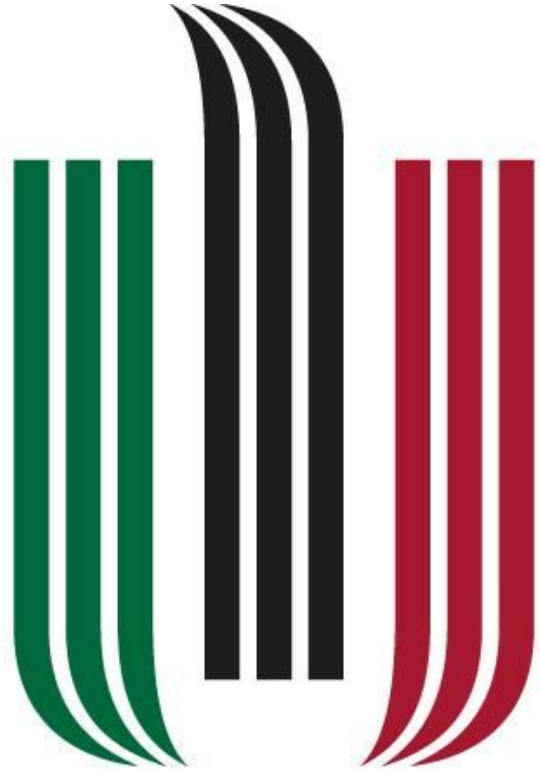


AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Podstawy Baz Danych

Dokumentacja Projektu „Konferencje”

Autorzy:
Michał Kołek

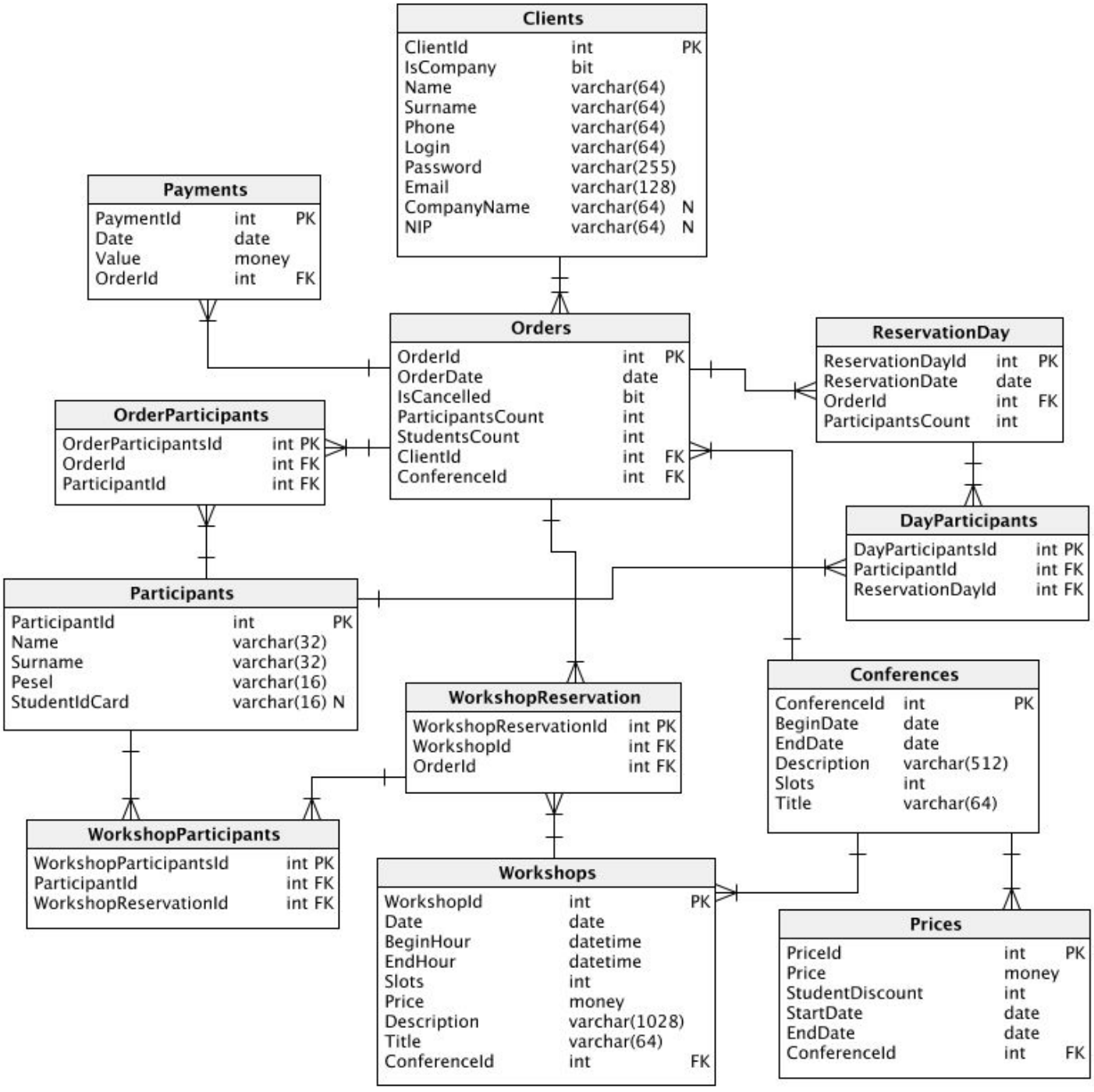
Prowadzący:
Dr inż. Robert Marcjan

Spis Treści

1.Wprowadzenie

Projekt miał na celu zaplanowanie i stworzenie systemu bazodanowego dla firmy organizującej konferencje. Klienci (indywidualni lub firmy) dokonują rezerwacji ustalonej liczby miejsc na dni konferencji. Lista uczestników nie musi być podawana natychmiast, oraz każdy sam wybiera dla siebie warsztaty. Cena konferencji zależy od terminu dokonania wpłaty (oraz dodatkowo może obowiązywać zniżka studencka). Warsztaty mogą być zarówno płatne jak i darmowe.

2.Schemat bazy danych



3.Tabele

3.1. Clients

Przechowuje informacje o klientach korzystających z systemu.

ClientId – Identyfikator klienta, wartość autoinkrementowana.
IsCompany – True/False czy klient jest firmą, czy prywatną osobą
Name – Imię klienta.
Surname – Nazwisko klienta.
Phone – Numer telefonu klienta (check regexujący wszelkie możliwe telefony na świecie)
Login – Pseudonim służący do logowania na stronie.
Password – Hash SHA1 hasła klienta (min. 5 znaków).
Email – Email klienta (regex na prawidłowy mail).
NIP – Identyfikator podatkowy.
CompanyName – Nazwa firmy, w przypadku klienta indywidualnego wynosi NULL.

```
CREATE TABLE Clients (
 ClientId    INT           NOT NULL IDENTITY,
 IsCompany   BIT           NOT NULL,
 Name        VARCHAR(64)   NOT NULL,
 Surname     VARCHAR(64)   NOT NULL,
 Phone       VARCHAR(64)   NOT NULL CHECK (Phone LIKE
   '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
 Login       VARCHAR(64)   NOT NULL,
 Password    VARCHAR(255) NOT NULL CHECK (LEN(Password) >= 5 ),
 Email       VARCHAR(128) NOT NULL CHECK (Email LIKE
   '[a-z,0-9,_,-]%@[a-z,0-9,_,-]%.[a-z][a-z]%'),
 CompanyName VARCHAR(64)   NULL,
 NIP         VARCHAR(64)   NULL,
 CONSTRAINT Clients_pk PRIMARY KEY (ClientId)
);
```

3.2. Conferences

Posiada informację na temat konferencji organizowanych przez firmę.

ConferenceId – Identyfikator konferencji, wartość autoinkrementowana.
BeginDate – Data rozpoczęcia konferencji.
EndDate – Data zakończenia konferencji (daty podlegają sprawdzeniu czy nie ma błędu logicznego).
Description – Opis konferencji.
Slots – Ilość miejsc (większa niż 0)
Title – Nazwa konferencji.

```
CREATE TABLE Conferences (
 ConferenceId INT           NOT NULL IDENTITY,
 BeginDate    DATE          NOT NULL,
 EndDate      DATE          NOT NULL,
 Description  VARCHAR(512) NOT NULL,
 Slots        INT           NOT NULL CHECK (Slots > 0),
 Title        VARCHAR(64)   NOT NULL,
 CONSTRAINT DateCheck CHECK (BeginDate <= EndDate),
 CONSTRAINT Conferences_pk PRIMARY KEY (ConferenceId)
);
```

3.3. DayParticipants

Pomocnicza tabela pozwalająca na zidentyfikowanie osób mających rezerwacje na dany dzień.

DayParticipantsId – Identyfikator konferencji, wartość autoinkrementowana.
ParticipantId – Identyfikator uczestnika z danego dnia rezerwacji.
ReservationDayId – Identyfikator danego dnia rezerwacji.

```
CREATE TABLE DayParticipants (
 DayParticipantsId INT NOT NULL IDENTITY,
 ParticipantId     INT NOT NULL,
 ReservationDayId  INT NOT NULL,
 CONSTRAINT DayParticipants_pk PRIMARY KEY (DayParticipantsId)
);

-- Reference: DayParticipants_Participants (table: DayParticipants)
ALTER TABLE DayParticipants
 ADD CONSTRAINT DayParticipants_Participants
FOREIGN KEY (ParticipantId)
REFERENCES Participants (ParticipantId);

-- Reference: DayParticipants_ReservationDay (table: DayParticipants)
ALTER TABLE DayParticipants
 ADD CONSTRAINT DayParticipants_ReservationDay
FOREIGN KEY (ReservationDayId)
REFERENCES ReservationDay (ReservationDayId);
```

3.4. OrderParticipants

Pomocnicza tabela pozwalająca na zidentyfikowanie osób mających rezerwacje w danym zamówieniu.

OrderParticipantsId – Identyfikator konferencji, wartość autoinkrementowana.
OrderId – Identyfikator danego zamówienia.
ParticipantId – Identyfikator uczestnika z danego dnia rezerwacji.

```
CREATE TABLE OrderParticipants (
 OrderParticipantsId INT NOT NULL IDENTITY,
 OrderId             INT NOT NULL,
 ParticipantId       INT NOT NULL,
 CONSTRAINT OrderParticipants_pk PRIMARY KEY (OrderParticipantsId)
);

-- Reference: OrderParticipants_Order (table: OrderParticipants)
ALTER TABLE OrderParticipants
 ADD CONSTRAINT OrderParticipants_Order
FOREIGN KEY (OrderId)
REFERENCES Orders (OrderId);

-- Reference: OrderParticipants_Participants (table: OrderParticipants)
ALTER TABLE OrderParticipants
 ADD CONSTRAINT OrderParticipants_Participants
FOREIGN KEY (ParticipantId)
REFERENCES Participants (ParticipantId);
```

3.5. Orders

Przechowuje informacje o zamówieniach klientów.

OrderId – Identyfikator zamówienia, wartość autoinkrementowana.
OrderDate – Data złożenia zamówienia.
IsCancelled – Czy zamówienie jest anulowane (w przypadku np. braku płatności przez 7 dni)
ParticipantsCount – Ilość uczestników.
StudentsCount – Ilość studentów.
ClientId – Identyfikator klienta składającego zamówienie.
ConferenceId – Identyfikator konferencji na którą jest zamówienie.

```
CREATE TABLE Orders (
 OrderId           INT  NOT NULL IDENTITY,
 OrderDate         DATE NOT NULL,
 IsCancelled       BIT  NOT NULL,
 ParticipantsCount INT  NOT NULL CHECK (ParticipantsCount >= 0),
 StudentsCount     INT  NOT NULL CHECK (StudentsCount >= 0),
 ClientId          INT  NOT NULL,
 ConferenceId      INT  NOT NULL,
 CONSTRAINT Orders_pk PRIMARY KEY (OrderId)
);

-- Reference: Order_Clients (table: Orders)
ALTER TABLE Orders
 ADD CONSTRAINT Order_Clients
FOREIGN KEY (ClientId)
REFERENCES Clients (ClientId);

-- Reference: Order_Conferences (table: Orders)
ALTER TABLE Orders
 ADD CONSTRAINT Order_Conferences
FOREIGN KEY (ConferenceId)
REFERENCES Conferences (ConferenceId);
```

3.6. Participants

Przechowuje informacje o uczestnikach.

ParticipantId – Identyfikator uczestnika, wartość autoinkrementowana.
Name – Data złożenia zamówienia.
Surname – Czy zamówienie jest anulowane (w przypadku np. braku płatności przez 7 dni)
Pesel – Ilość uczestników (regex na formę cyfrową peselu).
StudentIdCard – Ilość studentów.

```
CREATE TABLE Participants (
 ParticipantId INT         NOT NULL IDENTITY,
 Name          VARCHAR(32) NOT NULL,
 Surname       VARCHAR(32) NOT NULL,
 Pesel         VARCHAR(16) NOT NULL CHECK (Pesel LIKE
   '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
 StudentIdCard VARCHAR(16) NULL CHECK (StudentIdCard LIKE
   '[0-9][0-9][0-9][0-9][0-9][0-9]'),
 CONSTRAINT Participants_pk PRIMARY KEY (ParticipantId)
```

3.7. Payments

Przechowuje informacje o płatnościach dokonanych w ramach zamówienia.

PaymentId – Identyfikator płatności, wartość autoinkrementowana.
Date – Data płatności.
Value – Kwota płatności (nie mniejsza niż 0).
OrderId – Identyfikator zamówienia.

```
CREATE TABLE Payments (
 PaymentId INT   NOT NULL IDENTITY,
 Date      DATE  NOT NULL,
 Value     MONEY NOT NULL CHECK (Value > 0),
 OrderId   INT   NOT NULL,
 CONSTRAINT Payments_pk PRIMARY KEY (PaymentId)
);

-- Reference: Payments_Order (table: Payments)
ALTER TABLE Payments
 ADD CONSTRAINT Payments_Order
FOREIGN KEY (OrderId)
REFERENCES Orders (OrderId);
```

3.8. Prices

Przechowuje informacje o cenach konferencji między danymi dniami.

PriceId – Identyfikator ceny, wartość autoinkrementowana.
Price – Cena (nie mniejsza niż 0).
StudentDiscount – Procentowa zniżka dla studentów (nie większa niż 1).
StartDate – Od którego obowiązuje ta cena.
EndDate – Do którego obowiązuje ta cena.
ConferenceId – Identyfikator konferencji.

```
CREATE TABLE Prices (
 PriceId         INT           NOT NULL IDENTITY,
 Price           MONEY         NOT NULL CHECK (Price >= 0),
 StudentDiscount DECIMAL(3, 2) NOT NULL CHECK (StudentDiscount <= 1),
 StartDate       DATE          NOT NULL,
 EndDate         DATE          NOT NULL,
 ConferenceId    INT           NOT NULL,
 CONSTRAINT Prices_pk PRIMARY KEY (PriceId)
);

-- Reference: Prices_Conferences (table: Prices)
ALTER TABLE Prices
 ADD CONSTRAINT Prices_Conferences
FOREIGN KEY (ConferenceId)
REFERENCES Conferences (ConferenceId);
```

3.9. ReservationDay

Przechowuje informacje o rezerwacji danego dnia.

ReservationDayId – Identyfikator dnia rezerwacji, wartość autoinkrementowana.
ReservationDate – Data.
OrderId – Identyfikator zamówienia do którego należy rezerwacja tego dnia.
ParticipantsCount – Ilość uczestników danego dnia.

```
CREATE TABLE ReservationDay (
 ReservationDayId  INT  NOT NULL IDENTITY,
 ReservationDate   DATE NOT NULL,
 OrderId           INT  NOT NULL,
 ParticipantsCount INT  NOT NULL CHECK (ParticipantsCount > 0),
 CONSTRAINT ReservationDay_pk PRIMARY KEY (ReservationDayId)
);

-- Reference: ReservationDay_Order (table: ReservationDay)
ALTER TABLE ReservationDay
 ADD CONSTRAINT ReservationDay_Order
FOREIGN KEY (OrderId)
REFERENCES Orders (OrderId);
```

3.10. WorkshopParticipants

Przechowuje informacje o uczestnikach danego warsztatu.

WorkshopParticipantsId – Identyfikator, wartość autoinkrementowana.
ParticipantId – Identyfikator uczestnika warsztatu.
WorkshopReservationId – Identyfikator rezerwacji danego warsztatu.

```
CREATE TABLE WorkshopParticipants (
 WorkshopParticipantsId INT NOT NULL IDENTITY,
 ParticipantId          INT NOT NULL,
 WorkshopReservationId  INT NOT NULL,
 CONSTRAINT WorkshopParticipants_pk PRIMARY KEY (WorkshopParticipantsId)
);

-- Reference: WorkshopParticipants_Participants (table: WorkshopParticipants)
ALTER TABLE WorkshopParticipants
 ADD CONSTRAINT WorkshopParticipants_Participants
FOREIGN KEY (ParticipantId)
REFERENCES Participants (ParticipantId);

-- Reference: WorkshopParticipants_WorkshopReservation (table: WorkshopParticipants)
ALTER TABLE WorkshopParticipants
 ADD CONSTRAINT WorkshopParticipants_WorkshopReservation
FOREIGN KEY (WorkshopReservationId)
REFERENCES WorkshopReservation (WorkshopReservationId);
```

3.11. WorkshopReservation

Przechowuje informacje o rezerwacji danego warsztatu.

WorkshopReservationId – Identyfikator rezerwacji warsztatu, wartość autoinkrementowana.
WorkshopId – Identyfikator zarezerwowanego warsztatu.
OrderId – Identyfikator zamówienia do którego należy rezerwacja.

```
CREATE TABLE WorkshopReservation (
 WorkshopReservationId INT NOT NULL IDENTITY,
 WorkshopId            INT NOT NULL,
 OrderId               INT NOT NULL,
 CONSTRAINT WorkshopReservation_pk PRIMARY KEY (WorkshopReservationId)
);

-- Reference: WorkshopReservation_Order (table: WorkshopReservation)
ALTER TABLE WorkshopReservation
 ADD CONSTRAINT WorkshopReservation_Order
FOREIGN KEY (OrderId)
REFERENCES Orders (OrderId);

-- Reference: WorkshopReservation_Workshops (table: WorkshopReservation)
ALTER TABLE WorkshopReservation
 ADD CONSTRAINT WorkshopReservation_Workshops
FOREIGN KEY (WorkshopId)
REFERENCES Workshops (WorkshopId);
```

3.12. Workshops

Przechowuje informacje o dostępnych warsztatach.

WorkshopId – Identyfikator warsztatu, wartość autoinkrementowana.
Date – Data warsztatu.
BeginHour – Godzina rozpoczęcia.
EndHour – Godzina zakończenia. (z checkiem sprawdzającym logikę godzin)
Slots – Ilość miejsc (większa niż 0).
Price – Cena (może być równa 0).
Description – Opis warsztatu.
Title – Tytuł warsztatu
ConferenceId – Identyfikator konferencji do której należy warsztat.

```
CREATE TABLE Workshops (
 WorkshopId   INT           NOT NULL IDENTITY,
 Date         DATE          NOT NULL,
 BeginHour    DATETIME      NOT NULL,
 EndHour      DATETIME      NOT NULL,
 Slots        INT           NOT NULL CHECK (Slots > 0),
 Price        MONEY         NOT NULL,
 Description  VARCHAR(1028) NOT NULL,
 Title        VARCHAR(64)   NOT NULL,
 ConferenceId INT           NOT NULL,
 CONSTRAINT HourCheck CHECK (BeginHour < EndHour),
 CONSTRAINT Workshops_pk PRIMARY KEY (WorkshopId)
);
-- Reference: Workshops_Conferences (table: Workshops)
ALTER TABLE Workshops
 ADD CONSTRAINT Workshops_Conferences
FOREIGN KEY (ConferenceId)
REFERENCES Conferences (ConferenceId);
```

4.Widoki

4.1. Anulowane zamówienia

```
SELECT
 OrderId,
 ClientId,
 ConferenceId
FROM dbo.Orders
WHERE (IsCancelled = 1)
```

4.2. Wolne miejsca na konferencje

```
SELECT
dbo.Conferences.ConferenceId,
dbo.ReservationDay.ReservationDayId,
dbo.Conferences.Slots - SUM(dbo.Orders.ParticipantsCount) AS FreeSlots
FROM dbo.ReservationDay
INNER JOIN
dbo.Orders ON dbo.ReservationDay.OrderId = dbo.Orders.OrderId
LEFT OUTER JOIN
dbo.Conferences ON dbo.Orders.ConferenceId = dbo.Conferences.ConferenceId
WHERE (dbo.Orders.IsCancelled = 0)
GROUP BY dbo.ReservationDay.ReservationDayId, dbo.Conferences.Slots,
   dbo.Conferences.ConferenceId
```

4.3. Do kogo zadzwonić

```
SELECT
 cl.ClientId,
 ISNULL(cl.CompanyName, 'Not a company')       AS Company,
 cl.Phone,
 o.ParticipantsCount - COUNT(op.ParticipantId) AS pcount
FROM dbo.Orders AS o INNER JOIN
 dbo.Conferences AS c ON c.ConferenceId = o.ConferenceId
 INNER JOIN
 dbo.Clients AS cl ON cl.ClientId = o.ClientId
 LEFT OUTER JOIN
 dbo.OrderParticipants AS op ON op.OrderId = o.OrderId
WHERE (DATEDIFF(DAY, c.BeginDate, GETDATE()) < 14)
GROUP BY cl.ClientId, cl.CompanyName, o.ParticipantsCount, cl.Phone
HAVING (COUNT(op.ParticipantId) < o.ParticipantsCount)
```

4.4. Cena do zapłaty

```
SELECT
 o.OrderId,
 SUM(pa.Value) - (o.ParticipantsCount - o.StudentsCount * p.StudentDiscount) *
   p.Price AS ToPay
FROM dbo.Orders AS o INNER JOIN
 dbo.Conferences AS c ON o.ConferenceId = c.ConferenceId
 LEFT OUTER JOIN
 dbo.Prices AS p ON p.ConferenceId = c.ConferenceId
 LEFT OUTER JOIN
 dbo.Payments AS pa ON pa.OrderId = o.OrderId
WHERE (o.OrderDate BETWEEN p.StartDate AND p.EndDate)
GROUP BY o.OrderId, o.ParticipantsCount, o.StudentsCount, p.StudentDiscount, p.Price
```

4.5. Zamówienia do anulowania

```
SELECT
 dbo.Orders.OrderId,
 SUM(dbo.Payments.Value) - hmtp.ToPay AS Deficit
FROM dbo.Orders
 LEFT OUTER JOIN
 dbo.Payments ON dbo.Payments.OrderId = dbo.Orders.OrderId
 INNER JOIN
 dbo.how_much_to_pay AS hmtp ON hmtp.OrderId = dbo.Orders.OrderId
WHERE (DATEDIFF(DAY, dbo.Orders.OrderDate, dbo.Payments.Date) > 7)
GROUP BY dbo.Orders.OrderId, hmtp.ToPay
HAVING (SUM(dbo.Payments.Value) - hmtp.ToPay > 0)
```

4.6. Popularne konferencje

```
SELECT
 COUNT(op.ParticipantId) AS [Participants count],
 c.ConferenceId
FROM dbo.Conferences AS c LEFT OUTER JOIN
 dbo.Orders AS o ON c.ConferenceId = o.ConferenceId
 INNER JOIN
 dbo.OrderParticipants AS op ON op.OrderId = o.OrderId
GROUP BY c.ConferenceId
```

4.7. Popularne warsztaty

```
SELECT
 COUNT(wp.ParticipantId) AS [Participants count],
 w.WorkshopId
FROM dbo.Workshops AS w LEFT OUTER JOIN
 dbo.WorkshopReservation AS wr ON w.WorkshopId = wr.WorkshopId
 INNER JOIN
 dbo.WorkshopParticipants AS wp ON wp.WorkshopReservationId =
   wr.WorkshopReservationId
GROUP BY w.WorkshopId
```

4.8. Osoby prywatne

```
SELECT
 p.ParticipantId,
 p.Name,
 p.Surname
FROM dbo.Orders AS o INNER JOIN
 dbo.OrderParticipants AS op ON op.OrderId = o.OrderId
 INNER JOIN
 dbo.Participants AS p ON op.ParticipantId = p.ParticipantId
 INNER JOIN
 dbo.Clients AS c ON c.ClientId = o.ClientId
WHERE (c.IsCompany = 1) AND (o.IsCancelled = 0)
```

4.9. Zobacz klientów prywatnych

```
SELECT
 ClientId,
 Name,
 Surname,
 Phone,
 Email
FROM dbo.Clients
WHERE (IsCompany = 0)
```

4.10. Zobacz firmy

```
SELECT
 ClientId,
 CompanyName,
 Phone,
 Email
FROM dbo.Clients
WHERE (IsCompany = 1)
```

4.11. Wolne miejsca na warsztaty

```
SELECT
 w.WorkshopId,
 w.Slots - COUNT(wp.WorkshopParticipantsId) AS FreeSlots
FROM dbo.Workshops AS w LEFT OUTER JOIN
 dbo.WorkshopReservation AS wr ON wr.WorkshopId = w.WorkshopId
 INNER JOIN
 dbo.WorkshopParticipants AS wp ON wp.WorkshopReservationId =
   wr.WorkshopReservationId
GROUP BY w.WorkshopId, w.Slots
```

5.Procedury

5.1. Dodające

5.1.1. Dodanie konferencji

```
CREATE PROCEDURE [add_conference]
   @BeginDate   DATE,
   @EndDate     DATE,
   @Description VARCHAR(512),
   @Slots       INT,
   @Title       VARCHAR(64)
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   INSERT INTO Conferences
   (
     BeginDate,
     EndDate,
     Description,
     Slots,
     Title
   )
   VALUES
     (
       @BeginDate,
       @EndDate,
       @Description,
       @Slots,
       @Title
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding conference. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @errorMsg, 1
   END CATCH
 END
```

5.1.2. Dodanie klienta

```
CREATE PROCEDURE [add_client]
   @CompanyName VARCHAR(64) = NULL,
   @IsCompany   BIT = 0,
   @Name        VARCHAR(64),
   @Surname     VARCHAR(64),
   @Phone       VARCHAR(64),
   @NIP         VARCHAR(64),
   @Login       VARCHAR(64),
   @Password    NVARCHAR(4000),
   @Email       VARCHAR(128)
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF EXISTS
   (
       SELECT *
       FROM Clients
       WHERE Login = @Login
   )
     BEGIN
       THROW 52000, 'Login already used', 1
     END
   IF EXISTS
   (
       SELECT *
       FROM Clients
       WHERE Email = @Email
   )
     BEGIN
       THROW 52000, 'Email already used', 1
     END
   INSERT INTO Clients
   (
     CompanyName,
     IsCompany,
     Name,
     Surname,
     Phone,
     NIP,
     Login,
     Password,
     Email
   )
   VALUES
     (
       @CompanyName,
       @IsCompany,
       @Name,
       @Surname,
       @Phone,
       @NIP,
       @Login,
       HashBytes('SHA1', @Password),
       @Email
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Client. Error: '
```

```
                    + ERROR_MESSAGE();
    THROW 52000, @errorMsg, 1;
    END CATCH
 END
```

5.1.3. Dodanie zamówienia

```
CREATE PROCEDURE [add_order]
    @OrderDate          DATE,
    @IsCancelled        BIT = 0,
    @ClientId           INT,
    @ConferenceId       INT,
    @ParticipantsCount  INT = 0,
    @StudentsCount      INT = 0
AS
 BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
    IF NOT EXISTS
    (
        SELECT *
        FROM Clients
        WHERE ClientId = @ClientId
    )
      BEGIN
        THROW 52000, 'Provided ClientId does not exist', 1
      END
    IF NOT EXISTS
    (
        SELECT *
        FROM Conferences
        WHERE ConferenceId = @ConferenceId
    )
      BEGIN
        THROW 52000, 'Provided ConferenceId does not exist', 1
      END
    INSERT INTO Orders
    (
      OrderDate,
      IsCancelled,
      ClientId,
      ConferenceId,
      ParticipantsCount,
      StudentsCount
    )
    VALUES
      (
        @OrderDate,
        @IsCancelled,
        @ClientId,
        @ConferenceId,
        @ParticipantsCount,
        @StudentsCount
      )
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg NVARCHAR(2048);
    SET @errorMsg = 'Error during adding Order. Error: '
                    + ERROR_MESSAGE();
    THROW 52000, @ErrorMsg, 1;
    END CATCH
 END
```

5.1.4. Dodanie warsztatu

```
CREATE PROCEDURE [add_workshop]
   @Date          DATE,
   @BeginHour     DATETIME,
   @EndHour       DATETIME,
   @Slots         INT,
   @Price         MONEY,
   @Description   VARCHAR(512),
   @Title         VARCHAR(64),
   @ConferenceId INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Conferences
       WHERE ConferenceId = @ConferenceId
   )
     BEGIN
       THROW 52000, 'Conference with this Id does not exist', 1
     END
   IF EXISTS
   (
       SELECT *
       FROM Workshops
       WHERE ConferenceId = @ConferenceId
             AND [Date] = @Date
             AND Title = @Title
             AND BeginHour = @BeginHour
   )
     BEGIN
       THROW 52000, 'Provided workshop already exist (with the same ConferenceId,
   Date, Title and BeginHour)', 1
     END
   INSERT INTO Workshops
   (
     Date,
     BeginHour,
     EndHour,
     Slots,
     Price,
     Description,
     Title,
     ConferenceId
   )
   VALUES
     (
       @Date,
       @BeginHour,
       @EndHour,
       @Slots,
       @Price,
       @Description,
       @Title,
       @ConferenceId
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
```

```
   SET @errorMsg = 'Error during adding Workshop. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.1.5. Dodanie uczestnika

```
CREATE PROCEDURE [add_participant]
   @Name          VARCHAR(34),
   @Surname       VARCHAR(32),
   @Pesel         VARCHAR(16),
   @StudentIdCard VARCHAR(16) = NULL
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF EXISTS
   (
       SELECT *
       FROM Participants
       WHERE Pesel = @Pesel
   )
     BEGIN
       THROW 52000, 'Provided Person exist already', 1
     END
   INSERT INTO Participants
   (
     Name,
     Surname,
     Pesel,
     StudentIdCard
   )
   VALUES
     (
       @Name,
       @Surname,
       @Pesel,
       @StudentIdCard
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Participant. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.1.6. Dodanie cen

```
CREATE PROCEDURE [add_price]
   @Price MONEY,
   @StudentDiscount DECIMAL(3,2),
   @StartDate DATE,
   @EndDate DATE,
   @ConferenceId INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Conferences
       WHERE ConferenceId = @ConferenceId
   )
     BEGIN
       THROW 52000, 'Provided ConferenceId does not exist', 1
     END
   INSERT INTO Prices
   (
     Price,
     StudentDiscount,
     StartDate,
     EndDate,
     ConferenceId
   )
     VALUES
       (
         @Price,
         @StudentDiscount,
         @StartDate,
         @EndDate,
         @ConferenceId
       )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Price. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.1.7. Dodanie dni rezerwacji

```
CREATE PROCEDURE [add_reservationDay]
   @ReservationDate DATE,
   @OrderId INT,
   @ParticipantsCount INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Orders
       WHERE OrderId = @OrderId
   )
     BEGIN
       THROW 52000, 'Provided OrderId does not exist', 1
     END
   INSERT INTO ReservationDay
   (
     ReservationDate,
     OrderId,
     ParticipantsCount
   )
   VALUES
     (
       @ReservationDate,
       @OrderId,
       @ParticipantsCount
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Reservation Day. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.1.8. Dodanie płatności

```
CREATE PROCEDURE [add_payment]
   @Date DATE,
   @Value MONEY,
   @OrderId INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Orders
       WHERE OrderId = @OrderId
   )
     BEGIN
       THROW 52000, 'Provided OrderId does not exist', 1
     END
   INSERT INTO Payments
   (
     Date,
     Value,
     OrderId
   )
   VALUES
     (
       @Date,
       @Value,
       @OrderId
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Payment. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @errorMsg, 1;
   END CATCH
 END
```

5.1.9. Dodanie uczestników zamówienia

```
CREATE PROCEDURE [add_orderParticipants]
   @ParticipantId INT,
   @OrderId INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Participants
       WHERE ParticipantId = @ParticipantId
   )
     BEGIN
       THROW 52000, 'Provided ParticipantId does not exist', 1
     END
   IF NOT EXISTS
   (
       SELECT *
       FROM Orders
       WHERE OrderId = @OrderId
   )
     BEGIN
       THROW 52000, 'Provided OrderId does not exist', 1
     END
   INSERT INTO OrderParticipants
   (
     ParticipantId,
     OrderId
   )
   VALUES
     (
       @ParticipantId,
       @OrderId
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Order Participants. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.1.10. Dodanie uczestników danego dnia

```
CREATE PROCEDURE [add_dayParticipants]
   @ParticipantId INT,
   @ReservationDayId INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Participants
       WHERE ParticipantId = @ParticipantId
   )
     BEGIN
       THROW 52000, 'Provided ParticipantId does not exist', 1
     END
   IF NOT EXISTS
   (
       SELECT *
       FROM ReservationDay
       WHERE ReservationDayId = @ReservationDayId
   )
     BEGIN
       THROW 52000, 'Provided ReservationDayId does not exist', 1
     END
   INSERT INTO DayParticipants
   (
     ParticipantId,
     ReservationDayId
   )
   VALUES
     (
       @ParticipantId,
       @ReservationDayId
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Day Participants. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.1.11. Dodanie rezerwacji warsztatu

```
CREATE PROCEDURE [add_workshopReservation]
   @WorkshopId INT,
   @OrderId INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Orders
       WHERE OrderId = @OrderId
   )
     BEGIN
       THROW 52000, 'Provided OrderId does not exist', 1
     END
   IF NOT EXISTS
   (
       SELECT *
       FROM Workshops
       WHERE WorkshopId = @WorkshopId
   )
     BEGIN
       THROW 52000, 'Provided WorkshopId does not exist', 1
     END
   INSERT INTO WorkshopReservation
   (
     WorkshopId,
     OrderId
   )
   VALUES
     (
       @WorkshopId,
       @OrderId
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Workshop Reservation. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.1.12. Dodanie uczestników rezerwacji warsztatu

```
CREATE PROCEDURE [add_workshopParticipants]
   @ParticipantId INT,
   @WorkshopReservationId INT
AS
 BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
   IF NOT EXISTS
   (
       SELECT *
       FROM Participants
       WHERE ParticipantId = @ParticipantId
   )
     BEGIN
       THROW 52000, 'Provided ParticipantId does not exist', 1
     END
   IF NOT EXISTS
   (
       SELECT *
       FROM WorkshopReservation
       WHERE WorkshopReservationId = @WorkshopReservationId
   )
     BEGIN
       THROW 52000, 'Provided WorkshopReservationId does not exist', 1
     END
   INSERT INTO WorkshopParticipants
   (
     ParticipantId,
     WorkshopReservationId
   )
   VALUES
     (
       @ParticipantId,
       @WorkshopReservationId
     )
   END TRY
   BEGIN CATCH
   DECLARE @errorMsg NVARCHAR(2048);
   SET @errorMsg = 'Error during adding Workshop Participants. Error: '
                   + ERROR_MESSAGE();
   THROW 52000, @ErrorMsg, 1;
   END CATCH
 END
```

5.2. Wyświetlające

5.2.1. Uczestnicy konferencji

```
CREATE PROCEDURE conference_participants
   @confId INT
AS
 BEGIN
   SELECT
     p.ParticipantId,
     p.Name,
     p.Surname,
     p.Pesel
   FROM Orders o
     INNER JOIN OrderParticipants op ON op.OrderId = o.OrderId
     INNER JOIN Participants p ON p.ParticipantId = op.ParticipantId
     INNER JOIN Conferences c ON c.ConferenceId = o.ConferenceId
   WHERE c.ConferenceId = @confId AND IsCancelled = 0
 END
```

5.2.2. Uczestnicy warsztatu

```
CREATE PROCEDURE workshop_participants
   @workshopid INT
AS
 BEGIN
   SELECT
     p.ParticipantId,
     p.Name,
     p.Surname,
     p.Pesel
   FROM Workshops w
     INNER JOIN WorkshopReservation wr ON w.WorkshopId = wr.WorkshopId
     INNER JOIN WorkshopParticipants wp ON wp.WorkshopReservationId =
   wr.WorkshopReservationId
     INNER JOIN Participants p ON p.ParticipantId = wp.ParticipantID
   WHERE WorkshopId = @workshopid AND IsCancelled = 0
 END
```

5.2.3. Kwota za dane zamówienie

```
CREATE PROCEDURE [how_much_to_pay_by_order]
   @orderid INT
AS
 BEGIN
   SELECT
     o.OrderId,
     SUM(Value) - (ParticipantsCount - StudentsCount * StudentDiscount) * Price AS
   ToPay
   FROM Orders o
     INNER JOIN Conferences c ON o.ConferenceId = c.ConferenceId
     LEFT JOIN Prices p ON p.ConferenceId = c.ConferenceId
     LEFT JOIN Payments pa ON pa.OrderId = o.OrderId
   WHERE OrderDate BETWEEN p.StartDate AND p.EndDate AND o.OrderId = @orderid
   GROUP BY o.OrderId, ParticipantsCount, StudentsCount, StudentDiscount, Price
 END
```

5.2.4. Kwota do zapłacenia dla danego klienta

```
CREATE PROCEDURE [how_much_to_pay_by_client]
   @clientid INT
AS
 BEGIN
   SELECT
     c.ClientId,
     SUM(ToPay)
   FROM Clients c
     INNER JOIN Orders ord ON ord.ClientId = c.ClientId
     INNER JOIN how_much_to_pay hmtp ON hmtp.OrderId = ord.OrderId
   WHERE c.ClientId = @clientid
   GROUP BY c.ClientId
 END
```

5.2.5. Uczestnicy z danej firmy

```
CREATE PROCEDURE participants_from_company
   @NIP VARCHAR(64)
AS
 BEGIN
   SELECT
     p.ParticipantId,
     p.Name,
     p.Surname
   FROM Orders o
     INNER JOIN OrderParticipants op ON op.OrderId = o.OrderId
     INNER JOIN Participants p ON op.ParticipantId = p.ParticipantId
     INNER JOIN Clients c ON c.ClientId = o.ClientId
   WHERE c.NIP = @NIP AND IsCancelled = 0
 END
```

5.2.6. Ceny konferencji

```
CREATE PROCEDURE prices_of_conference
   @conferenceid INT
AS
 BEGIN
   SELECT
     c.ConferenceId,
     Price,
     StudentDiscount,
     StartDate AS [Cena od dnia],
     p.EndDate AS [Cena do dnia]
   FROM Conferences c INNER JOIN Prices p ON c.ConferenceId = p.ConferenceId
   WHERE @conferenceid = c.ConferenceId
   ORDER BY StartDate
 END
```

6.Indexy

Indeksy nasze mają na celu przyspieszenie wyszukiwania oraz korzystania z systemu bazodanowego. Oprócz tych które utworzyły się automatyczne wraz z CONSTRAINT PRIMARY KEY, dodaliśmy także:

```
CREATE INDEX orders_conferenceid_index ON Orders(ConferenceId);

CREATE INDEX orders_clientid_index ON Orders(ClientId);

CREATE INDEX payments_orderid_index ON Payments(OrderId);

CREATE INDEX reservationday_orderid_index ON ReservationDay(OrderId);

CREATE INDEX workshopreservation_workshopid_orderid_index ON
WorkshopReservation(WorkshopId, OrderId);

CREATE INDEX prices_conferenceid_index ON Prices(ConferenceId);

CREATE INDEX workshops_conferenceid_index ON Workshops(ConferenceId);

CREATE INDEX dayparticipants_participantid_reservationdayid_index ON
DayParticipants(ParticipantId, ReservationDayId);

CREATE INDEX orderparticipants_participantid_orderid_index ON
OrderParticipants(ParticipantId, OrderId);

CREATE INDEX start_time ON Workshops (BeginHour);

CREATE INDEX end_time ON Workshops (EndHour);

CREATE INDEX by_title ON Conferences (Title);
```

7.Triggery

7.1. Za mało miejsc na konferencje

```
CREATE TRIGGER too_few_conference_day_slots_trigger
 ON Orders
AFTER INSERT
AS
 BEGIN
   IF EXISTS
   (
       SELECT *
       FROM Orders o
         INNER JOIN inserted i ON i.ConferenceId = o.ConferenceId
         INNER JOIN Conferences c ON c.ConferenceId = o.ConferenceId
       GROUP BY i.ConferenceId, Slots
       HAVING SUM(o.ParticipantsCount) = Slots
   )
     BEGIN
       THROW 50001, 'There are not enough free slots for the conference', 1
     END
 END
```

7.2. Za mało miejsc na warsztaty

```
CREATE TRIGGER too_few_workshop_slots_trigger
 ON WorkshopReservation
AFTER INSERT
AS
 BEGIN
   IF EXISTS
   (
       SELECT *
       FROM WorkshopReservation wr
         INNER JOIN Workshops w ON w.WorkshopId = wr.WorkshopId
         INNER JOIN inserted i ON i.WorkshopReservationId = wr.WorkshopReservationId
       GROUP BY wr.WorkshopId, Slots
       HAVING COUNT(wr.WorkshopReservationId) = Slots
   )
     BEGIN
       THROW 50001, 'There are not enough free slots for the workshop', 1
     END
 END
```

7.3. Za dużo uczestników dodano do zamówienia

```
CREATE TRIGGER too_many_order_participants
 ON OrderParticipants
AFTER INSERT
AS
 BEGIN
   IF EXISTS
   (
       SELECT *
       FROM Orders o
         INNER JOIN OrderParticipants op ON op.OrderId = o.OrderId
         INNER JOIN inserted i ON i.OrderId = o.OrderId
       GROUP BY o.OrderId, ParticipantsCount
       HAVING COUNT(op.ParticipantId) = ParticipantsCount
   )
     BEGIN
       THROW 50001, 'There are too many participants for this order', 1
     END
 END
```

7.4. Sprawdzenie liczby studentów

```
CREATE TRIGGER check_students_count
 ON Participants
AFTER INSERT, UPDATE
AS
 BEGIN
   IF EXISTS
   (
       SELECT *
       FROM Participants p
         INNER JOIN OrderParticipants op ON op.ParticipantId = p.ParticipantId
         INNER JOIN inserted i ON i.ParticipantId = op.ParticipantId
         INNER JOIN Orders o ON op.OrderId = o.OrderId
       WHERE p.StudentIdCard IS NOT NULL OR i.StudentIdCard IS NOT NULL
       GROUP BY o.OrderId, StudentsCount
       HAVING COUNT(op.ParticipantId) > StudentsCount
   )
     BEGIN
       THROW 50001, 'The number of students is too big', 1
     END
 END
```

7.5. Czy konferencja ma więcej miejsc niż warsztaty

```
CREATE TRIGGER check_if_conference_has_more_slots_than_workshop
 ON Workshops
AFTER UPDATE, INSERT
AS
 BEGIN
   IF EXISTS
   (
       SELECT *
       FROM Conferences c
         INNER JOIN inserted i ON i.ConferenceId = c.ConferenceId
       WHERE i.Slots > c.Slots
   )
     BEGIN
       THROW 50001, 'There are not enough conference slots for this workshop', 1
     END
 END
```

7.6. Czy dzień rezerwacji jest w czasie trwania konferencji

```
CREATE TRIGGER is_reservation_day_during_conference
 ON ReservationDay
AFTER UPDATE, INSERT
AS
 BEGIN
   IF EXISTS
   (
       SELECT *
       FROM inserted i
         INNER JOIN Orders o ON i.OrderId = o.OrderId
         INNER JOIN Conferences c ON c.ConferenceId = o.ConferenceId
       WHERE i.ReservationDate BETWEEN c.BeginDate AND c.EndDate
   )
     BEGIN
       THROW 50001, 'There are too many participants for this order', 1
     END
 END
```

8.Proponowane role w systemie

Role przedstawione zostaną w sposób rosnących uprawnień, tj. każdy kolejny punkt będzie posiadał uprawnienia z punktu poprzedniego.

8.1. System www / Użytkownik

Uprawnienia witryny sieciowej, czyli także i korzystających z niej użytkowników. Zezwala na podgląd wszelakich konferencji i warsztatów, a oprócz tego jest zdolna do wywoływania procedur dodających zamówienia/rezerwacje.

8.2. Osoba zarządzająca zamówieniami

Rola posiadająca dostęp do list uczestników konferencji i warsztatów, oraz z opcją podglądu brakujących danych klientów. Oprócz tego może te braki także uzupełniać.

8.3. Koordynator konferencji

Dodawanie i edytowanie nadchodzących konferencji, warsztatów i progów cenowych.

8.4. Właściciel

Odczyt wszystkich tabel i widoków.

8.5. Administrator

Pełny dostęp z możliwością przeglądania i edytowania bazy danych.

9.Generator danych

W celu wygenerowania danych, skorzystaliśmy z oprogramowania SQL Data Generator firmy RedGate, posługując się pełną wersją Trial 14dniową (https://www.red-gate.com/products/sql-development/sql-data-generator/index).