo,@bookingDate)
     END
 ELSE PRINT 'Warsztat badz rezerwacja dnia konferencji nie
istnieje'
 END
go
```

AddConferenceDayBooking

Dodaje rezerwację dnia konferencji, sprawdza przy tym czy istnieje podany dzień konferencji oraz rezerwacja konferencji.

```
CREATE PROCEDURE AddConferenceDayBooking
@confDayID INT,
 @bookingID INT,
 @participantsNo INT
AS
BEGIN
 IF (exists(SELECT ConfDayID FROM [Conference Days] WHERE
ConfDayID=@confDayID)
   AND exists(SELECT BookingID FROM [Conference Bookings] WHERE
@bookingID=BookingID)
   BEGIN
     SET NOCOUNT ON
     INSERT INTO [Conf Day Bookings] (ConfDayID, BookingID,
ParticipantsNo)
     VALUES (@confDayID,@bookingID,@participantsNo)
   END
 ELSE PRINT 'Nie można dodać! Nie istnieje taki dzien konferencji
```

```
lub podana rezerwacja konferencji'
 END
GO
```

AddConferenceBooking

Dodaje rezerwację konferencji, sprawdziwszy czy istnieje podana konferencja oraz klient.

```
CREATE PROCEDURE AddConferenceBooking
@confID INT,
  @clientID INT,
  @bookingDate DATETIME
AS
BEGIN
  IF (exists(SELECT ClientID FROM Clients WHERE
ClientID=@clientID)
    AND exists(SELECT ConferenceID FROM Conferences WHERE
@confID=ConferenceID))
    BEGIN
      SET NOCOUNT ON
      INSERT INTO [Conference
Bookings](ConferenceID,ClientID,isCancelled,CancelDate,BookingDate
)
        VALUES (@confID,@clientID,0,NULL,@bookingDate)
    END
  ELSE PRINT 'Nie istnieje podana konferencja lub klient. Nie
można dodać'
END
GO
```

AddDiscount

Dodaje nową zniżkę, sprawdziwszy czy istnieje podana konferencja której zniżka ma dotyczyć.

```
CREATE PROCEDURE AddDiscount
@ConferenceID INT,
```

```
 @daysUntilStart INT,
 @discountPercent INT
AS
BEGIN
 IF exists(SELECT ConferenceID FROM Conferences WHERE
@ConferenceID=ConferenceID)
   BEGIN
     SET NOCOUNT ON
     INSERT INTO Discounts (ConferenceID, DaysUntilStart,
DiscountPercent)
       VALUES (@ConferenceID,@daysUntilStart,@discountPercent)
   END
 ELSE PRINT 'Nie isnieje podana konferencja'
END
GO
```

AddPayment

Dodaje płatność za rezerwację. Jeśli podana rezerwacja nie istnieje wyskakuje błąd.

```
CREATE PROCEDURE AddPayment
@BookingID INT,
  @paymentDate DATETIME,
  @value MONEY
AS
BEGIN
  IF exists(SELECT BookingID FROM [Conference Bookings] WHERE
BookingID=@BookingID)
    BEGIN
      SET NOCOUNT ON
      INSERT INTO Payments (BookingID, PaymentDate, Value)
        VALUES (@BookingID,@paymentDate,@value)
    END
  ELSE PRINT 'Podana rezerwacja nie istnieje'
END
GO
```

CancelConference

Anuluję podaną konferencję, co oznacza również anulowanie rezerwacji konferencji, wszystkich powiązanych warsztatów oraz ich rezerwacji.

```
CREATE PROCEDURE CancelConference @ConferenceID INT
AS
  BEGIN
    SET NOCOUNT ON
    UPDATE Conferences
      SET Conferences.isCancelled=1
    WHERE Conferences.ConferenceID=@ConferenceID
    UPDATE [Conference Bookings]
      SET [Conference Bookings].isCancelled=1
    WHERE [Conference Bookings].ConferenceID=@ConferenceID
    DECLARE @ConfDayID INT
    SELECT @ConfDayID = ConfDayID FROM [Conference Days] WHERE
ConferenceID=@ConferenceID
    UPDATE Workshops
      SET Workshops.isCancelled=1
    WHERE Workshops.ConfDayID=@ConfDayID
    UPDATE [Workshop Booking]
      SET [Workshop Booking].isCancelled=1
    WHERE [Workshop Booking].WorkshopID=(SELECT WorkshopID FROM
Workshops WHERE ConfDayID=@ConfDayID)
  END
```

CancelConferenceBooking

Anuluje rezerwację konferencji. Również automatycznie anuluje powiązane rezerwacje warsztatów.

```
CREATE PROCEDURE CancelConferenceBooking @ConferenceBookingID INT
AS
  BEGIN
    SET NOCOUNT ON
    UPDATE [Conference Bookings]
      SET [Conference Bookings].isCancelled=1
    WHERE [Conference Bookings].BookingID=@ConferenceBookingID
    UPDATE [Workshop Booking]
      SET [Workshop Booking].isCancelled=1
```

```
    WHERE [Workshop Booking].ConfDayBookingID=
          (SELECT ConfDayBookingID FROM [Conf Day Bookings] WHERE
BookingID=@ConferenceBookingID)
  END
```

CancelWorkshop

Anuluje warsztaty, co sprawi również, że rezerwacje warsztatów zostaną anulowane.

```
CREATE PROCEDURE CancelWorkshop @WorkshopID INT
AS
  BEGIN
    SET NOCOUNT ON
    UPDATE Workshops
      SET Workshops.isCancelled=1
    WHERE WorkshopID=@WorkshopID
    UPDATE [Workshop Booking]
      SET [Workshop Booking].isCancelled=1
    WHERE WorkshopID=@WorkshopID
  END
```

CancelWorkshopBooking

Anuluje rezerwację danego warsztatu, na podstawie podanego ID warsztatu oraz ID rezerwacji dnia konferencji.

```
CREATE PROCEDURE CancelWorkshopBooking
 @WorkshopID INT,
 @ConfDayBookingID INT
AS
 BEGIN
   SET NOCOUNT ON
   UPDATE [Workshop Booking]
     SET [Workshop Booking].isCancelled=1
   WHERE WorkshopID=@WorkshopID AND
ConfDayBookingID=@ConfDayBookingID
 END
```

---

8. Triggery

participants_limit_for_conf_day_reached

Trigger **participants_limit_for_conf_day_reached** sprawdza, czy nie został przekroczony limit osób zapisanych na jakiś dzień konferencji.

```
CREATE TRIGGER participants_limit_for_conf_day_reached
ON [Conf Day Bookings]
AFTER INSERT, UPDATE
AS
BEGIN
IF EXISTS(
  SELECT conf_D.ConfDayID, conf_d.ParticipantsLimit
  FROM [Conference Days] as conf_D
  JOIN [Conf Day Bookings] as cdb
      ON conf_D.ConfDayID = cdb.ConfDayID
  GROUP BY conf_D.ConfDayID, conf_d.ParticipantsLimit
  HAVING sum(cdb.ParticipantsNo) > conf_d.ParticipantsLimit AND
conf_D.ConfDayID = (SELECT ConfDayID from INSERTED)
)
BEGIN
  RAISERROR('Przekroczony limit miejc na dzien konferencji',
16,1);
  ROLLBACK TRANSACTION
END
END
GO
```

participants_limit_for_workshop_reached

Trigger sprawdza, czy nie został przekroczony limit osób zapisanych na konferencję.

```
CREATE TRIGGER participants_limit_for_workshop_reached
ON [Workshop Booking]
AFTER INSERT, UPDATE
AS
BEGIN
IF EXISTS(
 SELECT W.WorkshopID, W.ParticipantsLimit, WB.WorkshopID
 FROM [Workshops] as W
```

```
 JOIN [Workshop Booking] as WB
     ON W.WorkshopID=WB.WorkshopID
 GROUP BY W.WorkshopID, W.ParticipantsLimit, WB.WorkshopID
 HAVING sum(WB.ParticipantsNo) > W.ParticipantsLimit
 AND WB.WorkshopID = (SELECT WorkshopID from INSERTED)
)
BEGIN
 RAISERROR('Przekroczony limit miejsc na warsztat', 16,1);
 ROLLBACK TRANSACTION
END
END
GO
```

TooManyParticipantsForAConfDayBooking

Trigger sprawdza, czy liczba uczestników wpisanych do tabeli Participants nie przekracza ilości zadeklarowanej w rezerwacji na dzień konferencji.

```
CREATE TRIGGER TooManyParticipantsForAConfDayBooking
ON [Participants]
FOR INSERT
AS
BEGIN

 DECLARE @ConfDayBookingID AS int
 SET @ConfDayBookingID = (SELECT ConfDayBookingID FROM inserted)

 DECLARE @NrOfParticipants AS INT
 SET @NrOfParticipants =
 (SELECT COUNT(*) FROM Participants WHERE ConfDayBookingID =
@ConfDayBookingID)

 DECLARE @ParticipantsDeclaration AS int
 SET @ParticipantsDeclaration =
 (SELECT ParticipantsNo FROM [Conf Day Bookings] WHERE
ConfDayBookingID= @ConfDayBookingID )
 IF (@ParticipantsDeclaration < @NrOfParticipants)
BEGIN
  RAISERROR('Liczba uczestnikow dnia konferencji przekracza liczbe
zadeklarowana w rezerwacji', -1,-1);
```

```
  ROLLBACK TRANSACTION
  END
 END
END
```

WorkshopParticipantsNumberCheck

Trigger sprawdzający, czy liczba podanych uczestników warsztatu nie przekracza liczby zadeklarowanej w rezerwacji.

```
CREATE TRIGGER WorkshopParticipantsNumberCheck ON [Workshop
Participants]
FOR INSERT AS
BEGIN
DECLARE @WorkshopID AS int
SET @WorkshopID = (SELECT WorkshopID FROM inserted)
DECLARE @ParticipantsNr AS int
SET @ParticipantsNr = (SELECT COUNT(*) FROM [Workshop
Participants] WHERE WorkshopID =
@WorkshopID)
DECLARE @ParticipantsDeclared AS int
SET @ParticipantsDeclared = (SELECT ParticipantsNo FROM [Workshop
Booking] WHERE WorkshopID =
@WorkshopID)
IF (@ParticipantsDeclared < @ParticipantsNr)
BEGIN
RAISERROR('Podano wiecej uczestnikow warsztatu niz to wynika z
rezerwacji', -1, -1);
ROLLBACK TRANSACTION
END
END
GO
```

Add_Workshop_Error

Trigger sprawdza, czy dodany warsztat nie posiada większego limitu miejsc niż limit miejsc na dzień konferencji.

```
CREATE TRIGGER Add_Workshop_Error ON Workshops
AFTER INSERT
AS
BEGIN
DECLARE @WorkshopLimit int;
DECLARE @ConfDayLimit int;
SET @WorkshopLimit = (SELECT ParticipantsLimit FROM INSERTED);
SET @ConfDayLimit = (SELECT ParticipantsLimit FROM [Conference
Days] WHERE ConfDayID = (SELECT ConfDayID FROM INSERTED));
IF(@ConfDayLimit < @WorkshopLimit)
BEGIN
RAISERROR('Error: dodanie limitu miejsc na warsztat wiekszego niz
limit miejc na dzien konferencji', 16, 1)
ROLLBACK TRANSACTION
END
END
```

WorkshopTime_ConfDay

Sprawdza czy data warsztatów zawiera się w dniu konferencji.

```
CREATE TRIGGER WorkshopTime_ConfDay ON Workshops
AFTER INSERT
AS
BEGIN
DECLARE @start DATETIME;
DECLARE @end DATETIME;
SET @start = (SELECT StartTime FROM [Conference Days] WHERE
 [Conference Days].ConfDayID=(SELECT ConfDayID FROM inserted));
SET @end = (SELECT EndTime FROM [Conference Days] WHERE
 [Conference Days].ConfDayID=(SELECT ConfDayID FROM inserted));
IF((SELECT StartTime FROM inserted)<@start or (SELECT EndTime FROM
inserted)>@end)
BEGIN
RAISERROR('Error: złe daty w stosunku do dni konferencji', 16, 1)
ROLLBACK TRANSACTION
END
END
GO
```

9. Funkcje

participants_of_a_workshop

Funkcja zwracająca tabelę, w której są osoby zapisane na dany warsztat (wg. id warsztatu). Lista zawiera ID, imię i nazwisko uczestnika.

```
CREATE FUNCTION participants_of_a_workshop (@WorkshopID INT)
RETURNS TABLE
AS
RETURN
  SELECT WP.ParticipantID as WorkshopParticipantID, P.FirstName as
FirstName, P.LastName as LastName
  FROM [Workshop Participants] as WP JOIN Participants as P ON
WP.ParticipantID = P.ParticipantID
  WHERE WP.WorkshopID = @WorkshopID
GO
```

GetWorkshopLimit

Funkcja zwracająca limit miejsc na warsztat na podstawie podanego ID warsztatu.

```
CREATE FUNCTION GetWorkshopLimit
(
@WorkshopID int
)
RETURNS int
AS
BEGIN
RETURN ISNULL ((SELECT ParticipantsLimit FROM Workshops
              WHERE WorkshopID = @WorkshopID),0)
END
```

GetConferenceID

Funkcja zwracająca ID konferencji, na podstawie podanego dnia konferencji.

```
CREATE FUNCTION GetConferenceID
(
@ConfDayID int
)
RETURNS int
AS
BEGIN
RETURN(SELECT ConferenceID FROM [Conference Days]
WHERE ConfDayID = @ConfDayID)
```

GetConferenceDayFreePlaces

Funkcja zwraca ilość wolnych miejsc na dany dzień konferencji

```
CREATE FUNCTION GetConferenceDayFreePlaces
 (@confDayID int)
RETURNS INT
AS
BEGIN
 DECLARE @limit int =
 (SELECT ParticipantsLimit FROM [Conference Days] WHERE ConfDayID
= @confDayID)

 DECLARE @booked int =
 (SELECT sum(ParticipantsNo) from [Conf Day Bookings] group by
ConfDayID having ConfDayID = @confDayID )

 RETURN @limit - @booked
END
```

GetWorkshopNrOfBookedPlaces

Zwraca ilość zarezerwowanych miejsc na dany warsztat

```
CREATE FUNCTION GetWorkshopNrOfBookedPlaces
 (@workshopID int)
RETURNS INT
AS
BEGIN
```

```
 DECLARE @number int =
 ISNULL((SELECT sum(ParticipantsNo) from [Workshop Booking] group
by WorkshopID having WorkshopID = @workshopID),0)

 RETURN @number
END
```

GetWorkshopFreePlaces

Zwraca ilość wolnych miejsc na warsztat.

```
CREATE FUNCTION GetWorkshopFreePlaces
 (@workshopID int)
RETURNS INT
AS
BEGIN
 DECLARE @limit int = (ISNULL ((SELECT ParticipantsLimit FROM
Workshops
              WHERE WorkshopID = @WorkshopID),0))

 DECLARE @booked int =
 (ISNULL((SELECT sum(ParticipantsNo) from [Workshop Booking] group
by WorkshopID having WorkshopID = @workshopID),0))

 RETURN @limit - @booked
END
GO
```

10. Indeksy

Oto stworzone indeksy, oraz kody w sql które je definiują:

- `conf_days_CONFID_index` dodaje indeks do ConferenceID tabeli Conference Days
- `workshop_booking_CONFDAYBOOKINGID` dodaje indeks do ConfDayID tabeli WorkshopBookings

- `workshops_CONFDAYID` dodaje indeks do ConfDayID tabeli Workshops
- `payments_BOOKINGID` dodaje indeks do BookingID tabeli Payments
- `conference_bookings_CONFERENCEID` dodaje indeks do ConferenceID tabeli Conference Bookings
- `conference_bookings_CLIENTID` dodaje indeks do ClientID tabeli Conference Bookings

```
CREATE INDEX conf_days_CONFID_index ON [Conference
Days](ConferenceID);
CREATE INDEX workshop_booking_CONFDAYBOOKINGID ON [Workshop
Booking](ConfDayBookingID);
CREATE INDEX workshops_CONFDAYID ON [Workshops](ConfDayID);
CREATE INDEX payments_BOOKINGID ON [Payments](BookingID);
CREATE INDEX conference_bookings_CONFERENCEID ON [Conference
Bookings](ConferenceID);
CREATE INDEX conference_bookings_CLIENTID ON [Conference
Bookings](ClientID);
```

11. Role w systemie

W systemie wyróżnia się następujące role:

- **Administrator** - posiada dostęp do wszystkich tabel, widoków, funkcji i procedur
- **Pracownik firmy** - posiada dostęp do procedur, funkcji i widoków
- **Klient** - posiada dostęp do procedur:
  - AddConferenceBooking
  - AddConferenceDayBooking
  - AddNewClientCompany
  - AddNewClientIndividual
  - AddNewParticipant
  - AddWorkshopBooking
  - AddWorkshopParticipant
  - CancelConferenceBooking
- Widoków:
  - conference_days_available
  - workshops_available

12. Generator danych

Dane zostały wygenerowane za pomocą programu **RedGate SQL Data Generator** (http://www.redgate.com/products/sql-development/sql-data-generator).
Podczas generowania danych zostały wyłączone następujące triggery:

```
DISABLE TRIGGER participants_limit_for_conf_day_reached ON [Conf
Day Bookings];
DISABLE TRIGGER Add_Workshop_Error ON Workshops;
DISABLE TRIGGER TooManyParticipantsForAConfDayBooking ON
[Participants];
DISABLE TRIGGER participants_limit_for_workshop_reached  ON
[Workshop Booking];
DISABLE TRIGGER WorkshopParticipantsNumberCheck ON [Workshop
Participants];
```