AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Podstawy Baz Danych

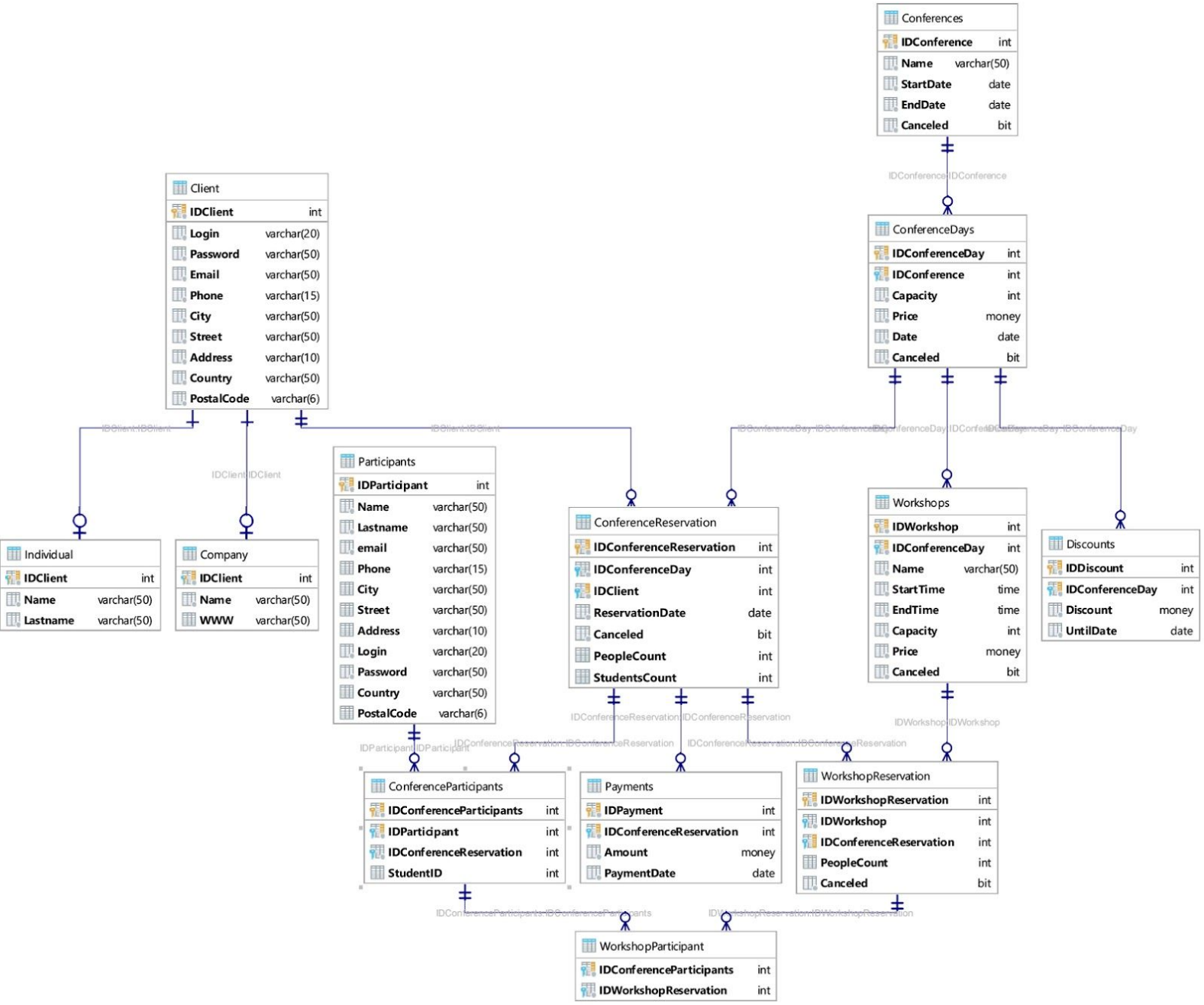
System zarządzania konferencjami - Dokumentacja

Grzegorz Różak **Paweł Zeller**

1. Wstęp

Celem projektu było zaprojektowanie oraz implementacja bazy danych dla firmy zajmującej się organizacją konferencji. Konferencje mogę być jedno lub kilkudniowe. Klienci dokonują rezerwacji na poszczególne dni konferencji oraz odbywające się na nich warsztaty, a następnie wprowadzają dane uczestników (do 2 tygodni przed konferencją). Poszczególne dni konferencji oraz warsztaty mogą być płatne. Cena warsztatów jest stała, natomiast cena dnia konferencji jest różna w zależności od daty rezerwacji. Im wcześniej została złożona rezerwacja, tym cena jest niższa. Dla konferencji jest również uwzględniana zniżka studencka. System będzie obsługiwany przez serwis internetowy, który powinien korzystać z procedur, funkcji udostępnionych przez bazę.

2. Schemat bazy danych



3. Tabele

Conferences

Tabela przechowuje podstawowe informacje o konferencji, nazwę, daty rozpoczęcia i zakończenia, oraz informacje dot. anulowania.

```
CREATE TABLE Conferences
(
 IDConference INT IDENTITY
   PRIMARY KEY,
 Name         VARCHAR(50)    NOT NULL,
 StartDate    DATE           NOT NULL,
 EndDate      DATE           NOT NULL,
 Canceled     BIT DEFAULT 0 NOT NULL
)
GO
```

Conference Days

Tabela przechowuje informacje o poszczególnych dniach konferencji, data, cena dnia, maksymalną ilość osób i informacje dot. anulowania dnia konferencji.

```
CREATE TABLE ConferenceDays
(
 IDConferenceDay INT IDENTITY
   PRIMARY KEY,
 IDConference    INT            NOT NULL
   CONSTRAINT FK_ConferenceDays_Conferences
   REFERENCES Conferences,
 Capacity        INT            NOT NULL,
 Price           MONEY          NOT NULL,
 Date            DATE           NOT NULL,
 Canceled        BIT DEFAULT 0 NOT NULL
)
GO
```

Discounts

Tabela przechowuje zniżki na dany dzień konferencji, datę do której obowiązuje dana zniżka i procent ceny, który należy zapłacić.

```
CREATE TABLE Discounts
(
 IDDiscount      INT IDENTITY
   PRIMARY KEY,
 IDConferenceDay INT   NOT NULL
   CONSTRAINT FK_Discounts_ConferenceDays
   REFERENCES ConferenceDays,
 Discount        MONEY NOT NULL,
 UntilDate       DATE  NOT NULL
)
```

Client

Tabela przechowuje informacje dot. klienta na konferencji, dane adresowe, kontaktowe, oraz dane do logowania do systemu. Jeśli dany klient jest osobą prywatną to dodatkowy rekord znajduje się również w tabeli Individual, jeśli jest firmą to w tabeli Company.

```
CREATE TABLE Client
(
 IDClient   INT IDENTITY
   PRIMARY KEY,
 Login      VARCHAR(20) NOT NULL,
 Password   VARCHAR(50) NOT NULL,
 Email      VARCHAR(50) NOT NULL
   CHECK ([Email] LIKE '%_@__%.__%'),
 Phone      VARCHAR(15) NOT NULL,
 City       VARCHAR(50) NOT NULL,
 Street     VARCHAR(50) NOT NULL,
 Address    VARCHAR(10) NOT NULL,
 Country    VARCHAR(50) NOT NULL,
 PostalCode VARCHAR(6)  NOT NULL
)
GO
```

Individual

Tabela przechowuje dane charakterystyczne dla indywidualnego klienta, imię i nazwisko. Klucz główny tej tabeli odpowiada kluczowi głównemu w tabeli Clients.

```
CREATE TABLE Individual
(
 IDClient INT         NOT NULL
   PRIMARY KEY
   CONSTRAINT FK_Individual_Client
   REFERENCES Client,
 Name     VARCHAR(50) NOT NULL,
 Lastname VARCHAR(50) NOT NULL
)
GO
```

Company

Tabela przechowuje dane dot. firmy, jako klienta, nazwę firmy i stronę internetową. Klucz główny tej tabeli odpowiada kluczowi głównemu w tabeli Clients.

```
CREATE TABLE Company
(
 IDClient INT         NOT NULL
   PRIMARY KEY
   CONSTRAINT FK_Company_Client
   REFERENCES Client,
 Name     VARCHAR(50) NOT NULL,
 WWW      VARCHAR(50)
   CHECK ([WWW] LIKE '%_.__%')
)
```

Participants

Tabela przechowuje dane uczestników konferencji. Obowiązkowo przechowywane są imię i nazwisko, email oraz dane logowania. Dane adresowe oraz telefon nie są obowiązkowe.

```
CREATE TABLE Participants
(
 IDParticipant INT IDENTITY
   PRIMARY KEY,
 Name          VARCHAR(50) NOT NULL,
 Lastname      VARCHAR(50) NOT NULL,
 email         VARCHAR(50) NOT NULL
   CHECK ([Email] LIKE '%_@__%.__%'),
 Phone         VARCHAR(15),
 City          VARCHAR(50),
 Street        VARCHAR(50),
 Address       VARCHAR(10),
 Login         VARCHAR(20) NOT NULL,
 Password      VARCHAR(50) NOT NULL,
 Country       VARCHAR(50),
 PostalCode    VARCHAR(6)
)
```

ConferenceReservation

Tabela przechowuje informacje o rezerwacjach składanych przez klientów na poszczególne dni konferencji - datę rezerwacji, ilość zapisanych osób oraz ewentualne anulowanie. Domyślnie data rezerwacji ustalana jest na datę dodania rekordu. Liczba zapisanych osób musi być liczbą dodatnią.

```
CREATE TABLE ConferenceReservation
(
 IDConferenceReservation INT IDENTITY
   PRIMARY KEY,
 IDConferenceDay         INT                    NOT NULL
   CONSTRAINT FK_ConferenceReservation_ConferenceDays
   REFERENCES ConferenceDays,
 IDClient                INT                    NOT NULL
   CONSTRAINT FK_ConferenceReservation_Client
   REFERENCES Client,
 ReservationDate         DATE DEFAULT getdate() NOT NULL,
 PeopleCount             INT
   CHECK ([PeopleCount] > 0),
 Canceled                BIT DEFAULT 0          NOT NULL
)
GO
```

Payments

Tabela przechowuje informacje o wpłatach dla danej rezerwacji konferencji - kwotę oraz datę wykonania płatności. Domyślnie data wpłaty ustalana jest na datę dodania rekordu. Kwota musi być dodatnia.

```
CREATE TABLE Payments
(
 IDPayment               INT IDENTITY
   PRIMARY KEY,
 IDConferenceReservation INT                    NOT NULL
   CONSTRAINT FK_Payments_ConferenceReservation
   REFERENCES ConferenceReservation,
 Amount                  MONEY                  NOT NULL
   CHECK ([Amount] > 0),
 PaymentDate             DATE DEFAULT getdate() NOT NULL
)
GO
```

ConferenceParticipants

Tabela przechowuje informacje o uczestnikach poszczególnych dni konferencji. Numer rezerwacji, z której został on zapisany, odnośnik do jego danych w postaci klucza obcego z tabeli Participants oraz ewentualny numer legitymacji studenckiej.

```
CREATE TABLE ConferenceParticipants
(
 IDConferenceParticipants INT IDENTITY
   PRIMARY KEY,
 IDParticipant            INT NOT NULL
   CONSTRAINT FK_ConferenceParticipants_Participants
   REFERENCES Participants,
 IDConferenceReservation  INT NOT NULL
   CONSTRAINT FK_ConferenceParticipants_ConferenceReservation
   REFERENCES ConferenceReservation,
 StudentID                INT
   CHECK ([StudentID] > 99)
)
GO
```

Workshops

Tabela przechowuje dane o warsztatach jakie są organizowane. Dzień, w którym warsztat się odbywa, nazwę warsztatu, godziny rozpoczęcia i zakończenia, ilość miejsc, cenę, która może wynosić 0 w przypadku, kiedy warsztat jest darmowy. Warsztat może zostać również odwołany, co umożliwia pole Canceled.

```
CREATE TABLE Workshops
(
 IDWorkshop      INT IDENTITY
   PRIMARY KEY,
 IDConferenceDay INT           NOT NULL
   CONSTRAINT FK_Workshops_ConferenceDays
   REFERENCES ConferenceDays,
 Name            VARCHAR(50)   NOT NULL,
 StartTime       TIME          NOT NULL,
```

```
 EndTime          TIME          NOT NULL,
 Capacity         INT           NOT NULL
   CHECK ([Capacity] > 0),
 Price            MONEY         NOT NULL
   CHECK ([Price] >= 0),
 Canceled         BIT DEFAULT 0 NOT NULL,
 CONSTRAINT end_time_after_start_CK
 CHECK ([StartTime] < [EndTime])
)
GO
```

WorkshopReservation

Tabela przechowuje informacje o rezerwacjach na poszczególne warsztaty. Są tworzone na podstawie uprzedniej rezerwacji konferencji oraz zawierają dodatnią liczbę osób i informacje o anulowaniu.

```
CREATE TABLE WorkshopReservation
(
 IDWorkshopReservation    INT IDENTITY
   PRIMARY KEY,
 IDWorkshop               INT           NOT NULL
   CONSTRAINT FK_WorkshopReservation_Workshops
   REFERENCES Workshops,
 IDConferenceReservation INT
   CONSTRAINT FK_RezerwacjaWarsztatu_RezerwacjaKonferencji
   REFERENCES ConferenceReservation,
 PeopleCount              INT
   CHECK ([PeopleCount] > 0),
 Canceled                 BIT DEFAULT 0 NOT NULL
)
GO
```

WorkshopParticipant

Tabela przechowuje uczestników na dane warsztaty. Jej rekordy odnoszą się do rezerwacji danego warsztatu oraz do uczestnika konferencji.

```
CREATE TABLE WorkshopParticipant
(
 IDConferenceParticipants INT NOT NULL
   CONSTRAINT FK_WorkshopParticipant_ConferenceParticipants
   REFERENCES ConferenceParticipants,
 IDWorkshopReservation    INT NOT NULL
   CONSTRAINT FK_WorkshopParticipant_WorkshopReservation
   REFERENCES WorkshopReservation
)
GO
```

4. Widoki

MissingParticipantData
Widok pokazuje tabelę, w której znajdują się dane wszystkich klientów, którzy nie uzupełnili swoich rezerwacji poprzez podanie odpowiedniej liczby uczestników. Wyświetla ilość pozostałych dni do konferencji, liczbę zapisanych osób oraz liczbę brakujących.

```
CREATE VIEW MissingParticipantData AS
 SELECT
   ConferenceReservation.IDClient,
   (
     SELECT Client.Phone
     FROM Client
     WHERE Client.IDClient = ConferenceReservation.IDClient
   )
AS 'Phone',
   isnull(
       (SELECT Individual.Name
        FROM Individual
          JOIN Client ON Individual.IDClient = Client.IDClient
        WHERE Client.IDClient = ConferenceReservation.IDClient
       ),
       (
         SELECT Company.Name
         FROM Company
           JOIN Client ON Company.IDClient = Client.IDClient
         WHERE Client.IDClient = ConferenceReservation.IDClient
       )
   )
AS 'Name',

   ConferenceReservation.IDConferenceReservation,
   ABS(DATEDIFF(DAY, ConferenceReservation.ReservationDate, Conferences.StartDate))
AS 'Days To Conference',
   ConferenceReservation.IDConferenceDay,
   ConferenceReservation.PeopleCount
AS 'People requested',
   (count(ConferenceParticipants.IDParticipant))
AS 'People registered'

 FROM dbo.ConferenceReservation
   JOIN ConferenceDays ON ConferenceReservation.IDConferenceDay =
ConferenceDays.IDConferenceDay
   JOIN Conferences ON ConferenceDays.IDConference = Conferences.IDConference
   FULL JOIN ConferenceParticipants
     ON ConferenceReservation.IDConferenceReservation =
ConferenceParticipants.IDConferenceReservation

 WHERE (ABS(DATEDIFF(DAY, ConferenceReservation.ReservationDate,
Conferences.StartDate)) < 14)
```

```
 GROUP BY
   ConferenceReservation.IDClient,
   ConferenceReservation.IDConferenceReservation,
   ConferenceReservation.IDConferenceDay,
   ConferenceReservation.ReservationDate, Conferences.StartDate,
   ConferenceReservation.PeopleCount
 HAVING (count(ConferenceParticipants.IDParticipant) <
ConferenceReservation.PeopleCount)
GO
```

CompanyOverview

Widok wyświetla dane wszystkich firm, które były klientami.

```
CREATE VIEW CompanyOverview AS
 SELECT
   Client.IDClient,
   Client.Login,
   Client.Password,
   Client.Email,
   Company.WWW,
   Client.Country,
   Client.City,
   Client.Street,
   Client.Address,
   'Company' AS "Type"
 FROM dbo.Client
   JOIN Company ON Client.IDClient = Company.IDClient
GO
```

IndividualOverview

Widok wyświetla dane wszystkich klientów indywidualnych.

```
CREATE VIEW IndividualOverview AS
 SELECT
   Individual.Name,
   Individual.Lastname,
   Client.IDClient,
   Client.Login,
   Client.Password,
   Client.Email,
   Client.Country,
   Client.City,
   Client.Street,
   Client.Address,
   'Individual' AS "Type"
 FROM dbo.Client
   JOIN Individual ON Client.IDClient = Individual.IDClient
GO
```

ConferenceParticipantsOverview

Widok wyświetla dane wszystkich uczestników.

```
CREATE VIEW ConferenceParticipantsOverview AS
 SELECT
   Participants.Name,
   Participants.Street,
   Participants.City,
   Participants.email,
   Participants.Login,
   Participants.Password,
   Conferences.Name AS 'Conference Name',
   ConferenceDays.IDConferenceDay
 FROM dbo.Participants
   JOIN ConferenceParticipants ON ConferenceParticipants.IDParticipant =
Participants.IDParticipant
   JOIN ConferenceReservation
     ON ConferenceReservation.IDConferenceReservation =
ConferenceParticipants.IDConferenceReservation
   LEFT OUTER JOIN ConferenceDays ON ConferenceDays.IDConferenceDay =
ConferenceReservation.IDConferenceDay
   LEFT OUTER JOIN Conferences ON ConferenceDays.IDConference =
Conferences.IDConference
 WHERE Conferences.EndDate >= (GETDATE())
GO
```

CanceledWorkshopReservation

Wyświetla listę rezerwacji warsztatów, które zostały anulowane. Pokazuje informacje o rezerwacji warsztatu oraz ID klienta, który ją złożył.

```
CREATE VIEW CanceledWorkshopsReservation AS
 SELECT
   WorkshopReservation.IDWorkshopReservation,
   WorkshopReservation.IDWorkshop,
   WorkshopReservation.PeopleCount,
   ConferenceReservation.IDConferenceReservation,
   ConferenceReservation.IDClient
 FROM dbo.WorkshopReservation
   JOIN ConferenceReservation
     ON WorkshopReservation.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
 WHERE (WorkshopReservation.Canceled) = 1
GO
```

ChargeForClientPerDay

```
CREATE VIEW ChargeForClientPerDay AS
 SELECT
   C.IDClient                                AS 'Client ID',
   CON.Name                                  AS 'Conference',
   CD.IDConferenceDay                        AS 'Days',
```

```
   CR.PeopleCount                              AS 'People Registered',
   ISNULL(CR.PeopleCount * CD.Price,0)         AS 'Charge for entrance',
   ISNULL(sum(WR.PeopleCount * WOR.Price),0) AS 'Charge for Workshop'
 FROM dbo.ConferenceReservation AS CR
   FULL JOIN ConferenceDays AS CD ON CR.IDConferenceDay = CD.IDConferenceDay
   JOIN Conferences AS CON ON CON.IDConference = CR.IDConferenceDay
   JOIN Client AS C ON CR.IDClient = C.IDClient
   LEFT JOIN WorkshopReservation AS WR ON CR.IDConferenceReservation =
WR.IDConferenceReservation
   LEFT JOIN Workshops AS WOR ON WR.IDWorkshop = WOR.IDWorkshop

 WHERE (ABS(DATEDIFF(DAY, CR.ReservationDate, CON.StartDate)) < 14)
 GROUP BY
   C.IDClient,
   CON.Name,
   CD.IDConferenceDay,
   CR.PeopleCount,
   CR.PeopleCount,
   CD.Price
GO
```

WorkshopOverview

Wyświetla listę z danymi ostatnich aktywnych i anulowanych warsztatów posortowaną według daty i godziny.

```
CREATE VIEW WorkshopOverview AS

SELECT TOP 20
    W.IDWorkshop                                                AS "Workshop ID",
    W.Name                                                      AS "Workshop name",
    W.IDConferenceDay                                           AS "Conference Day
ID",
    (SELECT Conferences.Name
     FROM Conferences
       JOIN ConferenceDays ON Conferences.IDConference =
ConferenceDays.IDConference
     WHERE ConferenceDays.IDConferenceDay = W.IDConferenceDay) AS "Conference",
    W.StartTime                                                 AS "Start time",
    W.EndTime                                                   AS "End time",
     (SELECT ConferenceDays.Date from ConferenceDays
         where W.IDConferenceDay = ConferenceDays.IDConferenceDay )AS "Date",
    W.Price                                                     AS "Entry price",
    'Active'                                                    AS "Availability"

  FROM dbo.Workshops AS W
  WHERE (W.Canceled) = 0
  GROUP BY W.IDConferenceDay, W.StartTime, W.Price, W.EndTime, W.IDWorkshop, W.Name

 UNION

     SELECT top 10
    W.IDWorkshop                                                AS "Workshop ID",
    W.Name                                                      AS "Workshop name",
    W.IDConferenceDay                                           AS "Conference Day
ID",
```

```
    (SELECT Conferences.Name
     FROM Conferences
       JOIN ConferenceDays ON Conferences.IDConference =
ConferenceDays.IDConference
     WHERE ConferenceDays.IDConferenceDay = W.IDConferenceDay) AS "Conference",
    W.StartTime                                                AS "Start time",
    W.EndTime                                                  AS "End time",
       (SELECT ConferenceDays.Date from ConferenceDays
         where W.IDConferenceDay = ConferenceDays.IDConferenceDay )AS "Date",
    W.Price                                                    AS "Entry price",
    'Canceled'                                                   AS "Availability"

  FROM dbo.Workshops AS W
  WHERE (W.Canceled) = 1
  GROUP BY W.IDConferenceDay, W.StartTime, W.Price, W.EndTime, W.IDWorkshop, W.Name
  ORDER BY Date DESC, W.StartTime DESC
go
```

UpcomingConferences

Wyświetla listę wszystkich dni konferencji, które są zarejestrowane w bazie, ale jeszcze się nie odbyły/skończyły.

```
CREATE VIEW UpcomingConferences AS
 SELECT
   c.Name,
   c.StartDate,
   c.EndDate,
   days.IDConferenceDay,
   isnull((
            SELECT SUM(ConferenceParticipants.IDParticipant)
            FROM ConferenceParticipants
              JOIN ConferenceReservation
                ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
            WHERE days.IDConferenceDay = ConferenceReservation.IDConferenceDay
          ), 0) AS 'Participants'

 FROM Conferences AS c
   FULL JOIN ConferenceDays AS days ON c.IDConference = days.IDConference
 WHERE (c.EndDate >= GETDATE())
 GROUP BY c.IDConference, c.Name, c.StartDate, c.EndDate, days.IDConferenceDay
GO
```

ConferenceDayParticipantsList

Wyświetla listę wszystkich uczestników z podziałem na dni konferencji i rezerwacje konfernecji.

```
CREATE VIEW ConferenceDayParticipantsList AS
 SELECT
   ConferenceDays.IDConferenceDay,
   ConferenceReservation.IDConferenceReservation,
   Participants.Name,
   Participants.Lastname,
   Participants.email
```

```
 FROM ConferenceDays
   JOIN ConferenceReservation ON ConferenceDays.IDConferenceDay =
ConferenceReservation.IDConferenceDay
   JOIN ConferenceParticipants
     ON ConferenceReservation.IDConferenceReservation =
ConferenceParticipants.IDConferenceReservation
   JOIN Participants ON ConferenceParticipants.IDParticipant =
Participants.IDParticipant
 GROUP BY ConferenceDays.IDConferenceDay,
   ConferenceReservation.IDConferenceReservation,
   Participants.Name,
   Participants.Lastname,
   Participants.email
GO
```

PopularClients

Wyświetla listę klientów posortowaną według ilości w sumie złożonych rezerwacji przez danego klienta.

```
CREATE VIEW PopularClients AS
 SELECT TOP 100
   c.IDClient,
   c.Login,
   (
     isnull(
         (SELECT name
          FROM Individual
          WHERE Individual.IDClient = c.IDClient),
         (SELECT name
          FROM Company
          WHERE Company.IDClient = c.IDClient)
     )
   ) AS 'Customer Name',
   (
     SELECT count(ConferenceReservation.IDConferenceReservation)
     FROM ConferenceReservation
     WHERE ConferenceReservation.IDClient = c.IDClient
   ) AS 'No. of reservation'

 FROM Client AS c
 ORDER BY
   (
     SELECT count(ConferenceReservation.IDConferenceReservation)
     FROM ConferenceReservation
     WHERE ConferenceReservation.IDClient = c.IDClient
   ) DESC
GO
```

ConferenceReservationPaymentOverview

Wyświetla płatności dla wszystkich nieanulowanych rezerwacji konferencji - Kwotę należną, zapłaconą oraz pozostałą.

```
CREATE VIEW ConferenceReservationPaymentOverview AS
 SELECT
   Conferences.Name,
```

```
   Conferences.IDConference,
   ConferenceDays.IDConferenceDay,
   ConferenceDays.Price,
   ConferenceReservation.IDConferenceReservation,
   (dbo.count_paid(ConferenceReservation.IDConferenceReservation))    AS 'Paid
amount',
   (dbo.count_payment(ConferenceReservation.IDConferenceReservation)) AS 'Due
amount'
 FROM Conferences
   JOIN ConferenceDays ON Conferences.IDConference = ConferenceDays.IDConference
   JOIN ConferenceReservation ON ConferenceDays.IDConferenceDay =
ConferenceReservation.IDConferenceDay
 WHERE Conferences.Canceled = 0 AND ConferenceDays.Canceled = 0 AND
ConferenceReservation.Canceled = 0
GO
```

5. Widoki z parametrem

Payments From Client

Wyświetla płatności otrzymane od danego klienta (id klienta)

```
CREATE PROCEDURE VIEW_payments_from_client(@id INT)
AS
 (
   SELECT
     Client.IDClient,
     (
       (SELECT Individual.Name
        FROM Individual
        WHERE Individual.IDClient = Client.IDClient)
       + (SELECT Individual.Lastname
          FROM Individual
          WHERE Individual.IDClient = Client.IDClient)
     )
                          AS 'Name',
     ConferenceReservation.IDConferenceReservation,
     ConferenceDays.IDConferenceDay,
     Conferences.Name     AS 'Conference Name',
     Payments.Amount,
     Payments.PaymentDate AS 'Payment date'

   FROM Client
     LEFT JOIN ConferenceReservation ON Client.IDClient =
ConferenceReservation.IDClient
     JOIN ConferenceDays ON ConferenceReservation.IDConferenceDay =
ConferenceDays.IDConferenceDay
     JOIN Conferences ON ConferenceDays.IDConference = Conferences.IDConference
```

```
    JOIN Payments ON ConferenceReservation.IDConferenceReservation =
Payments.IDConferenceReservation
   WHERE Client.IDClient = @id
   GROUP BY Client.IDClient, ConferenceReservation.IDConferenceReservation,
     ConferenceDays.IDConferenceDay, Conferences.Name, Payments.Amount,
Payments.PaymentDate
 )
GO
```

Payments For Conference

Wyświetla listę płatności za daną konferencję.

```
CREATE PROCEDURE VIEW_payments_for_conference(@id INT)
AS
 (
   SELECT
     Conferences.IDConference                      AS 'Conference id',
     Conferences.Name                              AS 'Conference Name',
     ConferenceDays.IDConferenceDay                AS 'Conference day id',
     ConferenceDays.Date                              'Conference day date',
     ConferenceReservation.IDConferenceReservation AS 'Conference reservation id',
     Client.IDClient                               AS 'Client id',
     Payments.Amount                               AS 'Paid amount',
     Payments.PaymentDate                          AS 'Payment date'

   FROM Conferences
     JOIN ConferenceDays ON Conferences.IDConference = ConferenceDays.IDConference
     LEFT JOIN ConferenceReservation ON ConferenceDays.IDConferenceDay =
ConferenceReservation.IDConferenceDay
     JOIN Client ON ConferenceReservation.IDClient = Client.IDClient
     LEFT JOIN Payments ON ConferenceReservation.IDConferenceReservation =
Payments.IDConferenceReservation
   WHERE Conferences.IDConference = @id
 )
GO
```

Prices For Conference

Pokazuje listę obowiązujących cen za każdy dzień danej konferencji, dokładne informacje o danej konferencji, uwzględnia zniżki i ich daty obowiązywania

```
CREATE PROCEDURE VIEW_prices_for_conference(@id INT)
AS
 (
   SELECT
     Conferences.IDConference       AS 'Conference id',
     Conferences.Name               AS 'Conference Name',
     Conferences.StartDate          AS 'Start date',
     Conferences.EndDate            AS 'End date',
     ConferenceDays.IDConferenceDay AS 'Conference day id',
     ConferenceDays.Date            AS 'Conference day date',
     ConferenceDays.Price           AS 'Price',
```

```
      Discounts.UntilDate            AS 'Last discount day',
      (1 - Discounts.Discount) * 100 AS 'Discount [%]'

   FROM Conferences
     JOIN ConferenceDays ON Conferences.IDConference = ConferenceDays.IDConference
     LEFT JOIN Discounts ON ConferenceDays.IDConferenceDay =
Discounts.IDConferenceDay

   WHERE Conferences.IDConference = @id
 )
GO
```

Participants For Conference Day

Pokazuje listę uczestników na dany dzień konferencji (otrzymując jako parametr id dnia konferencji)

```
CREATE PROCEDURE VIEW_participants_for_conference_day(@id INT)
AS
 (
   SELECT
     ConferenceDays.IDConferenceDay,
     ConferenceDays.Date,
     Conferences.Name,
     Participants.Name,
     Participants.Lastname,
     Participants.Phone,
     Participants.email,
     Participants.Address,
     Participants.Street,
     Participants.City,
     Participants.Country
   FROM ConferenceDays
     JOIN Conferences ON ConferenceDays.IDConference = Conferences.IDConference
     JOIN ConferenceReservation ON ConferenceDays.IDConferenceDay =
ConferenceReservation.IDConferenceDay
     JOIN ConferenceParticipants
       ON ConferenceReservation.IDConferenceReservation =
ConferenceParticipants.IDConferenceReservation
     JOIN Participants ON ConferenceParticipants.IDParticipant =
Participants.IDParticipant
   WHERE ConferenceDays.IDConferenceDay = @id

 )
GO
```

Generate ID

Procedura wyświetlająca dane do identyfikatorów dla uczestników na daną konferencję (jako parametr otrzymuje id konferencji)

```
CREATE PROCEDURE VIEW_GENERATE_ID(@id INT)
AS
```

```
 RETURN SELECT
          ROW_NUMBER()
          OVER (
            ORDER BY Participants.IDParticipant ASC ) AS ID,
          Participants.Name,
          Participants.Lastname,
          Participants.email,
          Conferences.Name                              AS 'Conference name'
        FROM Participants
          JOIN ConferenceParticipants ON Participants.IDParticipant =
ConferenceParticipants.IDParticipant
          JOIN ConferenceReservation
            ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
          JOIN ConferenceDays ON ConferenceReservation.IDConferenceDay =
ConferenceDays.IDConferenceDay
          JOIN Conferences ON ConferenceDays.IDConference =
Conferences.IDConference
        WHERE @id = ConferenceDays.IDConference
GO


CREATE PROCEDURE VIEW_participants_for_Workhop(@IDWorkshop INT)
AS
 BEGIN
   SELECT
     Participants.Name,
     Participants.Lastname,
     Participants.email,
     Participants.Phone,
     Participants.City,
     Participants.street,
     Participants.Address,
     Participants.PostalCode,
     Participants.Country
   FROM WorkshopParticipant
     JOIN ConferenceParticipants
       ON ConferenceParticipants.IDConferenceParticipants =
WorkshopParticipant.IDConferenceParticipants
     JOIN Participants
       ON Participants.IDParticipant = ConferenceParticipants.IDParticipant
     JOIN WorkshopReservation
       ON WorkshopParticipant.IDWorkshopReservation =
WorkshopReservation.IDWorkshopReservation
     JOIN Workshops
       ON Workshops.IDWorkshop = WorkshopReservation.IDWorkshop
   WHERE Workshops.IDWorkshop = @IDWorkshop
 END
GO
```

Most Attednded Workshops

Widok pokazuje listę warsztatów odbywających się podczas danej konferencji, porządkując je ze względu na ilość uczestników.

```
CREATE PROCEDURE VIEW_most_attended_workshops(@IDConference INT = NULL)
AS
 BEGIN
   IF @IDConference IS NOT NULL
     BEGIN
       SELECT
         Conferences.Name AS 'ConferenceName',
         Workshops.Name   AS 'WorkshopName',
         count(*)         AS 'attendees'
       FROM ConferenceDays
         JOIN Workshops
           ON Workshops.IDConferenceDay = ConferenceDays.IDConferenceDay
         JOIN Conferences
           ON Conferences.IDConference = ConferenceDays.IDConference
         JOIN workshopreservation
           ON WorkshopReservation.IDWorkshop = Workshops.IDWorkshop
         JOIN WorkshopParticipant
           ON WorkshopParticipant.IDWorkshopReservation =
WorkshopReservation.IDWorkshopReservation
       WHERE Conferences.IDConference = @IDConference AND Workshops.Canceled = 0
       GROUP BY Conferences.Name, Workshops.Name
       ORDER BY count(*) DESC
     END
   ELSE
     BEGIN
       SELECT
         Conferences.Name AS 'ConferenceName',
         Workshops.Name   AS 'WorkshopName',
         count(*)         AS 'attendees'
       FROM Workshops
         JOIN ConferenceDays
           ON Workshops.IDConferenceDay = ConferenceDays.IDConferenceDay
         JOIN Conferences
           ON Conferences.IDConference = ConferenceDays.IDConference
         JOIN workshopreservation
           ON WorkshopReservation.IDWorkshop = Workshops.IDWorkshop
         JOIN WorkshopParticipant
           ON WorkshopParticipant.IDWorkshopReservation =
WorkshopReservation.IDWorkshopReservation
       WHERE Workshops.Canceled = 0
       GROUP BY Conferences.Name, Workshops.Name
       ORDER BY count(*) DESC
     END
 END
GO
```

Workshop Reservations For Conference Reservations
Widok pokazuje listę rezerwacji warsztatów z danej rezerwacji konferencji (jako parametr bierze id rezerwacji konferencji)

```
CREATE PROCEDURE
VIEW_workshop_reservations_for_ConferenceReservation(@IDConferenceReservation INT)
AS
 BEGIN
   SELECT
     Workshops.Name,
     WorkshopReservation.Canceled,
     WorkshopReservation.PeopleCount
   FROM WorkshopReservation
     JOIN Workshops
       ON Workshops.IDWorkshop = WorkshopReservation.IDWorkshop
   WHERE WorkshopReservation.IDConferenceReservation = 1
   ORDER BY WorkshopReservation.canceled, WorkshopReservation.IDWorkshop
 END
GO
```

6. Procedury i Funkcje

Lista dostępnych procedur i funkcji:

6.1 Procedury dodające:

- AddConference
- AddParticipant
- AddPricePoint
- AddConferenceReservation
- AddConferenceDay
- AddWorkshop
- AddToPayment
- AddWorkshopReservation
- AddConferenceParticipant
- AddClient

6.2 Procedury edytujące:

- CancelWorkshopReservation
- CancelWorkshop
- MakeWorkshopReservationActive
- MakeWorkshopActive
- Sign_to_workshop
- User_sign_to_workshop
- EditClient

- EditParticipant
- EditWorkshopCapacity
- EditConferenceDayCapacity
- EditPeopleCountInWorkshopReservation
- MakeConferenceReservationActive
- CancelConferenceReservation
- CancelConferenceDay
- CancelConference
- EditPeopleCountInConferenceReservation
- EditPeopleCountInReservationUSERLEVEL
- DeleteConferenceParticipant
- DeleteWorkshopParticipant

6.3 Funkcje i procedury pomocnicze

- Students_Count
- Find_price_per_day
- Price_workshop
- IsClientACompany
- Count_payment_for_workshops
- Count_payment
- Cancel_unpaid_reservation
- FreePlacesForConferenceDay
- FreePlacesForWorkshop
- Count_paid
- Count_payment_ESTIMATED

6.1 Procedury dodajace

AddConference

Procedura dodaje nową konferencje do bazy danych.

```
CREATE PROCEDURE dbo.addConference
 @Name varchar(50),
 @StartDate date,
 @EndDate date
AS
BEGIN

 INSERT INTO Conferences(
   Name,StartDate,EndDate
 )
 VALUES(@Name, @StartDate, @EndDate)
 END
go
```

AddParticipant

Procedura dodająca do bazy danych nowego uczestnika.

```
CREATE PROCEDURE dbo.addParticipant
   @Name varchar(50),
   @Lastname varchar(50),
   @email varchar(50),
   @Phone varchar(15) ,
   @City varchar(50) ,
   @Street varchar(50),
   @Address varchar(10),
   @Login varchar(20),
   @Password varchar(50),
   @Country varchar(50)
AS
BEGIN

 INSERT INTO Participants(
   Name,Lastname,email,Phone,City,Street,Address,Login,Password,Country
 )
VALUES(@Name, @Lastname,
@email,@Phone,@City,@Street,@Address,@Login,@Password,@Country)
 END
go
```

AddPricePoint

Procedura dodaje nowy próg cenowy do podanego dnia konferencji.

```
CREATE PROCEDURE dbo.addPricePoint
 @DayID INT,
 @date DATE,
 @price MONEY
AS
BEGIN
SET NOCOUNT ON
IF @date > GETDATE()
 BEGIN
 INSERT INTO Discounts(
  IDConferenceDay, Discount, UntilDate
 )
 VALUES(@DayID,@price,@date)
 END
 ELSE
PRINT 'Cant change past data'
end
go
```

AddConferenceReservation

Procedura dodaje nową rezerwacje na podaną konferencję na podstawie danego clientID.

```
CREATE PROCEDURE dbo.addConferenceReservation
 @dayid INT,
```

```
 @Clientid INT,
 @count INT

AS
BEGIN

 INSERT INTO ConferenceReservation(
  IDConferenceDay, IDClient, ReservationDate, Canceled, PeopleCount
 )
 VALUES(@dayid,@Clientid,getdate(),0,@count)
 END
go
```

AddConferenceDay

Procedura dodaje nowy dzień do już istniejącej konferencji.

```
CREATE PROCEDURE dbo.addConferenceDay
 @IdConference int,
 @capacity int,
 @price FLOAT,
 @data DATE
AS
BEGIN
 INSERT INTO ConferenceDays(
   IDConference, Capacity, Price,Date
 )
 VALUES( @IdConference, @capacity, @price,@data)
 END
go
```

AddWorkshop

Procedura dodaje nowy warsztat do podanego dnia konferencji.

```
CREATE PROCEDURE dbo.addWorkshop
 @IDConferenceDay     int,
 @Name   VARCHAR(50),
 @StartTime     TIME,
 @EndTime       TIME,
 @Capacity     VARCHAR(50),
 @Price   float,
 @Canceled BIT

AS
BEGIN
 INSERT INTO Workshops(
   IDConferenceDay, Name, StartTime, EndTime, Capacity, Price, Canceled
 )
 VALUES(@IDConferenceDay,@Name,@StartTime,@EndTime,@Capacity,@Price,@Canceled)
 END
go
```

AddToPayment

Procedura dodaje nową zapłatę od klienta za daną rezerwację konferencji.

```
CREATE PROCEDURE dbo.addtoPayment
 @ConferenceID int,
 @Amount MONEY,
 @Date   DATE
 as
 BEGIN
   INSERT INTO Payments(
    IDConferenceReservation, Amount, PaymentDate
   )
   VALUES (@ConferenceID,@Amount,@Date)
 END
go
```

AddWorkshopReservation

Procedura dodaje nową rezerwację warsztatu na podstawie podanej rezerwacji konferencji.

```
CREATE PROCEDURE addWorkshopReservation
 @IDWorkshop INT,
 @IDConferenceReservation INT,
 @PeopleCount INT

AS
BEGIN

 INSERT INTO WorkshopReservation(
  IDWorkshop, Canceled, IDConferenceReservation, PeopleCount
 )
 VALUES(@IDWorkshop,0, @IDConferenceReservation,@PeopleCount)
 END
```

AddConferenceParticipant

Procedura dodaje dane nowego uczestnika do rezerwacji konferencji (wpisuje uczestnika do rezerwacji)

```
CREATE PROCEDURE dbo.addConferenceParticipant
 @IDParticipant      int,
 @IDConferenceReservation   int,
 @StudentsID int =null
AS
BEGIN

 INSERT INTO ConferenceParticipants(
   IDParticipant, IDConferenceReservation, StudentID
 )
 VALUES(@IDParticipant,@IDConferenceReservation,@StudentsID)

 END
go
```

AddClient

Procedura dodaje nowego klienta do bazy dancyh. W zależności od @isCompany, musi zostać podany @CompanyName lub @Name i @Lastname. Zostanie stworzony wpis w tabeli Clients oraz w odpowiedniej Individual lub Company

```
CREATE PROCEDURE dbo.addClient
 @isCompany bit,
 @Login      VARCHAR(20),
 @Password   VARCHAR(50),
   @Email      VARCHAR(50),
   @Phone      VARCHAR(15),
   @City       VARCHAR(50),
   @Street     VARCHAR(50),
   @Country    VARCHAR(50),
   @PostalCode VARCHAR(6),
   @Address    VARCHAR(10),
 @CompanyName varchar(50)=NULL,
 @WWW         varchar(50)=NULL,
 @Name        varchar(50)=NULL,
 @LastName    varchar(50)=NULL
AS
BEGIN
 DECLARE @fail BIT
 SET @fail=0
 IF @isCompany=1
   BEGIN
     IF @CompanyName is NULL
       BEGIN
         SET @fail=1
         RAISERROR ('CompanyName missing',16,1)
         ROLLBACK TRANSACTION
       END
   END
 IF @isCompany=0
   BEGIN
     IF @Name is null or @LastName is NULL
       BEGIN
         SET @fail=1
         RAISERROR ('Name or LastName missing',16,1)
         ROLLBACK TRANSACTION
       END
   END

if @fail=0
 BEGIN
   INSERT INTO Client(
     Login, Password, email, City, Street, Country, PostalCode, Address, Phone
   )
   VALUES(@Login, @Password,
@Email,@City,@Street,@Country,@PostalCode,@Address,@Phone)
   DECLARE @ID int
   SET @ID =(select IDClient
             from dbo.Client
```

```
            where dbo.Client.Login=@Login)

   IF @isCompany=1
     BEGIN
         INSERT INTO Company(IDClient,Name,WWW) VALUES(@ID,@CompanyName,@WWW)
     END
   ELSE
     BEGIN
         INSERT INTO Individual(IDClient,Name,LastName) VALUES(@ID,@Name,@LastName)
     END
   END
 END
go
```

6.2 Procedury edytujące

CancelWorkshopReservation

Funkcja anulująca rezerwacje warsztatu na podstawie podanego id rezerwacji.

```
CREATE PROCEDURE dbo.cancelWorkshopReservation
 @IDWorkshopReservation int
 AS
BEGIN
  SET NOCOUNT ON
  UPDATE dbo.WorkshopReservation
  SET dbo.WorkshopReservation.Canceled=1
  where dbo.WorkshopReservation.IDWorkshopReservation=@IDWorkshopReservation
END
go
```

CancelWorkshop

Procedura anulująca warsztat o podanym id.

```
CREATE PROCEDURE dbo.cancelWorkshop
 @IDWorkshop int
 AS
 BEGIN

  SET NOCOUNT ON
  UPDATE dbo.WorkshopReservation
   SET dbo.WorkshopReservation.Canceled=1
   where dbo.WorkshopReservation.IDWorkshop=@IdWorkshop

   UPDATE dbo.Workshops
   SET dbo.Workshops.Canceled=1
   where dbo.Workshops.IDWorkshop=@IDWorkshop
 END
go
```

MakeWorkshopReservationActive
Procedura aktywuje uprzednio anulowaną rezerwacje warsztatu.

```
CREATE PROCEDURE dbo.makeWorkshopReservationActive
 @IDWorkshopReservation int
AS
BEGIN
 SET NOCOUNT ON
 UPDATE dbo.WorkshopReservation
 SET dbo.WorkshopReservation.Canceled=0
 WHERE dbo.WorkshopReservation.IDWorkshopReservation=@IDWorkshopReservation
END
go
```

MakeWorkshopActive
Procedura aktywuje uprzednio anulowany warsztat.

```
CREATE PROCEDURE dbo.makeWorkshopActive
 @IDWorkshop int
 AS
 BEGIN
   SET NOCOUNT ON
   UPDATE dbo.Workshops
   SET dbo.Workshops.Canceled=0
   WHERE dbo.Workshops.IDWorkshop=@IDWorkshop

 END
go
```

Sign_to_workshop
Procedure assigns given conference participant to workshop (makes new workshop participant record), pod warunkiem, że osoba jest zapisana na dzień konferencji, w którym odbywa się warsztat.

```
CREATE PROCEDURE dbo.sign_to_workshop
 @ConferenceparticiantID int,
 @WorkshopReservation int
AS
BEGIN
SET NOCOUNT ON

 INSERT INTO WorkshopParticipant(IDConferenceParticipants, IDWorkshopReservation)
 VALUES(
 @ConferenceparticiantID,
 @WorkshopReservation
)
END
go
```

User_sign_to_workshop

Umożliwia użytkownikowi wpisanie się na wybrany warsztat, pod warunkiem, że w bazie jest na ten warsztat rezerwacja odpowiadająca rezerwacji konferencji użytkwnika.

```
CREATE PROCEDURE user_sign_to_workshop
@ConferenceparticiantID int,
@WorkshopID int
AS
BEGIN
SET NOCOUNT ON
DECLARE @WorkshopReservationID int
set @WorkshopReservationID = (
  SELECT TOP 1 WorkshopReservation.IDWorkshopReservation
  FROM WorkshopReservation
    JOIN Workshops ON Workshops.IDWorkshop = WorkshopReservation.IDWorkshop
    JOIN ConferenceReservation
      ON WorkshopReservation.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
    JOIN ConferenceParticipants
      ON ConferenceReservation.IDConferenceReservation =
ConferenceParticipants.IDConferenceReservation
  WHERE Workshops.IDWorkshop=@WorkshopID
        and ConferenceParticipants.IDConferenceParticipants =
@ConferenceparticiantID
)
if(@WorkshopReservationID IS NOT NULL )
  BEGIN
  exec sign_to_workshop @ConferenceparticiantID,@WorkshopReservationID
   end
 ELSE
   RAISERROR('Can not add to workshop', 16, 1)
END
go
```

EditClient

Procedura edytująca istniejącego klienta.

```
CREATE PROCEDURE editClient(
 @isCompany bit,
 @Login varchar(20),
 @Password varchar(50)=NULL,
 @email varchar(50)=NULL,
 @Phone varchar(15)=NULL,
 @City varchar(50)=NULL,
 @Street varchar(50)=NULL,
 @Country varchar(50)=NULL,
 @PostalCode varchar(6)=NULL,
 @Address varchar(10)=NULL,
 @CompanyName varchar(50)=NULL,
 @WWW varchar(50)=NULL,
 @Name varchar(50)=NULL,
 @LastName varchar(50)=NULL
```

```
)
AS
BEGIN
  SET NOCOUNT ON

DECLARE @ID as int
SET @ID=(select IDClient
        from dbo.Client
        where dbo.Client.Login=@Login)


  IF @IsCompany=0
    BEGIN
      IF @Name is not NULL
        BEGIN
          UPDATE dbo.Individual
          SET dbo.Individual.Name=@Name
          WHERE dbo.Individual.IDClient=@ID
        END
      IF @Lastname is not NULL
        BEGIN
          UPDATE dbo.Individual
          SET dbo.Individual.LastName=@LastName
          WHERE dbo.Individual.IDClient=@ID
        END
    END
  ELSE
    BEGIN
      IF @CompanyName is not NULL
        BEGIN
          UPDATE dbo.Company
          SET dbo.Company.Name=@CompanyName
          WHERE dbo.Company.IDClient=@ID
        END
      IF @WWW is not NULL
        BEGIN
          UPDATE dbo.Company
          SET dbo.Company.WWW=@WWW
          WHERE dbo.Company.IDClient=@ID
        END
    END

  IF @Password is not NULL
    BEGIN
      UPDATE dbo.Client
      SET dbo.Client.Password=@Password
      where dbo.Client.IDClient=@ID
    END
  IF @email is not NULL
    BEGIN
      UPDATE dbo.Client
      SET dbo.Client.email=@email
      where dbo.Client.IDClient=@ID
    END
```

```
    IF @Phone is not NULL
      BEGIN
        UPDATE dbo.Client
        SET dbo.Client.Phone=@Phone
        where dbo.Client.IDClient=@ID
      END
    IF @City is not NULL
      BEGIN
        UPDATE dbo.Client
        SET dbo.Client.City=@City
        where dbo.Client.IDClient=@ID
      END
    IF @Street is not NULL
      BEGIN
        UPDATE dbo.Client
        SET dbo.Client.Street=@Street
        where dbo.Client.IDClient=@ID
      END
    IF @Country is not NULL
      BEGIN
        UPDATE dbo.Client
        SET dbo.Client.Country=@Country
        where dbo.Client.IDClient=@ID
      END
    IF @PostalCode is not NULL
      BEGIN
        UPDATE dbo.Client
        SET dbo.Client.PostalCode=@PostalCode
        where dbo.Client.IDClient=@ID
      END
    IF @Address is not NULL
      BEGIN
        UPDATE dbo.Client
        SET dbo.Client.Address=@Address
        where dbo.Client.IDClient=@ID
      END
 END
go
```

EditParticipant

Procedura edytująca istniejącego w bazie danych uczestnika .

```
CREATE PROCEDURE editParticipant(
 @Name varchar(50)=NULL,
 @Lastname varchar(50)=NULL,
 @email varchar(50)=NULL,
 @Phone varchar(15)=NULL,
 @City varchar(50)=NULL,
 @Street varchar(50)=NULL,
 @Address varchar(10)=NULL,
 @Login varchar(50),
 @Password varchar(50)=NULL,
 @Country varchar(50)=NULL,
 @PostalCode varchar(6)=NULL
```

```
)
AS
BEGIN
 SET NOCOUNT ON

 IF @Name is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.Name=@Name
     where dbo.Participants.Login=@Login
   END

 IF @Lastname is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.Lastname=@Lastname
     where dbo.Participants.Login=@Login
   END

 IF @email is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.email=@email
     where dbo.Participants.Login=@Login
   END

 IF @Phone is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.Phone=@Phone
     where dbo.Participants.Login=@Login
   END

 IF @City is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.City=@City
     where dbo.Participants.Login=@Login
   END

 IF @Street is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.Street=@Street
     where dbo.Participants.Login=@Login
   END

 IF @Address is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.Address=@Address
     where dbo.Participants.Login=@Login
   END
```

```
 IF @Password is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.Password=@Password
     where dbo.Participants.Login=@Login
   END

 IF @Country is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.Country=@Country
     where dbo.Participants.Login=@Login
   END

 IF @PostalCode is not NULL
   BEGIN
     UPDATE dbo.Participants
     SET dbo.Participants.PostalCode=@PostalCode
     where dbo.Participants.Login=@Login
   END
END
go
```

EditWorkshopCapacity

Procedura umożliwia edycję liczby możliwych miejsc na warsztat.

```
CREATE PROCEDURE editWorkshopCapacity(
 @IDWorkshop int,
 @NewCapacity int
)
 AS
 BEGIN
   SET NOCOUNT ON
   if not exists(
     select *
     from dbo.Workshops
     where dbo.Workshops.IDWorkshop=@IDWorkshop
   )
     BEGIN
       RAISERROR ('The is no workshop with given ID', 16,1)
       ROLLBACK TRANSACTION
     END
   ELSE
     BEGIN
       UPDATE dbo.Workshops
       SET dbo.Workshops.Capacity=@NewCapacity
       WHERE dbo.Workshops.IDWorkshop=@IDWorkshop
     END
 END
go
```

EditConferenceDayCapacity

Procedura umozliwia edycję liczby osób mogących wejść na dany dzień konferencji.

```
CREATE PROCEDURE editConferenceDayCapacity(
 @IDConferenceDay int,
 @NewCapacity int
)
 AS
BEGIN
 SET NOCOUNT ON
 IF not exists(
     select *
     from dbo.ConferenceDays
     where dbo.ConferenceDays.IDConferenceDay=@IDConferenceDay
   )
     BEGIN
       RAISERROR ('The is no Conference Day with given ID', 16,1)
       ROLLBACK TRANSACTION
     END
 ELSE
   BEGIN
     UPDATE dbo.ConferenceDays
     SET dbo.ConferenceDays.Capacity=@NewCapacity
     WHERE dbo.ConferenceDays.IDConferenceDay=@IDConferenceDay
   END
END
go
```

EditPeopleCountInWorkshopReservation

Umożliwia klientowi edycję liczby osób w rezerwacji warsztatu.

```
CREATE PROCEDURE editPeopleCountInWorkshopReservation(
 @IDWorkshopReservation int,
 @NewPeopleCount int
)
AS
 BEGIN
   SET NOCOUNT ON

   IF EXISTS(select *
             from WorkshopReservation
             where
WorkshopReservation.IDWorkshopReservation=@IDWorkshopReservation)
   BEGIN
     UPDATE WorkshopReservation
     SET WorkshopReservation.PeopleCount=@NewPeopleCount
     WHERE WorkshopReservation.IDWorkshopReservation=@IDWorkshopReservation
   END
   ELSE
   RAISERROR('There is no reservation with given ID',16,1)
 END
go
```

MakeConferenceReservationActive
Procedura umżliwia aktywowanie uprzednio anulowanej rezerwacji konferencji. Działa kaskadowo, więc uprzednio anulowane rezerwacje warsztatu znowu stają się aktywne(o ile warsztat nie jest anulowany)

```
CREATE PROCEDURE makeConferenceReservationActive
 @IDConferenceReservation int
 AS
 BEGIN
   SET NOCOUNT ON
   UPDATE dbo.ConferenceReservation
   SET dbo.ConferenceReservation.Canceled=0
   where dbo.ConferenceReservation.IDConferenceReservation=@IDConferenceReservation


   UPDATE dbo.WorkshopReservation
   SET dbo.WorkshopReservation.Canceled=0
   WHERE WorkshopReservation.IDWorkshopReservation IN  (select
WorkshopReservation.IDWorkshopReservation
               from WorkshopReservation
               join Workshops
               on Workshops.IDWorkshop=WorkshopReservation.IDWorkshop
               where
WorkshopReservation.IDConferenceReservation=@IDConferenceReservation and
Workshops.Canceled=0)
 END
go
```

CancelConferenceReservation
Procedura umożliwia anulowanie danej rezerwacji. Działa kaskadowo, w chwili anulowania rezerwacji konferencji, anulowane zostają również rezerwacje warsztatów zrobione z danej rezerwacji.

```
create PROCEDURE dbo.cancelConferenceReservation
 @IDConferenceReservation int
 AS
 BEGIN
   SET NOCOUNT ON
   UPDATE dbo.ConferenceReservation
   SET dbo.ConferenceReservation.Canceled=1
   where dbo.ConferenceReservation.IDConferenceReservation=@IDConferenceReservation

   UPDATE dbo.WorkshopReservation
   SET dbo.WorkshopReservation.Canceled=1
   WHERE dbo.WorkshopReservation.IDConferenceReservation=@IDConferenceReservation
 END
go
```

CancelConferenceDay

Procedura umożliwia anulowanie danego dnia konferencji. Działa kaskadowo, anulowane zostają też wszystkie rezerwacje na ten dzień konferencji, oraz wszystkie rezerwacje warsztatów.

```
CREATE PROCEDURE cancelConferenceDay (
 @IDConferenceDay int
)
 AS
BEGIN
 SET NOCOUNT ON
 UPDATE ConferenceDays
 SET ConferenceDays.Canceled=1
 WHERE ConferenceDays.IDConferenceDay=@IDConferenceDay

 CREATE TABLE #IDs (ID int)
 INSERT INTO #IDs(ID) (
     select IDConferenceReservation
     from ConferenceReservation
     where ConferenceReservation.IDConferenceDay=@IDConferenceDay
 )

 Declare @ID int

 While exists(select * from #IDs)
 BEGIN
   set @ID=(Select top 1 ID from #IDs)

   exec cancelConferenceReservation @ID

   Delete from #IDs where ID=@ID
 END
 DROP TABLE #IDs
END
go
```

CancelConference

Procedura umożliwiająca anulowanie konferencji. Działa kaskadowo, anulowane zostają również wszystkie dni konferencji i rezerwacje złożone na dni i warsztaty.

```
CREATE PROCEDURE cancelConference(
 @IDConference int
)
 AS
 BEGIN
   SET NOCOUNT ON

   UPDATE Conferences
   SET Conferences.Canceled=1
   WHERE Conferences.IDConference=@IDConference

   CREATE TABLE #IDC (ID int)
   INSERT INTO #IDC(ID) (
```

```
      select IDConferenceDay
      from ConferenceDays
      where ConferenceDays.IDConference=@IDConference
    )
    Declare @IDC int

    while exists(select * from #IDC)
    BEGIN
      set @IDC=(select top 1 ID from #IDC)
      exec cancelConferenceDay @IDC
      delete from #IDC where ID=@IDC
    END
    DROP TABLE #IDC
 END
go
```

EditPeopleCountInConferenceReservation

Procedura zmienia ilość zadeklarowanych przez klienta osób w ramach rezerwacji konferencji. Można ją wykonać zawsze w celach administracyjncyh.

```
CREATE PROCEDURE editPeopleCountInConferenceReservation (
 @IDConferenceReservation int,
 @NewPeopleCount int
)
AS
 BEGIN
   SET NOCOUNT ON

   IF (EXISTS(select *
              from ConferenceReservation
              where
ConferenceReservation.IDConferenceReservation=@IDConferenceReservation)

     BEGIN
       UPDATE ConferenceReservation
       SET ConferenceReservation.PeopleCount=@NewPeopleCount
       WHERE ConferenceReservation.IDConferenceReservation=@IDConferenceReservation
     END
   ELSE
   RAISERROR ('There is no reservation with given ID',16,1)
 END
go
```

EditPeopleCountInReservationUSERLEVEL

Umożliwia klientowi aktualizację miejsc w rezewacji konferencji. Sprawdza, czy nie jest za późno na wprowadzenie takich zmian.

```
CREATE PROCEDURE editPeopleCountInReservationUSERLEVEL(
 @IDClient int,
 @IDConferenceReservation int,
 @NewPeopleCount int
```

```
)
 AS
BEGIN
 SET NOCOUNT ON

 if(@IDClient != null)
 BEGIN
   if (SELECT
datediff(DAY,ConferenceReservation.ReservationDate,Conferences.StartDate) FROM
ConferenceReservation
       join ConferenceDays on ConferenceReservation.IDConferenceDay =
ConferenceDays.IDConferenceDay
       join Conferences on ConferenceDays.IDConference = Conferences.IDConference
       where
ConferenceReservation.IDConferenceReservation=@IDConferenceReservation) > 14
      or (
        (SELECT IDClient from ConferenceReservation
           where IDConferenceReservation=@IDConferenceReservation) != @IDClient
      )
     BEGIN
       RAISERROR ('Too late to make reservation changes, or mismatched ID', 16,1)
       ROLLBACK TRANSACTION
     END
 END

 UPDATE dbo.ConferenceReservation
   set dbo.ConferenceReservation.PeopleCount = @NewPeopleCount

END
go
```

DeleteConferenceParticipant

Porcedura umożliwia usunięcie uczestnika z listy na dany dzień konferencji. Z poziomu klienta procedura wywoła się automatycznie z jego idClienta, będzie mógł edytować tylko swoje rezerwacje. Z poziomu administratora, funkcja przyjmnie IDClient=null, co umożliwia dostęp do wszytkich rekordów.

```
CREATE PROCEDURE deleteConferenceParticipant(
 @IDClient                INT = NULL,
 @IDConferenceParticipant INT
)
AS
 BEGIN
   SET NOCOUNT ON

   IF @IDClient IS NOT NULL
     BEGIN
       IF DATEDIFF(DAY, GETDATE(), (
         SELECT date
         FROM ConferenceDays
           JOIN ConferenceReservation
```

```
            ON ConferenceReservation.IDConferenceDay =
ConferenceDays.IDConferenceDay
          JOIN ConferenceParticipants
            ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
         WHERE ConferenceParticipants.IDConferenceParticipants =
@IDConferenceParticipant
       )) > 14
       BEGIN
         DELETE
         FROM WorkshopParticipant
         WHERE IDConferenceParticipants IN
               (SELECT ConferenceParticipants.IDConferenceParticipants
                FROM ConferenceParticipants
                  JOIN ConferenceReservation
                    ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
                WHERE ConferenceReservation.IDClient = @IDClient AND
                      ConferenceParticipants.IDConferenceParticipants =
@IDConferenceParticipant)

         DELETE
         FROM ConferenceParticipants
         WHERE IDConferenceParticipants IN
               (SELECT ConferenceParticipants.IDConferenceParticipants
                FROM ConferenceParticipants
                  JOIN ConferenceReservation
                    ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
                WHERE ConferenceReservation.IDClient = @IDClient AND
                      ConferenceParticipants.IDConferenceParticipants =
@IDConferenceParticipant)
       END
       ELSE
         BEGIN
           RAISERROR ('It is too late to edit participant list',16,1)
         END
     END
   ELSE
     BEGIN
       DELETE
       FROM WorkshopParticipant
       WHERE IDConferenceParticipants IN
             (SELECT ConferenceParticipants.IDConferenceParticipants
              FROM ConferenceParticipants
                JOIN ConferenceReservation
                  ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
              WHERE ConferenceParticipants.IDConferenceParticipants =
@IDConferenceParticipant)
       DELETE
       FROM ConferenceParticipants
       WHERE IDConferenceParticipants IN
             (SELECT ConferenceParticipants.IDConferenceParticipants
```

```
              FROM ConferenceParticipants
                JOIN ConferenceReservation
                  ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
              WHERE ConferenceParticipants.IDConferenceParticipants =
@IDConferenceParticipant)
     END
 END
go
```

DeleteWorkshopParticipant

Porcedura umożliwia usunięcie uczestnika z listy na dany warsztat (z danej rezerwacji warsztatu). Z poziomu klienta procedura wywoła się automatycznie z jego idClienta, będzie mógł edytować tylko swoje rezerwacje. Z poziomu administratora, funkcja przyjmnie IDClient=null, co umożliwia dostęp do wszytkich rekordów.

```
CREATE PROCEDURE deleteWorkshopParticipant(
 @IDClient                INT = NULL,
 @IDConferenceParticipant INT
)
AS
 BEGIN
   SET NOCOUNT ON

   IF @IDClient IS NOT NULL
     BEGIN
       DELETE
       FROM WorkshopParticipant
       WHERE IDConferenceParticipants IN
             (SELECT ConferenceParticipants.IDConferenceParticipants
              FROM ConferenceParticipants
                JOIN ConferenceReservation
                  ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
              WHERE ConferenceReservation.IDClient = @IDClient AND
                    ConferenceParticipants.IDConferenceParticipants =
@IDConferenceParticipant)
     END
   ELSE
     BEGIN
       DELETE
       FROM WorkshopParticipant
       WHERE IDConferenceParticipants IN
             (SELECT ConferenceParticipants.IDConferenceParticipants
              FROM ConferenceParticipants
                JOIN ConferenceReservation
                  ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
              WHERE ConferenceParticipants.IDConferenceParticipants =
@IDConferenceParticipant)
     END
 END
```

6.3 Funkcje pomocnicze

Students_Count
Funkcja oblicza ilość studentów dla danej rezerwacji konferencji.

```
CREATE FUNCTION students_count
(
@param int
)
RETURNS int
AS
BEGIN
DECLARE @suma int
SET @suma = ISNULL((
SELECT count(*) from ConferenceParticipants
 join ConferenceReservation on ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
 WHERE ConferenceReservation.IDConferenceReservation=@param
 AND ConferenceParticipants.StudentID is NOT NULL
                  ),0)
RETURN @suma
END
go
```

Find_price_per_day
Oblicza cenę, którą trzeba zapłacić za dany dzień konferencji, biorąc pod uwagę zniżkę wynikającą z wcześniejszej rezeracji. W przypadku braku zaplanowanych zniżek, funkcja podaje cenę dnia 1:1

```
CREATE FUNCTION find_price_per_day
(
@param int
)
RETURNS money
AS
BEGIN
DECLARE @suma MONEY
DECLARE @price MONEY

SET @suma = isnull((SELECT top 1 Discounts.Discount FROM ConferenceReservation
JOIN ConferenceDays ON ConferenceReservation.IDConferenceDay =
ConferenceDays.IDConferenceDay
JOIN Discounts ON ConferenceDays.IDConferenceDay = Discounts.IDConferenceDay
WHERE ConferenceReservation.IDConferenceReservation=@param
AND ConferenceReservation.ReservationDate< Discounts.UntilDate
ORDER BY Discounts.UntilDate ASC ),1)


SET @price = (SELECT ConferenceDays.Price from ConferenceDays
 JOIN ConferenceReservation on ConferenceDays.IDConferenceDay =
ConferenceReservation.IDConferenceDay
```

```
where ConferenceReservation.IDConferenceReservation=@param)
RETURN (@suma*@price)
END
go
```

Price_workshop

Funkcja oblicza cenę, jaką klient będzie musiał zapłacić za zarezerwowane miejsca na warsztaty dla danej rezerwacji warsztatu. Przyjmuje jako parametr IDWorkshopReservation.

```
CREATE FUNCTION price_workshop
(
@param int
)
RETURNS MONEY
AS
BEGIN
DECLARE @suma MONEY
SET @suma=ISNULL((SELECT WorkshopReservation.PeopleCount*Workshops.Price
FROM WorkshopReservation JOIN Workshops ON
WorkshopReservation.IDWorkshop=Workshops.IDWorkshop
WHERE IDWorkshopReservation=@param),0)
RETURN @suma
END
go
```

IsClientACompany

Funkcja zwraca 1, jeżeli podany idClient reprezentuje firme, 0 jeżeli jest to klient indywidualny.

```
CREATE FUNCTION isClientACompany(
 @ID int
)
 RETURNS bit
 AS
 BEGIN
   IF exists(
     select *
     from Individual
     where Individual.IDClient=@ID
   )
     RETURN 0;
   RETURN 1;
 END
go
```

Count_payment_for_workshops

Funkcja zwraca cenę, jaką klient będzie musiał zapłacić za rezerwacje warsztatów, dla danej rezerwacji konferencji.

```
CREATE FUNCTION count_payment_for_workshops
(
@param int
)
RETURNS MONEY
AS
BEGIN
DECLARE @suma MONEY

SET @suma =isnull((Select
SUM(dbo.price_workshop(WorkshopReservation.IDWorkshopReservation)) FROM
WorkshopReservation
 join ConferenceReservation on WorkshopReservation.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
 join Workshops on WorkshopReservation.IDWorkshop = Workshops.IDWorkshop
 WHERE ConferenceReservation.IDConferenceReservation=@param AND
Workshops.Canceled=0),0)

 RETURN @suma
end
go



--Oblicza caly koszt dla danej
--idRezerwacjikonferencji
```

Count_payment

Funkcja oblicza pełny koszt rezerwacji z uwzględnieniem zniżek studenckich wpisanych uczestników i ceną warsztatów.

```
CREATE FUNCTION count_payment
(
@param int
)
RETURNS MONEY
AS
BEGIN
DECLARE @suma MONEY

SET @suma = ISNULL((SELECT (
ConferenceReservation.PeopleCount*dbo.find_price_per_day(ConferenceReservation.IDCo
nferenceReservation)
-
dbo.students_count(ConferenceReservation.IDConferenceReservation)*dbo.find_price_pe
r_day(ConferenceReservation.IDConferenceReservation)/2 )
FROM ConferenceReservation WHERE IDConferenceReservation=@param),0)

declare @suma2 MONEY
exec @suma2 =count_payment_for_workshops @param
```

```
SET @suma2 = @suma2 +@suma

RETURN @suma2
END
go
```

Cancel_unpaid_reservation

Funkcja po wywołaniu automatycznie kasuje rezerwacje, które nie zostały opłacone i są starsze, niż 7 dni.

```
create PROCEDURE dbo.cancel_unpaid_reservation
AS
BEGIN
--select zwaraca rezerwacje starsze niz 7 dni, ktore jeszcze sa aktywne i
nieoplacone
declare @id int
declare cur CURSOR LOCAL for
 (
   SELECT CR.IDConferenceReservation
   FROM ConferenceReservation AS CR
     INNER JOIN
     Payments AS P ON P.IDConferenceReservation = CR.IDConferenceReservation
   WHERE CR.ReservationDate < DATEADD(DAY, -7, GETDATE()) AND CR.Canceled = 0
   GROUP BY CR.IDConferenceReservation
   HAVING Sum(P.Amount) < dbo.count_payment(CR.IDConferenceReservation)
   UNION
   (
     SELECT CR.IDConferenceReservation
     FROM ConferenceReservation AS CR
     WHERE NOT exists(
         SELECT Payments.IDConferenceReservation
         FROM Payments
         WHERE Payments.IDConferenceReservation = CR.IDConferenceReservation
     ) AND CR.ReservationDate < DATEADD(DAY, -7, GETDATE()) AND CR.Canceled = 0
   )
 )
open cur
fetch next from cur into @id
WHILE @@FETCH_STATUS = 0
BEGIN

     EXEC cancelConferenceReservation @id
     fetch next from cur into @id

END
CLOSE cur
DEALLOCATE cur

end
go
```

FreePlacesForConferenceDay

Funkcja zwraca ilość wolnych miejsc na dany dzień konferencji. W obliczeiach pomija anulowane rezerwacje.

```
CREATE FUNCTION freePlacesForConferenceDay(
 @IDConferenceDay int
)
 RETURNS INT
 AS
 BEGIN
   RETURN(
     select ConferenceDays.Capacity-ISNULL((select sum(PeopleCount)
                                    from ConferenceReservation
                                    where
ConferenceReservation.IDConferenceDay=@IDConferenceDay and
                                          ConferenceReservation.Canceled=0
                                    group by ConferenceReservation.IDConferenceDay
                                    ), 0)
     from ConferenceDays
     where ConferenceDays.IDConferenceDay=@IDConferenceDay

   )
 END
go
```

FreePlacesForWorkshop

Funkcja zwraca ilość wolnych miejsc na warsztat. Sprawdza istnienie warsztatu i nie liczy anulowanych rezerwacjji warsztatu.

```
CREATE FUNCTION freePlacesForWorkshop(
 @IDWorkshop int
)
 RETURNS INT
 AS
 BEGIN
   IF exists(select *
             from Workshops
             where Workshops.IDWorkshop=@IDWorkshop and Workshops.Canceled=1)
     BEGIN
       RETURN NULL
     END

   RETURN(
     select Workshops.Capacity-ISNULL((select sum(PeopleCount)
                                    from WorkshopReservation
                                    where
WorkshopReservation.IDWorkshop=@IDWorkshop and
                                          WorkshopReservation.Canceled=0
                                    group by WorkshopReservation.IDWorkshop
                                    ), 0)
     from Workshops
     where Workshops.IDWorkshop=@IDWorkshop
```

```
   )
 END
go
```

Count_paid

Oblicza ile zapłacono dotychczas za daną rezerwację konferencji.

```
CREATE FUNCTION count_paid
(
@param int
)
RETURNS MONEY
AS
BEGIN
DECLARE @suma MONEY

SET @suma = ISNULL((
   SELECT sum(Payments.Amount)from Payments
   where IDConferenceReservation=@param
                 ),0)

RETURN @suma
END
go
```

Count_payment_ESTIMATED

Funkcja oblicza przewidywany koszt rezerwacji, bez wyspecyfikowanej listy uczestników.

```
CREATE FUNCTION count_payment_ESTIMATED
(
@param int
)
RETURNS MONEY
AS
BEGIN
DECLARE @suma MONEY

SET @suma = ISNULL((SELECT (
ConferenceReservation.PeopleCount*dbo.find_price_per_day(ConferenceReservation.IDCo
nferenceReservation)
-
ConferenceReservation.StudentsCount*dbo.find_price_per_day(ConferenceReservation.ID
ConferenceReservation)/2 )
FROM ConferenceReservation WHERE IDConferenceReservation=@param),0)

declare @suma2 MONEY
exec @suma2 =count_payment_for_workshops @param
SET @suma2 = @suma2 +@suma

RETURN @suma2
END
```

7. Triggery

Lista triggerów zaimplementowanych w bazie:

- login_unique
- login_unique_participants
- can_add_day
- can_add_day_same_date
- can_decrease_conf_day_capacity
- conference_day_canceled
- add_to_conference_day
- is_participant_signed_conference_day
- is_reservation_canceled
- change_people_count
- check_reservation_date
- conference_reservation_cancel
- limit_participants_per_day
- check_conference_dates
- can_add_discount
- check_ConferenceReservation_if_canceled
- check_payment_date
- conference_payment
- is_participant_signed_workshop
- is_time_collision
- participant_on_conference_day
- workshop_limit
- workshop_reservation_cancel
- can_decrease_workshop_capacity
- check_workshop_hours

Opisy poszczególnych triggerów oraz ich kod:

login_unique trigger zapobiega sytuacji, w której dwóch klientów miałoby ten sam login.

```
CREATE TRIGGER login_unique
 ON Client
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @login AS VARCHAR(20)
```

```
   SET @login = (SELECT Login
                 FROM inserted)

   DECLARE @Counts AS INT
   SET @Counts = (SELECT count(Client.IDClient)
                  FROM Client
                  WHERE Client.Login = @login
   )

   IF (1 < @Counts)
     BEGIN
       RAISERROR ('That login already exists', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

login_unique_participants - trigger zapobiega sytuacji, w której dwóch uczestników miałoby ten samo login.

```
CREATE TRIGGER login_unique_participants
 ON Participants
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @login AS VARCHAR(20)
   SET @login = (SELECT Login
                 FROM inserted)

   DECLARE @Counts AS INT
   SET @Counts = (SELECT count(Participants.IDParticipant)
                  FROM Participants
                  WHERE Participants.Login = @login
   )

   IF (1 < @Counts)
     BEGIN
       RAISERROR ('That login already exists', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

can_add_day - sprawdza czy dodany dzień konferencji znajduje się w granicach StartDate i EndDate danej konferencji.

```
CREATE TRIGGER can_add_day
 ON ConferenceDays
AFTER INSERT AS
 BEGIN
   IF NOT exists(SELECT *
```

```
                FROM inserted
                  JOIN Conferences ON Conferences.IDConference =
inserted.IDConference
                WHERE (inserted.Date >= Conferences.StartDate) AND (inserted.Date
<= Conferences.EndDate)
   )
     BEGIN
       RAISERROR ('Can not add day', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

can_add_day_same_date - sprawdza czy dzień konferencji nie został dodany dwukrotnie

```
CREATE TRIGGER can_add_day_same_date
 ON ConferenceDays
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @idConfDay AS INT
   SET @idConfDay = (SELECT inserted.IDConferenceDay
                     FROM inserted)

   DECLARE @date AS DATE
   SET @date = (SELECT inserted.Date
                FROM inserted)

   DECLARE @Conf AS INT
   SET @Conf = (SELECT Conferences.IDConference
                FROM Conferences
                  JOIN ConferenceDays ON Conferences.IDConference =
ConferenceDays.IDConference
                WHERE ConferenceDays.IDConferenceDay = @idConfDay)

   IF (
     (SELECT count(ConferenceDays.IDConferenceDay)
      FROM ConferenceDays
        JOIN Conferences ON ConferenceDays.IDConference = Conferences.IDConference
      WHERE ConferenceDays.Date = @date
            AND ConferenceDays.IDConference = @Conf) > 1
   )

     BEGIN
       RAISERROR ('Can not add day-wrong date', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

can_decrease_conf_day_capacity - sprawdza czy można zmniejszyć liczbę miejsc na dzień konferencji. Nie można tego zrobić jeśli nowa pojemność miałaby być mniejsza niż liczba zarezerwowanych miejsc.

```
CREATE TRIGGER can_decrease_conf_day_capacity
 ON ConferenceDays
AFTER UPDATE AS
 BEGIN
   IF exists(SELECT *
             FROM inserted AS dc
               JOIN ConferenceReservation AS cr
                 ON cr.IDConferenceDay = dc.IDConferenceDay
             WHERE cr.Canceled=0
             GROUP BY cr.IDConferenceDay, dc.Capacity
             HAVING sum(cr.PeopleCount) > dc.Capacity
   )
     BEGIN
       RAISERROR ('Can not decrease capacity of conference day. Too many
reservations', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

conference_day_canceled - sprawdza czy nie został dodany dzień konferencji lub usunięto anulowanie dnia na anulowaną konferencję.

```
CREATE TRIGGER conference_day_canceled
 ON ConferenceDays
AFTER INSERT, UPDATE
AS
 BEGIN
   IF EXISTS(
       SELECT *
       FROM inserted
         JOIN Conferences
           ON Conferences.IDConference = inserted.IDConference
       WHERE (Conferences.Canceled = 1 AND inserted.Canceled = 0)
   )
     BEGIN
       RAISERROR ('Can not add Conference Day for canceled Conference', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

add_to_conference_day - sprawdza czy podczas dodawania kolejnych uczestników dla danej rezerwacji nie została przekroczona ilość osób zarezerwowanych. Nie można dodać więcej osób niż się zarezerwowało.

```
CREATE TRIGGER add_to_conference_reservation
 ON ConferenceParticipants
AFTER INSERT AS
 BEGIN

   DECLARE @ConfDayBookingID AS INT
   SET @ConfDayBookingID = (SELECT IDConferenceReservation
                            FROM inserted)

   DECLARE @ParticipantsCount AS INT
   SET @ParticipantsCount = (SELECT COUNT(*)
                             FROM ConferenceParticipants
                             WHERE ConferenceParticipants.IDConferenceReservation =
@ConfDayBookingID)
   PRINT @ParticipantsCount

   DECLARE @ParticipantsNo AS INT
   SET @ParticipantsNo = (SELECT ConferenceReservation.PeopleCount
                          FROM ConferenceReservation
                          WHERE ConferenceReservation.IDConferenceReservation =
@ConfDayBookingID)
   PRINT @ParticipantsNo
   IF (@ParticipantsNo < @ParticipantsCount)
     BEGIN
       RAISERROR ('Too much participants', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

is_participant_signed_conference_day - sprawdza czy dany uczestnik dnia konferencji nie został dodany dwukrotnie (przez jednego klienta lub wielu).

```
CREATE TRIGGER is_participant_signed_conference_day
 ON ConferenceParticipants
AFTER INSERT, UPDATE AS
 BEGIN
   IF (SELECT count(*)
       FROM inserted ins
         JOIN ConferenceReservation cr
           ON cr.IDConferenceReservation = ins.IDConferenceReservation
         JOIN ConferenceReservation cr2
           ON cr.IDConferenceDay = cr2.IDConferenceDay
         JOIN ConferenceParticipants cp
           ON cp.IDConferenceReservation = cr2.IDConferenceReservation
       WHERE ins.IDParticipant = cp.IDParticipant
      ) > 1
     BEGIN
       RAISERROR ('Participant is already signed', 16, 1)
       ROLLBACK TRANSACTION
```

```
    END
 END
GO
```

is_reservation_canceled - sprawdza czy dodawani do dnia konferencji uczestnicy, nie są dodawani z anulowanych rezerwacji

```
CREATE TRIGGER is_reservtion_canceled
 ON ConferenceParticipants
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @id AS INT = (SELECT IDConferenceReservation
                         FROM inserted)

   IF exists(
       SELECT ConferenceReservation.IDConferenceReservation
       FROM ConferenceReservation
       WHERE ConferenceReservation.Canceled = 1
             AND ConferenceReservation.IDConferenceReservation = @id
   )
     BEGIN
       RAISERROR ('This reservation has been canceled', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

change_people_count - sprawdza czy można zmniejszyć ilość osób w rezerwacji. Nie można tego zrobić jeśli lista zgłoszonych uczestników jest większa niż nowa liczba osób w rezerwacji.

```
CREATE TRIGGER change_people_count
 ON ConferenceReservation
AFTER UPDATE AS
 BEGIN

   DECLARE @conferenceID AS INT
   SET @conferenceID = (
     SELECT inserted.IDConferenceReservation
     FROM inserted
   )

   IF ((SELECT count(ConferenceParticipants.IDParticipant)
        FROM ConferenceParticipants
        WHERE ConferenceParticipants.IDConferenceReservation = @conferenceID)
       > (SELECT PeopleCount
          FROM ConferenceReservation
          WHERE ConferenceReservation.IDConferenceReservation = @conferenceID))
     BEGIN
       RAISERROR ('Can not decrease people count, too much participants', 16, 1)
```

```
      ROLLBACK TRANSACTION
    END
 END
GO
```

check_reservation_date - sprawdza czy data rezerwacji na dany dzień konferencji jest mniejsza od daty dnia konferencji

```
CREATE TRIGGER check_reservation_date
 ON ConferenceReservation
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @data AS DATE
   SET @data = (SELECT inserted.ReservationDate
                FROM inserted)

   DECLARE @idConferenceDay AS INT
   SET @idConferenceDay = (SELECT inserted.IDConferenceDay
                           FROM inserted)

   IF (
     (SELECT Conferences.StartDate
      FROM Conferences
        JOIN ConferenceDays ON Conferences.IDConference =
ConferenceDays.IDConference
      WHERE ConferenceDays.IDConferenceDay = @idConferenceDay)
     < @data
   )
     BEGIN
       RAISERROR ('Reservation after conference date', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

conference_reservation_cancel - sprawdza czy nie została złożona rezerwacja na anulowany dzień konferencji

```
CREATE TRIGGER conference_reservation_cancel
 ON ConferenceReservation
AFTER INSERT, UPDATE
AS
 BEGIN
   IF EXISTS(
       SELECT *
       FROM inserted
         JOIN ConferenceDays
           ON ConferenceDays.IDConferenceDay = inserted.IDConferenceDay
       WHERE (ConferenceDays.Canceled = 1 AND inserted.Canceled = 0)
   )
```

```
    BEGIN
      RAISERROR ('Can not make reservation for canceled ConferenceDay', 16, 1)
      ROLLBACK TRANSACTION
    END
 END
GO
```

limit_participants_per_day - sprawdza czy ilość osób zarezerwowanych przez klientów nie jest większa niż pojemność danego dnia konferencji. Nie można zarezerwować więcej miejsc niż podany limit.

```
CREATE TRIGGER limit_participants_per_day
 ON ConferenceReservation
AFTER INSERT, UPDATE AS
 BEGIN
   IF exists(SELECT *
             FROM inserted
               JOIN ConferenceDays ON inserted.IDConferenceDay =
ConferenceDays.IDConferenceDay
               JOIN ConferenceReservation ON ConferenceDays.IDConferenceDay =
ConferenceReservation.IDConferenceDay
             WHERE
               inserted.IDConferenceDay = ConferenceReservation.IDConferenceDay AND
ConferenceReservation.Canceled = 0
             GROUP BY ConferenceReservation.IDConferenceDay,
ConferenceDays.Capacity
             HAVING sum(ConferenceReservation.PeopleCount) >
ConferenceDays.Capacity
   )
     BEGIN
       RAISERROR ('Can not add more participants to conference day', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

check_conference_dates - sprawdza czy data rozpoczęcia konferencji jest przed datą jej zakończenia

```
CREATE TRIGGER check_conference_dates
 ON Conferences
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @startdata AS DATE
   SET @startdata = (SELECT inserted.StartDate
                     FROM inserted)

   DECLARE @enddata AS DATE
   SET @enddata = (SELECT inserted.EndDate
                   FROM inserted)
```

```
   IF (
     @startdata > @enddata
   )
     BEGIN
       RAISERROR ('Start date can not be after end date', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

can_add_discount - sprawdza czy nie została dodana zniżka po rozpoczęciu konferencji

```
CREATE TRIGGER can_add_discount
 ON Discounts
AFTER INSERT, UPDATE AS
 BEGIN
   IF exists(SELECT *
             FROM inserted
               JOIN ConferenceDays ON ConferenceDays.IDConferenceDay =
inserted.IDConferenceDay
               JOIN Conferences ON Conferences.IDConference =
ConferenceDays.IDConferenceDay
             WHERE inserted.UntilDate > Conferences.StartDate
   )
     BEGIN
       RAISERROR ('Can not add discount after the conference begining', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
```

check_ConferenceReservation_if_canceled - sprawdza czy potencjalna płatność nie zostałaby dokonana na anulowaną rezerwacje

```
CREATE TRIGGER check_ConferenceReservation_if_canceled
 ON Payments
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @idConf AS INT
   SET @idConf = (SELECT inserted.IDConferenceReservation
                  FROM inserted)

   IF exists(SELECT *
             FROM ConferenceReservation
             WHERE (ConferenceReservation.IDConferenceReservation = @idConf)
                   AND (ConferenceReservation.Canceled = 1)
   )
     BEGIN
       RAISERROR ('Reservation has been canceled', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
```

```
GO
```

check_payment_date - sprawdza czy potencjalna płatność nie zostałaby dokonana po rozpoczęciu konferencji

```
CREATE TRIGGER check_payment_date
 ON Payments
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @idConf AS INT
   SET @idConf = (SELECT inserted.IDConferenceReservation
                  FROM inserted)

   DECLARE @date AS DATE
   SET @date = (SELECT inserted.PaymentDate
                FROM inserted)

   IF (
     (SELECT ConferenceReservation.ReservationDate
      FROM ConferenceReservation
      WHERE ConferenceReservation.IDConferenceReservation = @idConf) < @date
   )

     BEGIN
       RAISERROR ('Payment is too late', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
```

conference_payment

```
CREATE TRIGGER conference_payment
 ON Payments
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @ConferenceReservation AS INT
   SET @ConferenceReservation = (SELECT IDConferenceReservation
                                 FROM inserted)

   DECLARE @PaidAmount AS FLOAT
   SET @PaidAmount = (SELECT sum(Payments.Amount)
                      FROM Payments
                      WHERE Payments.IDConferenceReservation =
@ConferenceReservation
   )


   IF (@PaidAmount > dbo.count_payment(@ConferenceReservation))
     BEGIN
       RAISERROR ('Too big amount.', 16, 1)
```

```
      ROLLBACK TRANSACTION
    END
 END
GO
```

is_participant_signed_workshop - sprawdza czy uczestnik nie został zapisany więcej niż raz na dany warsztat

```
CREATE TRIGGER is_participant_signed_workshop
 ON WorkshopParticipant
AFTER INSERT, UPDATE
AS
 BEGIN
   IF (SELECT count(*)
       FROM inserted ins
         JOIN WorkshopReservation wr
           ON wr.IDWorkshopReservation = ins.IDWorkshopReservation
         JOIN WorkshopParticipant wp
           ON wp.IDWorkshopReservation = wr.IDWorkshopReservation
       WHERE wp.IDConferenceParticipants = ins.IDConferenceParticipants
      ) > 1
     BEGIN
       RAISERROR ('Participant is already signed to workshop`', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

is_time_collision - sprawdza czy uczestnik może być zapisany na dany warsztat. Jeśli jest już zapisany na inny i godziny trwania tych warsztatów nachodzą na siebie to wpis jest anulowany

```
CREATE TRIGGER is_time_collision
 ON WorkshopParticipant
AFTER INSERT, UPDATE
AS
 BEGIN

   DECLARE @insertedIDConferenceParticipants AS INT
   SET @insertedIDConferenceParticipants = (SELECT
inserted.IDConferenceParticipants
                                            FROM inserted)

   DECLARE @insertedIDIDWorkshopReservation AS INT
   SET @insertedIDIDWorkshopReservation = (SELECT inserted.IDWorkshopReservation
                                           FROM inserted)


   DECLARE @User AS INT
   SET @User = (SELECT ConferenceParticipants.IDConferenceParticipants
                FROM ConferenceParticipants
```

```
                WHERE ConferenceParticipants.IDConferenceParticipants =
@insertedIDConferenceParticipants
   )

   DECLARE @Workshop AS INT
   SET @Workshop = (SELECT Workshops.IDWorkshop
                    FROM Workshops
                      JOIN WorkshopReservation ON Workshops.IDWorkshop =
WorkshopReservation.IDWorkshop
                    WHERE @insertedIDIDWorkshopReservation =
WorkshopReservation.IDWorkshopReservation
   )

   DECLARE @start AS TIME
   SET @start = (SELECT Workshops.StartTime
                 FROM Workshops
                 WHERE Workshops.IDWorkshop = @Workshop
   )

   DECLARE @end AS TIME
   SET @end = (SELECT Workshops.EndTime
               FROM Workshops
               WHERE Workshops.IDWorkshop = @Workshop
   )

   IF (
        SELECT count(Workshops.IDWorkshop)
        FROM Workshops
          JOIN WorkshopReservation ON Workshops.IDWorkshop =
WorkshopReservation.IDWorkshop
          JOIN WorkshopParticipant
            ON WorkshopReservation.IDWorkshopReservation =
WorkshopParticipant.IDWorkshopReservation
          JOIN ConferenceParticipants
            ON WorkshopParticipant.IDConferenceParticipants =
ConferenceParticipants.IDConferenceParticipants
        WHERE @User = ConferenceParticipants.IDConferenceParticipants
              AND (
                (@start <= Workshops.EndTime AND @start >= Workshops.StartTime)
                OR
                (@end <= Workshops.EndTime AND @start >= Workshops.StartTime)
              )
      ) > 1
     BEGIN
       RAISERROR ('Participant is already on a different workshop at that time`',
16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

participant_on_conference_day - sprawdza czy uczestni warsztatu jest również uczestnikiem dnia konferencji, w którym odbywa się ten warsztat

```
CREATE TRIGGER participant_on_conference_day
 ON WorkshopParticipant
AFTER INSERT, UPDATE AS
 BEGIN
   IF NOT EXISTS(SELECT *
                 FROM inserted
                   JOIN WorkshopReservation
                     ON WorkshopReservation.IDWorkshopReservation =
inserted.IDWorkshopReservation
                   JOIN Workshops
                     ON Workshops.IDWorkshop = WorkshopReservation.IDWorkshop
                   JOIN ConferenceParticipants
                     ON ConferenceParticipants.IDConferenceParticipants =
inserted.IDConferenceParticipants
                   JOIN ConferenceReservation
                     ON ConferenceParticipants.IDConferenceReservation =
ConferenceReservation.IDConferenceReservation
                 WHERE Workshops.IDConferenceDay =
ConferenceReservation.IDConferenceDay
   )
     BEGIN
       RAISERROR ('Person is not participant of Conference Day', 16, 1);
       ROLLBACK TRANSACTION
     END
 END
GO
```

workshop_limit - sprawdza czy nie został przekroczony limit uczestników na dany warsztat

```
CREATE TRIGGER workshop_limit
 ON WorkshopReservation
AFTER INSERT, UPDATE AS
 BEGIN
   IF exists(
       SELECT *
       FROM inserted
         JOIN Workshops ON inserted.IDWorkshop = Workshops.IDWorkshop
         JOIN WorkshopReservation ON Workshops.IDWorkshop =
WorkshopReservation.IDWorkshop
       WHERE inserted.IDWorkshop = WorkshopReservation.IDWorkshop AND
WorkshopReservation.Canceled = 0
       GROUP BY WorkshopReservation.IDWorkshop, Workshops.IDWorkshop,
Workshops.Capacity
       HAVING sum(WorkshopReservation.PeopleCount) > Workshops.Capacity
   )
     BEGIN
       RAISERROR ('Can not add more participants to the workshop', 16, 1)
       ROLLBACK TRANSACTION
```

```
    END
 END
GO
```

workshop_reservation_cancel - sprawdza czy nie jest tworzona rezerwacja na anulowany warsztat

```
CREATE TRIGGER workshop_reservation_cancel
 ON WorkshopReservation
AFTER INSERT, UPDATE
AS
 BEGIN
   IF EXISTS(SELECT *
             FROM inserted
               JOIN Workshops
                 ON Workshops.IDWorkshop = inserted.IDWorkshop
             WHERE Workshops.Canceled = 1 AND inserted.Canceled = 0)
     BEGIN
       RAISERROR ('Can not make reservation for canceled workshop', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

can_decrease_workshop_capacity - sprawdza czy można zmniejszyć liczbę pojemność warsztatu. Nie można tego zrobić jeśli została już zarezerwowana większa liczba miejsc niż podana liczba

```
CREATE TRIGGER can_decrease_workshop_capacity
 ON Workshops
AFTER UPDATE AS
 BEGIN
   IF EXISTS(SELECT 'Yes'
             FROM inserted
               INNER JOIN
               WorkshopReservation ON inserted.IDWorkshop =
WorkshopReservation.IDWorkshop
             GROUP BY WorkshopReservation.IDWorkshop, inserted.Capacity
             HAVING Sum(WorkshopReservation.PeopleCount) > inserted.Capacity
   )
     BEGIN
       RAISERROR ('Cannot decrease workshop capacity, too many reservations.', 16,
       1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

check_workshop_hours - sprawdza czy godzina rozpoczęcia jest (prawidłowo) przed godziną zakończenia warsztatu

```
CREATE TRIGGER check_workshop_hours
 ON Workshops
AFTER INSERT, UPDATE AS
 BEGIN

   DECLARE @starttime AS TIME
   SET @starttime = (SELECT inserted.StartTime
                     FROM inserted)

   DECLARE @endtime AS TIME
   SET @endtime = (SELECT inserted.EndTime
                   FROM inserted)

   IF (
     @starttime > @endtime
   )
     BEGIN
       RAISERROR ('Start time can not be later than end time', 16, 1)
       ROLLBACK TRANSACTION
     END
 END
GO
```

8. Indeksy

W celu usprawnienia wyszukiwania danych dodaliśmy indexy do niektórych kluczy obcych, co przy większych ilościach danych może skutkować poprawą wydajności. Są to indeksy nieklastrowane, o domyślnej specyfikacji:

PAD_INDEX = OFF,
STATISTICS_NORECOMPUTE = OFF,
SORT_IN_TEMPDB = OFF,
DROP_EXISTING = OFF,
ONLINE = OFF,
ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON

Index_ConferenceReservation

```
create index Index_ConferenceReservation
on ConferenceReservation (IDClient)
go
```

Index_Payments

```
create index Index_Payments
on Payments (IDConferenceReservation)
go
```

Index_Conferenceparticipants

```
create index Index_Conferenceparticipants
on ConferenceParticipants (IDParticipant)
go
```

Index_Workshops

```
create index Index_Workshops
on Workshops (IDConferenceDay)
go
```

Index_WorkshopReservation

```
create index Index_WorkshopReservation
on WorkshopReservation (IDConferenceReservation)
go
```

Index_WorkshopParticipants

```
create index Index_WorkshopParticipants
on WorkshopParticipant (IDConferenceParticipants)
go
```

Index_Days

```
create index Index_Days
on ConferenceDays (IDConference)
go
```

Index_Discounts

```
create index Index_Discounts
on Discounts (IDConferenceDay)
go
```

9. Role w systemie

- Administrator bazy danych - ma dostęp do wszystkich zaimplementowanych procedur i funkcji, jak i bezpośrednio do rekordów bazy danych.

- Pracownik - ma dostęp do wszyskich porcedur administracyjnych z bazy, z wyłączeniem funkcji pomocniczych odpowiedzialnych za obliczanie kosztów, czy ilości wolnych miejsc.

- Klient - ma dostęp (niekoniecznie bezpośredni z uwagi na poleganie na indeksach podanych jako parametr)
  do wybrancyh procedur:
  - addConferenceReservation
  - addConferenceParticipant
  - addtoPayment
  - addWorkshopReservation
  - deleteWorkshopParticipant
  - deleteConferencePartiipant
  - editPeopleCountInReservationUSERLEVEL
  - editPeopleCountInWorkshopReservation
  - sign_to_workshop
  - cancelWorkshopReservation
  - cancelConferenceReservation
  - makeWorkshopReservationActive
  - makeConferenceReservationActive

- Uczestnik ma dostęp do:
  - user_sign_to_workshop

10. Dane

Ze względu na podział klientów na dwie dodatkowe tabele do generowania danych klientów użyliśmy funkcji addClient. Dzięki temu każdy klient będzie miał swoje record z tym samym ID w tabeli Clients i dokładnie jednej z dwóch tabel Individual lub Company. Przy pomocy generatora Mocharoo wygenerowaliśmy przykładowe dane i zmieniliśmy inserty na naszą funkcję. Przykładowe recordy wyglądają tatk:

```
	addClient 1, 'slamare', '7vsiY0wcU48', 'slamare@state.tx.us', '191-443-8515',
'Caruaru', 'Declaration', 'Brazil', '55000-000', '6', 'Kamba', 'joomla.org', 'Sheri',
'Lamar'; GO
	addClient 1, 'htordofff', 'S64IoLPDB', 'htordofff@forbes.com', '403-469-0061',
'Sokołów Podlaski', 'Old Gate', 'Poland', '08-301', '29', 'DabZ', 'constantcontact.com',
'Hayley', 'Tordoff'; GO
	addClient 0, 'jsapseedg', 'eASSowC85LI', 'jsapseedg@indiatimes.com', '972-880-1291',
'Cahuacho', 'Bobwhite', 'Peru', '32-961', '29', 'Demimbu', 'clickbank.net', 'Jemima',
'Sapseed'; GO
```

Podstawowe dane dot. użytkowników i konferencji zostały wygenerowane za pomocą Red Gate SQL Data Generator.