Dokumentacja projektu bazy danych

Krzysztof Blecharczyk, Przemysław Jabłecki

Zawartość

1. Uproszczona specyfikacja wymagań do projektu

1.1 Opis funkcji z punktu widzenia użytkowników

1.1.1 Lista aktorów i ich funkcje

Organizator:
-tworzy konferencje,
-tworzy warsztaty,
-ustala stawki za konferencję/warsztaty oraz aktualizuje je w zależności od terminu,
-udostępnia listę konferencji oraz warsztatów,
-przyjmuje rezerwacje,
-przyjmuje płatności z uwzględnieniem wszystkich czynników wpływających na należność za udział
-komunikuje się z klientem jeśli nie poda listy uczestników konferencji do 2 tygodni od terminu konferencji,
-anuluje rezerwacje jeśli nie została opłacona lub w przypadku braku uzupełnienia danych
-wystawia faktury dla klientów,
-weryfikuje, czy dany uczestnik może brać udział w konkretnych warsztatach
-nadaje identyfikator uczestnikom,
-tworzy statystyki (lista osób, płatności, najczęstsi klienci).

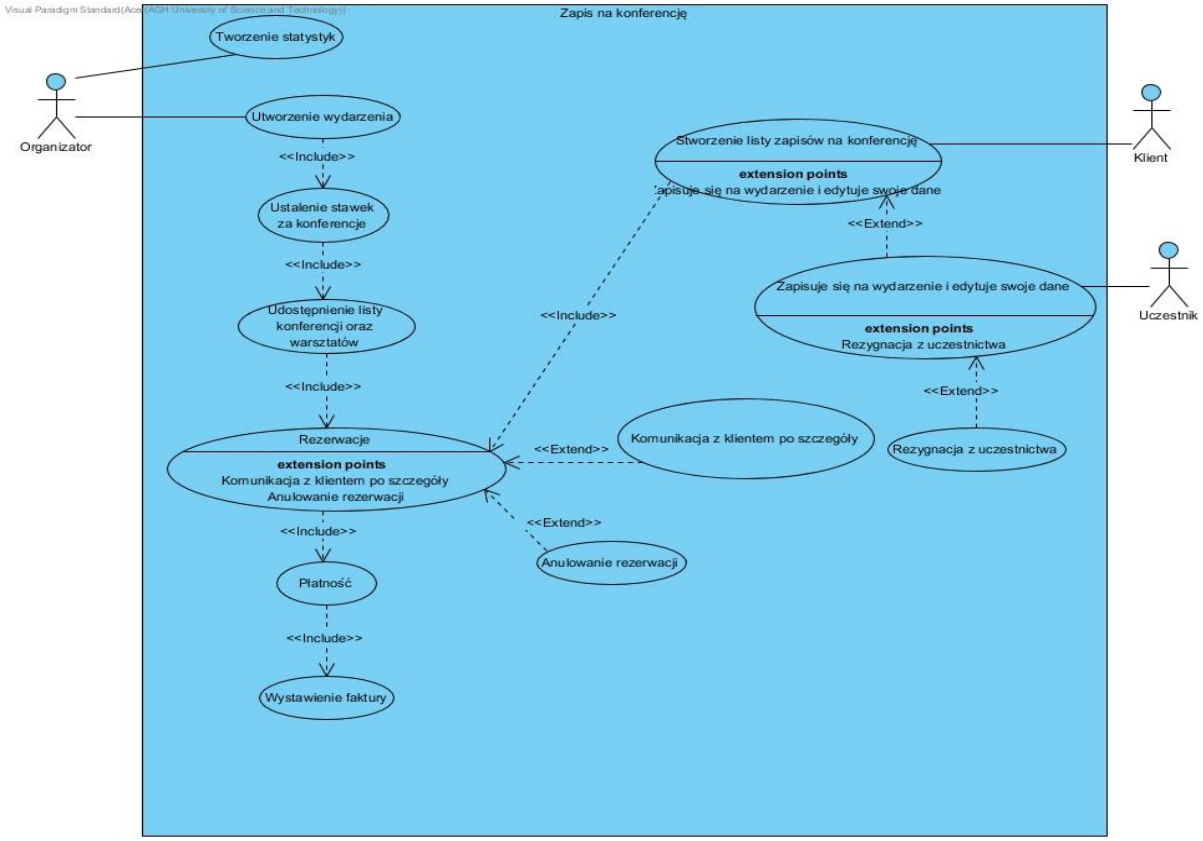
Klient firmowy:
-rezerwuje ilość miejsc na konferencję,
-zgłasza uczestników na konferencję i warsztaty (pod warunkiem, że są wolne miejsca),
-zbiera i udostępnia dane uczestników, w tym dane osobowe oraz przysługujące im zniżki,
-wnosi opłatę,
-anuluje rezerwację.

Klient indywidualny:
-rezerwuje jedno miejsce na konferencję,
-zgłasza siebie na konferencję,
-uiszcza opłatę za rezerwację,
-anuluje rezerwację.

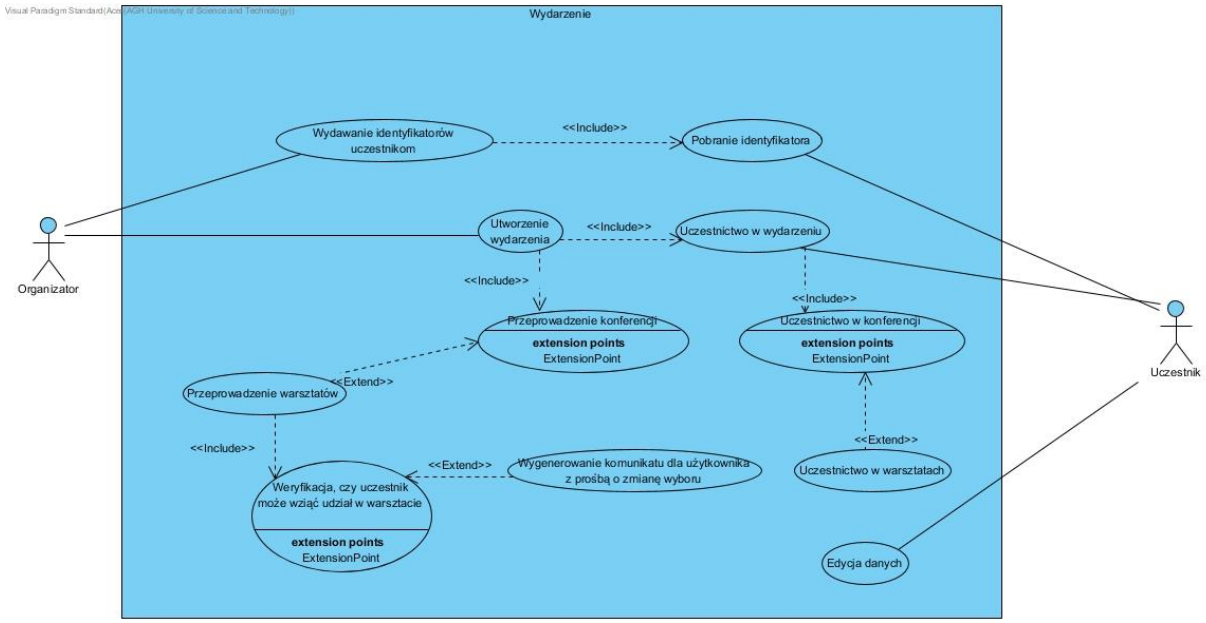
Uczestnik:
-edytuje dane personalne,
-zapisuje się na konkretne wydarzenia,
-pobiera swój identyfikator.

Klient indywidualny
-rezerwuje miejsce na konferencję/warsztaty,
-wnosi opłatę,
-anuluje rezerwację,
-edytuje dane.

1.1.2 Diagramy Use Case

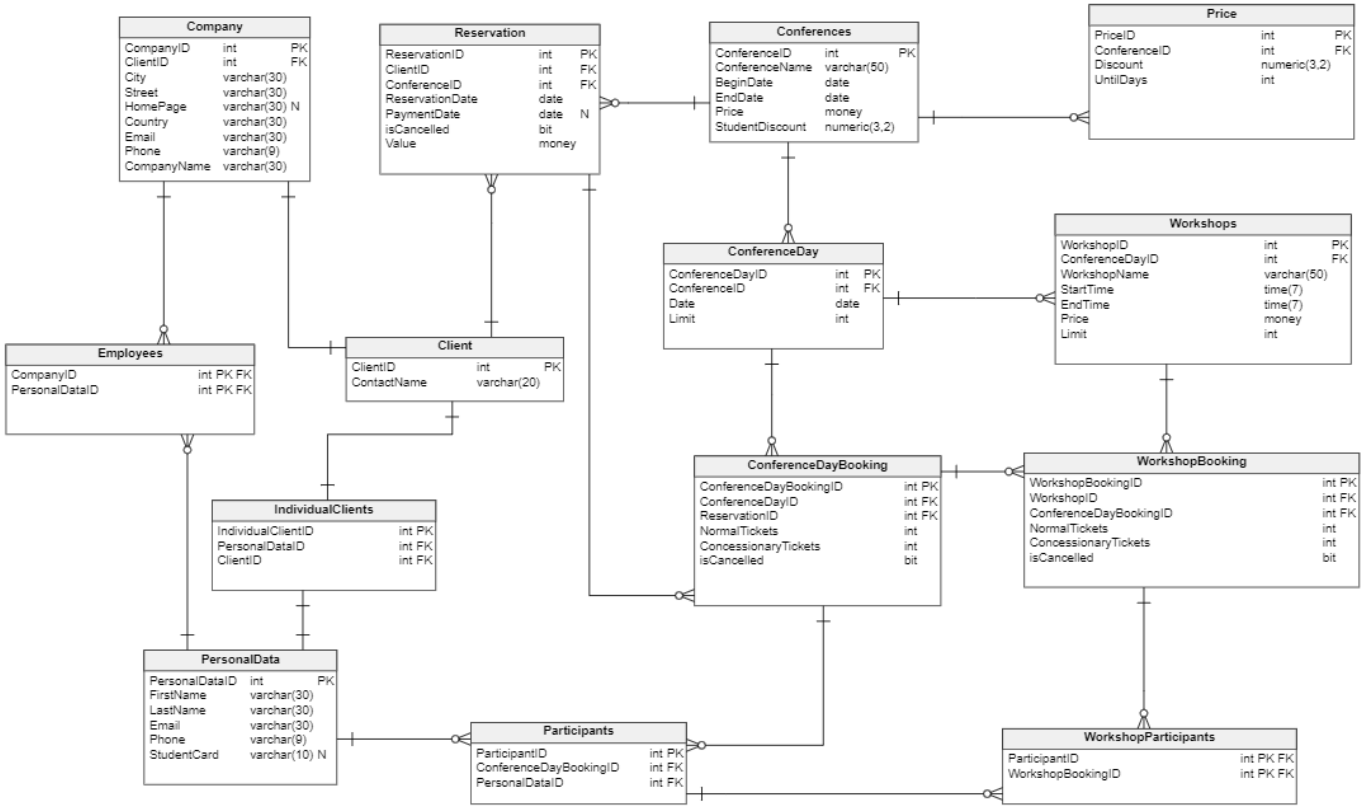


Rysunek 1 Diagram Use Case "Zapis na konferencję"



Rysunek 2 Diagram Use Case "Wydarzenie"

2. Diagram ER bazy danych



Rysunek 3 Diagram bazy danych

3. Opis tabel

3.1 Company

Tabela stanowi zbiór Informacji o Klientach Firmowych którzy mogą dokonywać rezerwacji konferencji. Tabela składa się z pól CompanyID, będącej identyfikatorem danej firmy (klucz główny), ClientID która pozwala na identyfikację firmy jako klienta dokonującego zamówienie, danych adresowych siedziby firmy (m.in. są to pola City, Street, Country), danych kontaktowych firmy – pola Email, Phone, HomePage.

```
1.  -- Table: Company
2.  CREATE TABLE Company (
3.      CompanyID int  NOT NULL IDENTITY,
4.      ClientID int  NOT NULL,
5.      City varchar(30)  NOT NULL,
6.      Street varchar(30)  NOT NULL,
7.      HomePage varchar(30)  NULL,
8.      Country varchar(30)  NOT NULL,
9.      Email varchar(30)  NOT NULL CHECK (Email like '%@%'),
10.     Phone varchar(9)  NOT NULL CHECK ((LEN(Phone)=9 AND ISNUMERIC(Phone)=1)),
11.     CompanyName varchar(30)  NOT NULL,
12.     CONSTRAINT company_uniques UNIQUE (Email, Phone, CompanyName),
13.     CONSTRAINT Company_pk PRIMARY KEY  (CompanyID)
14. );
```

3.2 Individual Clients

Tabela IndividualClients stanowi informacje o Kliencie Indywidualnym zarejestrowanym w bazie danych. Pole IndividualClientID jako klucz główny identyfikuje klienta indywidualnego, PersonalDataID jest kluczem obcym odwołującym się do rekordu w tabeli PersonalData, i stanowi połączenie do danych personalnych klienta indywidualnego, ClientID jest identyfikatorem klienta.

```
1.  -- Table: IndividualClients
2.  CREATE TABLE IndividualClients (
3.      IndividualClientID int  NOT NULL IDENTITY,
4.      PersonalDataID int  NOT NULL,
5.      ClientID int  NOT NULL,
6.      CONSTRAINT IndividualClients_pk PRIMARY KEY  (IndividualClientID)
7.  );
```

3.3 Client

Client jest tabelą przechowującą ClientID – klucz główny identyfikujący klienta oraz ContactName – nazwę kontaktową klienta.

```
1.  -- Table: Client
2.  CREATE TABLE Client (
3.      ClientID int  NOT NULL IDENTITY,
4.      ContactName varchar(20)  NOT NULL,
5.      CONSTRAINT Client_unique UNIQUE (ContactName),
6.      CONSTRAINT Client_pk PRIMARY KEY  (ClientID)
7.  );
```

3.4 PersonalData

PersonalData pełni w bazie danych funkcję magazynu informacji o danych personalnych Klientów Indywidualnych oraz Pracowników (Tabela Employees). W jej skład wchodzą pola: PersonalDataID- identyfikujące dane personalne osoby, FirstName, LastName – odpowiednio imię i nazwisko , dane kontaktowe – Email oraz Phone – które są unikalne w tabeli PersonalData. Dodatkowo mamy pole StudentCard które może przyjmować wartość NULL w przypadku gdy właściciel danych personalnych nie jest studentem, w przeciwnym przypadku zawiera numer legitymacji.

```
1.  -- Table: PersonalData
2.  CREATE TABLE PersonalData (
3.      PersonalDataID int  NOT NULL IDENTITY,
4.      FirstName varchar(30)  NOT NULL,
5.      LastName varchar(30)  NOT NULL,
6.      Email varchar(30)  NOT NULL CHECK (Email like '%@%'),
7.      Phone varchar(9)  NOT NULL CHECK ((LEN(Phone)=9 AND ISNUMERIC(Phone)=1)),
8.      StudentCard varchar(10)  NULL,
9.      CONSTRAINT PersonalData_uniques UNIQUE (Email, StudentCard, Phone),
10.     CONSTRAINT PersonalData_pk PRIMARY KEY  (PersonalDataID)
11. );
```

3.5 Employees

Employees – tabela przechowująca pracowników firm które dokonują rezerwacji konferencji. CompanyID – identyfikator firmy, do której należy dany pracownik, PersonalDataID – powiązujące pracownika z jego danymi personalnymi.

```
1.  -- Table: Employees
2.  CREATE TABLE Employees (
3.      CompanyID int  NOT NULL,
4.      PersonalDataID int  NOT NULL,
5.      CONSTRAINT PersonalDataID PRIMARY KEY  (CompanyID,PersonalDataID)
6.  );
```

3.6 Participants

ParticipantID –identyfikator uczestnika danej konferencji, ConferenceDayBookindID – klucz obcy wiążący uczestnika konferencji z dniem konferencji na który jest zapisany, PersonalDataID – identyfikator danych personalnych z tabeli PersonalDataID

```
1.  -- Table: Participants
2.  CREATE TABLE Participants (
3.      ParticipantID int  NOT NULL IDENTITY,
4.      ConferenceDayBookingID int  NOT NULL,
5.      PersonalDataID int  NOT NULL,
6.      CONSTRAINT Participants_pk PRIMARY KEY  (ParticipantID)
7.  );
```

3.7 WorkshopParticipants

Tabela jest zbiorem informacji o tym który uczestnik został zapisany na który warsztat. ParticipantID – identyfikator uczestnika, WorkshopBookingID – identyfikator rezerwacji warsztatu w którym uczestnik bierze udział.

```
1.  -- Table: WorkshopParticipants
2.  CREATE TABLE WorkshopParticipants (
3.      ParticipantID int  NOT NULL,
4.      WorkshopBookingID int  NOT NULL,
5.      CONSTRAINT WorkshopParticipants_pk PRIMARY KEY  (ParticipantID,WorkshopBookingID)
6.  );
```

3.8 Conferences

Tabela Conferences to zbiór informacji o koferencjach jakie są organizowane przez firmę. ConferenceID – identyfikator konferencji, ConferenceName – nazwa konferencji, BeginDate, EndDate – odpowiednio daty początku i zakończenia konferencji, Price – cena konferencji oraz StudentDiscount – zniżka studencka.

```
1.  -- Table: Conferences
2.  CREATE TABLE Conferences (
3.      ConferenceID int  NOT NULL IDENTITY,
4.      ConferenceName varchar(50)  NOT NULL,
5.      BeginDate date  NOT NULL,
6.      EndDate date  NOT NULL,
7.      Price money  NOT NULL CHECK (Price >= 0.0),
8.      StudentDiscount numeric(3,2)  NOT NULL DEFAULT 0 CHECK (StudentDiscount BETWEEN 0 AND 1.0),
9.      CONSTRAINT validConfDates CHECK (EndDate >= BeginDate),
10.     CONSTRAINT Conferences_pk PRIMARY KEY  (ConferenceID)
11. );
```

3.9 ConferenceDay

ConferenceDay – tabela która stanowi rozpiskę dni danej konferencji. Zawiera powiązanie z daną konferencją – ConferenceID, identyfikator dnia konferencji – ConferenceDayID, data tego dnia, Limit – limit uczestników dnia konferencji.

```
1.  -- Table: ConferenceDay
2.  CREATE TABLE ConferenceDay (
3.      ConferenceDayID int  NOT NULL IDENTITY,
4.      ConferenceID int  NOT NULL,
5.      Date date  NOT NULL,
6.      Limit int  NOT NULL CHECK (Limit >= 0),
7.      CONSTRAINT ConfDay_unique UNIQUE (Date),
8.      CONSTRAINT ConferenceDay_pk PRIMARY KEY  (ConferenceDayID)
9.  );
```

3.10 Workshops

Tabela gromadzi informacje o Warsztatach, które są rozszerzeniem konferencji. WorkshopID – identyfikator warsztatu, ConferenceDayID – powiązanie z dniem w którym odbywa się dany warsztat, StartTime, EndTime – odpowiednio godzina rozpoczęcia i zakończenia konferencji, Price – cena za warsztat, oraz Limit – limit miejsc uczestników.

```
1.  -- Table: Workshops
2.  CREATE TABLE Workshops (
3.      WorkshopID int  NOT NULL IDENTITY,
4.      ConferenceDayID int  NOT NULL,
5.      WorkshopName varchar(50)  NOT NULL,
6.      StartTime time(7)  NOT NULL,
7.      EndTime time(7)  NOT NULL,
8.      Price money  NOT NULL CHECK (Price >= 0),
9.      Limit int  NOT NULL CHECK (Limit >= 0),
10.     CONSTRAINT ProperTime CHECK (EndTime > StartTime),
11.     CONSTRAINT Workshops_pk PRIMARY KEY  (WorkshopID)
12. );
```

3.11 ConferenceDayBooking

ConferenceDayBooking – tabela zawierająca Rezerwacje danego dnia konferencji. ConferenceDayBookingID – identyfikator rezerwacji dnia, ConferenceDayID – identyfikator dnia który jest rezerwowany, ReservationID – identyfikator rezerwacji do której należy rezerwacja dnia, NormalTickets – ilość biletów normanych na rezerwowany dzień, ConcessionaryTickets – ilość studentów zapisywanych na dany dzień oraz isCanelled – przechowująca informację, czy rezerwacja danego dnia została usunięta czy nie.

```
1.  -- Table: ConferenceDayBooking
2.  CREATE TABLE ConferenceDayBooking (
3.      ConferenceDayBookingID int  NOT NULL IDENTITY,
4.      ConferenceDayID int  NOT NULL,
5.      ReservationID int  NOT NULL,
6.      NormalTickets int  NOT NULL CHECK (NormalTickets >= 0),
7.      ConcessionaryTickets int  NOT NULL CHECK (ConcessionaryTickets >= 0),
8.      isCancelled bit  NOT NULL DEFAULT 0,
9.      CONSTRAINT ConferenceDayBooking_pk PRIMARY KEY  (ConferenceDayBookingID)
10. );
```

3.12 WorkshopBooking

Tabela analogiczna do poprzedniej, przechowująca rezerwacje warsztatów. WorkshopBookingID – identyfikator rezerwacji warsztatu, WorkshopID – identyfikator warsztatu który jest rezerwowany, ConferenceDayBookingID – identyfikator rezerwacji dnia który rozszerzamy o rezerwację warsztatu, NormalTickets i ConcessionaryTickets – ilość biletów normalnych i ulgowych (dla studentów), isCancelled – informacja czy rezerwacja warsztatu została usunięta czy też nie.

```
1.  -- Table: WorkshopBooking
2.  CREATE TABLE WorkshopBooking (
3.      WorkshopBookingID int  NOT NULL IDENTITY,
4.      WorkshopID int  NOT NULL,
5.      ConferenceDayBookingID int  NOT NULL,
6.      NormalTickets int  NOT NULL CHECK (NormalTickets >= 0),
7.      ConcessionaryTickets int  NOT NULL CHECK (ConcessionaryTickets >= 0),
8.      isCancelled bit  NOT NULL DEFAULT 0,
9.      CONSTRAINT WorkshopBooking_pk PRIMARY KEY  (WorkshopBookingID)
```

```
);
```

3.13 Reservation

Tabela Reservation jest tabelą przechowującą informacje o rezerwacjach. ReservationID to identyfikator rezerwacji, ClientID identyfikator klienta dokonującego rezerwacji, ConferenceID identyfikator konferencji, ReservationDate data dokonania rezerwacji, PaymentDate – jeśli rezerwacja nie została opłacona przyjmuje wartość NULL, jeśli została to przechowuje datę uiszczenia opłaty, isCancelled informacja o tym, czy rezerwacja została anulowana, Value – koszt rezerwacji.

```
-- Table: Reservation
CREATE TABLE Reservation (
    ReservationID int  NOT NULL IDENTITY,
    ClientID int  NOT NULL,
    ConferenceID int  NOT NULL,
    ReservationDate date  NOT NULL,
    PaymentDate date  NULL,
    isCancelled bit  NOT NULL DEFAULT 0,
    Value money  NOT NULL,
    CONSTRAINT Reservation_pk PRIMARY KEY  (ReservationID)
);
```

3.14 Price

Tabela Price to tabela regulująca czasowe zniżki cen dla danej konferencji. PriceID – identyfikator ceny, ConferenceID – identyfikator konferencji, Discount – obniżka ceny przy dokonaniu rezerwacji w danym okresie czasu, UntilDays – ilość dni do danej konferencji, wyznacza przedziały czasu dla aktualnej obniżki tzn. najmniejsza wartość UntilDays oznacza najnowszą obniżkę.

```
-- Table: Price
CREATE TABLE Price (
    PriceID int  NOT NULL IDENTITY,
    ConferenceID int  NOT NULL,
    Discount numeric(3,2)  NOT NULL DEFAULT 0.0 CHECK (Discount BETWEEN 0 AND 1.0),
    UntilDays int  NOT NULL CHECK (UntilDays >= 0),
    CONSTRAINT Price_pk PRIMARY KEY  (PriceID)
);
```

4. Deklaracje kluczy obcych

```
-- foreign keys
-- Reference: Company_Client (table: Company)
ALTER TABLE Company ADD CONSTRAINT Company_Client
    FOREIGN KEY (ClientID)
    REFERENCES Client (ClientID);

-- Reference: ConferenceDayBooking_ConferenceDay (table: ConferenceDayBooking)
ALTER TABLE ConferenceDayBooking ADD CONSTRAINT ConferenceDayBooking_ConferenceDay
    FOREIGN KEY (ConferenceDayID)
    REFERENCES ConferenceDay (ConferenceDayID);

-- Reference: ConferenceDayBooking_Reservation (table: ConferenceDayBooking)
ALTER TABLE ConferenceDayBooking ADD CONSTRAINT ConferenceDayBooking_Reservation
    FOREIGN KEY (ReservationID)
    REFERENCES Reservation (ReservationID);
```

```sql

-- Reference: ConferenceDay_Conferences (table: ConferenceDay)
ALTER TABLE ConferenceDay ADD CONSTRAINT ConferenceDay_Conferences
    FOREIGN KEY (ConferenceID)
    REFERENCES Conferences (ConferenceID);

-- Reference: Employees_Company (table: Employees)
ALTER TABLE Employees ADD CONSTRAINT Employees_Company
    FOREIGN KEY (CompanyID)
    REFERENCES Company (CompanyID);

-- Reference: Employees_PersonalData (table: Employees)
ALTER TABLE Employees ADD CONSTRAINT Employees_PersonalData
    FOREIGN KEY (PersonalDataID)
    REFERENCES PersonalData (PersonalDataID);

-- Reference: IndividualClients_Client (table: IndividualClients)
ALTER TABLE IndividualClients ADD CONSTRAINT IndividualClients_Client
    FOREIGN KEY (ClientID)
    REFERENCES Client (ClientID);

-- Reference: IndividualClients_PersonalData (table: IndividualClients)
ALTER TABLE IndividualClients ADD CONSTRAINT IndividualClients_PersonalData
    FOREIGN KEY (PersonalDataID)
    REFERENCES PersonalData (PersonalDataID);

-- Reference: Participants_ConferenceDayBooking (table: Participants)
ALTER TABLE Participants ADD CONSTRAINT Participants_ConferenceDayBooking
    FOREIGN KEY (ConferenceDayBookingID)
    REFERENCES ConferenceDayBooking (ConferenceDayBookingID);

-- Reference: Participants_PersonalData (table: Participants)
ALTER TABLE Participants ADD CONSTRAINT Participants_PersonalData
    FOREIGN KEY (PersonalDataID)
    REFERENCES PersonalData (PersonalDataID);

-- Reference: Price_Conferences (table: Price)
ALTER TABLE Price ADD CONSTRAINT Price_Conferences
    FOREIGN KEY (ConferenceID)
    REFERENCES Conferences (ConferenceID);

-- Reference: Reservation_Client (table: Reservation)
ALTER TABLE Reservation ADD CONSTRAINT Reservation_Client
    FOREIGN KEY (ClientID)
    REFERENCES Client (ClientID);

-- Reference: Reservation_Conferences (table: Reservation)
ALTER TABLE Reservation ADD CONSTRAINT Reservation_Conferences
    FOREIGN KEY (ConferenceID)
    REFERENCES Conferences (ConferenceID);

-- Reference: WorkshopBooking_ConferenceDayBooking (table: WorkshopBooking)
ALTER TABLE WorkshopBooking ADD CONSTRAINT WorkshopBooking_ConferenceDayBooking
    FOREIGN KEY (ConferenceDayBookingID)
    REFERENCES ConferenceDayBooking (ConferenceDayBookingID);

-- Reference: WorkshopBooking_Workshops (table: WorkshopBooking)
ALTER TABLE WorkshopBooking ADD CONSTRAINT WorkshopBooking_Workshops
    FOREIGN KEY (WorkshopID)
    REFERENCES Workshops (WorkshopID);


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

-- Reference: WorkshopParticipants_WorkshopBooking (table: WorkshopParticipants)
ALTER TABLE WorkshopParticipants ADD CONSTRAINT WorkshopParticipants_WorkshopBooking
    FOREIGN KEY (WorkshopBookingID)
    REFERENCES WorkshopBooking (WorkshopBookingID);

-- Reference: Workshops_ConferenceDay (table: Workshops)
ALTER TABLE Workshops ADD CONSTRAINT Workshops_ConferenceDay
    FOREIGN KEY (ConferenceDayID)
    REFERENCES ConferenceDay (ConferenceDayID);
```

5. Widoki

5.1. Płatności klientów

```
--płatności klientów
create view  clients_payments_view
as
    select NameSurname as ClientName, isCancelled, Value
    from ( select PersonalDataID, FirstName + ' ' + LastName as NameSurname from PersonalData) as pd
        join IndividualClients as ic
        on pd.PersonalDataID = ic.PersonalDataID
        join Client as c
        on c.ClientID = ic.ClientID
        join Reservation as r
        on r.ClientID = c.ClientID
    group by NameSurname, isCancelled, Value

    union

    select CompanyName as ClientName, isCancelled, Value
    from Company
        join Client as c
        on c.ClientID = Company.ClientID
        join Reservation as r
        on r.ClientID = c.ClientID
    group by CompanyName, isCancelled, Value
go
```

5.2. Przychody firmy

```
--przychody
create view income
AS
    SELECT year(r.PaymentDate) as year, month(r.PaymentDate) as month, sum(r.value) as income
    FROM Reservation as r
    WHERE r.Paymentdate is not null and r.isCancelled = 0
    GROUP BY year(r.PaymentDate), month(r.PaymentDate)
    WITH ROLLUP
go
--ID klientów z ilością ich rezerwacji
create view client_popularity
as
    SELECT c.ClientID, count(*) as liczba
    from Client as C
    join Reservation as r
    on r.clientID=c.clientID and r.isCancelled = 0
    group by c.clientID
go
```

5.3 Nieopłacone rezerwacje które jeszcze nie zostały anulowane

```
--nieopłacone rezerwacje (jeszcze nie anulowane)
create view unpaid_reservations
as
    select reservationID, clientID
    from reservation
    where paymentdate is  null and isCancelled = 0
go
```

5.4 Lista firm które nie wprowadziły żadnych danych personalnych swoich pracowników

```
--lista firm, które nie wprowadziły żadnych danych swoich pracowników
create view no_participants_data_view
as
    select r.clientID, r.reservationID, r.conferenceID
    from Reservation as r
```

```
6.      join ConferenceDayBooking as cdb
7.      on cdb.ReservationID=r.ReservationID
8.      join Participants as p
9.      on p.ConferenceDayBookingID=cdb.ConferenceDayBookingID
10.     group by cdb.ConferenceDayBookingid, r.clientID, r.reservationID, r.conferenceID
11.     having count(ParticipantID) = 0
12. go
```

5.5 Rezerwacje które muszą zostać opłacone do jutra, bo zostaną anulowane

```
1.  --rezerwacje, które muszą być opłacone do jutra, bo zostaną anulowane
2.  create view by_tomorrow_should_be_paid_view
3.  as
4.      select ReservationID, ClientID
5.      from Reservation
6.      where isCancelled = 0 and paymentdate is null and datediff(day, ReservationDate, getdate()) = 6
7.  go
```

5.6 Lista konferencji z liczbą wolnych i zarezerwowanych miejsc na dany dzień

```
1.  --widok konferencji z liczbą wolnych i zarezerwowanych miejsc na dany dzień
2.
3.  create view conf_day_free_reserved_seats_view
4.  as
5.      select cd.ConferenceDayID, cd.Date, isNull([dbo].[funcConferenceDayFreePlaces] ( cd.ConferenceDayID),cd.
    Limit) as free , cd.Limit
6.      from ConferenceDay as cd
7.  go
```

5.7 Lista warsztatów, wraz z wolnymi i zarezerwowanymi miejscami

```
1.  --widok przedstawiający warsztaty wraz z wolnymi i zarezerwowanymi miejscami
2.  create view workshop_free_reserved_seats_view
3.  as
4.      select w.WorkshopID, isNull(dbo.funcWorkshopFreePlaces(w.WorkshopID),w.Limit) as free , w.Limit
5.      from Workshops as w
6.  go
```

5.8 Lista konferencji wraz z datami rozpoczęcia i zakończenia

```
1.  --widok przedstawiający listę konferencji wraz z ich datami rozpoczęcia i zakończenia
2.
3.  create view conferences_list_view
4.  as
5.      select c.ConferenceID, c.ConferenceName, c.BeginDate, c.EndDate
6.      from Conferences as c
7.  go
```

5.9 Nadchodzące konferencje (wszystkie)

```
1.  --widok przedstawiający nachodzące konferencje
2.
3.  create view upcoming_conferences_view
4.  as
5.      select c.ConferenceID, c.ConferenceName, c.BeginDate
6.      from Conferences as c
7.      where DATEDIFF(day,GETDATE(),c.BeginDate)>0
8.  go
```

5.10 Popularność warsztatów

```
1.  --popularność warsztatów
2.
3.  create view workshops_popularity_view
4.  as
```

```
5.      select w.WorkshopID, count(wb.WorkshopBookingID) as numberOfReservations, w.Limit -
    isNull(dbo.funcWorkshopFreePlaces(w.WorkshopID),w.Limit) as totalTakenSeats
6.      from Workshops as w
7.          left join WorkshopBooking as wb on wb.WorkshopID = w.WorkshopID
8.      group by w.WorkshopID,w.Limit
9.  go
```

5.11 Popularność dni konferencji

```
1.  --popularność dni konferencji
2.
3.  create view conference_days_popularity_view
4.  as
5.      select cd.ConferenceDayID, count(cdb.ConferenceDayBookingID) as numberOfReservations, cd.Limit -
    isNull(dbo.funcConferenceDayFreePlaces( cd.ConferenceDayID),cd.Limit) as totalTakenSeats
6.      from ConferenceDay as cd
7.          left join ConferenceDayBooking as cdb on cdb.ConferenceDayID = cd.ConferenceDayID
8.      group by cd.ConferenceDayID,cd.Limit
9.  go
```

6. Funkcje

6.1 Dni konferencji dla danej konferencji

```
1.  --zwraca wpisy w conferencedays dla podanego id konferencji
2.
3.  CREATE FUNCTION funcConferenceDays
4.      (
5.          @ConferenceID INT
6.      )
7.      RETURNS @days TABLE
8.      (
9.          ConferenceDayID INT,
10.         Date DATE,
11.         ParticipantsLimit INT
12.     )
13. AS
14.     BEGIN
15.         INSERT INTO @days
16.         SELECT ConferenceDayID, Date, Limit
17.         FROM ConferenceDay
18.         WHERE ConferenceID=@conferenceID
19.         RETURN;
20.     END
21. go
```

6.2 Ilość wolnych miejsc dla danej konferencji

```
1.  --Zwraca ilość wolnych miejsc na dany dzień konferencji.
2.
3.  CREATE FUNCTION funcConferenceDayFreePlaces
4.      (
5.          @ConferenceDayID INT
6.      )
7.      RETURNS INT
8.  AS
9.      BEGIN
10.         RETURN
11.         (
12.         SELECT cd.Limit - (sum(cdb.NormalTickets + cdb.ConcessionaryTickets))
13.         FROM ConferenceDay as cd
14.         LEFT JOIN ConferenceDayBooking as cdb
15.         ON cd.ConferenceDayID=cdb.ConferenceDayID AND cdb.isCancelled=0
16.         WHERE cd.ConferenceDayID=@ConferenceDayID
17.         GROUP BY cdb.ConferenceDayID, cd.Limit
18.         )
19.     END
```

```
go
```

6.3 Ilość wolnych miejsc na warsztacie

```
--zwraca ilość wolnych miejsc na warsztacie
CREATE FUNCTION funcWorkshopFreePlaces
    (
        @WorkshopID INT
    )
    RETURNS INT
AS
    BEGIN
        RETURN
        (
        SELECT w.Limit - (sum(wb.ConcessionaryTickets + wb.NormalTickets))
        FROM Workshops as w
        LEFT JOIN WorkshopBooking as wb
        ON w.WorkshopID=wb.WorkshopID AND wb.isCancelled=0
        WHERE w.WorkshopID=@WorkshopID
        GROUP BY w.WorkshopID, w.Limit
        )
    END
    go
```

6.4 Uczestnicy danej konferencji

```
--zwraca uczestników konferencji
CREATE FUNCTION funcConferenceDayParticipants
    (
        @ConferenceID INT
    )
    RETURNS @ParticipantsInfo TABLE
    (
        tConferenceID INT,
        tConferenceDayID INT,
        tParticipantID INT,
        tFirstName VARCHAR(30),
        tLastName VARCHAR(30),
        tEmail VARCHAR(30),
        tPhone VARCHAR(9),
        tStudentCart VARCHAR(10)
    )
AS
    BEGIN
        INSERT INTO @ParticipantsInfo
        SELECT c.ConferenceID, cd.ConferenceDayID, p.ParticipantID, pd.FirstName, pd.LastName, pd.Email, pd.
Phone, pd.StudentCard
        FROM Conferences AS c
        JOIN ConferenceDay AS cd
        ON c.ConferenceID=cd.ConferenceID
        JOIN ConferenceDayBooking AS cdb
        ON cdb.ConferenceDayID=cd.ConferenceDayID
        JOIN Participants AS p
        ON p.ConferenceDayBookingID=cdb.ConferenceDayBookingID
        JOIN PersonalData AS pd
        ON pd.PersonalDataID=p.PersonalDataID
        WHERE c.ConferenceID=@ConferenceID AND cdb.isCancelled = 0
        RETURN
    END
go
```

6.5 Uczestnicy warsztatu

```
--uczestnicy warsztatu
CREATE FUNCTION funcWorkshopParticipants
    (
        @WorkshopID INT
    )
    RETURNS @ParticipantsInfo TABLE
    (
        tWorkshopID INT,
        tConferenceDayID INT,
        tParticipantID INT,
        tFirstName VARCHAR(30),
        tLastName VARCHAR(30),
```

```
        tEmail VARCHAR(30),
        tPhone VARCHAR(9),
        tStudentCart VARCHAR(10)
    )
AS
    BEGIN
        INSERT INTO @ParticipantsInfo
        SELECT w.WorkshopID, cd.ConferenceDayID, p.ParticipantID, pd.FirstName, pd.LastName, pd.Email, pd.Phone, pd.StudentCard
        FROM Workshops AS w
        JOIN ConferenceDay AS cd
        ON w.ConferenceDayID=cd.ConferenceDayID
        JOIN ConferenceDayBooking AS cdb
        ON cdb.ConferenceDayID=cd.ConferenceDayID
        JOIN Participants AS p
        ON p.ConferenceDayBookingID=cdb.ConferenceDayBookingID
        JOIN PersonalData AS pd
        ON pd.PersonalDataID=p.PersonalDataID
        WHERE w.WorkshopID=@WorkshopID AND cdb.isCancelled = 0
        RETURN
    END
go
```

6.6 Aktualna obniżka dotycząca danej rezerwacji

```
CREATE FUNCTION funcReservationDiscount
    (
        @ReservationID INT
    )
    RETURNS NUMERIC(3,2)
AS
    BEGIN
        RETURN
        (
        SELECT isNull(( select top 1 Discount
                        from Price where UntilDays > DATEDIFF(DAY,r.ReservationDate,c.BeginDate) and PriceID = p.PriceID
                        order by UntilDays asc),0.0)
        FROM Reservation AS r
        JOIN Conferences as c
        ON r.ConferenceID=c.ConferenceID
        JOIN Price as p
        ON p.ConferenceID=c.ConferenceID
        WHERE r.ReservationID=@ReservationID
    )
    end
go
```

6.7 Koszt dnia konferencji dla zamówienia

```
--koszt dnia konferencji dla zamówienia

create function funcTotalCostOfConfday
    (
        @ReservationID int
    )
    returns money
as
begin
    return
    (
        select sum( cdb.NormalTickets * c.Price + cdb.ConcessionaryTickets * c.Price * (1-c.StudentDiscount))
        from ConferenceDayBooking as cdb
            join Reservation as r on r.ReservationID = cdb.ReservationID
            join Conferences as c on c.ConferenceID = r.ConferenceID
        where r.ReservationID = @ReservationID
        group by r.ReservationID
    )
end
go
```

6.8 Koszt warsztatów dla zamówienia

```
1.  --koszt warsztatów dla zamówienia
2.  create function funcTotalCostOfWorkshops
3.      (
4.          @ReservationID int
5.      )
6.      returns money
7.  as
8.  begin
9.      return
10.     (
11.         select isNull(sum(wb.NormalTickets * w.Price + wb.ConcessionaryTickets * w.Price *(1-
    c.StudentDiscount)),0)
12.         from Reservation as r
13.             left join ConferenceDayBooking as cdb on cdb.ReservationID = r.ReservationID
14.             left join WorkshopBooking as wb on wb.ConferenceDayBookingID = cdb.ConferenceDayBookingID
15.             left join Workshops as w on w.WorkshopID = wb.WorkshopID
16.             join ConferenceDay as cd on cd.ConferenceDayID = w.ConferenceDayID
17.             join Conferences as c on c.ConferenceID = cd.ConferenceID
18.         where r.ReservationID = @ReservationID
19.         group by r.ReservationID
20.
21.     )
22. end
23. go
```

6.9 Całkowity koszt rezerwacji

```
1.  -- całkowity koszt rezerwacji
2.  create function funcTotalReservationCost
3.      (
4.          @ReservationID int
5.      )
6.      returns money
7.  as
8.  begin
9.      return
10.     (
11.         cast((1-
    dbo.funcReservationDiscount(@ReservationID)) * (dbo.funcTotalCostOfWorkshops(@ReservationID) + dbo.funcTotal
    CostOfConfday(@ReservationID))  as money)
12.     )
13.
14. end
15. go
```

7. Procedury

7.1. Procedury wstawiające

7.1.1 Dodawanie ceny

```
1.  --dodawanie ceny
2.  CREATE PROCEDURE [dbo].[procAddPrice]
3.      @ConferenceID int,
4.      @Discount numeric(3,2),
5.      @UntilDays int
6.  AS
7.  BEGIN
8.      SET NOCOUNT ON
9.      BEGIN TRY
10.     IF NOT EXISTS
11.     (
12.     SELECT * FROM Conferences
13.     where ConferenceID = @ConferenceID
14.     )
15.     BEGIN
```

```
        ;THROW 52000, 'Conference does not exist. ' ,1
    END
    INSERT INTO Price
    (
    ConferenceID,
    Discount,
    UntilDays
    )
    VALUES
    (
    @ConferenceID,
    @Discount,
    @UntilDays
    )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
```

7.1.2 Dodaj konferencję

```
--dodaj konferencję
create procedure [dbo].[procAddConference]
    @Conf_name varchar(50),
    @Begin_date date,
    @End_date date,
    @Price money,
    @Student_disc numeric(3,2),
    @Limit int,
    @Disc numeric(3,2),
    @Until int
AS
BEGIN
    set nocount on
    begin transaction;
    BEGIN TRY
        IF (@Begin_date > @End_date or @Limit < 0 or @Disc <0 or @Disc >1 or @Student_disc < 0 or @Student_disc > 1)
            BEGIN
                ; throw 52000, 'Wrong data', 1
            END

        insert into Conferences(ConferenceName,BeginDate,EndDate,Price,StudentDiscount)
        values (@Conf_name,@Begin_date,@End_date,@Price,@Student_disc)
    IF @@ERROR <> 0
    begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
    end

    declare @confID INT
    set @confID = @@IDENTITY
    declare @duration int
    declare @iterator int
    set @duration = DATEDIFF(dd, @Begin_date, @End_date)
    set @iterator = 0

    while @iterator <= @duration
    begin
        insert into ConferenceDay (ConferenceID, Date, Limit) values (@confID, cast(DATEADD(dd, @iterator, @Begin_date) as date),@Limit)

        IF @@ERROR <> 0
        begin
            rollback transaction;
            RAISERROR('Error, transaction not completed!',16,-1)
        end
        if @iterator = 0
        begin
            exec procAddPrice @ConferenceID=@confID, @Discount = @Disc, @UntilDays = @Until;
            IF @@ERROR <> 0
            begin
                RAISERROR('Error, transaction not completed!',16,-1)
                rollback transaction;
            end
        end
        set @iterator = @iterator + 1
    end
```

```
    commit transaction;
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
            = 'Cannot add conference . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
        rollback transaction;
    END CATCH
END
GO
```

7.1.3 Tworzenie nowego warsztatu

```
--tworzenie nowego warsztatu
create procedure [dbo].[procAddWorkshop]
    @ConfDayID INT,
    @WorkshopName VARCHAR(50),
    @StartTime TIME(7),
    @EndTime TIME(7),
    @Price money,
    @WorkshopLimit INT
AS
BEGIN
    set nocount on
    BEGIN TRY
        IF NOT EXISTS
            ( select * from ConferenceDay where ConferenceDayID = @ConfDayID )
        BEGIN
            THROW 52000, 'Conference day does not exist', 1
        END
        IF EXISTS
            (select * from Workshops
             where WorkshopName = @WorkshopName and StartTime = @StartTime and EndTime = @EndTime)
        BEGIN
            THROW 52000, 'Such workshop already exists', 1
        END
        insert into Workshops (ConferenceDayID, WorkshopName, StartTime, EndTime, Price, Limit)
        values (@ConfDayID, @WorkshopName, @StartTime, @EndTime, @Price, @WorkshopLimit)
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar (2048)
            = 'Cannot add workshop . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
GO
```

7.1.4 Dodanie rezerwacji na warsztat przez klienta firmowego

```
--Dodanie rezerwacji przez klienta firmowego
create PROCEDURE [dbo].[procAddWorkshopBookingCompany]
    @WorkshopID int,
    @ConferenceDayBookingID int,
    @StudentsNo int,
    @NormalNo int
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
    begin transaction;
    IF NOT EXISTS
        (
        SELECT * FROM Workshops
        WHERE WorkShopID = @WorkShopID
        )
    BEGIN
        ;THROW 52000, 'Workshop does not exist. ' ,1
    END
    IF NOT EXISTS
        (
        SELECT * FROM ConferenceDayBooking
        WHERE  ConferenceDayBookingID = @ConferenceDayBookingID
        )
```

```
    BEGIN
        ;THROW 52000, 'ConferenceDayBooking does not exist. ' ,1
    END
    IF (@StudentsNo + @NormalNo = 0) or @StudentsNo < 0 or @NormalNo < 0
    BEGIN
        ;THROW 52000, 'Not acceptable tickets values ' ,1
    END

    if (select ConferenceDayID from ConferenceDayBooking where @ConferenceDayBookingID = ConferenceDayBookin
gID) <> (select ConferenceDayID from Workshops where @WorkshopID = WorkshopID)
    BEGIN
        ;THROW 52000, 'Workshop does not belong to that conference day' ,1
    END

    INSERT INTO WorkshopBooking
        (
        WorkShopID,
        ConferenceDayBookingID,
        NormalTickets,
        ConcessionaryTickets,
        isCancelled
        )
    VALUES
        (
        @WorkshopID,
        @ConferenceDayBookingID,
        @StudentsNo,
        @NormalNo,
        0
        )
    commit transaction;
    END TRY
    BEGIN CATCH
        rollback transaction;
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO
```

7.1.5 Dodanie rezerwacji na warsztat przez klienta indywidualnego

```
-- Dodanie rezerwacji na warsztat przez klienta indywidualnego
CREATE PROCEDURE [dbo].[procAddWorkshopBookingIndividual]
    @WorkshopID int,
    @ConferenceDayBookingID int,
    @StudentCard varchar(10) = null
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
        begin transaction;
    IF NOT EXISTS
        (
        SELECT * FROM Workshops
        WHERE WorkShopID = @WorkShopID
        )
    BEGIN
        ;THROW 52000, 'Workshop does not exist. ' ,1
    END
    IF NOT EXISTS
        (
        SELECT * FROM ConferenceDayBooking
        WHERE  ConferenceDayBookingID = @ConferenceDayBookingID
        )
    BEGIN
        ;THROW 52000, 'ConferenceDayBooking does not exist. ' ,1
    END
    if @StudentCard is not null and not exists (select *
    from Reservation as r
        left join Client as c on c.ClientID = r.ClientID
        left join IndividualClients as ic on ic.ClientID = c.ClientID
        left join PersonalData as pd on pd.PersonalDataID = ic.PersonalDataID
        where @StudentCard = pd.StudentCard)
    BEGIN
    THROW 52000, 'Student Card does not exists', 1
    END
    if (select ConferenceDayID from ConferenceDayBooking where @ConferenceDayBookingID = ConferenceDayBookin
gID) <> (select ConferenceDayID from Workshops where @WorkshopID = WorkshopID)
    BEGIN
```

```
        ;THROW 52000, 'Workshop does not belong to that conference day' ,1
    END
    if @StudentCard is not null
    BEGIN
        insert into WorkshopBooking (WorkshopID,ConferenceDayBookingID, NormalTickets, ConcessionaryTickets,
 isCancelled)
        values ( @WorkshopID, @ConferenceDayBookingID, 0, 1, 0)
    END
    if @StudentCard is null
    BEGIN
        insert into WorkshopBooking (WorkshopID,ConferenceDayBookingID, NormalTickets, ConcessionaryTickets,
 isCancelled)
        values ( @WorkshopID, @ConferenceDayBookingID, 1, 0, 0)
    END
        commit transaction;
    END TRY
    BEGIN CATCH
        rollback transaction;
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO
```

7.1.6 Utworzenie rezerwacji przez klienta indywidualnego

```
--Utworzenie rezerwacji przez klienta indywidualnego
CREATE PROCEDURE [dbo].[procAddReservationIndividual]
    @ClientID int,
    @ConferenceID int,
    @ReservationDate date,
    @PaymentDate date,
    @isCancelled bit = 0,
    @Value money
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
    IF NOT EXISTS
    (
    SELECT * FROM Client
    WHERE ClientID = @ClientID
    )
    BEGIN
        ;THROW 52000, 'Client does not exist. ' ,1
    END
    IF NOT EXISTS
    (
    SELECT * FROM IndividualClients as ic
    WHERE ic.ClientID = @ClientID
    )
    BEGIN
        ;THROW 52000, 'Client is not individual person. ' ,1
    END
    IF NOT EXISTS
    (
    SELECT * FROM Conferences
    WHERE ConferenceID = @ConferenceID
    )
    BEGIN
        ;THROW 52000, 'Conference does not exist. ' ,1
    END
    INSERT INTO Reservation
        (
        ClientID,
        ConferenceID,
        ReservationDate,
        PaymentDate,
        isCancelled,
        Value
        )
        VALUES
        (
        @ClientID,
        @ConferenceID,
        @ReservationDate,
        @PaymentDate,
        @isCancelled,
        @Value
        )
```

```
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO
```

7.1.7 Utworzenie rezerwacji przez klienta firmowego

```
--Utworzenie rezerwacji przez klienta firmowego
CREATE PROCEDURE [dbo].[procAddReservationCompany]
    @ClientID int,
    @ConferenceID int,
    @ReservationDate date,
    @PaymentDate date,
    @isCancelled bit = 0,
    @Value money
AS
BEGIN
    SET NOCOUNT ON
    BEGIN TRY
    IF NOT EXISTS
    (
    SELECT * FROM Client
    WHERE ClientID = @ClientID
    )
    BEGIN
        ;THROW 52000, 'Client does not exist. ' ,1
    END
    IF NOT EXISTS
    (
    SELECT * FROM Company as c
    WHERE c.ClientID = @ClientID
    )
    BEGIN
        ;THROW 52000, 'Client is not company. ' ,1
    END
    IF NOT EXISTS
    (
    SELECT * FROM Conferences
    WHERE ConferenceID = @ConferenceID
    )
    BEGIN
        ;THROW 52000, 'Conference does not exist. ' ,1
    END
    INSERT INTO Reservation
        (
        ClientID,
        ConferenceID,
        ReservationDate,
        PaymentDate,
        isCancelled,
        Value
        )
        VALUES
        (
        @ClientID,
        @ConferenceID,
        @ReservationDate,
        @PaymentDate,
        @isCancelled,
        @Value
        )
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO
```

7.1.8 Dodanie danych personalnych

```
--dodawanie danych personalnych
```

```
create procedure [dbo].[procAddPerson]
    @FirstName varchar(30),
    @LastName varchar(30),
    @Email varchar(30),
    @Phone varchar(9),
    @StudentCard varchar (10) = null
as
begin
    set nocount on;
    begin try
    begin transaction;
    if exists
    (
    select * from PersonalData
    where Email=@Email or Phone=@Phone
    )
    BEGIN
            ; THROW 52000, 'Participant already exists.',1
        END
    insert into
    PersonalData
    (
    FirstName,
    LastName,
    Email,
    Phone,
    StudentCard
    )
    values
    (
    @FirstName,
    @LastName,
    @Email,
    @Phone,
    @StudentCard
    )
    commit transaction;
    end try
    BEGIN CATCH
    rollback transaction;
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add Person. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO
```

7.1.9 Dodawanie klienta firmowego

```
--dodawanie klienta firmowego
create procedure [dbo].[procAddCompanyClient]
    @ContactName varchar(20),
    @City varchar(30),
    @Street varchar(30),
    @HomePage varchar(30) = null,
    @Country varchar(30),
    @Email varchar(30),
    @Phone varchar(9),
    @CompanyName varchar(30)
AS
BEGIN
    set nocount on
    BEGIN TRY
    begin transaction;
    IF EXISTS
        (select * from Client where ContactName = @ContactName) or EXISTS
        (select * from Company
         where City = @City and Street = @Street and Email = @Email
            and Country = @Country and Phone = @Phone and CompanyName = @CompanyName)
    BEGIN
        THROW 52000, 'Client/Company already exist', 1
    END

    declare @tempID INT

    insert into Client( ContactName )
    values (@ContactName)
    IF @@ERROR <> 0
        begin
            rollback transaction;
            RAISERROR('Error, transaction not completed!',16,-1)
```

```
        end
    set @tempID = @@IDENTITY

    if exists (select * from Company where Email = @Email or Phone = @Phone or CompanyName = @CompanyName)
    BEGIN
        THROW 52000, 'Client/Company already exist', 1
    END

    if @Email not like '%@%' or LEN(@Phone) <> 9 or ISNUMERIC(@Phone) <> 1
    BEGIN
        THROW 52000, 'wrong data', 1
    END
    insert into Company(ClientID, City, Street, HomePage, Country, Email, Phone, CompanyName)
    values (@tempID, @City, @Street, @HomePage, @Country, @Email, @Phone, @CompanyName)
    commit transaction;
    END TRY
    BEGIN CATCH
    rollback transaction;
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot add Company Client . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
GO
```

7.1.10 Dodawanie klienta indywidualnego

```
--dodawanie klienta indywidualnego

create procedure [dbo].[procAddIndividualClient]
    @ContactName varchar (20),
    @FirstName varchar(30),
    @LastName varchar(30),
    @Email varchar(30),
    @Phone varchar(9),
    @StudentCard varchar (10) = null
as
begin
    set nocount on;
    begin try
    begin transaction;
    if  exists
        (select * from PersonalData
        where Email=@Email or Phone=@Phone or @StudentCard = StudentCard)
    BEGIN
        ; THROW 52000, 'Person already exists.',1
    END

    if @Email not like '%@%' or LEN(@Phone) <> 9 or ISNUMERIC(@Phone) <> 1
    BEGIN
        THROW 52000, 'wrong data', 1
    END

    if exists (select * from Client where ContactName = @ContactName)
    BEGIN
        ; THROW 52000, 'Client already exists.',1
    END
    insert into Client(ContactName) values (@ContactName)
    IF @@ERROR <> 0
    begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
    end
    declare @ClientID int
    set @ClientID = @@IDENTITY

    exec procAddPerson  @FirstName, @LastName, @Email, @Phone, @StudentCard
    IF @@ERROR <> 0
    begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
    end
    declare @id int = @@IDENTITY
    insert into IndividualClients(PersonalDataID, ClientID) values (@id, @ClientID)
    IF @@ERROR <> 0
    begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
```

```
    end
    commit transaction;
    end try
    BEGIN CATCH
    rollback transaction;
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add IndividualClient. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO
```

7.1.11 Dodawanie pracownika

```
--dodawanie pracownika

create procedure procAddEmployee
    @firstName varchar(30),
    @lastName varchar(30),
    @Email varchar(30),
    @Phone varchar(9),
    @CompID int,
    @StudentCard varchar(10) = null
as
begin
    set nocount on
    begin try
        begin transaction
        if exists (select * from PersonalData where Email = @Email and Phone = @Phone and @firstName = First
Name and @lastName = LastName)
        begin
            THROW 52000, 'Employee Data already exists', 1
        end

        declare @empID int
        exec dbo.procAddPerson @firstName, @lastName, @Email,@Phone, @StudentCard

        IF @@ERROR <> 0
        begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
        end

        set @empID = @@IDENTITY
        insert into Employees (CompanyID, PersonalDataID) values (@CompID, @empID)

        IF @@ERROR <> 0
        begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
        end

        commit transaction;
    end try
    begin CATCH
    rollback transaction;
    declare @errorMsg nvarchar (2048)
            = 'Cannot add Company Client . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    end CATCH
end
GO
```

7.1.12 Dodawanie rezerwacji dnia konferencji przez klienta indywidualnego

```
--dodawanie bookingu dnia konferencji (indywidualny klient)
create procedure procAddConferenceDayBookingIndividual
    @ConferenceDayID INT,
    @ReservationID INT,
    @StudentCard varchar(10) =  null
as
BEGIN
SET NOCOUNT ON
    BEGIN TRY
```

```
    begin transaction;
    if NOT EXISTS
        (select * from ConferenceDay where ConferenceDayID = @ConferenceDayID)
    BEGIN
    THROW 52000, 'Conference day does not exists ', 1
    END
    if NOT EXISTS
        (select * from Reservation where ReservationID = @ReservationID)
    BEGIN
    THROW 52000, 'Reservation does not exists ', 1
    END
    if (select ConferenceID from Reservation where ReservationID = @ReservationID )<> (select ConferenceID f
rom ConferenceDay where ConferenceDayID = @ConferenceDayID )
    BEGIN
    THROW 52000, 'Diffrenct conference is reserved', 1
    END
    if @StudentCard is not null and not exists (select *
    from Reservation as r
        left join Client as c on c.ClientID = r.ClientID
        left join IndividualClients as ic on ic.ClientID = c.ClientID
        left join PersonalData as pd on pd.PersonalDataID = ic.PersonalDataID
        where @StudentCard = pd.StudentCard)
    BEGIN
    THROW 52000, 'Student Card does not exists', 1
    END
    if @StudentCard is not null
    BEGIN
        insert into ConferenceDayBooking (ConferenceDayID, ReservationID, NormalTickets, ConcessionaryTicket
s, isCancelled)
        values ( @ConferenceDayID, @ReservationID, 0, 1, 0)
    END
    if @StudentCard is null
    BEGIN
        insert into ConferenceDayBooking (ConferenceDayID, ReservationID, NormalTickets, ConcessionaryTicket
s, isCancelled)
        values ( @ConferenceDayID, @ReservationID, 1, 0, 0)
    END
    commit transaction;
    END TRY
    BEGIN CATCH
    rollback transaction;
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot add Conference day booking . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
GO
```

7.1.13 Dodawanie rezerwacji dnia konferencji przez klienta firmowego

```
--dodawanie bookingu firmy na dzień konferencji
create procedure procAddConferenceDayBookingCompany
    @ConferenceDayID INT,
    @ReservationID INT,
    @NormalTickets INT,
    @ConcessionaryTickets INT
as
BEGIN
SET NOCOUNT ON
    BEGIN TRY
    begin transaction;

    if NOT EXISTS
        (select * from ConferenceDay where ConferenceDayID = @ConferenceDayID)
    BEGIN
    THROW 52000, 'Conference day does not exists ', 1
    END

    if (select ConferenceID from Reservation where ReservationID = @ReservationID )<> (select ConferenceID f
rom ConferenceDay where ConferenceDayID = @ConferenceDayID )
    BEGIN
    THROW 52000, 'Diffrenct conference is reserved', 1
    END

    if NOT EXISTS
        (select * from Reservation where ReservationID = @ReservationID)
    BEGIN
    THROW 52000, 'Reservation does not exists ', 1
    END

```

```
30.     IF (@NormalTickets + @ConcessionaryTickets = 0) or @ConcessionaryTickets < 0 or @NormalTickets < 0
31.     BEGIN
32.         ;THROW 52000, 'Not acceptable tickets values ' ,1
33.     END
34.
35.     insert into ConferenceDayBooking (ConferenceDayID, ReservationID, NormalTickets, ConcessionaryTickets, i
    sCancelled)
36.     values (@ConferenceDayID, @ReservationID, @NormalTickets, @ConcessionaryTickets, 0)
37.
38.     commit transaction;
39.     END TRY
40.     BEGIN CATCH
41.     rollback transaction;
42.     DECLARE @errorMsg nvarchar (2048)
43.             = 'Cannot add Conference day booking . Error message : '
44.                 + ERROR_MESSAGE () ;
45.         ; THROW 52000 , @errorMsg ,1
46.     END CATCH
47. END
48. GO
```

7.1.14 Dodawanie klienta indywidualnego jako członka konferencji

```
1.  --Dodaj indywidualnego członka konferencji
2.  create procedure [dbo].[procAddConferenceIndividualParticipant]
3.      @ConferenceDayBookingID int,
4.      @FirstName varchar(30),
5.      @LastName varchar (30),
6.      @Email varchar (30),
7.      @Phone varchar (9)
8.  as
9.  begin
10. set nocount on
11. begin try
12.     begin transaction;
13.     if not exists (select * from PersonalData where FirstName = @FirstName and LastName =@LastName and Email
     =  @Email and @Phone= Phone  )
14.     begin
15.     ; throw 52000, 'Given Person does not exist ',1
16.     end
17.     if not exists ( select * from ConferenceDayBooking where ConferenceDayBookingID = @ConferenceDayBookingI
    D)
18.     begin
19.     ; throw 52000, 'Wrong ConferenceDayBookingID. ConferenceDayBookingID does not exist ',1
20.     end
21.     declare @PersonalDataID int = (select PersonalDataID from PersonalData where FirstName = @FirstName and
    LastName = @LastName and Email = @Email and Phone = @Phone)
22.     if not exists ( select * from IndividualClients where PersonalDataID = @PersonalDataID)
23.     begin
24.     ; throw 52000, 'This person is not IndividualClient ',1
25.     end
26.     insert into Participants(ConferenceDayBookingID, PersonalDataID)
27.     values (@ConferenceDayBookingID, @PersonalDataID)
28.     commit transaction;
29.     end try
30.     begin catch
31.     rollback transaction;
32.         DECLARE @errorMsg nvarchar(2048)
33.         = 'Can not add ConferenceIndividualParticipant. Error message: ' + ERROR_MESSAGE();
34.         ;THROW 52000, @errorMsg,1
35.     END CATCH
36. END
37. GO
38.
```

7.1.15. Dodaj pracownika firmy jako uczestnika konferencji

```
1.  --Dodaj firmowego Uczestika konferencji
2.  create procedure [dbo].[procAddConferenceCompanyParticipant]
3.      @ConferenceDayBookingID int,
4.      @FirstName varchar(30),
5.      @LastName varchar (30),
6.      @Email varchar (30),
7.      @Phone varchar (9),
8.      @CompanyID int,
```

```
    @StudentCard varchar(10) = null
as
begin
set nocount on;
begin try
begin transaction;
    if not exists ( select * from ConferenceDayBooking where ConferenceDayBookingID = @ConferenceDayBookingID)
    begin
        ; throw 52000, 'Wrong ConferenceDayBookingID. ConferenceDayBookingID does not exist ',1
    end
    if exists (select * from PersonalData where FirstName = @FirstName and LastName = @LastName and  Email = @Email and Phone = @Phone)
    begin
        declare @PersonalDataID int = (select PersonalDataID from PersonalData where FirstName = @FirstName and LastName = @LastName and  Email = @Email and Phone = @Phone)
        if exists (select * from Employees where PersonalDataID=@PersonalDataID and CompanyID = @CompanyID)

        begin
            insert into Participants (ConferenceDayBookingID, PersonalDataID)
            values (@ConferenceDayBookingID, @PersonalDataID)
        end
    end
if not exists (select * from PersonalData where FirstName = @FirstName and LastName = @LastName and  Email = @Email and Phone = @Phone)
    begin
    exec procAddPerson @FirstName, @LastName, @Email, @Phone, @StudentCard
    IF @@ERROR <> 0
        begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
        end
    declare @PDataID int  = @@Identity
    insert into Employees (CompanyID, PersonalDataID) values (@CompanyID, @PDataID)
    IF @@ERROR <> 0
        begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
        end
    insert into Participants (ConferenceDayBookingID, PersonalDataID) values (@ConferenceDayBookingID, @PDataID)
    IF @@ERROR <> 0
        begin
        RAISERROR('Error, transaction not completed!',16,-1)
        rollback transaction;
        end
    end
    commit transaction;
    end try
    begin catch
    rollback transaction;
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add ConferenceCompanyParticipant. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO


```

7.1.16. Dodaj firmowego uczestnika warsztatów

```
-- Dodaj firmowego uczestnika warsztatów

 create procedure [dbo].[procAddWorkshopCompanyParticipant]
    @ParticipantID int,
    @WorkshopBookingID int
as
begin
set nocount on;
begin try
begin transaction;
    if not exists ( select * from Participants where ParticipantID = @ParticipantID)
    begin
        ; throw 52000, 'Wrong ParticipantID. ParticipantID does not exist ',1
    end
    if not exists ( select * from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
    begin
        ; throw 52000, 'Wrong WorkshopBookingID. WorkshopBookingID does not exist ',1
```

```
    end
    if (select ConferenceDayBookingID from Participants where ParticipantID = @ParticipantID) <>
       (select ConferenceDayBookingID from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
    begin
        ; throw 52000, 'Participant can not take part in this workshop (wrong conferencedaybookingid) ',1
    end
        insert into WorkshopParticipants(ParticipantID, WorkshopBookingID)
        values (@ParticipantID, @WorkshopBookingID)
commit transaction;
    end try
    begin catch
    rollback transaction;
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add WorkshopParticipant. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
GO
```

7.1.17 Dodaj klienta indywidualnego jako uczestnika warsztatów

```
--Dodaj uczestnika warsztatów
create procedure [dbo].[proccAddWorkshopIndividualParticipant]
    @ParticipantID int,
    @WorkshopBookingID int
as
begin
set nocount on;
begin try
begin transaction;
    if not exists ( select * from Participants where ParticipantID = @ParticipantID)
    begin
        ; throw 52000, 'Wrong ParticipantID. ParticipantID does not exist ',1
    end
    if not exists ( select * from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
    begin
        ; throw 52000, 'Wrong WorkshopBookingID. WorkshopBookingID does not exist ',1
    end
    if not exists ( select * from IndividualClients as ic
                    where ic.PersonalDataID = (select PersonalDataID from Participants where ParticipantID =
 @ParticipantID))
    begin
        ; throw 52000, 'Participant is not individual client ',1
    end
    if (select ConferenceDayBookingID from Participants where ParticipantID = @ParticipantID) <>
       (select ConferenceDayBookingID from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
    begin
        ; throw 52000, 'Participant can not take part in this workshop (wrong conferencedaybookingid) ',1
    end
    insert into WorkshopParticipants(ParticipantID, WorkshopBookingID)
    values (@ParticipantID, @WorkshopBookingID)
    commit transaction;
    end try
    begin catch
    rollback transaction;
        DECLARE @errorMsg nvarchar(2048)
        = 'Can not add WorkshopParticipant. Error message: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END

GO
```

7.2 Procedury statystyczne

7.2.1 Procedura pokazująca ilość rezerwacji dla danego klienta (uporządkowane malejąco)

```
--popularnosc klientów (ilosc rezerwacji)
CREATE PROCEDURE show_clients_popularity
```

```
as
BEGIN
    SELECT  Name, COUNT(*) AS Number
    FROM (select PersonalDataID, FirstName + ' ' +LastName as Name from PersonalData) as pd
    JOIN IndividualClients as ic
    ON pd.PersonalDataID=ic.PersonalDataID
    JOIN Client as c
    ON ic.ClientID=c.ClientID
    JOIN Reservation as r
    ON r.ClientID = c.ClientID
    GROUP BY Name

    UNION

    SELECT co.CompanyName as Name, COUNT(*) AS Number
    FROM Company as co
    JOIN Client as cl
    ON co.ClientID=cl.ClientID
    JOIN Reservation as r
    ON r.ClientID=cl.ClientID
    GROUP BY co.CompanyName
END
go
```

7.2.2 Lista uczestników dni konferencji dla wybranej konferencji

```
--lista uczestników dni konferencji dla wybranej konferencji
create PROCEDURE proc_showConferenceDaysParticipants
    @confID int
as
begin
    select c.ConferenceName, cd.Date, pd.FirstName,  pd.LastName
    from Conferences as c
        join ConferenceDay as cd
        on cd.ConferenceID = c.ConferenceID
        join ConferenceDayBooking as cdb
        on cdb.ConferenceDayID = cd.ConferenceDayID
        join Participants as p
        on p.ConferenceDayBookingID = cdb.ConferenceDayBookingID
        join PersonalData as pd
        on pd.PersonalDataID = p.PersonalDataID
    where c.ConferenceID = @confID
    group by ConferenceName, cd.Date,  pd.FirstName,  pd.LastName
    order by 1,2,4
end
go
```

7.2.3 Lista uczestników dla poszczególnych warsztatów w obrębie konferencji

```
-- lista uczestników dla danego warsztaty w obrębie konferencji
create PROCEDURE proc_ShowWorkshopParticipants
    @confID int
as
begin
    select ConferenceName, WorkshopName, pd.FirstName, pd.LastName
    from Conferences as c
        join ConferenceDay as cd
        on cd.ConferenceID = c.ConferenceID
        join Workshops as w
        on w.ConferenceDayID = cd.ConferenceDayID
        join WorkshopBooking as wb
        on wb.WorkshopID = w.WorkshopID
        join WorkshopParticipants as wp
        on wp.WorkshopBookingID = wb.WorkshopBookingID
        join Participants as p
        on p.ParticipantID = wp.ParticipantID
        join PersonalData as pd
        on pd.PersonalDataID = p.PersonalDataID
    where c.ConferenceID = @confID
    group by ConferenceName, WorkshopName, pd.FirstName, pd.LastName
end
go
```

7.2.4 Lista wydarzeń które odbywają się na danej konferencji

```
-- lista wydarzen w obrębie konferencji

create procedure proc_Events
    @ConferenceID INT
AS
BEGIN
    set nocount on
    BEGIN TRY
    if NOT EXISTS (select * from Conferences where ConferenceID = @ConferenceID)
    BEGIN
        THROW 52000, 'Conference does not exists ', 1
    END
    select c.ConferenceName, cd.ConferenceDayID, (DATEDIFF(dd, c.BeginDate, cd.Date) + 1) as ConferenceDayNo
, isNull(w.WorkshopName, 'no workshop') as WorkshopName
    from Conferences as c
        join ConferenceDay as cd
        on c.ConferenceID = cd.ConferenceID
        left join Workshops as w
        on w.ConferenceDayID = cd.ConferenceDayID
    where c.ConferenceID = @ConferenceID
    group by c.ConferenceName, cd.ConferenceDayID, DATEDIFF(dd, c.BeginDate, cd.Date), w.WorkshopName
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot add Conference day booking . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.2.5 Lista uczestników danego warsztatu

```
--uczestnicy danego warsztatu
create procedure workshop_participants
 @WID int
as
BEGIN
    BEGIN TRY
    set nocount on
    if not exists (select * from Workshops where WorkshopID = @WID)
    begin
        THROW 52000, 'Workshop does not exists ', 1
    end
    select wp.ParticipantID, pd.FirstName, pd.LastName, pd.Phone
    from Workshops as w
    left join WorkshopBooking as wb
    on wb.workshopID=w.workshopID
    left join WorkshopParticipants as wp
    on wp.WorkshopBookingID=wb.WorkshopBookingID
    left join Participants as p
    on p.ParticipantID=wp.ParticipantID
    left join personaldata as pd
    on pd.PersonalDataID = p.PersonalDataID
    where w.WorkshopID = @WID
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot find participants . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.2.6 Lista uczestników danej konferencji

```
--uczestnicy danej konferencji
create procedure conference_participants
 @CID int
as
BEGIN
    BEGIN TRY
    set nocount on
    if not exists (select * from Conferences where ConferenceID = @CID)
```

```
    begin
        THROW 52000, 'Conference does not exists ', 1
    end
    select p.ParticipantID, pd.FirstName, pd.LastName, pd.Phone
    from Conferences as c
    join ConferenceDay as cd
    on cd.ConferenceID=c.ConferenceID
    join ConferenceDayBooking as cdb
    on cdb.ConferenceDayID=cd.ConferenceDayID
    join Participants as p
    on p.ConferenceDayBookingID = cdb.ConferenceDayBookingID
    join personaldata as pd
    on pd.PersonalDataID = p.PersonalDataID
    where c.ConferenceID = @CID
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot find participants . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.2.7 Lista uczestników danego dnia konferencji

```
--uczestnicy danego dnia konferencji
create procedure conferenceday_participants
 @CDID int
as
BEGIN
    BEGIN TRY
    set nocount on
    if not exists (select * from ConferenceDay where ConferenceDayID = @CDID)
    begin
        THROW 52000, 'ConferenceDay does not exists ', 1
    end
    select p.ParticipantID, pd.FirstName, pd.LastName, pd.Phone
    from ConferenceDay as cd
    join ConferenceDayBooking as cdb
    on cdb.ConferenceDayID=cd.ConferenceDayID
    join Participants as p
    on p.ConferenceDayBookingID = cdb.ConferenceDayBookingID
    join personaldata as pd
    on pd.PersonalDataID = p.PersonalDataID
    where cd.ConferenceDayID = @CDID
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot find participants . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.2.8 Lista konferencji danego uczestnika

```
-- konferencje danego uczestnika
create procedure participant_conferences
 @PID int
as
BEGIN
    BEGIN TRY
    set nocount on
    if not exists (select * from Participants where ParticipantID = @PID)
    begin
        THROW 52000, 'CParticipant does not exists ', 1
    end
    select c.ConferenceID, c.ConferenceName
    from Participants as p
        join ConferenceDayBooking as cdb on cdb.ConferenceDayBookingID = p.ConferenceDayBookingID
        join ConferenceDay as cd on cdb.ConferenceDayID = cd.ConferenceDayID
        join Conferences as c on c.ConferenceID = cd.ConferenceID
    where p.ParticipantID = @PID
    END TRY
    BEGIN CATCH
```

```
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot find participant . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.2.9 Wpływy z rezerwacji dla danej konferencji

```
-- wpływy z rezerwacji dla wybranej konfernecji
create procedure conference_incomes
 @CID int
as
BEGIN
    BEGIN TRY
    set nocount on
    if not exists (select * from Conferences where ConferenceID = @CID)
    begin
        THROW 52000, 'Conference does not exists ', 1
    end
    select r.ReservationID, sum(r.Value) as Sum
    from Reservation as r
    where r.ConferenceID = @CID and r.isCancelled = 0
    group by r.ReservationID
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot find conference . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.2.10 Lista konferencji które odbędą się w przeciągu podanej liczby dni

```
-- nadchodzące konferencje w przeciągu x dni
create procedure comming_conferences
 @days int
as
BEGIN
    BEGIN TRY
    set nocount on
    if (@days<0)
    begin
        THROW 52000, 'a negative number ', 1
    end
    select c.ConferenceID, c.ConferenceName
    from Conferences as c
    where c.Begindate >= getdate() and c.begindate<=dateadd(day,@days,getdate())
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot find participant . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.2.11 Lista wydarzeń na które podany uczestnik został zapisany

```
--procedura-widok sprawdzenie na co jest się zapisanym

create procedure enrollments
    @id int
as
BEGIN
    BEGIN TRY
    set nocount on
```

```
    if not exists (select * from Participants where ParticipantID = @id)
    BEGIN
        THROW 52000, 'Participant does not exists ', 1
    END
    select w.WorkshopID, w.StartTime, w.EndTime, cd.ConferenceDayID, cd.Date
    from Participants as p
         join ConferenceDayBooking as cdb on cdb.ConferenceDayBookingID = p.ConferenceDayBookingID
         join WorkshopBooking as wb on wb.ConferenceDayBookingID = cdb.ConferenceDayBookingID
         join ConferenceDay as cd on cd.ConferenceDayID = cdb.ConferenceDayID
         join Workshops as w on w.WorkshopID = wb.WorkshopID
    where p.ParticipantID = @id and wb.isCancelled = 0 and cdb.isCancelled = 0

    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot find participant . Error message : '
              + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

7.3 Procedury aktualizujące

7.3.1 Zmiana limitu miejsc na warsztacie

```
--ZMIANA LIMITU MIEJSC NA WARSZTACIE
create PROCEDURE update_workshop_participants_limit
    @WorkShopID int,
    @newLimit int
AS
BEGIN
SET NOCOUNT ON
    BEGIN TRY
    DECLARE @diff int
    set @diff = @newLimit - (SELECT Limit from Workshops where WorkShopID = @WorkShopID);

    IF dbo.funcWorkshopFreePlaces(@WorkShopID) < (-1 * @diff)
    BEGIN
        THROW 52000, 'There are too many registered participants to resize this workshop',1
    END

        UPDATE Workshops
        SET Limit = @newLimit
        where WorkShopID = @WorkShopID
    END TRY
    BEGIN CATCH
        DECLARE @errorMsg nvarchar(2048)
        = 'cannot change limit: ' + ERROR_MESSAGE();
        ;THROW 52000, @errorMsg,1
    END CATCH
END
go
```

7.3.2 Zmiana limitu dnia konferencji

```
--zmiana limitu dnia konferencji
CREATE PROCEDURE update_conferenceday_participants_limit
    @ConferenceDayID int,
    @newLimit int
AS
    BEGIN
    DECLARE @diff int
    set @diff = @newLimit - (SELECT Limit from ConferenceDay where ConferenceDayID = @ConferenceDayID);

    IF dbo.funcConferenceDayFreePlaces(@ConferenceDayID) < (-1 * @diff)
    BEGIN
        THROW 52000, 'There are too many registered participants to resize this conferenceday',1
        END

        UPDATE ConferenceDay
        SET Limit = @newLimit
```

```
        where ConferenceDayID = @ConferenceDayID
    END
go
```

7.3.3 Unieważnienie nieopłaconych zamówień

```
--unieważnianie nieopłaconych zamówień
create procedure cancel_reservations
as
BEGIN
    update Reservation
    set Reservation.isCancelled = 1
    from Reservation
        join Conferences as c
        on c.ConferenceID = Reservation.ConferenceID
    where Reservation.PaymentDate is  null
    and DATEDIFF(dd,Reservation.ReservationDate,GETDATE()) > 7
END
go
```

7.3.4 Wprowadzenie daty płatności

```
--wprowadzanie daty płatności
create procedure proc_setPaymentDate
    @ReservationID INT,
    @PaymentDate DATE
as
BEGiN
SET NOCOUNT ON
    BEGIN TRY
    if not exists (select * from Reservation where ReservationID = @ReservationID)
    BEGIN
        THROW 52000, 'Reservation does not exist ', 1
    END

    if exists (select * from Reservation where ReservationID = @ReservationID and PaymentDate is not null)
    BEGIN
        THROW 52000, 'PaymentDate already exists ', 1
    END

    if exists (select * from Reservation where ReservationID = @ReservationID and isCancelled = 1)
    BEGIN
        THROW 52000, 'Reservation is cancelled ', 1
    END

    if DATEDIFF(dd, (select ReservationDate from Reservation where ReservationID = @ReservationID ), @Paymen
tDate) < 0
    BEGIN
        THROW 52000, 'Wrong date', 1
    END

    update Reservation
    set Reservation.PaymentDate = @PaymentDate
    where Reservation.ReservationID = @ReservationID
    END TRY
    BEGIN CATCH
    DECLARE @errorMsg nvarchar (2048)
            = 'Cannot add payment date . Error message : '
                + ERROR_MESSAGE () ;
        ; THROW 52000 , @errorMsg ,1
    END CATCH
END
go
```

8. Triggery

8.1 Poprawna liczba studentów

```
--
 trigger / czy liczba studentow zapowiedzianych zgadza sie z rzeczywistą przy pelnym wypelnieniu uczestnikow
 dla zadanej rezerwacji

create trigger appropriate_students_amount
 on Participants
 after insert, update
 as
 begin

    if exists
    (
    select *
    from Inserted as p
        join PersonalData as pd
        on pd.PersonalDataID = p.PersonalDataID
        join ConferenceDayBooking as cdb
        on cdb.ConferenceDayBookingID = p.ConferenceDayBookingID
    group by pd.PersonalDataID, cdb.ConcessionaryTickets, cdb.NormalTickets
    having cdb.ConcessionaryTickets != (select count(pd_inner.PersonalDataID)
                                        from PersonalData as pd_inner
                                        where pd.PersonalDataID = pd_inner.PersonalDataID and pd_inner.Stude
ntCard is not null)
            and (cdb.ConcessionaryTickets+cdb.NormalTickets) = (select count(pd_inner.PersonalDataID)
                                        from PersonalData as pd_inner
                                        where pd.PersonalDataID = pd_inner.PersonalDataID)
    )
    BEGIN
        THROW 50001 , 'Number of students is not equal to declared number of students in booking ' , 1
        ROLLBACK TRANSACTION
    END
end
```

8.2 Mniej zapisanych na warsztat niż na konferencję

```
--mniej zapisanych na warsztat niż na konferencję

create TRIGGER  [dbo].[workshop_participants_fewer_than_day_participants_trigger]
    ON [dbo].[WorkshopBooking]
    AFTER INSERT, UPDATE
AS
    BEGIN
        IF EXISTS(
            SELECT *
            FROM inserted as i
            JOIN ConferenceDayBooking as cdb
            ON cdb.ConferenceDayBookingID=i.ConferenceDayBookingID
            WHERE (i.NormalTickets+i.ConcessionaryTickets)>(cdb.NormalTickets+cdb.ConcessionaryTickets)
        )
        BEGIN
            THROW 50001, 'All workshop participants must be day attendees', 1
        END
    END
```

8.3 Większy limit na konferencji niż na warsztatach

```
--
sprawdzenie czy w dniu konferencji jest wiekszy limit na uczestnikow niz na warsztatach (bo każdy uczestnik
warsztatu musi byc zapisany na ten dzien!!!


create trigger appropriate_limits
on Workshops
after insert, update
as
begin
    if exists
    (
        select *
        from Inserted as i
            join ConferenceDay as cd
            on cd.ConferenceDayID = i.ConferenceDayID
        where i.Limit > cd.Limit
    )
    begin
```

```
        THROW 50001 , 'Workshop cannot have greater limit of participant`s amount than conference day than ' , 1
        ROLLBACK TRANSACTION
    end
end
```

8.4 Sprawdzenie czy zapisanych na warsztat nie jest więcej niż na konferencję

```
--
sprawdzenie, czy przypadkiem liczba zarejestrowanych na wartsztat nie jest wieksza niz zarejestrowanych na dzien konferencji

create trigger workshop_participants_conf_participants
on WorkshopBooking
after insert, update
as
begin
    if exists
    (
        select *
        from Inserted as i
            join ConferenceDayBooking as cdb
            on cdb.ConferenceDayBookingID = i.ConferenceDayBookingID
        where (i.ConcessionaryTickets + i.NormalTickets) > (cdb.ConcessionaryTickets + cdb.NormalTickets)
    )
    begin
        THROW 50001 , 'There cannot be more participants on workshop than conference day' , 1
        ROLLBACK TRANSACTION
    end
end
```

8.5 Sprawdzenie, czy dni konferencji które rezerwujemy są w zarezerwowanej konferencji

```
-- sprawdzenie, czy w jednym zamowieniu na pewno są ujęte dni rezerwacji na dany dzien konferencji

create trigger conf_days_in_one_conf
on ConferenceDayBooking
after insert
as
begin
    if exists
    (
        select *
        from Inserted as i
            join Reservation as r
            on r.ReservationID = i.ReservationID
            join ConferenceDays as cd
            on cd.ConferenceDayID = i.ConferenceDayID
            join Conferences as c1
            on c1.ConferenceID = cd.ConferenceID
            join Conferences as c2
            on c2.ConferenceID = r.ConferenceID
        where c2.ConferenceID != c1.ConferenceID
    )
    begin
        THROW 50001 , 'Cannot book conference days from different conferences in one order! ' , 1
        ROLLBACK TRANSACTION
    end
end
```

8.6 Nienachodzenie czasu warsztatów

```
create trigger [dbo].[overlapping_workshop_time]
on [dbo].[WorkshopParticipants]
after insert
as
begin
    if exists
    (
        select *
        from Inserted as i
            join WorkshopBooking as wb1 on wb1.WorkshopBookingID = i.WorkshopBookingID
            join Workshops as w1 on w1.WorkshopID = wb1.WorkshopID
        where i.ParticipantID in
        (select wp.ParticipantID
```

```
14.          from WorkshopParticipants as wp
15.              join WorkshopBooking as wb on wb.WorkshopBookingID = wp.WorkshopBookingID
16.              join Workshops as w on w.WorkshopID = wb.WorkshopID
17.          where wp.ParticipantID = i.ParticipantID and  w.StartTime < w1.EndTime and w1.StartTime < w.EndTime
    AND w1.WorkshopID <> wb.WorkshopID)
18.      )
19.      begin
20.          THROW 50001 , 'Workshops cannor overlap each other' , 1
21.          ROLLBACK TRANSACTION
22.      end
23.  end
```

8.7 Sprawdzenie, czy zniżki czasowe na daną konferencję malejąc

```
1.   --opadające zniżki
2.
3.   create trigger monotonous_threshold_of_prices
4.   on Price
5.   after insert, update
6.   as
7.   begin
8.       Declare @PreviousPriceDiscount numeric(3,2)
9.       set @PreviousPriceDiscount = isNull((select top 1 Discount from Price as p
10.                                  where p.ConferenceID = (select ConferenceID from Inserted)
11.                                  and p.UntilDays > (select UntilDays from Inserted)
12.                                  order by p.UntilDays asc),1.0)
13.      Declare @NextPriceDiscount numeric(3,2)
14.      set @NextPriceDiscount = isNull((select top 1 Discount from Price as p
15.                                 where p.ConferenceID = (select ConferenceID from Inserted)
16.                                 and p.UntilDays < (select UntilDays from Inserted)
17.                                 order by p.UntilDays desc),0.0)
18.      if @PreviousPriceDiscount < (select Discount from Inserted) or @NextPriceDiscount > (select Discount fro
     m Inserted)
19.      begin
20.          THROW 50001 , 'Prices are not in correct order' ,1
21.          ROLLBACK TRANSACTION
22.      end
23.  end
```

8.8 Propagacja anulacji zamówienia

```
1.   --propagacja anulacji zamówienia
2.
3.   CREATE TRIGGER [dbo].[order_canceled_trigger]
4.       ON [dbo].[Reservation]
5.       AFTER UPDATE
6.   AS
7.       BEGIN
8.
9.           UPDATE wb
10.          SET wb.isCancelled = i.isCancelled
11.          FROM WorkshopBooking wb
12.              JOIN ConferenceDayBooking cdb ON wb.ConferenceDayBookingID=cdb.ConferenceDayBookingID
13.              JOIN ConferenceDay as cd ON cd.ConferenceDayID = cdb.ConferenceDayID
14.              JOIN Conferences as c ON c.ConferenceID = cd.ConferenceID
15.              JOIN Inserted i ON c.ConferenceID = i.ConferenceID
16.
17.          UPDATE cdb
18.          SET cdb.isCancelled = i.isCancelled
19.          FROM ConferenceDayBooking cdb
20.              JOIN ConferenceDay as cd ON cd.ConferenceDayID = cdb.ConferenceDayID
21.              JOIN Conferences as c ON c.ConferenceID = cd.ConferenceID
22.              JOIN Inserted i ON c.ConferenceID = i.ConferenceID
23.      END
```

8.9 Sprawdzenie czy wystarczyło miejsc na konferencję

```
1.   --sprawdzenie, czy wystarczylo miejsc w konf
2.   CREATE TRIGGER [dbo].[too_few_conference_day_register]
3.   ON [dbo].[ConferenceDayBooking]
4.       AFTER INSERT
5.   AS
6.       BEGIN
7.           IF EXISTS
8.           (
9.           SELECT * FROM inserted AS i
10.          WHERE dbo.funcConferenceDayFreePlaces(i.ConferenceDayID)<0
```

```
        )
    BEGIN
        THROW 50001, 'Too few free places to register this many conference day attendees.', 1
    END
END
GO
```

8.10 Sprawdzenie, czy wystarczyło miejsc na warsztat

```
--sprawdzenie, czy wystarczylo miejsc w warsztacie

CREATE TRIGGER [dbo].[too_few_workshop_places_trigger]
ON [dbo].[WorkshopBooking]
    AFTER INSERT
AS
    BEGIN
        IF EXISTS
        (
            SELECT * FROM inserted AS i
            WHERE dbo.funcWorkshopFreePlaces(i.WorkshopID) < 0
        )
        BEGIN
            THROW 50001, 'Too few free places to register this many workshop attendees.',1
        END
    END
GO
```

8.11 Sprawdzenie, czy liczba uczestników nie jest większa niż liczba zarezerwowanych miejsc

```
--Sprawdzenie limitu przypisania uczestników do rezerwacji

CREATE  TRIGGER [dbo].[attendees_not_above_reservation_trigger]
ON [dbo].[Participants]
    AFTER INSERT
AS
    BEGIN
        IF EXISTS
        (
            SELECT *
            FROM inserted AS i
            GROUP BY i.ConferenceDayBookingID
            HAVING EXISTS(
                SELECT *
                FROM ConferenceDayBooking cdb
                WHERE cdb.ConferenceDayBookingID=i.ConferenceDayBookingID
                    AND (cdb.NormalTickets + cdb.ConcessionaryTickets) <     (SELECT COUNT(*)
                                                                              FROM Participants as p
                                                                              WHERE p.ConferenceDayBookingID=c
db.ConferenceDayBookingID
                                                                              GROUP BY p.ConferenceDayBookingI
D)
            )

        )
        BEGIN
            THROW 50001, 'Number of attendees must not exceed number of reservations', 1
        END
    END
GO
```

8.12 Sprawdzenie limitu przypisania uczestników do warsztatu

```
--Sprawdzenie limitu przypisania uczestników do warsztatu

CREATE  TRIGGER [dbo].[attendees_not_above_reservation_trigger_workshops]
ON [dbo].[WorkshopParticipants]
    AFTER INSERT
AS
    BEGIN
        IF EXISTS
        (
            SELECT *
            FROM inserted AS i
            GROUP BY i.WorkshopBookingID
            HAVING EXISTS(
```

```
                SELECT *
                FROM WorkshopBooking wb
                WHERE wb.WorkshopBookingID=i.WorkshopBookingID
                    AND (wb.ConcessionaryTickets + wb.NormalTickets)  <  (SELECT COUNT(*)
                                                FROM WorkshopParticipants as wp
                                                WHERE wp.WorkshopBookingID=wb.WorkshopBookingID
                                                GROUP BY wp.WorkshopBookingID)
            )

        )
        BEGIN
            THROW 50001, 'Number of attendees must not exceed number of reservations', 1
        END
    END
GO
```

8.13 Opłacenie po anulacji

```
-- opłacenie po anulacji
CREATE TRIGGER [dbo].[check_payments_opportunity]
ON [dbo].[Reservation]
    AFTER INSERT, UPDATE
AS
    BEGIN
    IF EXISTS
    (
        SELECT * FROM inserted AS i
        WHERE (isCancelled = 1)
    )
    BEGIN
        THROW 50001, 'Payment cannot be added',1
    END
END
```

8.14 Zamówienie po dniu konferencji

```
--zamówienie po dniu konferencji
CREATE TRIGGER [dbo].[order_before_conference_day_trigger]
    ON [dbo].[Reservation]
    AFTER INSERT, UPDATE
AS
    BEGIN
        IF EXISTS(
        SELECT *
        FROM inserted AS i
            JOIN Conferences as c
            ON c.ConferenceID = i.ConferenceID
            WHERE c.EndDate<=i.ReservationDate
        )
    BEGIN
        THROW 50001, 'Order cannot be placed for conference day in the past', 1
    END
END
```

9. Indeksy

Oprócz indeksów związanych z kluczami obcymi zdecydowaliśmy się wprowadzić dodatkowe indeksy:

```
CREATE INDEX ConferenceDay_idx on ConferenceDay (Date ASC)
;

CREATE INDEX BeginDate_idx on Conferences (BeginDate ASC)
;

CREATE INDEX EndDate_idx on Conferences (EndDate ASC)
;

CREATE INDEX discount_idx on Price (Discount ASC)
;
```

```
12.
13. CREATE INDEX untildays_idx on Price (UntilDays ASC)
14. ;
15.
16. CREATE INDEX ReservationDate_idx on Reservation (ReservationDate ASC)
17. ;
18.
19. CREATE INDEX StartTime_idx on Workshops (StartTime ASC)
20. ;
21.
22. CREATE INDEX EndTime_idx on Workshops (EndTime ASC)
23. ;
```

10. Role w systemie

10.1 Administrator

Osoba znająca język SQL do obsługiwania sytuacji losowych, posiada możliwość rozbudowy i polepszenia bazy danych. Dostęp do wszystkich procedur składowanych i widoków.

10.2 Pracownik firmy

Osoba odpowiedzialna za obsługiwanie zamówień, która będzie się kontaktowała z klientem i pomagała mu w przypadku problemów z rejestracją.

Pracownik ma dostęp do wszystkich procedur funkcji oraz widoków.

10.3 Klient

Osoba składająca zamówienia i dokonująca rezerwacji.

10.3.1 Procedury

Niektóre procedury są dostępne tylko dla Klienta Indywidualnego, a niektóre tylko dla Klienta Firmowego.

Klient Indywidualny:

- procAddWorkshopBookingIndividual
- procAddReservationIndividual
- procAddConferenceDayBookingIndividual
- procAddConferenceIndividalParticipant
- procAddWorkshopIndividualParticipant
- enrollments

KlientFirmowy:

- procAddWorkshopBookingCompany

- procAddReservationCompany
- procAddEmployee
- procAddConferenceDayBookingCompany
- procAddConferenceComanyParticipant
- procAddWorkshopCompanyParticipant

Procedury wspólne:

- proc_showConferenceDaysPArticipants
- proc_ShowWorkshopParticipants
- proc_Events
- workshop_participants
- conference_participants
- conferenceday_participants
- participant_conferences
- comming_conferences

10.3.2 Funkcje

- funcConferenceDays
- funcConferenceDayFreePlaces
- funcWorkshopFreePlaces
- funcReservationDiscount
- funcTotalCostOfConfday
- funcTotalCostOfWorkshops
- funcTotalReservationCost

10.3.3 Widoki

- conf_day_free_reserved_seats_view
- workshop_free_reserved_seats_view
- conferences_list_view
- upcoming_conferences_view

10.4 Uczestnik

Uczestnik konferencji to osoba biorąca udział w konferencji lub w warsztacie.

10.4.1 Procedury

- participant_conferences
- enrollments