

# Dokumentacja projektu bazy danych

---

*Krzysztof Bleharczyk, Przemysław Jabłecki*

## Zawartość

1.	Uproszczona specyfikacja wymagań do projektu.....	4
1.1	Opis funkcji z punktu widzenia użytkowników.....	4
1.1.1	Lista aktorów i ich funkcje.....	4
1.1.2	Diagramy Use Case .....	5
2.	Diagram ER bazy danych .....	6
3.	Opis tabel.....	7
3.1	Company.....	7
3.2	Individual Clients .....	7
3.3	Client.....	7
3.4	PersonalData .....	8
3.5	Employees .....	8
3.6	Participants.....	8
3.7	WorkshopParticipants .....	9
3.8	Conferences.....	9
3.9	ConferenceDay .....	9
3.10	Workshops.....	9
3.11	ConferenceDayBooking .....	10
3.12	WorkshopBooking .....	10
3.13	Reservation.....	11
3.14	Price .....	11
4.	Deklaracje kluczy obcych.....	11
5.	Widoki.....	13
5.1.	Płatności klientów .....	13
5.2.	Przychody firmy .....	13
5.3	Nieopłacone rezerwacje które jeszcze nie zostały anulowane .....	13
5.4	Lista firm które nie wprowadziły żadnych danych personalnych swoich pracowników .....	13
5.5	Rezerwacje które muszą zostać opłacone do jutra, bo zostaną anulowane .....	14
5.6	Lista konferencji z liczbą wolnych i zarezerwowanych miejsc na dany dzień .....	14
5.7	Lista warsztatów, wraz z wolnymi i zarezerwowanymi miejscami.....	14

5.8	Lista konferencji wraz z datami rozpoczęcia i zakończenia .....	14
5.9	Nadchodzące konferencje (wszystkie) .....	14
5.10	Popularność warsztatów .....	14
5.11	Popularność dni konferencji.....	15
6.	Funkcje .....	15
6.1	Dni konferencji dla danej konferencji.....	15
6.2	Ilość wolnych miejsc dla danej konferencji .....	15
6.3	Ilość wolnych miejsc na warsztacie .....	16
6.4	Uczestnicy danej konferencji.....	16
6.5	Uczestnicy warsztatu .....	16
6.6	Aktualna obniżka dotycząca danej rezerwacji.....	17
6.7	Koszt dnia konferencji dla zamówienia .....	17
6.8	Koszt warsztatów dla zamówienia .....	18
6.9	Całkowity koszt rezerwacji .....	18
7.	Procedury .....	18
7.1.	Procedury wstawiające.....	18
7.1.1	Dodawanie ceny .....	18
7.1.2	Dodaj konferencję .....	19
7.1.3	Tworzenie nowego warsztatu .....	20
7.1.4	Dodanie rezerwacji na warsztat przez klienta firmowego .....	20
7.1.5	Dodanie rezerwacji na warsztat przez klienta indywidualnego .....	21
7.1.6	Utworzenie rezerwacji przez klienta indywidualnego.....	22
7.1.7	Utworzenie rezerwacji przez klienta firmowego.....	23
7.1.8	Dodanie danych personalnych .....	23
7.1.9	Dodawanie klienta firmowego .....	24
7.1.10	Dodawanie klienta indywidualnego .....	25
7.1.11	Dodawanie pracownika .....	26
7.1.12	Dodawanie rezerwacji dnia konferencji przez klienta indywidualnego .....	26
7.1.13	Dodawanie rezerwacji dnia konferencji przez klienta firmowego .....	27
7.1.14	Dodawanie klienta indywidualnego jako członka konferencji .....	28
7.1.15.	Dodaj pracownika firmy jako uczestnika konferencji.....	28
7.1.16.	Dodaj firmowego uczestnika warsztatów .....	29
7.1.17	Dodaj klienta indywidualnego jako uczestnika warsztatów.....	30
7.2	Procedury statystyczne .....	30

7.2.1 Procedura pokazująca ilość rezerwacji dla danego klienta (uporządkowane malejąco) .....	30
7.2.2 Lista uczestników dni konferencji dla wybranej konferencji.....	31
7.2.3 Lista uczestników dla poszczególnych warsztatów w obrębie konferencji.....	31
7.2.4 Lista wydarzeń które odbywają się na danej konferencji .....	32
7.2.5 Lista uczestników danego warsztatu .....	32
7.2.6 Lista uczestników danej konferencji.....	32
7.2.7 Lista uczestników danego dnia konferencji.....	33
7.2.8 Lista konferencji danego uczestnika.....	33
7.2.9 Wpływy z rezerwacji dla danej konferencji .....	34
7.2.10 Lista konferencji które odbędą się w przeciągu podanej liczby dni .....	34
7.2.11 Lista wydarzeń na które podany uczestnik został zapisany .....	34
7.3 Procedury aktualizujące .....	35
7.3.1 Zmiana limitu miejsc na warsztacie .....	35
7.3.2 Zmiana limitu dnia konferencji.....	35
7.3.3 Unieważnienie nieopłaconych zamówień .....	36
7.3.4 Wprowadzenie daty płatności.....	36
8. Triggery.....	36
8.1 Poprawna liczba studentów .....	37
8.2 Mniej zapisanych na warsztat niż na konferencję.....	37
8.3 Większy limit na konferencji niż na warsztatach.....	37
8.4 Sprawdzenie czy zapisanych na warsztat nie jest więcej niż na konferencję.....	38
8.5 Sprawdzenie, czy dni konferencji które rezerwujemy są w zarezerwowanej konferencji .....	38
8.6 Nienachodzenie czasu warsztatów.....	38
8.7 Sprawdzenie, czy zniżki czasowe na daną konferencję maleją .....	39
8.8 Propagacja anulacji zamówienia .....	39
8.9 Sprawdzenie czy wystarczyło miejsc na konferencję .....	39
8.10 Sprawdzenie, czy wystarczyło miejsc na warsztat.....	40
8.11 Sprawdzenie, czy liczba uczestników nie jest większa niż liczba zarezerwowanych miejsc .	40
8.12 Sprawdzenie limitu przypisania uczestników do warsztatu .....	40
8.13 Opłacenie po anulacji .....	41
8.14 Zamówienie po dniu konferencji.....	41
9. Indeksy.....	41
10. Role w systemie.....	42
10.1 Administrator .....	42

10.2 Pracownik firmy.....	42
10.3 Klient.....	42
10.3.1 Procedury .....	42
10.3.2 Funkcje .....	43
10.3.3 Widoki.....	43
10.4 Uczestnik .....	43
10.4.1 Procedury .....	43

## 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, najczęstszy 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.

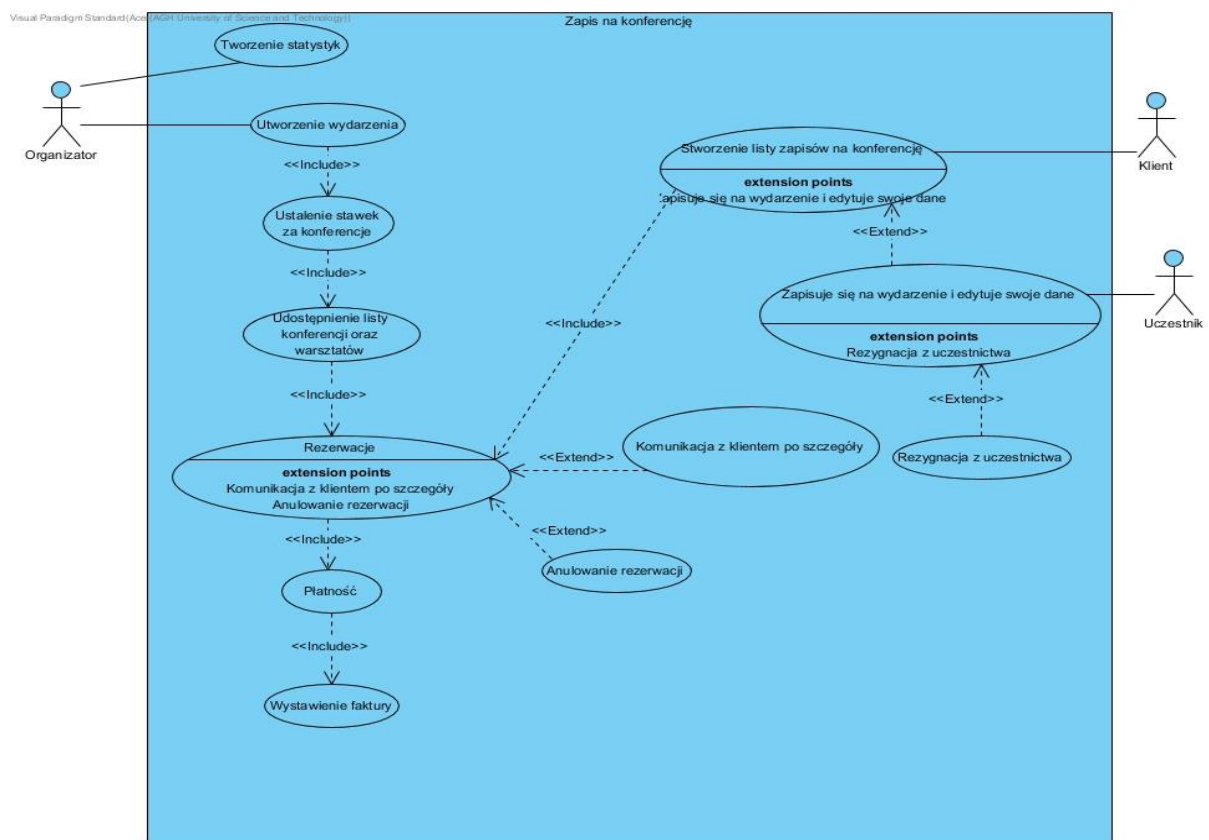
### **Uczestnik:**

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

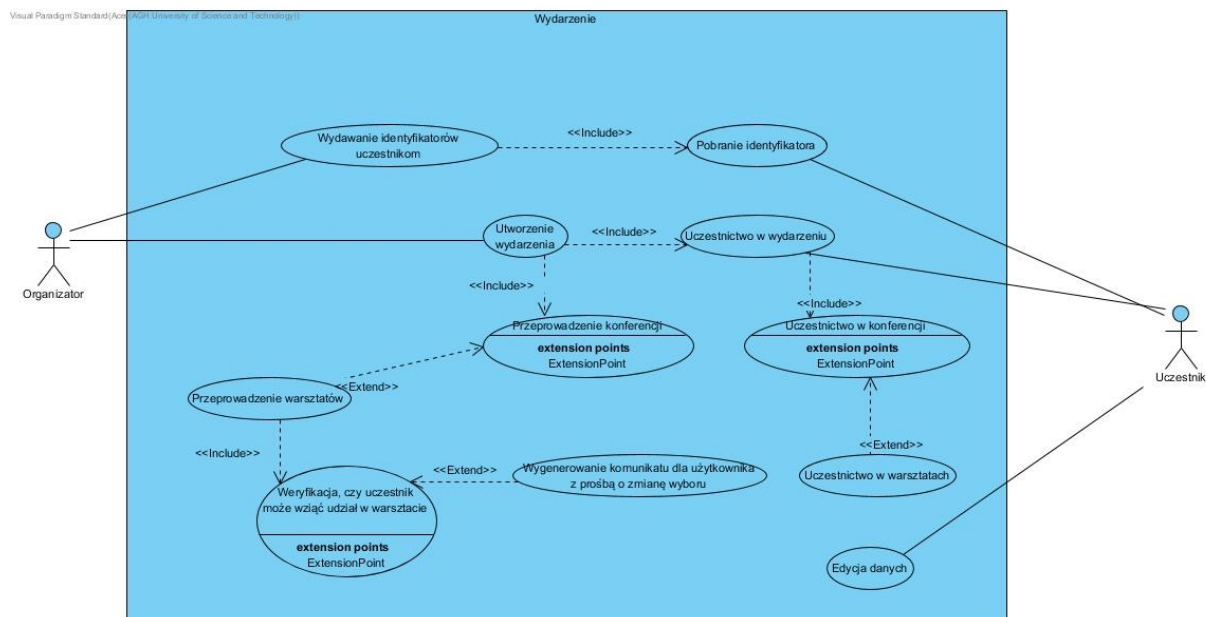
### **Klient indywidualny**

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

## **1.1.2 Diagramy Use Case**

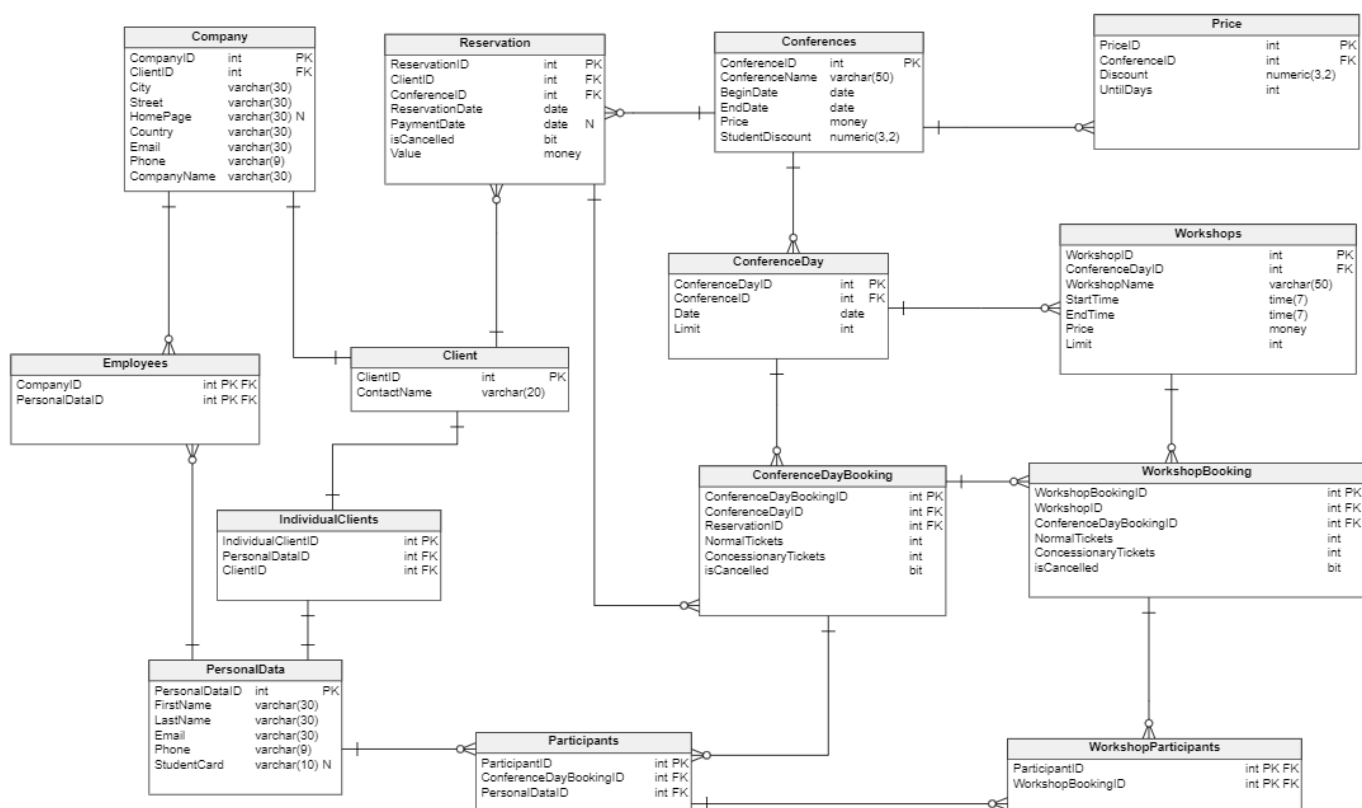


Rysunek 1 Diagram Use Case "Zapis na konferencję"



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 wchodzi 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 konferencjach 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 rozpisę 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 normalnych na rezerwowany dzień, ConcessionaryTickets – ilość studentów zapisywanych na dany dzień oraz isCancelled – 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)

```

```
10. );
```

### 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.

```
1. -- Table: Reservation
2. CREATE TABLE Reservation (
3.     ReservationID int NOT NULL IDENTITY,
4.     ClientID int NOT NULL,
5.     ConferenceID int NOT NULL,
6.     ReservationDate date NOT NULL,
7.     PaymentDate date NULL,
8.     isCancelled bit NOT NULL DEFAULT 0,
9.     Value money NOT NULL,
10.    CONSTRAINT Reservation_pk PRIMARY KEY (ReservationID)
11. );
```

### 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.

```
1. -- Table: Price
2. CREATE TABLE Price (
3.     PriceID int NOT NULL IDENTITY,
4.     ConferenceID int NOT NULL,
5.     Discount numeric(3,2) NOT NULL DEFAULT 0.0 CHECK (Discount BETWEEN 0 AND 1.0),
6.     UntilDays int NOT NULL CHECK (UntilDays >= 0),
7.     CONSTRAINT Price_pk PRIMARY KEY (PriceID)
8. );
```

## 4. Deklaracje kluczy obcych

```
1. -- foreign keys
2. -- Reference: Company_Client (table: Company)
3. ALTER TABLE Company ADD CONSTRAINT Company_Client
4.     FOREIGN KEY (ClientID)
5.     REFERENCES Client (ClientID);
6.
7. -- Reference: ConferenceDayBooking_ConferenceDay (table: ConferenceDayBooking)
8. ALTER TABLE ConferenceDayBooking ADD CONSTRAINT ConferenceDayBooking_ConferenceDay
9.     FOREIGN KEY (ConferenceDayID)
10.    REFERENCES ConferenceDay (ConferenceDayID);
11.
12. -- Reference: ConferenceDayBooking_Reservation (table: ConferenceDayBooking)
13. ALTER TABLE ConferenceDayBooking ADD CONSTRAINT ConferenceDayBooking_Reservation
14.     FOREIGN KEY (ReservationID)
15.     REFERENCES Reservation (ReservationID);
```

```

16.
17. -- Reference: ConferenceDay_Conferences (table: ConferenceDay)
18. ALTER TABLE ConferenceDay ADD CONSTRAINT ConferenceDay_Conferences
19.     FOREIGN KEY (ConferenceID)
20.     REFERENCES Conferences (ConferenceID);
21.
22. -- Reference: Employees_Company (table: Employees)
23. ALTER TABLE Employees ADD CONSTRAINT Employees_Company
24.     FOREIGN KEY (CompanyID)
25.     REFERENCES Company (CompanyID);
26.
27. -- Reference: Employees_PersonalData (table: Employees)
28. ALTER TABLE Employees ADD CONSTRAINT Employees_PersonalData
29.     FOREIGN KEY (PersonalDataID)
30.     REFERENCES PersonalData (PersonalDataID);
31.
32. -- Reference: IndividualClients_Client (table: IndividualClients)
33. ALTER TABLE IndividualClients ADD CONSTRAINT IndividualClients_Client
34.     FOREIGN KEY (ClientID)
35.     REFERENCES Client (ClientID);
36.
37. -- Reference: IndividualClients_PersonalData (table: IndividualClients)
38. ALTER TABLE IndividualClients ADD CONSTRAINT IndividualClients_PersonalData
39.     FOREIGN KEY (PersonalDataID)
40.     REFERENCES PersonalData (PersonalDataID);
41.
42. -- Reference: Participants_ConferenceDayBooking (table: Participants)
43. ALTER TABLE Participants ADD CONSTRAINT Participants_ConferenceDayBooking
44.     FOREIGN KEY (ConferenceDayBookingID)
45.     REFERENCES ConferenceDayBooking (ConferenceDayBookingID);
46.
47. -- Reference: Participants_PersonalData (table: Participants)
48. ALTER TABLE Participants ADD CONSTRAINT Participants_PersonalData
49.     FOREIGN KEY (PersonalDataID)
50.     REFERENCES PersonalData (PersonalDataID);
51.
52. -- Reference: Price_Conferences (table: Price)
53. ALTER TABLE Price ADD CONSTRAINT Price_Conferences
54.     FOREIGN KEY (ConferenceID)
55.     REFERENCES Conferences (ConferenceID);
56.
57. -- Reference: Reservation_Client (table: Reservation)
58. ALTER TABLE Reservation ADD CONSTRAINT Reservation_Client
59.     FOREIGN KEY (ClientID)
60.     REFERENCES Client (ClientID);
61.
62. -- Reference: Reservation_Conferences (table: Reservation)
63. ALTER TABLE Reservation ADD CONSTRAINT Reservation_Conferences
64.     FOREIGN KEY (ConferenceID)
65.     REFERENCES Conferences (ConferenceID);
66.
67. -- Reference: WorkshopBooking_ConferenceDayBooking (table: WorkshopBooking)
68. ALTER TABLE WorkshopBooking ADD CONSTRAINT WorkshopBooking_ConferenceDayBooking
69.     FOREIGN KEY (ConferenceDayBookingID)
70.     REFERENCES ConferenceDayBooking (ConferenceDayBookingID);
71.
72. -- Reference: WorkshopBooking_Workshops (table: WorkshopBooking)
73. ALTER TABLE WorkshopBooking ADD CONSTRAINT WorkshopBooking_Workshops
74.     FOREIGN KEY (WorkshopID)
75.     REFERENCES Workshops (WorkshopID);
76.
77.
78. -- Reference: WorkshopParticipants_Participants (table: WorkshopParticipants)
79. ALTER TABLE WorkshopParticipants ADD CONSTRAINT WorkshopParticipants_Participants
80.     FOREIGN KEY (ParticipantID)
81.     REFERENCES Participants (ParticipantID);
82.
83. -- Reference: WorkshopParticipants_WorkshopBooking (table: WorkshopParticipants)
84. ALTER TABLE WorkshopParticipants ADD CONSTRAINT WorkshopParticipants_WorkshopBooking
85.     FOREIGN KEY (WorkshopBookingID)
86.     REFERENCES WorkshopBooking (WorkshopBookingID);
87.
88. -- Reference: Workshops_ConferenceDay (table: Workshops)
89. ALTER TABLE Workshops ADD CONSTRAINT Workshops_ConferenceDay
90.     FOREIGN KEY (ConferenceDayID)
91.     REFERENCES ConferenceDay (ConferenceDayID);

```

## 5. Widoki

### 5.1. Płatności klientów

```
1. --płatności klientów
2. create view clients_payments_view
3. as
4.     select NameSurname as ClientName, isCancelled, Value
5.     from ( select PersonalDataID, FirstName + ' ' + LastName as NameSurname from PersonalData) as pd
6.         join IndividualClients as ic
7.         on pd.PersonalDataID = ic.PersonalDataID
8.         join Client as c
9.         on c.ClientID = ic.ClientID
10.        join Reservation as r
11.        on r.ClientID = c.ClientID
12.    group by NameSurname, isCancelled, Value
13.
14.    union
15.
16.    select CompanyName as ClientName, isCancelled, Value
17.    from Company
18.        join Client as c
19.        on c.ClientID = Company.ClientID
20.        join Reservation as r
21.        on r.ClientID = c.ClientID
22.    group by CompanyName, isCancelled, Value
23. go
```

### 5.2. Przychody firmy

```
1. --przychody
2. create view income
3. AS
4.     SELECT year(r.PaymentDate) as year, month(r.PaymentDate) as month, sum(r.value) as income
5.     FROM Reservation as r
6.     WHERE r.Paymentdate is not null and r.isCancelled = 0
7.     GROUP BY year(r.PaymentDate), month(r.PaymentDate)
8.     WITH ROLLUP
9. go
10. --ID klientów z ilością ich rezerwacji
11. create view client_popularity
12. as
13.     SELECT c.ClientID, count(*) as liczba
14.     from Client as C
15.     join Reservation as r
16.     on r.clientID=c.clientID and r.isCancelled = 0
17.     group by c.clientID
18. go
```

### 5.3 Nieopłacone rezerwacje które jeszcze nie zostały anulowane

```
1. --nieopłacone rezerwacje (jeszcze nie anulowane)
2. create view unpaid_reservations
3. as
4.     select reservationID, clientID
5.     from reservation
6.     where paymentdate is null and isCancelled = 0
7. go
```

### 5.4 Lista firm które nie wprowadziły żadnych danych personalnych swoich pracowników

```
1. --lista firm, które nie wprowadziły żadnych danych swoich pracowników
2. create view no_participants_data_view
3. as
4.     select r.clientID, r.reservationID, r.conferenceID
5.     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.
6.      Limit) as free , cd.Limit
7.      from ConferenceDay as cd

```

## 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 nadchodzące 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

```

```
20. go
```

### 6.3 Ilość wolnych miejsc na warsztacie

```
1. --zwraca ilość wolnych miejsc na warsztacie
2. CREATE FUNCTION funcWorkshopFreePlaces
3. (
4.     @WorkshopID INT
5. )
6. RETURNS INT
7. AS
8. BEGIN
9.     RETURN
10.    (
11.        SELECT w.Limit - (sum(wb.ConcessionaryTickets + wb.NormalTickets))
12.        FROM Workshops as w
13.        LEFT JOIN WorkshopBooking as wb
14.        ON w.WorkshopID=wb.WorkshopID AND wb.isCancelled=0
15.        WHERE w.WorkshopID=@WorkshopID
16.        GROUP BY w.WorkshopID, w.Limit
17.    )
18. END
19. go
```

### 6.4 Uczestnicy danej konferencji

```
1. --zwraca uczestników konferencji
2. CREATE FUNCTION funcConferenceDayParticipants
3. (
4.     @ConferenceID INT
5. )
6. RETURNS @ParticipantsInfo TABLE
7. (
8.     tConferenceID INT,
9.     tConferenceDayID INT,
10.    tParticipantID INT,
11.    tFirstName VARCHAR(30),
12.    tLastName VARCHAR(30),
13.    tEmail VARCHAR(30),
14.    tPhone VARCHAR(9),
15.    tStudentCart VARCHAR(10)
16. )
17. AS
18. BEGIN
19.     INSERT INTO @ParticipantsInfo
20.     SELECT c.ConferenceID, cd.ConferenceDayID, p.ParticipantID, pd.FirstName, pd.LastName, pd.Email, pd.
Phone, pd.StudentCard
21.     FROM Conferences AS c
22.     JOIN ConferenceDay AS cd
23.     ON c.ConferenceID=cd.ConferenceID
24.     JOIN ConferenceDayBooking AS cdb
25.     ON cdb.ConferenceDayID=cd.ConferenceDayID
26.     JOIN Participants AS p
27.     ON p.ConferenceDayBookingID=cdb.ConferenceDayBookingID
28.     JOIN PersonalData AS pd
29.     ON pd.PersonalDataID=p.PersonalDataID
30.     WHERE c.ConferenceID=@ConferenceID AND cdb.isCancelled = 0
31.     RETURN
32. END
33. go
```

### 6.5 Uczestnicy warsztatu

```
1. --uczestnicy warsztatu
2. CREATE FUNCTION funcWorkshopParticipants
3. (
4.     @WorkshopID INT
5. )
6. RETURNS @ParticipantsInfo TABLE
7. (
8.     tWorkshopID INT,
9.     tConferenceDayID INT,
10.    tParticipantID INT,
11.    tFirstName VARCHAR(30),
12.    tLastName VARCHAR(30),
```



```

13.         tEmail VARCHAR(30),
14.         tPhone VARCHAR(9),
15.         tStudentCard VARCHAR(10)
16.     )
17. AS
18. BEGIN
19.     INSERT INTO @ParticipantsInfo
20.     SELECT w.WorkshopID, cd.ConferenceDayID, p.ParticipantID, pd.FirstName, pd.LastName, pd.Email, pd.Phone, pd.StudentCard
21.     FROM Workshops AS w
22.     JOIN ConferenceDay AS cd
23.     ON w.ConferenceDayID=cd.ConferenceDayID
24.     JOIN ConferenceDayBooking AS cdb
25.     ON cdb.ConferenceDayID=cd.ConferenceDayID
26.     JOIN Participants AS p
27.     ON p.ConferenceDayBookingID=cdb.ConferenceDayBookingID
28.     JOIN PersonalData AS pd
29.     ON pd.PersonalDataID=p.PersonalDataID
30.     WHERE w.WorkshopID=@WorkshopID AND cdb.isCancelled = 0
31.     RETURN
32. END
33. go

```

## 6.6 Aktualna obniżka dotycząca danej rezerwacji

```

1. CREATE FUNCTION funcReservationDiscount
2. (
3.     @ReservationID INT
4. )
5. RETURNS NUMERIC(3,2)
6. AS
7. BEGIN
8.     RETURN
9.     (
10.        SELECT isNull(( select top 1 Discount
11.                        from Price where UntilDays > DATEDIFF(DAY,r.ReservationDate,c.BeginDate) and PriceID
12.                        = p.PriceID
13.                        order by UntilDays asc),0.0)
14.        FROM Reservation AS r
15.        JOIN Conferences AS c
16.        ON r.ConferenceID=c.ConferenceID
17.        JOIN Price AS p
18.        ON p.ConferenceID=c.ConferenceID
19.        WHERE r.ReservationID=@ReservationID
20.    )
21. end
22. go

```

## 6.7 Koszt dnia konferencji dla zamówienia

```

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

# 7. Procedury

## 7.1. Procedury wstawiające

### 7.1.1 Dodawanie ceny

```
1.  --dodawanie ceny
2.  CREATE PROCEDURE [dbo].[procAddPrice]
3.  (
4.      @ConferenceID int,
5.      @Discount numeric(3,2),
6.      @UntilDays int
7.  )
8.  AS
9.  BEGIN
10.     SET NOCOUNT ON
11.     BEGIN TRY
12.         IF NOT EXISTS
13.         (
14.             SELECT * FROM Conferences
15.             where ConferenceID = @ConferenceID
16.         )
17.         BEGIN
```

```

16.         ;THROW 52000, 'Conference does not exist. ' ,1
17.     END
18.     INSERT INTO Price
19.     (
20.     ConferenceID,
21.     Discount,
22.     UntilDays
23.     )
24.     VALUES
25.     (
26.     @ConferenceID,
27.     @Discount,
28.     @UntilDays
29.     )
30. END TRY
31. BEGIN CATCH
32.     DECLARE @errorMsg nvarchar(2048)
33.     = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
34.     ;THROW 52000, @errorMsg,1
35. END CATCH
36. END

```

## 7.1.2 Dodaj konferencję

```

1.  --dodaj konferencję
2.  create procedure [dbo].[procAddConference]
3.      @Conf_name varchar(50),
4.      @Begin_date date,
5.      @End_date date,
6.      @Price money,
7.      @Student_disc numeric(3,2),
8.      @Limit int,
9.      @Disc numeric(3,2),
10.     @Until int
11. AS
12. BEGIN
13.     set nocount on
14.     begin transaction;
15.     BEGIN TRY
16.         IF (@Begin_date > @End_date or @Limit < 0 or @Disc <0 or @Disc >1 or @Student_disc < 0 or @Student_d
isc > 1)
17.             BEGIN
18.                 ; throw 52000, 'Wrong data', 1
19.             END
20.
21.             insert into Conferences (ConferenceName,BeginDate,EndDate,Price,StudentDiscount)
22.             values (@Conf_name,@Begin_date,@End_date,@Price,@Student_disc)
23.         IF @@ERROR <> 0
24.             begin
25.                 RAISERROR('Error, transaction not completed!',16,-1)
26.                 rollback transaction;
27.             end
28.
29.             declare @confID INT
30.             set @confID = @@IDENTITY
31.             declare @duration int
32.             declare @iterator int
33.             set @duration = DATEDIFF(dd, @Begin_date, @End_date)
34.             set @iterator = 0
35.
36.             while @iterator <= @duration
37.             begin
38.                 insert into ConferenceDay (ConferenceID, Date, Limit) values (@confID, cast (DATEADD(dd, @iterator, @
Begin_date) as date),@Limit)
39.
40.                 IF @@ERROR <> 0
41.                     begin
42.                         rollback transaction;
43.                         RAISERROR('Error, transaction not completed!',16,-1)
44.                     end
45.                 if @iterator = 0
46.                     begin
47.                         exec procAddPrice @ConferenceID=@confID, @Discount = @Disc, @UntilDays = @Until;
48.                         IF @@ERROR <> 0
49.                             begin
50.                                 RAISERROR('Error, transaction not completed!',16,-1)
51.                                 rollback transaction;
52.                             end
53.                         end
54.                         set @iterator = @iterator + 1
55.                     end

```

```

56.     commit transaction;
57.     END TRY
58.     BEGIN CATCH
59.         DECLARE @errorMsg nvarchar (2048)
60.             = 'Cannot add conference . Error message : '
61.               + ERROR_MESSAGE () ;
62.         ; THROW 52000 , @errorMsg ,1
63.         rollback transaction;
64.     END CATCH
65. END
66. GO

```

### 7.1.3 Tworzenie nowego warsztatu

```

1.  --tworzenie nowego warsztatu
2.  create procedure [dbo].[procAddWorkshop]
3.      @ConfDayID INT,
4.      @WorkshopName VARCHAR(50),
5.      @StartTime TIME(7),
6.      @EndTime TIME(7),
7.      @Price money,
8.      @WorkshopLimit INT
9.  AS
10. BEGIN
11.     set nocount on
12.     BEGIN TRY
13.         IF NOT EXISTS
14.             ( select * from ConferenceDay where ConferenceDayID = @ConfDayID )
15.         BEGIN
16.             THROW 52000, 'Conference day does not exist', 1
17.         END
18.         IF EXISTS
19.             (select * from Workshops
20.              where WorkshopName = @WorkshopName and StartTime = @StartTime and EndTime = @EndTime)
21.         BEGIN
22.             THROW 52000, 'Such workshop already exists', 1
23.         END
24.         insert into Workshops (ConferenceDayID, WorkshopName, StartTime, EndTime, Price, Limit)
25.         values (@ConfDayID, @WorkshopName, @StartTime, @EndTime, @Price, @WorkshopLimit)
26.     END TRY
27.     BEGIN CATCH
28.         DECLARE @errorMsg nvarchar (2048)
29.             = 'Cannot add workshop . Error message : '
30.               + ERROR_MESSAGE () ;
31.         ; THROW 52000 , @errorMsg ,1
32.     END CATCH
33. END
34. GO

```

### 7.1.4 Dodanie rezerwacji na warsztat przez klienta firmowego

```

1.  --Dodanie rezerwacji przez klienta firmowego
2.  create PROCEDURE [dbo].[procAddWorkshopBookingCompany]
3.      @WorkshopID int,
4.      @ConferenceDayBookingID int,
5.      @StudentsNo int,
6.      @NormalNo int
7.  AS
8.  BEGIN
9.      SET NOCOUNT ON
10.     BEGIN TRY
11.         begin transaction;
12.         IF NOT EXISTS
13.             (
14.                 SELECT * FROM Workshops
15.                 WHERE WorkShopID = @WorkShopID
16.             )
17.         BEGIN
18.             ;THROW 52000, 'Workshop does not exist. ' ,1
19.         END
20.         IF NOT EXISTS
21.             (
22.                 SELECT * FROM ConferenceDayBooking
23.                 WHERE ConferenceDayBookingID = @ConferenceDayBookingID
24.             )

```

```

25. BEGIN
26.     ;THROW 52000, 'ConferenceDayBooking does not exist. ' ,1
27. END
28. IF (@StudentsNo + @NormalNo = 0) or @StudentsNo < 0 or @NormalNo < 0
29. BEGIN
30.     ;THROW 52000, 'Not acceptable tickets values ' ,1
31. END
32.
33. if (select ConferenceDayID from ConferenceDayBooking where @ConferenceDayBookingID = ConferenceDayBookin
gID) <> (select ConferenceDayID from Workshops where @WorkshopID = WorkshopID)
34. BEGIN
35.     ;THROW 52000, 'Workshop does not belong to that conference day' ,1
36. END
37.
38. INSERT INTO WorkshopBooking
39. (
40.     WorkShopID,
41.     ConferenceDayBookingID,
42.     NormalTickets,
43.     ConcessionaryTickets,
44.     isCancelled
45. )
46. VALUES
47. (
48.     @WorkshopID,
49.     @ConferenceDayBookingID,
50.     @StudentsNo,
51.     @NormalNo,
52.     0
53. )
54. commit transaction;
55. END TRY
56. BEGIN CATCH
57.     rollback transaction;
58.     DECLARE @errorMsg nvarchar(2048)
59.     = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
60.     ;THROW 52000, @errorMsg,1
61. END CATCH
62. END
63. GO

```

### 7.1.5 Dodanie rezerwacji na warsztat przez klienta indywidualnego

```

1. -- Dodanie rezerwacji na warsztat przez klienta indywidualnego
2. CREATE PROCEDURE [dbo].[procAddWorkshopBookingIndividual]
3.     @WorkshopID int,
4.     @ConferenceDayBookingID int,
5.     @StudentCard varchar(10) = null
6. AS
7. BEGIN
8.     SET NOCOUNT ON
9.     BEGIN TRY
10.         begin transaction;
11.         IF NOT EXISTS
12.         (
13.             SELECT * FROM Workshops
14.             WHERE WorkShopID = @WorkShopID
15.         )
16.         BEGIN
17.             ;THROW 52000, 'Workshop does not exist. ' ,1
18.         END
19.         IF NOT EXISTS
20.         (
21.             SELECT * FROM ConferenceDayBooking
22.             WHERE ConferenceDayBookingID = @ConferenceDayBookingID
23.         )
24.         BEGIN
25.             ;THROW 52000, 'ConferenceDayBooking does not exist. ' ,1
26.         END
27.         if @StudentCard is not null and not exists (select *
28.         from Reservation as r
29.         left join Client as c on c.ClientID = r.ClientID
30.         left join IndividualClients as ic on ic.ClientID = c.ClientID
31.         left join PersonalData as pd on pd.PersonalDataID = ic.PersonalDataID
32.         where @StudentCard = pd.StudentCard)
33.         BEGIN
34.             THROW 52000, 'Student Card does not exists', 1
35.         END
36.         if (select ConferenceDayID from ConferenceDayBooking where @ConferenceDayBookingID = ConferenceDayBookin
gID) <> (select ConferenceDayID from Workshops where @WorkshopID = WorkshopID)
37.         BEGIN

```

```

38.         ;THROW 52000, 'Workshop does not belong to that conference day' ,1
39.     END
40.     if @StudentCard is not null
41.     BEGIN
42.         insert into WorkshopBooking (WorkshopID,ConferenceDayBookingID, NormalTickets, ConcessionaryTickets,
isCancelled)
43.         values ( @WorkshopID, @ConferenceDayBookingID, 0, 1, 0)
44.     END
45.     if @StudentCard is null
46.     BEGIN
47.         insert into WorkshopBooking (WorkshopID,ConferenceDayBookingID, NormalTickets, ConcessionaryTickets,
isCancelled)
48.         values ( @WorkshopID, @ConferenceDayBookingID, 1, 0, 0)
49.     END
50.     commit transaction;
51. END TRY
52. BEGIN CATCH
53.     rollback transaction;
54.     DECLARE @errorMsg nvarchar(2048)
55.     = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
56.     ;THROW 52000, @errorMsg,1
57. END CATCH
58. END
59. GO

```

### 7.1.6 Utworzenie rezerwacji przez klienta indywidualnego

```

1.  --Utworzenie rezerwacji przez klienta indywidualnego
2.  CREATE PROCEDURE [dbo].[procAddReservationIndividual]
3.      @ClientID int,
4.      @ConferenceID int,
5.      @ReservationDate date,
6.      @PaymentDate date,
7.      @isCancelled bit = 0,
8.      @Value money
9.  AS
10. BEGIN
11.     SET NOCOUNT ON
12.     BEGIN TRY
13.         IF NOT EXISTS
14.         (
15.             SELECT * FROM Client
16.             WHERE ClientID = @ClientID
17.         )
18.         BEGIN
19.             ;THROW 52000, 'Client does not exist. ' ,1
20.         END
21.         IF NOT EXISTS
22.         (
23.             SELECT * FROM IndividualClients as ic
24.             WHERE ic.ClientID = @ClientID
25.         )
26.         BEGIN
27.             ;THROW 52000, 'Client is not individual person. ' ,1
28.         END
29.         IF NOT EXISTS
30.         (
31.             SELECT * FROM Conferences
32.             WHERE ConferenceID = @ConferenceID
33.         )
34.         BEGIN
35.             ;THROW 52000, 'Conference does not exist. ' ,1
36.         END
37.         INSERT INTO Reservation
38.         (
39.             ClientID,
40.             ConferenceID,
41.             ReservationDate,
42.             PaymentDate,
43.             isCancelled,
44.             Value
45.         )
46.         VALUES
47.         (
48.             @ClientID,
49.             @ConferenceID,
50.             @ReservationDate,
51.             @PaymentDate,
52.             @isCancelled,
53.             @Value
54.         )

```

```

55.     END TRY
56.     BEGIN CATCH
57.         DECLARE @errorMsg nvarchar(2048)
58.         = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
59.         ;THROW 52000, @errorMsg,1
60.     END CATCH
61. END
62. GO

```

### 7.1.7 Utworzenie rezerwacji przez klienta firmowego

```

1.  --Utworzenie rezerwacji przez klienta firmowego
2.  CREATE PROCEDURE [dbo].[procAddReservationCompany]
3.      @ClientID int,
4.      @ConferenceID int,
5.      @ReservationDate date,
6.      @PaymentDate date,
7.      @isCancelled bit = 0,
8.      @Value money
9.  AS
10. BEGIN
11.     SET NOCOUNT ON
12.     BEGIN TRY
13.         IF NOT EXISTS
14.         (
15.             SELECT * FROM Client
16.             WHERE ClientID = @ClientID
17.         )
18.         BEGIN
19.             ;THROW 52000, 'Client does not exist. ',1
20.         END
21.         IF NOT EXISTS
22.         (
23.             SELECT * FROM Company as c
24.             WHERE c.ClientID = @ClientID
25.         )
26.         BEGIN
27.             ;THROW 52000, 'Client is not company. ',1
28.         END
29.         IF NOT EXISTS
30.         (
31.             SELECT * FROM Conferences
32.             WHERE ConferenceID = @ConferenceID
33.         )
34.         BEGIN
35.             ;THROW 52000, 'Conference does not exist. ',1
36.         END
37.         INSERT INTO Reservation
38.         (
39.             ClientID,
40.             ConferenceID,
41.             ReservationDate,
42.             PaymentDate,
43.             isCancelled,
44.             Value
45.         )
46.         VALUES
47.         (
48.             @ClientID,
49.             @ConferenceID,
50.             @ReservationDate,
51.             @PaymentDate,
52.             @isCancelled,
53.             @Value
54.         )
55.     END TRY
56.     BEGIN CATCH
57.         DECLARE @errorMsg nvarchar(2048)
58.         = 'Can not add conference day. Error message: ' + ERROR_MESSAGE();
59.         ;THROW 52000, @errorMsg,1
60.     END CATCH
61. END
62. GO

```

### 7.1.8 Dodanie danych personalnych

```

1.  --dodawanie danych personalnych

```

```

2. create procedure [dbo].[procAddPerson]
3.     @FirstName varchar(30),
4.     @LastName varchar(30),
5.     @Email varchar(30),
6.     @Phone varchar(9),
7.     @StudentCard varchar(10) = null
8. as
9. begin
10.     set nocount on;
11.     begin try
12.         begin transaction;
13.         if exists
14.         (
15.             select * from PersonalData
16.             where Email=@Email or Phone=@Phone
17.         )
18.         BEGIN
19.             ; THROW 52000, 'Participant already exists.',1
20.         END
21.         insert into
22.         PersonalData
23.         (
24.             FirstName,
25.             LastName,
26.             Email,
27.             Phone,
28.             StudentCard
29.         )
30.         values
31.         (
32.             @FirstName,
33.             @LastName,
34.             @Email,
35.             @Phone,
36.             @StudentCard
37.         )
38.         commit transaction;
39.     end try
40.     BEGIN CATCH
41.         rollback transaction;
42.         DECLARE @errorMsg nvarchar(2048)
43.             = 'Can not add Person. Error message: ' + ERROR_MESSAGE();
44.             ;THROW 52000, @errorMsg,1
45.     END CATCH
46. END
47. GO

```

### 7.1.9 Dodawanie klienta firmowego

```

1. --dodawanie klienta firmowego
2. create procedure [dbo].[procAddCompanyClient]
3.     @ContactName varchar(20),
4.     @City varchar(30),
5.     @Street varchar(30),
6.     @HomePage varchar(30) = null,
7.     @Country varchar(30),
8.     @Email varchar(30),
9.     @Phone varchar(9),
10.    @CompanyName varchar(30)
11. AS
12. BEGIN
13.     set nocount on
14.     BEGIN TRY
15.         begin transaction;
16.         IF EXISTS
17.         (select * from Client where ContactName = @ContactName) or EXISTS
18.         (select * from Company
19.             where City = @City and Street = @Street and Email = @Email
20.             and Country = @Country and Phone = @Phone and CompanyName = @CompanyName)
21.         BEGIN
22.             THROW 52000, 'Client/Company already exist', 1
23.         END
24.
25.         declare @tempID INT
26.
27.         insert into Client( ContactName )
28.         values (@ContactName)
29.         IF @@ERROR <> 0
30.             begin
31.                 rollback transaction;
32.                 RAISERROR('Error, transaction not completed!',16,-1)

```



```

33.         end
34.         set @tempID = @@IDENTITY
35.
36.         if exists (select * from Company where Email = @Email or Phone = @Phone or CompanyName = @CompanyName)
37.         BEGIN
38.             THROW 52000, 'Client/Company already exist', 1
39.         END
40.
41.         if @Email not like '%@%' or LEN(@Phone) <> 9 or ISNUMERIC(@Phone) <> 1
42.         BEGIN
43.             THROW 52000, 'wrong data', 1
44.         END
45.         insert into Company(ClientID, City, Street, HomePage, Country, Email, Phone, CompanyName)
46.         values (@tempID, @City, @Street, @HomePage, @Country, @Email, @Phone, @CompanyName)
47.         commit transaction;
48.     END TRY
49.     BEGIN CATCH
50.         rollback transaction;
51.         DECLARE @errorMsg nvarchar (2048)
52.             = 'Cannot add Company Client . Error message : '
53.               + ERROR_MESSAGE () ;
54.         ; THROW 52000 , @errorMsg ,1
55.     END CATCH
56. END
57. GO

```

### 7.1.10 Dodawanie klienta indywidualnego

```

1.  --dodawanie klienta indywidualnego
2.
3.  create procedure [dbo].[procAddIndividualClient]
4.      @ContactName varchar (20),
5.      @FirstName varchar(30),
6.      @LastName varchar(30),
7.      @Email varchar(30),
8.      @Phone varchar(9),
9.      @StudentCard varchar (10) = null
10. as
11. begin
12.     set nocount on;
13.     begin try
14.         begin transaction;
15.         if exists
16.             (select * from PersonalData
17.              where Email=@Email or Phone=@Phone or @StudentCard = StudentCard)
18.         BEGIN
19.             ; THROW 52000, 'Person already exists.',1
20.         END
21.
22.         if @Email not like '%@%' or LEN(@Phone) <> 9 or ISNUMERIC(@Phone) <> 1
23.         BEGIN
24.             THROW 52000, 'wrong data', 1
25.         END
26.
27.         if exists (select * from Client where ContactName = @ContactName)
28.         BEGIN
29.             ; THROW 52000, 'Client already exists.',1
30.         END
31.         insert into Client(ContactName) values (@ContactName)
32.         IF @@ERROR <> 0
33.         begin
34.             RAISERROR('Error, transaction not completed!',16,-1)
35.             rollback transaction;
36.         end
37.         declare @ClientID int
38.         set @ClientID = @@IDENTITY
39.
40.         exec procAddPerson @FirstName, @LastName, @Email, @Phone, @StudentCard
41.         IF @@ERROR <> 0
42.         begin
43.             RAISERROR('Error, transaction not completed!',16,-1)
44.             rollback transaction;
45.         end
46.         declare @id int = @@IDENTITY
47.         insert into IndividualClients(PersonalDataID, ClientID) values (@id, @ClientID)
48.         IF @@ERROR <> 0
49.         begin
50.             RAISERROR('Error, transaction not completed!',16,-1)
51.             rollback transaction;

```

```

52.     end
53.     commit transaction;
54. end try
55. BEGIN CATCH
56.     rollback transaction;
57.     DECLARE @errorMsg nvarchar(2048)
58.     = 'Can not add IndividualClient. Error message: ' + ERROR_MESSAGE();
59.     ;THROW 52000, @errorMsg,1
60. END CATCH
61. END
62. GO

```

### 7.1.11 Dodawanie pracownika

```

1.  --dodawanie pracownika
2.
3.  create procedure procAddEmployee
4.      @firstName varchar(30),
5.      @lastName varchar(30),
6.      @Email varchar(30),
7.      @Phone varchar(9),
8.      @CompID int,
9.      @StudentCard varchar(10) = null
10. as
11. begin
12.     set nocount on
13.     begin try
14.         begin transaction
15.         if exists (select * from PersonalData where Email = @Email and Phone = @Phone and @firstName = First
16.             Name and @lastName = LastName)
17.             THROW 52000, 'Employee Data already exists', 1
18.         end
19.
20.         declare @empID int
21.         exec dbo.procAddPerson @firstName, @lastName, @Email, @Phone, @StudentCard
22.
23.         IF @@ERROR <> 0
24.         begin
25.             RAISERROR('Error, transaction not completed!',16,-1)
26.             rollback transaction;
27.         end
28.
29.         set @empID = @@IDENTITY
30.         insert into Employees (CompanyID, PersonalDataID) values (@CompID, @empID)
31.
32.         IF @@ERROR <> 0
33.         begin
34.             RAISERROR('Error, transaction not completed!',16,-1)
35.             rollback transaction;
36.         end
37.
38.         commit transaction;
39.     end try
40.     begin CATCH
41.         rollback transaction;
42.         declare @errorMsg nvarchar (2048)
43.         = 'Cannot add Company Client . Error message : '
44.         + ERROR_MESSAGE () ;
45.         ; THROW 52000 , @errorMsg ,1
46.     end CATCH
47. end
48. GO

```

### 7.1.12 Dodawanie rezerwacji dnia konferencji przez klienta indywidualnego

```

1.  --dodawanie bookingu dnia konferencji (indywidualny klient)
2.  create procedure procAddConferenceDayBookingIndividual
3.      @ConferenceDayID INT,
4.      @ReservationID INT,
5.      @StudentCard varchar(10) = null
6.  as
7.  BEGIN
8.  SET NOCOUNT ON
9.  BEGIN TRY

```

```

10.     begin transaction;
11.     if NOT EXISTS
12.         (select * from ConferenceDay where ConferenceDayID = @ConferenceDayID)
13.     BEGIN
14.         THROW 52000, 'Conference day does not exists ', 1
15.     END
16.     if NOT EXISTS
17.         (select * from Reservation where ReservationID = @ReservationID)
18.     BEGIN
19.         THROW 52000, 'Reservation does not exists ', 1
20.     END
21.     if (select ConferenceID from Reservation where ReservationID = @ReservationID )<> (select ConferenceID f
rom ConferenceDay where ConferenceDayID = @ConferenceDayID )
22.     BEGIN
23.         THROW 52000, 'Diffrenct conference is reserved', 1
24.     END
25.     if @StudentCard is not null and not exists (select *
26.         from Reservation as r
27.             left join Client as c on c.ClientID = r.ClientID
28.             left join IndividualClients as ic on ic.ClientID = c.ClientID
29.             left join PersonalData as pd on pd.PersonalDataID = ic.PersonalDataID
30.             where @StudentCard = pd.StudentCard)
31.     BEGIN
32.         THROW 52000, 'Student Card does not exists', 1
33.     END
34.     if @StudentCard is not null
35.     BEGIN
36.         insert into ConferenceDayBooking (ConferenceDayID, ReservationID, NormalTickets, ConcessionaryTicket
s, isCancelled)
37.         values ( @ConferenceDayID, @ReservationID, 0, 1, 0)
38.     END
39.     if @StudentCard is null
40.     BEGIN
41.         insert into ConferenceDayBooking (ConferenceDayID, ReservationID, NormalTickets, ConcessionaryTicket
s, isCancelled)
42.         values ( @ConferenceDayID, @ReservationID, 1, 0, 0)
43.     END
44.     commit transaction;
45. END TRY
46. BEGIN CATCH
47.     rollback transaction;
48.     DECLARE @errorMsg nvarchar (2048)
49.         = 'Cannot add Conference day booking . Error message : '
50.         + ERROR_MESSAGE () ;
51.         ; THROW 52000 , @errorMsg ,1
52.     END CATCH
53. END
54. GO

```

### 7.1.13 Dodawanie rezerwacji dnia konferencji przez klienta firmowego

```

1. --dodawanie bookingu firmy na dzień konferencji
2. create procedure procAddConferenceDayBookingCompany
3.     @ConferenceDayID INT,
4.     @ReservationID INT,
5.     @NormalTickets INT,
6.     @ConcessionaryTickets INT
7. as
8. BEGIN
9.     SET NOCOUNT ON
10.    BEGIN TRY
11.        begin transaction;
12.
13.        if NOT EXISTS
14.            (select * from ConferenceDay where ConferenceDayID = @ConferenceDayID)
15.        BEGIN
16.            THROW 52000, 'Conference day does not exists ', 1
17.        END
18.
19.        if (select ConferenceID from Reservation where ReservationID = @ReservationID )<> (select ConferenceID f
rom ConferenceDay where ConferenceDayID = @ConferenceDayID )
20.        BEGIN
21.            THROW 52000, 'Diffrenct conference is reserved', 1
22.        END
23.
24.        if NOT EXISTS
25.            (select * from Reservation where ReservationID = @ReservationID)
26.        BEGIN
27.            THROW 52000, 'Reservation does not exists ', 1
28.        END
29.

```

```

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 Uczestnika 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,

```

```

9.      @StudentCard varchar(10) = null
10.  as
11.  begin
12.  set nocount on;
13.  begin try
14.  begin transaction;
15.  if not exists ( select * from ConferenceDayBooking where ConferenceDayBookingID = @ConferenceDayBookingI
D)
16.  begin
17.      ; throw 52000, 'Wrong ConferenceDayBookingID. ConferenceDayBookingID does not exist ',1
18.  end
19.  if exists (select * from PersonalData where FirstName = @FirstName and LastName = @LastName and Email =
@Email and Phone = @Phone)
20.  begin
21.      declare @PersonalDataID int = (select PersonalDataID from PersonalData where FirstName = @FirstName
and LastName = @LastName and Email = @Email and Phone = @Phone)
22.      if exists (select * from Employees where PersonalDataID=@PersonalDataID and CompanyID = @CompanyID)
23.      begin
24.          insert into Participants (ConferenceDayBookingID, PersonalDataID)
25.          values (@ConferenceDayBookingID, @PersonalDataID)
26.      end
27.  end
28.  if not exists (select * from PersonalData where FirstName = @FirstName and LastName = @LastName and Email =
@Email and Phone = @Phone)
29.  begin
30.  exec procAddPerson @FirstName, @LastName, @Email, @Phone, @StudentCard
31.  IF @@ERROR <> 0
32.  begin
33.      RAISERROR('Error, transaction not completed!',16,-1)
34.      rollback transaction;
35.  end
36.  declare @PDataID int = @@Identity
37.  insert into Employees (CompanyID, PersonalDataID) values (@CompanyID, @PDataID)
38.  IF @@ERROR <> 0
39.  begin
40.      RAISERROR('Error, transaction not completed!',16,-1)
41.      rollback transaction;
42.  end
43.  insert into Participants (ConferenceDayBookingID, PersonalDataID) values (@ConferenceDayBookingID, @PDat
aID)
44.  IF @@ERROR <> 0
45.  begin
46.      RAISERROR('Error, transaction not completed!',16,-1)
47.      rollback transaction;
48.  end
49.  end
50.  commit transaction;
51.  end try
52.  begin catch
53.  rollback transaction;
54.      DECLARE @errorMsg nvarchar(2048)
55.      = 'Can not add ConferenceCompanyParticipant. Error message: ' + ERROR_MESSAGE();
56.      ;THROW 52000, @errorMsg,1
57.  END CATCH
58.  END
59.  GO
60.
61.

```

## 7.1.16. Dodaj firmowego uczestnika warsztatów

```

1.  -- Dodaj firmowego uczestnika warsztatów
2.
3.  create procedure [dbo].[procAddWorkshopCompanyParticipant]
4.  @ParticipantID int,
5.  @WorkshopBookingID int
6.  as
7.  begin
8.  set nocount on;
9.  begin try
10.  begin transaction;
11.  if not exists ( select * from Participants where ParticipantID = @ParticipantID)
12.  begin
13.      ; throw 52000, 'Wrong ParticipantID. ParticipantID does not exist ',1
14.  end
15.  if not exists ( select * from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
16.  begin
17.      ; throw 52000, 'Wrong WorkshopBookingID. WorkshopBookingID does not exist ',1

```

```

18.     end
19.     if (select ConferenceDayBookingID from Participants where ParticipantID = @ParticipantID) <>
20.         (select ConferenceDayBookingID from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
21.     begin
22.         ; throw 52000, 'Participant can not take part in this workshop (wrong conferencedaybookingid) ',1
23.     end
24.     insert into WorkshopParticipants(ParticipantID, WorkshopBookingID)
25.     values (@ParticipantID, @WorkshopBookingID)
26.     commit transaction;
27.     end try
28.     begin catch
29.         rollback transaction;
30.         DECLARE @errorMsg nvarchar(2048)
31.         = 'Can not add WorkshopParticipant. Error message: ' + ERROR_MESSAGE();
32.         ;THROW 52000, @errorMsg,1
33.     END CATCH
34. END
35. GO

```

### 7.1.17 Dodaj klienta indywidualnego jako uczestnika warsztatów

```

1.  --Dodaj uczestnika warsztatów
2.  create procedure [dbo].[proccAddWorkshopIndividualParticipant]
3.      @ParticipantID int,
4.      @WorkshopBookingID int
5.  as
6.  begin
7.      set nocount on;
8.      begin try
9.          begin transaction;
10.         if not exists ( select * from Participants where ParticipantID = @ParticipantID)
11.             begin
12.                 ; throw 52000, 'Wrong ParticipantID. ParticipantID does not exist ',1
13.             end
14.         if not exists ( select * from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
15.             begin
16.                 ; throw 52000, 'Wrong WorkshopBookingID. WorkshopBookingID does not exist ',1
17.             end
18.         if not exists ( select * from IndividualClients as ic
19.                         where ic.PersonalDataID = (select PersonalDataID from Participants where ParticipantID =
20. @ParticipantID))
21.             begin
22.                 ; throw 52000, 'Participant is not individual client ',1
23.             end
24.         if (select ConferenceDayBookingID from Participants where ParticipantID = @ParticipantID) <>
25.             (select ConferenceDayBookingID from WorkshopBooking where WorkshopBookingID = @WorkshopBookingID)
26.             begin
27.                 ; throw 52000, 'Participant can not take part in this workshop (wrong conferencedaybookingid) ',1
28.             end
29.         insert into WorkshopParticipants(ParticipantID, WorkshopBookingID)
30.         values (@ParticipantID, @WorkshopBookingID)
31.         commit transaction;
32.     end try
33.     begin catch
34.         rollback transaction;
35.         DECLARE @errorMsg nvarchar(2048)
36.         = 'Can not add WorkshopParticipant. Error message: ' + ERROR_MESSAGE();
37.         ;THROW 52000, @errorMsg,1
38.     END CATCH
39. END
40. GO

```

## 7.2 Procedury statystyczne

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

```

1.  --popularnosc klientow (ilosc rezerwacji)
2.  CREATE PROCEDURE show_clients_popularity

```

```

3.  as
4.  BEGIN
5.      SELECT  Name, COUNT(*) AS Number
6.      FROM (select PersonalDataID, FirstName + ' ' + LastName as Name from PersonalData) as pd
7.      JOIN IndividualClients as ic
8.      ON pd.PersonalDataID=ic.PersonalDataID
9.      JOIN Client as c
10.     ON ic.ClientID=c.ClientID
11.     JOIN Reservation as r
12.     ON r.ClientID = c.ClientID
13.     GROUP BY Name
14.
15.     UNION
16.
17.     SELECT co.CompanyName as Name, COUNT(*) AS Number
18.     FROM Company as co
19.     JOIN Client as cl
20.     ON co.ClientID=cl.ClientID
21.     JOIN Reservation as r
22.     ON r.ClientID=cl.ClientID
23.     GROUP BY co.CompanyName
24. END
25. go

```

## 7.2.2 Lista uczestników dni konferencji dla wybranej konferencji

```

1.  --lista uczestników dni konferencji dla wybranej konferencji
2.  create PROCEDURE proc_showConferenceDaysParticipants
3.      @confID int
4.  as
5.  begin
6.      select c.ConferenceName, cd.Date, pd.FirstName, pd.LastName
7.      from Conferences as c
8.      join ConferenceDay as cd
9.      on cd.ConferenceID = c.ConferenceID
10.     join ConferenceDayBooking as cdb
11.     on cdb.ConferenceDayID = cd.ConferenceDayID
12.     join Participants as p
13.     on p.ConferenceDayBookingID = cdb.ConferenceDayBookingID
14.     join PersonalData as pd
15.     on pd.PersonalDataID = p.PersonalDataID
16.     where c.ConferenceID = @confID
17.     group by ConferenceName, cd.Date, pd.FirstName, pd.LastName
18.     order by 1,2,4
19. end
20. go

```

## 7.2.3 Lista uczestników dla poszczególnych warsztatów w obrębie konferencji

```

1.  -- lista uczestników dla danego warsztatu w obrębie konferencji
2.  create PROCEDURE proc_ShowWorkshopParticipants
3.      @confID int
4.  as
5.  begin
6.      select ConferenceName, WorkshopName, pd.FirstName, pd.LastName
7.      from Conferences as c
8.      join ConferenceDay as cd
9.      on cd.ConferenceID = c.ConferenceID
10.     join Workshops as w
11.     on w.ConferenceDayID = cd.ConferenceDayID
12.     join WorkshopBooking as wb
13.     on wb.WorkshopID = w.WorkshopID
14.     join WorkshopParticipants as wp
15.     on wp.WorkshopBookingID = wb.WorkshopBookingID
16.     join Participants as p
17.     on p.ParticipantID = wp.ParticipantID
18.     join PersonalData as pd
19.     on pd.PersonalDataID = p.PersonalDataID
20.     where c.ConferenceID = @confID
21.     group by ConferenceName, WorkshopName, pd.FirstName, pd.LastName
22. end
23. go

```

## 7.2.4 Lista wydarzeń które odbywają się na danej konferencji

```
1.  -- lista wydarzen w obrębie konferencji
2.
3.  create procedure proc_Events
4.  @ConferenceID INT
5.  AS
6.  BEGIN
7.      set nocount on
8.      BEGIN TRY
9.          if NOT EXISTS (select * from Conferences where ConferenceID = @ConferenceID)
10.         BEGIN
11.             THROW 52000, 'Conference does not exists ', 1
12.         END
13.         select c.ConferenceName, cd.ConferenceDayID, (DATEDIFF(dd, c.BeginDate, cd.Date) + 1) as ConferenceDayNo
14.         , isNull(w.WorkshopName, 'no workshop') as WorkshopName
15.         from Conferences as c
16.         join ConferenceDay as cd
17.         on c.ConferenceID = cd.ConferenceID
18.         left join Workshops as w
19.         on w.ConferenceDayID = cd.ConferenceDayID
20.         where c.ConferenceID = @ConferenceID
21.         group by c.ConferenceName, cd.ConferenceDayID, DATEDIFF(dd, c.BeginDate, cd.Date), w.WorkshopName
22.     END TRY
23.     BEGIN CATCH
24.         DECLARE @errorMsg nvarchar (2048)
25.         = 'Cannot add Conference day booking . Error message : '
26.         + ERROR_MESSAGE () ;
27.         ; THROW 52000 , @errorMsg ,1
28.     END CATCH
29. go
```

## 7.2.5 Lista uczestników danego warsztatu

```
1.  --uczestnicy danego warsztatu
2.  create procedure workshop_participants
3.  @WID int
4.  as
5.  BEGIN
6.      BEGIN TRY
7.          set nocount on
8.          if not exists (select * from Workshops where WorkshopID = @WID)
9.         begin
10.             THROW 52000, 'Workshop does not exists ', 1
11.         end
12.         select wp.ParticipantID, pd.FirstName, pd.LastName, pd.Phone
13.         from Workshops as w
14.         left join WorkshopBooking as wb
15.         on wb.workshopID=w.workshopID
16.         left join WorkshopParticipants as wp
17.         on wp.WorkshopBookingID=wb.WorkshopBookingID
18.         left join Participants as p
19.         on p.ParticipantID=wp.ParticipantID
20.         left join personaldata as pd
21.         on pd.PersonalDataID = p.PersonalDataID
22.         where w.WorkshopID = @WID
23.     END TRY
24.     BEGIN CATCH
25.         DECLARE @errorMsg nvarchar (2048)
26.         = 'Cannot find participants . Error message : '
27.         + ERROR_MESSAGE () ;
28.         ; THROW 52000 , @errorMsg ,1
29.     END CATCH
30. END
31. go
```

## 7.2.6 Lista uczestników danej konferencji

```
1.  --uczestnicy danej konferencji
2.  create procedure conference_participants
3.  @CID int
4.  as
5.  BEGIN
6.      BEGIN TRY
7.          set nocount on
8.          if not exists (select * from Conferences where ConferenceID = @CID)
```



```

9.      begin
10.         THROW 52000, 'Conference does not exists ', 1
11.      end
12.      select p.ParticipantID, pd.FirstName, pd.LastName, pd.Phone
13.      from Conferences as c
14.      join ConferenceDay as cd
15.      on cd.ConferenceID=c.ConferenceID
16.      join ConferenceDayBooking as cdb
17.      on cdb.ConferenceDayID=cd.ConferenceDayID
18.      join Participants as p
19.      on p.ConferenceDayBookingID = cdb.ConferenceDayBookingID
20.      join personaldata as pd
21.      on pd.PersonalDataID = p.PersonalDataID
22.      where c.ConferenceID = @CID
23.      END TRY
24.      BEGIN CATCH
25.      DECLARE @errorMsg nvarchar (2048)
26.      = 'Cannot find participants . Error message : '
27.      + ERROR_MESSAGE () ;
28.      ; THROW 52000 , @errorMsg ,1
29.      END CATCH
30.  END
31.  go

```

## 7.2.7 Lista uczestników danego dnia konferencji

```

1.  --uczestnicy danego dnia konferencji
2.  create procedure conferenceday_participants
3.  @CDID int
4.  as
5.  BEGIN
6.      BEGIN TRY
7.          set nocount on
8.          if not exists (select * from ConferenceDay where ConferenceDayID = @CDID)
9.          begin
10.             THROW 52000, 'ConferenceDay does not exists ', 1
11.          end
12.          select p.ParticipantID, pd.FirstName, pd.LastName, pd.Phone
13.          from ConferenceDay as cd
14.          join ConferenceDayBooking as cdb
15.          on cdb.ConferenceDayID=cd.ConferenceDayID
16.          join Participants as p
17.          on p.ConferenceDayBookingID = cdb.ConferenceDayBookingID
18.          join personaldata as pd
19.          on pd.PersonalDataID = p.PersonalDataID
20.          where cd.ConferenceDayID = @CDID
21.          END TRY
22.          BEGIN CATCH
23.          DECLARE @errorMsg nvarchar (2048)
24.          = 'Cannot find participants . Error message : '
25.          + ERROR_MESSAGE () ;
26.          ; THROW 52000 , @errorMsg ,1
27.          END CATCH
28.  END
29.  go

```

## 7.2.8 Lista konferencji danego uczestnika

```

1.  -- konferencje danego uczestnika
2.  create procedure participant_conferences
3.  @PID int
4.  as
5.  BEGIN
6.      BEGIN TRY
7.          set nocount on
8.          if not exists (select * from Participants where ParticipantID = @PID)
9.          begin
10.             THROW 52000, 'CParticipant does not exists ', 1
11.          end
12.          select c.ConferenceID, c.ConferenceName
13.          from Participants as p
14.          join ConferenceDayBooking as cdb on cdb.ConferenceDayBookingID = p.ConferenceDayBookingID
15.          join ConferenceDay as cd on cdb.ConferenceDayID = cd.ConferenceDayID
16.          join Conferences as c on c.ConferenceID = cd.ConferenceID
17.          where p.ParticipantID = @PID
18.          END TRY
19.          BEGIN CATCH

```

```

20. DECLARE @errorMsg nvarchar (2048)
21.         = 'Cannot find participant . Error message : '
22.           + ERROR_MESSAGE () ;
23.         ; THROW 52000 , @errorMsg ,1
24.     END CATCH
25. END
26. go

```

## 7.2.9 Wpływy z rezerwacji dla danej konferencji

```

1. -- wpływy z rezerwacji dla wybranej konferencji
2. create procedure conference_incomes
3.     @CID int
4. as
5. BEGIN
6.     BEGIN TRY
7.         set nocount on
8.         if not exists (select * from Conferences where ConferenceID = @CID)
9.         begin
10.             THROW 52000, 'Conference does not exists ', 1
11.         end
12.         select r.ReservationID, sum(r.Value) as Sum
13.         from Reservation as r
14.         where r.ConferenceID = @CID and r.isCancelled = 0
15.         group by r.ReservationID
16.     END TRY
17.     BEGIN CATCH
18.         DECLARE @errorMsg nvarchar (2048)
19.         = 'Cannot find conference . Error message : '
20.           + ERROR_MESSAGE () ;
21.         ; THROW 52000 , @errorMsg ,1
22.     END CATCH
23. END
24. go

```

## 7.2.10 Lista konferencji które odbędą się w przeciągu podanej liczby dni

```

1. -- nadchodzące konferencje w przeciągu x dni
2. create procedure coming_conferences
3.     @days int
4. as
5. BEGIN
6.     BEGIN TRY
7.         set nocount on
8.         if (@days<0)
9.         begin
10.             THROW 52000, 'a negative number ', 1
11.         end
12.         select c.ConferenceID, c.ConferenceName
13.         from Conferences as c
14.         where c.BeginDate >= getdate() and c.BeginDate<=dateadd(day,@days,getdate())
15.     END TRY
16.     BEGIN CATCH
17.         DECLARE @errorMsg nvarchar (2048)
18.         = 'Cannot find participant . Error message : '
19.           + ERROR_MESSAGE () ;
20.         ; THROW 52000 , @errorMsg ,1
21.     END CATCH
22. END
23. go

```

## 7.2.11 Lista wydarzeń na które podany uczestnik został zapisany

```

1. --procedura-widok sprawdzenie na co jest się zapisanym
2.
3. create procedure enrollments
4.     @id int
5. as
6. BEGIN
7.     BEGIN TRY
8.         set nocount on

```

```

9.      if not exists (select * from Participants where ParticipantID = @id)
10.      BEGIN
11.          THROW 52000, 'Participant does not exists ', 1
12.      END
13.      select w.WorkshopID, w.StartTime, w.EndTime, cd.ConferenceDayID, cd.Date
14.      from Participants as p
15.      join ConferenceDayBooking as cdb on cdb.ConferenceDayBookingID = p.ConferenceDayBookingID
16.      join WorkshopBooking as wb on wb.ConferenceDayBookingID = cdb.ConferenceDayBookingID
17.      join ConferenceDay as cd on cd.ConferenceDayID = cdb.ConferenceDayID
18.      join Workshops as w on w.WorkshopID = wb.WorkshopID
19.      where p.ParticipantID = @id and wb.isCancelled = 0 and cdb.isCancelled = 0
20.
21.      END TRY
22.      BEGIN CATCH
23.      DECLARE @errorMsg nvarchar (2048)
24.          = 'Cannot find participant . Error message : '
25.          + ERROR_MESSAGE () ;
26.          ; THROW 52000 , @errorMsg ,1
27.      END CATCH
28.  END
29.  go

```

## 7.3 Procedury aktualizujące

### 7.3.1 Zmiana limitu miejsc na warsztacie

```

1.  --ZMIANA LIMITU MIEJSC NA WARSZTACIE
2.  create PROCEDURE update_workshop_participants_limit
3.      @WorkShopID int,
4.      @newLimit int
5.  AS
6.  BEGIN
7.      SET NOCOUNT ON
8.      BEGIN TRY
9.          DECLARE @diff int
10.         set @diff = @newLimit - (SELECT Limit from Workshops where WorkShopID = @WorkShopID);
11.
12.         IF dbo.funcWorkshopFreePlaces(@WorkShopID) < (-1 * @diff)
13.         BEGIN
14.             THROW 52000, 'There are too many registered participants to resize this workshop',1
15.         END
16.
17.         UPDATE Workshops
18.         SET Limit = @newLimit
19.         where WorkShopID = @WorkShopID
20.     END TRY
21.     BEGIN CATCH
22.         DECLARE @errorMsg nvarchar(2048)
23.         = 'cannot change limit: ' + ERROR_MESSAGE();
24.         ;THROW 52000, @errorMsg,1
25.     END CATCH
26. END
27. go

```

### 7.3.2 Zmiana limitu dnia konferencji

```

1.  --zmiana limitu dnia konferencji
2.  CREATE PROCEDURE update_conferenceday_participants_limit
3.      @ConferenceDayID int,
4.      @newLimit int
5.  AS
6.  BEGIN
7.      DECLARE @diff int
8.      set @diff = @newLimit - (SELECT Limit from ConferenceDay where ConferenceDayID = @ConferenceDayID);
9.
10.     IF dbo.funcConferenceDayFreePlaces(@ConferenceDayID) < (-1 * @diff)
11.     BEGIN
12.         THROW 52000, 'There are too many registered participants to resize this conferenceday',1
13.     END
14.
15.     UPDATE ConferenceDay
16.     SET Limit = @newLimit

```

```

17.         where ConferenceDayID = @ConferenceDayID
18.     END
19. go

```

### 7.3.3 Unieważnienie nieopłaconych zamówień

```

1.  --unieważnianie nieopłaconych zamówień
2.  create procedure cancel_reservations
3.  as
4.  BEGIN
5.      update Reservation
6.      set Reservation.isCancelled = 1
7.      from Reservation
8.      join Conferences as c
9.      on c.ConferenceID = Reservation.ConferenceID
10.     where Reservation.PaymentDate is null
11.     and DATEDIFF(dd,Reservation.ReservationDate,GETDATE()) > 7
12. END
13. go

```

### 7.3.4 Wprowadzenie daty płatności

```

1.  --wprowadzanie daty płatności
2.  create procedure proc_setPaymentDate
3.  @ReservationID INT,
4.  @PaymentDate DATE
5.  as
6.  BEGIN
7.      SET NOCOUNT ON
8.      BEGIN TRY
9.          if not exists (select * from Reservation where ReservationID = @ReservationID)
10.         BEGIN
11.             THROW 52000, 'Reservation does not exist ', 1
12.         END
13.
14.         if exists (select * from Reservation where ReservationID = @ReservationID and PaymentDate is not null)
15.         BEGIN
16.             THROW 52000, 'PaymentDate already exists ', 1
17.         END
18.
19.         if exists (select * from Reservation where ReservationID = @ReservationID and isCancelled = 1)
20.         BEGIN
21.             THROW 52000, 'Reservation is cancelled ', 1
22.         END
23.
24.         if DATEDIFF(dd, (select ReservationDate from Reservation where ReservationID = @ReservationID ), @PaymentDate) < 0
25.         BEGIN
26.             THROW 52000, 'Wrong date', 1
27.         END
28.
29.         update Reservation
30.         set Reservation.PaymentDate = @PaymentDate
31.         where Reservation.ReservationID = @ReservationID
32.     END TRY
33.     BEGIN CATCH
34.         DECLARE @errorMsg nvarchar (2048)
35.         = 'Cannot add payment date . Error message : '
36.         + ERROR_MESSAGE () ;
37.         ; THROW 52000 , @errorMsg ,1
38.     END CATCH
39. END
40. go

```

## 8. Triggery

## 8.1 Poprawna liczba studentów

```
1.  --
   trigger / czy liczba studentow zapowiedzianych zgadza sie z rzeczywista przy pelnym wypelnieniu uczestnikow
   dla zadanej rezerwacji
2.
3.  create trigger appropriate_students_amount
4.  on Participants
5.  after insert, update
6.  as
7.  begin
8.
9.      if exists
10.     (
11.         select *
12.         from Inserted as p
13.         join PersonalData as pd
14.         on pd.PersonalDataID = p.PersonalDataID
15.         join ConferenceDayBooking as cdb
16.         on cdb.ConferenceDayBookingID = p.ConferenceDayBookingID
17.         group by pd.PersonalDataID, cdb.ConcessionaryTickets, cdb.NormalTickets
18.         having cdb.ConcessionaryTickets != (select count(pd_inner.PersonalDataID)
19.                                             from PersonalData as pd_inner
20.                                             where pd.PersonalDataID = pd_inner.PersonalDataID and pd_inner.Stude
ntCard is not null)
21.         and (cdb.ConcessionaryTickets+cdb.NormalTickets) = (select count(pd_inner.PersonalDataID)
22.                                                            from PersonalData as pd_inner
23.                                                            where pd.PersonalDataID = pd_inner.PersonalDataID)
24.     )
25.     BEGIN
26.         THROW 50001 , 'Number of students is not equal to declared number of students in booking ' , 1
27.         ROLLBACK TRANSACTION
28.     END
29. end
```

## 8.2 Mniej zapisanych na warsztat niż na konferencję

```
1.  --mniej zapisanych na warsztat niż na konferencję
2.
3.  create TRIGGER [dbo].[workshop_participants_fewer_than_day_participants_trigger]
4.  ON [dbo].[WorkshopBooking]
5.  AFTER INSERT, UPDATE
6.  AS
7.  BEGIN
8.      IF EXISTS(
9.          SELECT *
10.         FROM inserted as i
11.         JOIN ConferenceDayBooking as cdb
12.         ON cdb.ConferenceDayBookingID=i.ConferenceDayBookingID
13.         WHERE (i.NormalTickets+i.ConcessionaryTickets)>(cdb.NormalTickets+cdb.ConcessionaryTickets)
14.     )
15.     BEGIN
16.         THROW 50001, 'All workshop participants must be day attendees', 1
17.     END
18. END
```

## 8.3 Większy limit na konferencji niż na warsztatach

```
1.  --
   sprawdzenie czy w dniu konferencji jest wiekszy limit na uczestnikow niz na warsztatach (bo każdy uczestnik
   warsztatu musi byc zapisany na ten dzien!!!)
2.
3.
4.  create trigger appropriate_limits
5.  on Workshops
6.  after insert, update
7.  as
8.  begin
9.      if exists
10.     (
11.         select *
12.         from Inserted as i
13.         join ConferenceDay as cd
14.         on cd.ConferenceDayID = i.ConferenceDayID
15.         where i.Limit > cd.Limit
16.     )
17.     begin
```

```

18.      THROW 50001 , 'Workshop cannot have greater limit of participant's amount than conference day than '
19.      , 1
20.      ROLLBACK TRANSACTION
21.  end

```

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

```

1.  --
2.  -- sprawdzenie, czy przypadkiem liczba zarejestrowanych na warsztat nie jest większa niż zarejestrowanych na d
3.  --   dzień konferencji
4.  create trigger workshop_participants_conf_participants
5.  on WorkshopBooking
6.  after insert, update
7.  as
8.  begin
9.      if exists
10.     (
11.         select *
12.         from Inserted as i
13.         join ConferenceDayBooking as cdb
14.         on cdb.ConferenceDayBookingID = i.ConferenceDayBookingID
15.         where (i.ConcessionaryTickets + i.NormalTickets) > (cdb.ConcessionaryTickets + cdb.NormalTickets)
16.     )
17.     begin
18.         THROW 50001 , 'There cannot be more participants on workshop than conference day' , 1
19.         ROLLBACK TRANSACTION
20.     end
21. end

```

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

```

1.  -- sprawdzenie, czy w jednym zamówieniu na pewno są ujęte dni rezerwacji na dany dzień konferencji
2.
3.  create trigger conf_days_in_one_conf
4.  on ConferenceDayBooking
5.  after insert
6.  as
7.  begin
8.      if exists
9.      (
10.         select *
11.         from Inserted as i
12.         join Reservation as r
13.         on r.ReservationID = i.ReservationID
14.         join ConferenceDays as cd
15.         on cd.ConferenceDayID = i.ConferenceDayID
16.         join Conferences as c1
17.         on c1.ConferenceID = cd.ConferenceID
18.         join Conferences as c2
19.         on c2.ConferenceID = r.ConferenceID
20.         where c2.ConferenceID != c1.ConferenceID
21.     )
22.     begin
23.         THROW 50001 , 'Cannot book conference days from different conferences in one order! ' , 1
24.         ROLLBACK TRANSACTION
25.     end
26. end

```

## 8.6 Nienachodzenie czasu warsztatów

```

1.  create trigger [dbo].[overlapping_workshop_time]
2.  on [dbo].[WorkshopParticipants]
3.  after insert
4.  as
5.  begin
6.      if exists
7.      (
8.         select *
9.         from Inserted as i
10.         join WorkshopBooking as wb1 on wb1.WorkshopBookingID = i.WorkshopBookingID
11.         join Workshops as w1 on w1.WorkshopID = wb1.WorkshopID
12.         where i.ParticipantID in
13.         (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
18.         AND w1.WorkshopID <> wb.WorkshopID
19.     )
20. begin
21.     THROW 50001 , 'Workshops cannor overlap each other' , 1
22.     ROLLBACK TRANSACTION
23. end

```

## 8.7 Sprawdzenie, czy zniżki czasowe na daną konferencję maleją

```

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
19. m Inserted)
20.      begin
21.          THROW 50001 , 'Prices are not in correct order' ,1
22.          ROLLBACK TRANSACTION
23.      end
24. 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 wystarczyło 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

```

```

11.     )
12.     BEGIN
13.         THROW 50001, 'Too few free places to register this many conference day attendees.', 1
14.     END
15. END
16. GO

```

## 8.10 Sprawdzenie, czy wystarczyło miejsc na warsztat

```

1.  --sprawdzenie, czy wystarczyło miejsc w warsztacie
2.
3.  CREATE TRIGGER [dbo].[too_few_workshop_places_trigger]
4.  ON [dbo].[WorkshopBooking]
5.  AFTER INSERT
6.  AS
7.  BEGIN
8.      IF EXISTS
9.      (
10.         SELECT * FROM inserted AS i
11.         WHERE dbo.funcWorkshopFreePlaces(i.WorkshopID) < 0
12.      )
13.      BEGIN
14.         THROW 50001, 'Too few free places to register this many workshop attendees.',1
15.      END
16.  END
17. GO

```

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

```

1.  --Sprawdzenie limitu przypisania uczestników do rezerwacji
2.
3.  CREATE TRIGGER [dbo].[attendees_not_above_reservation_trigger]
4.  ON [dbo].[Participants]
5.  AFTER INSERT
6.  AS
7.  BEGIN
8.      IF EXISTS
9.      (
10.         SELECT *
11.         FROM inserted AS i
12.         GROUP BY i.ConferenceDayBookingID
13.         HAVING EXISTS(
14.             SELECT *
15.             FROM ConferenceDayBooking cdb
16.             WHERE cdb.ConferenceDayBookingID=i.ConferenceDayBookingID
17.                 AND (cdb.NormalTickets + cdb.ConcessionaryTickets) <
18.                     (SELECT COUNT(*)
19.                      FROM Participants as p
20.                      WHERE p.ConferenceDayBookingID=c
21.                      GROUP BY p.ConferenceDayBookingID)
22.         )
23.      )
24.      BEGIN
25.         THROW 50001, 'Number of attendees must not exceed number of reservations', 1
26.      END
27.  END
28. GO

```

## 8.12 Sprawdzenie limitu przypisania uczestników do warsztatu

```

1.  --Sprawdzenie limitu przypisania uczestników do warsztatu
2.
3.  CREATE TRIGGER [dbo].[attendees_not_above_reservation_trigger_workshops]
4.  ON [dbo].[WorkshopParticipants]
5.  AFTER INSERT
6.  AS
7.  BEGIN
8.      IF EXISTS
9.      (
10.         SELECT *
11.         FROM inserted AS i
12.         GROUP BY i.WorkshopBookingID
13.         HAVING EXISTS(

```



```

14.         SELECT *
15.         FROM WorkshopBooking wb
16.         WHERE wb.WorkshopBookingID=i.WorkshopBookingID
17.             AND (wb.ConcessionaryTickets + wb.NormalTickets) < (SELECT COUNT(*)
18.                             FROM WorkshopParticipants as wp
19.                             WHERE wp.WorkshopBookingID=wb.WorkshopBookingID
20.                             GROUP BY wp.WorkshopBookingID)
21.     )
22.
23. )
24. BEGIN
25.     THROW 50001, 'Number of attendees must not exceed number of reservations', 1
26. END
27. END
28. GO

```

## 8.13 Opłacenie po anulacji

```

1. -- opłacenie po anulacji
2. CREATE TRIGGER [dbo].[check_payments_opportunity]
3. ON [dbo].[Reservation]
4. AFTER INSERT, UPDATE
5. AS
6. BEGIN
7.     IF EXISTS
8.     (
9.         SELECT * FROM inserted AS i
10.        WHERE (isCancelled = 1)
11.    )
12.    BEGIN
13.        THROW 50001, 'Payment cannot be added',1
14.    END
15. END

```

## 8.14 Zamówienie po dniu konferencji

```

1. --zamówienie po dniu konferencji
2. CREATE TRIGGER [dbo].[order_before_conference_day_trigger]
3. ON [dbo].[Reservation]
4. AFTER INSERT, UPDATE
5. AS
6. BEGIN
7.     IF EXISTS(
8.         SELECT *
9.         FROM inserted AS i
10.        JOIN Conferences as c
11.        ON c.ConferenceID = i.ConferenceID
12.        WHERE c.EndDate<=i.ReservationDate
13.    )
14.    BEGIN
15.        THROW 50001, 'Order cannot be placed for conference day in the past', 1
16.    END
17. END

```

## 9. Indeksy

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

```

1. CREATE INDEX ConferenceDay_idx on ConferenceDay (Date ASC)
2. ;
3.
4. CREATE INDEX BeginDate_idx on Conferences (BeginDate ASC)
5. ;
6.
7. CREATE INDEX EndDate_idx on Conferences (EndDate ASC)
8. ;
9.
10. CREATE INDEX discount_idx on Price (Discount ASC)
11. ;

```

```

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 obsługi 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
- procAddConferenceIndividualParticipant
- procAddWorkshopIndividualParticipant
- enrollments

#### Klient Firmowy:

- 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