

Spis treści

Spis treści.....	1
Wprowadzenie	4
Cel.....	4
Aktorzy.....	4
Administrator	4
Pracownik firmy.....	4
Klient biznesowy.....	4
Uczestnik	4
Schemat bazy.....	5
Tabele	6
Conferences.....	6
ConferenceDays	7
ConferenceCosts.....	8
Workshops.....	9
ConferenceBooking	10
ConferenceDayBooking.....	11
WorkshopBooking	12
Payments	13
Participants.....	14
DayParticipants.....	15
WorkshopParticipants	16
Clients.....	17
Indeksy.....	18
Widoki	19
View_MostPopularWorkshops.....	19
View_MostPopularConferences.....	20
View_MostPopularConferencesByStudents	21
View_MostPopularWorkshopsByStudents	22
View_MostProfitableConference	23
View_MostProfitableWorkshops	24
View_WorkshopsFreePlaces	25
View_ConferenceFreePlaces	26
View_AvailableConferenceDays.....	27
View_AvailableWorkshops.....	28

View_ClientsActivity	29
View_MostProfitableClients	30
Procedury	31
Dodające	31
Procedure_AddConferenceDay	31
Procedure_AddWorkshop	32
Procedure_AddConferenceCost	33
Procedure_AddConferenceBooking	34
Procedure_AddConferenceDayBooking	35
Procedure_AddWorkshopBooking	36
Procedure_AddDayParticipant	37
Procedure_AddWorkshopParticipant	38
Procedure_AddParticipant	39
Procedure_AddClient	40
Aktualizujące	41
Procedure_RemoveConference	41
Procedure_UpdateConferenceDetails	42
Procedure_UpdateWorkshopDetails	44
Procedure_CancelConferenceBooking	46
Procedure_CancelConferenceDayBooking	47
Procedure_UpdateWorkshopNumberOfParticipants	48
Procedure_UpdateConferenceDayNumberOfParticipants	49
Procedure_CancelConferenceBookingWithoutPayingAfterSevenDays	50
Wyświetlające	51
Procedure_ShowConferenceDaysAmountOfParticipants	51
Procedure_ShowListOfEventsOfConference	52
Funkcje	53
Function_FreeDayPlaces	53
Function_FreeWorkshopPlaces	54
Function_BookingFreeStudentPlaces	55
Function_DaysOfConference	56
Function_ConferenceDayParticipants	57
Function_WorkshopsPerConference	58
Function_WorkshopDate	59
Function_BookingDaysCost	60
Function_BookingWorkshopCost	61

Function_TotalBookingCost	62
Function_WorkshopListForParticipant.....	63
Function_ConferencesDaysListForParticipant	64
Function_ClientsOrdersList	65
Triggery.....	66
Trigger_TooFewFreePlacesForDayBooking.....	66
Trigger_TooFewFreePlacesForWorkshopBooking	67
Trigger_LessPlacesForDayThanForWorkshop	68
Trigger_NotEnoughBookedPlacesForDay	69
Trigger_NotEnoughBookedPlacesForWorkshop	70
Trigger_TooFewPlacesAfterDecreasingDayCapacity.....	71
Trigger_TooFewPlacesAfterDecreasingWorkshopCapacity	72
Trigger_BookingDayInDifferentConference	73
Trigger_BookingDayAlreadyExists.....	74
Trigger_BookingWorkshopInDifferentDay	75
Trigger_ArePriceThresholdsMonotonous	76

Wprowadzenie

Cel

Tworzona baza danych wspomagania firmy organizującej konferencje. Firma organizuje konferencje, które mogą być jedno- lub kilkudniowe. Klienci powinni móc rejestrować się na konferencje za pomocą systemu www. Klientami mogą być zarówno indywidualne osoby jak i firmy, natomiast uczestnikami konferencji są osoby (firma nie musi podawać od razu przy rejestracji listy uczestników - może zarezerwować odpowiednią ilość miejsc na określone dni oraz na warsztaty, natomiast na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić - a jeśli sama nie uzupełni do tego czasu, to pracownicy dzwonią do firmy i ustalają takie informacje). Każdy uczestnik konferencji otrzymuje identyfikator imienny (+ ew. informacja o firmie na nim). Dla konferencji kilkudniowych, uczestnicy mogą rejestrować się na dowolne z tych dni.

Aktorzy

Administrator

Osoba znajdująca się na bazach danych, potrafiąca obsłużyć błędy oraz w miarę możliwości poprawiająca i rozszerzająca działanie bazy. Ma dostęp do wszystkich procedur i widoków.

Pracownik firmy

Osoba odpowiadająca za przyjmowanie zamówień od klientów biznesowych, egzekwowanie opłat za zamówione konferencje oraz pomagająca w przypadku problemów z rejestracją.

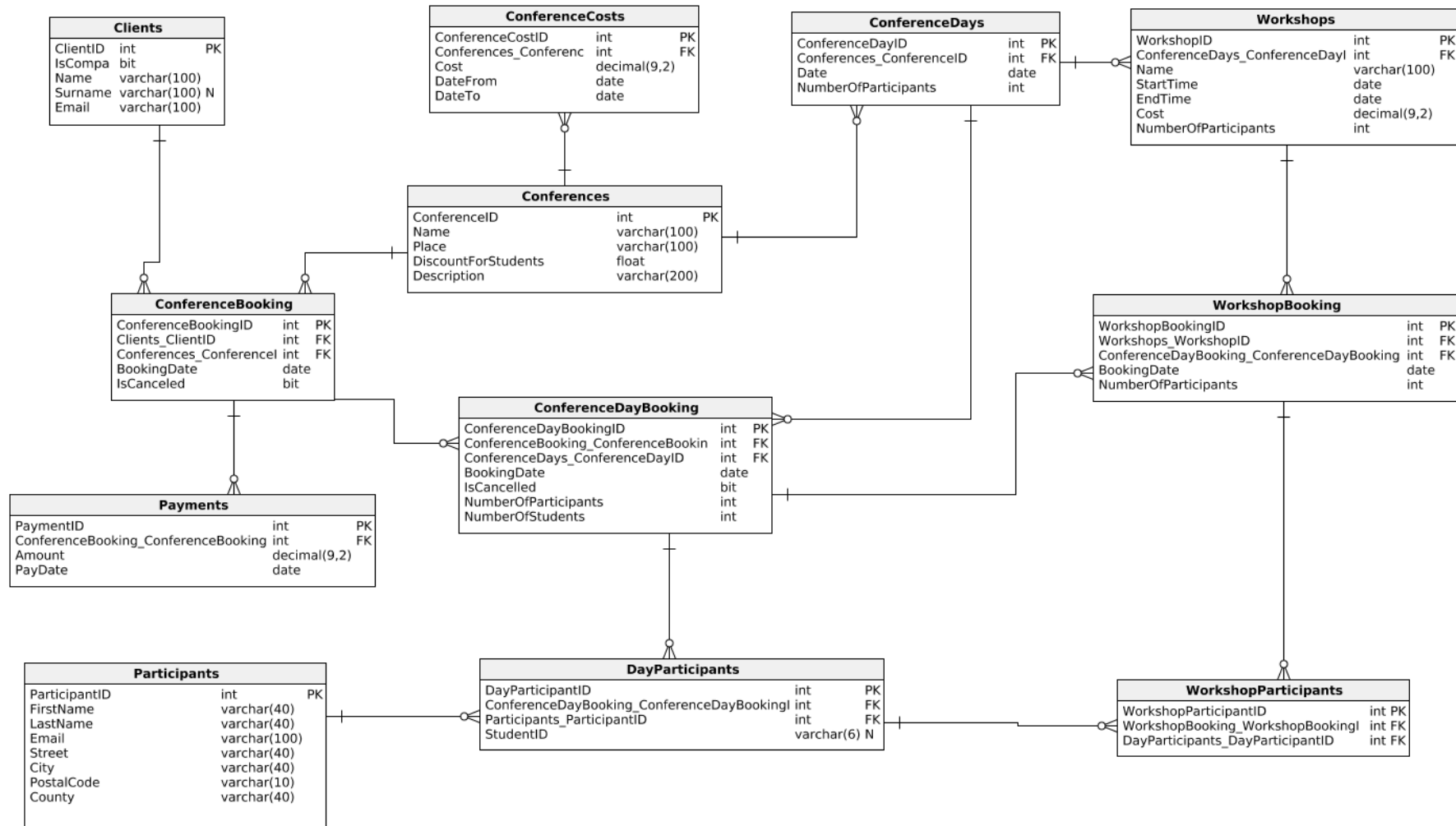
Klient biznesowy

Osoba składająca zamówienie oraz dodająca uczestników do poszczególnych konferencji i warsztatów za pomocą zewnętrznego serwisu.

Uczestnik

Osoba uczestnicząca w konferencji oraz wybranych warsztatach. Zostaje ona wprowadzona do bazy za pośrednictwem Klienta Biznesowego. Nie posiada dostępu do żadnych funkcji w systemie.

Schemat bazy



Tabele

Conferences

```
CREATE TABLE Conferences (  
    ConferenceID INT NOT NULL IDENTITY,  
    Name VARCHAR(100) NOT NULL,  
    Place VARCHAR(100) NOT NULL,  
    DiscountForStudents FLOAT NOT NULL DEFAULT 0,  
    Description VARCHAR(200) NOT NULL,  
    CONSTRAINT ProperDiscountForStudents CHECK (  
        DiscountForStudents >= 0  
        AND DiscountForStudents <= 100  
    ),  
    CONSTRAINT ProperDescription CHECK (LEN(Description) > 5),  
    CONSTRAINT Conferences_pk PRIMARY KEY (ConferenceID)  
);
```

Nazwa kolumny	Typ danych	Inne informacje
ConferenceID	INT	PK
Name	VARCHAR(100)	
Place	VARCHAR(100)	
DiscountForStudents	FLOAT	
Description	VARCHAR(100)	

ConferenceDays

```
CREATE TABLE ConferenceDays (  
    ConferenceDayID INT NOT NULL IDENTITY,  
    Conferences_ConferenceID INT NOT NULL,  
    DATE DATE NOT NULL,  
    NumberOfParticipants INT NOT NULL,  
    CONSTRAINT PositiveNumberOfConferenceParticipants CHECK  
(NumberOfParticipants > 0),  
    CONSTRAINT ConferenceDays_pk PRIMARY KEY (ConferenceDayID)  
);
```

```
-- Reference: Conferences_ConferenceDays (table: ConferenceDays)  
ALTER TABLE ConferenceDays ADD CONSTRAINT Conferences_ConferenceDays  
FOREIGN KEY (Conferences_ConferenceID) REFERENCES Conferences  
(ConferenceID)
```

Nazwa kolumny	Typ danych	Inne informacje
ConferenceDayID	INT	PK
Conferences_ConferenceID	INT	FK
DATE	DATE	
NumberOfParticipants	INT	MUST BE POSITIVE

ConferenceCosts

```
CREATE TABLE ConferenceCosts (  
    ConferenceCostID INT NOT NULL IDENTITY,  
    Conferences_ConferenceID INT NOT NULL,  
    Cost DECIMAL(9, 2) NOT NULL,  
    DateFrom DATE NOT NULL,  
    DateTo DATE NOT NULL,  
    CONSTRAINT ProperDayDifference CHECK (DateFrom <= DateTo),  
    CONSTRAINT NonnegativeCostConf CHECK (Cost >= 0),  
    CONSTRAINT ConferenceCosts_pk PRIMARY KEY (ConferenceCostID)  
);
```

```
-- Reference: Conferences_ConferenceDay (table: ConferenceCosts)  
ALTER TABLE ConferenceCosts ADD CONSTRAINT Conferences_ConferenceDay  
FOREIGN KEY (Conferences_ConferenceID) REFERENCES Conferences  
(ConferenceID)
```

Nazwa kolumny	Typ danych	Inne informacje
ConferenceCostID	INT	PK
Conferences_ConferenceID	INT	FK
Cost	DECIMAL(9,2)	Cost >= 0
DateFrom	DATE	DateFrom <= DateTo
DateTo	DATE	DateFrom <= DateTo

Workshops

```
CREATE TABLE Workshops (
    WorkshopID INT NOT NULL IDENTITY,
    ConferenceDays_ConferenceDayID INT NOT NULL,
    Name VARCHAR(100) NOT NULL,
    StartTime DATE NOT NULL,
    EndTime DATE NOT NULL,
    Cost DECIMAL(9, 2) NOT NULL DEFAULT 0,
    NumberOfParticipants INT NOT NULL DEFAULT 10,
    CONSTRAINT NonnegativeCostWorkshop CHECK (Cost >= 0),
    CONSTRAINT ProperTimeDifferance CHECK (StartTime <= EndTime),
    CONSTRAINT PosiviteWorkshopParticipants CHECK (NumberOfParticipants
> 0),
    CONSTRAINT Workshops_pk PRIMARY KEY (WorkshopID)
);
```

```
-- Reference: Workshops_ConferenceDays (table: Workshops)
```

```
ALTER TABLE Workshops ADD CONSTRAINT Workshops_ConferenceDays FOREIGN KEY
(ConferenceDays_ConferenceDayID) REFERENCES ConferenceDays
(ConferenceDayID)
```

Nazwa kolumny	Typ danych	Inne informacje
WorkshopID	INT	PK
ConferenceDays_ConferenceID	INT	FK
Name	VARCHAR(100)	
StartTime	DATE	StartTime <= EndTime
EndTime	DATE	StartTime <= EndTime
Cost	DECIMAL(9,2)	Cost >= 0
NumberOfParticipants	INT	MUST BE POSITIVE

ConferenceBooking

```
CREATE TABLE ConferenceBooking (  
    ConferenceBookingID INT NOT NULL IDENTITY,  
    Clients_ClientID INT NOT NULL,  
    Conferences_ConferenceID INT NOT NULL,  
    BookingDate DATE NOT NULL DEFAULT getdate(),  
    IsCanceled BIT NOT NULL DEFAULT 0,  
    CONSTRAINT ConferenceBooking_pk PRIMARY KEY (ConferenceBookingID)  
);
```

```
-- Reference: BookingConference_Clients (table: ConferenceBooking)  
ALTER TABLE ConferenceBooking ADD CONSTRAINT BookingConference_Clients  
FOREIGN KEY (Clients_ClientID) REFERENCES Clients (ClientID)
```

```
-- Reference: BookingConference_Conferences (table: ConferenceBooking)  
ALTER TABLE ConferenceBooking ADD CONSTRAINT BookingConference_Conferences  
FOREIGN KEY (Conferences_ConferenceID) REFERENCES Conferences  
(ConferenceID)
```

Nazwa kolumny	Typ danych	Inne informacje
ConferenceBookingID	INT	PK
Clients_ClientID	INT	FK
Conferences_ConferenceID	INT	FK
BookingDate	DATE	
IsCanceled	BIT	

ConferenceDayBooking

```
CREATE TABLE ConferenceDayBooking (
    ConferenceDayBookingID INT NOT NULL IDENTITY,
    ConferenceBooking_ConferenceBookingID INT NOT NULL,
    ConferenceDays_ConferenceDayID INT NOT NULL,
    BookingDate DATE NOT NULL DEFAULT getdate(),
    IsCancelled BIT NOT NULL DEFAULT 0,
    NumberOfParticipants INT NOT NULL,
    NumberOfStudents INT NOT NULL DEFAULT 0,
    CONSTRAINT PositiveNumberOfParticipants CHECK (NumberOfParticipants > 0),
    CONSTRAINT NonnegativeNumberOfStudents CHECK (NumberOfStudents >= 0),
    CONSTRAINT ProperNumberOfStudents CHECK (NumberOfParticipants >=
NumberOfStudents),
    CONSTRAINT ConferenceDayBooking_pk PRIMARY KEY (ConferenceDayBookingID)
);
```

```
-- Reference: ConferenceDayBooking_ConferenceBooking (table:
ConferenceDayBooking)
```

```
ALTER TABLE ConferenceDayBooking ADD CONSTRAINT
ConferenceDayBooking_ConferenceBooking FOREIGN KEY
(ConferenceBooking_ConferenceBookingID) REFERENCES ConferenceBooking
(ConferenceBookingID)
```

```
-- Reference: ConferenceDayBooking_ConferenceDays (table:
ConferenceDayBooking)
```

```
ALTER TABLE ConferenceDayBooking ADD CONSTRAINT
ConferenceDayBooking_ConferenceDays FOREIGN KEY
(ConferenceDays_ConferenceDayID) REFERENCES ConferenceDays
(ConferenceDayID)
```

Nazwa kolumny	Typ danych	Inne informacje
ConferenceDayBookingID	INT	PK
ConferenceBooking_ConferenceBookingID	INT	FK
ConferenceDays_ConferenceDayID	INT	FK
BookingDate	DATE	
IsCanceled	BIT	
NumberOfParticipants	INT	POSITIVE AND NO SMALLER THAN NUMBEROFSTUDENTS
NumberOfStudents	INT	POSITIVE

WorkshopBooking

```
CREATE TABLE WorkshopBooking (  
  WorkshopBookingID INT NOT NULL IDENTITY,  
  Workshops_WorkshopID INT NOT NULL,  
  ConferenceDayBooking_ConferenceDayBookingID INT NOT NULL,  
  BookingDate DATE NOT NULL DEFAULT getdate(),  
  NumberOfParticipants INT NOT NULL,  
  CONSTRAINT PositiveWorkshopNumberOfParticipants CHECK  
  (NumberOfParticipants > 0),  
  CONSTRAINT WorkshopBooking_pk PRIMARY KEY (WorkshopBookingID)  
);  
  
-- Reference: WorkshopBooking_ConferenceDayBooking (table: WorkshopBooking)  
ALTER TABLE WorkshopBooking ADD CONSTRAINT  
WorkshopBooking_ConferenceDayBooking FOREIGN KEY  
(ConferenceDayBooking_ConferenceDayBookingID) REFERENCES  
ConferenceDayBooking (ConferenceDayBookingID)  
  
-- Reference: WorkshopBooking_Workshops (table: WorkshopBooking)  
ALTER TABLE WorkshopBooking ADD CONSTRAINT WorkshopBooking_Workshops  
FOREIGN KEY (Workshops_WorkshopID) REFERENCES Workshops (WorkshopID)
```

Nazwa kolumny	Typ danych	Inne informacje
WorkshopBookingID	INT	PK
Workshops_WorkshopID	INT	FK
ConferenceDayBooking_ConferenceDayBookingID	INT	FK
BookingDate	DATE	
NumberOfParticipants	BIT	POSITIVE

Payments

```
CREATE TABLE Payments (  
    PaymentID INT NOT NULL IDENTITY,  
    ConferenceBooking_ConferenceBookingID INT NOT NULL,  
    Amount DECIMAL(9, 2) NOT NULL,  
    PayDate DATE NOT NULL DEFAULT getdate(),  
    CONSTRAINT PositiveValue CHECK (Amount > 0),  
    CONSTRAINT Payments_pk PRIMARY KEY (PaymentID)  
);
```

-- Reference: Payments_ConferenceBooking (table: Payments)

```
ALTER TABLE Payments ADD CONSTRAINT Payments_ConferenceBooking FOREIGN KEY  
(ConferenceBooking_ConferenceBookingID) REFERENCES ConferenceBooking  
(ConferenceBookingID)
```

Nazwa kolumny	Typ danych	Inne informacje
PaymentID	INT	PK
ConferenceBooking_ConferenceBookingID	INT	FK
Amount	DECIMAL(9,2)	POSITIVE
PayDate	DATE	

Participants

```
CREATE TABLE Participants (  
    ParticipantID INT NOT NULL IDENTITY,  
    FirstName VARCHAR(40) NOT NULL,  
    LastName VARCHAR(40) NOT NULL,  
    Email VARCHAR(100) NOT NULL,  
    Street VARCHAR(40) NOT NULL,  
    City VARCHAR(40) NOT NULL,  
    PostalCode VARCHAR(10) NOT NULL,  
    Country VARCHAR(40) NOT NULL,  
    CONSTRAINT Participants_pk PRIMARY KEY (ParticipantID)  
);
```

Nazwa kolumny	Typ danych	Inne informacje
ParticipantID	INT	PK
FirstName	VARCHAR(40)	
LastName	VARCHAR(40)	
Email	VARCHAR(100)	
Street	VARCHAR(40)	
City	VARCHAR(40)	
PostalCode	VARCHAR(10)	
Country	VARCHAR(40)	

DayParticipants

```
CREATE TABLE DayParticipants (  
  DayParticipantID INT NOT NULL IDENTITY,  
  ConferenceDayBooking_ConferenceDayBookingID INT NOT NULL,  
  Participants_ParticipantID INT NOT NULL,  
  StudentID VARCHAR(6) NULL DEFAULT NULL,  
  CONSTRAINT DayParticipants_pk PRIMARY KEY (DayParticipantID)  
);
```

```
-- Reference: DayParticipants_ConferenceDayBooking (table: DayParticipants)  
ALTER TABLE DayParticipants ADD CONSTRAINT  
DayParticipants_ConferenceDayBooking FOREIGN KEY  
(ConferenceDayBooking_ConferenceDayBookingID) REFERENCES  
ConferenceDayBooking (ConferenceDayBookingID)
```

```
-- Reference: Participants_DayParticipants (table: DayParticipants)  
ALTER TABLE DayParticipants ADD CONSTRAINT Participants_DayParticipants  
FOREIGN KEY (Participants_ParticipantID) REFERENCES Participants  
(ParticipantID)
```

Nazwa kolumny	Typ danych	Inne informacje
DayParticipantID	INT	PK
ConferenceDayBooking_ConferenceDayBookingID	INT	FK
Participants_ParticipantID	INT	FK
StudentID	VARCHAR(6)	

WorkshopParticipants

```
CREATE TABLE WorkshopParticipants (  
    WorkshopParticipantID INT NOT NULL IDENTITY,  
    WorkshopBooking_WorkshopBookingID INT NOT NULL,  
    DayParticipants_DayParticipantID INT NOT NULL,  
    CONSTRAINT WorkshopParticipants_pk PRIMARY KEY  
    (WorkshopParticipantID)  
);  
  
-- Reference: DayParticipants_WorkshopParticipants (table:  
WorkshopParticipants)  
ALTER TABLE WorkshopParticipants ADD CONSTRAINT  
DayParticipants_WorkshopParticipants FOREIGN KEY  
(DayParticipants_DayParticipantID) REFERENCES DayParticipants  
(DayParticipantID)  
  
-- Reference: WorkshopParticipants_WorkshopBooking (table:  
WorkshopParticipants)  
ALTER TABLE WorkshopParticipants ADD CONSTRAINT  
WorkshopParticipants_WorkshopBooking FOREIGN KEY  
(WorkshopBooking_WorkshopBookingID) REFERENCES WorkshopBooking  
(WorkshopBookingID)
```

Nazwa kolumny	Typ danych	Inne informacje
WorkshopParticipantID	INT	PK
WorkshopBooking_WorkshopBookingID	INT	FK
DayParticipants_DayParticipantID	INT	FK

Clients

```
CREATE TABLE Clients (  
    ClientID INT NOT NULL IDENTITY,  
    IsCompany BIT NOT NULL,  
    Name VARCHAR(100) NOT NULL,  
    Surname VARCHAR(100) NULL,  
    Email VARCHAR(100) NOT NULL,  
    CONSTRAINT Clients_pk PRIMARY KEY (ClientID)  
);
```

Nazwa kolumny	Typ danych	Inne informacje
ClientID	INT	PK
IsCompany	BIT	
Name	VARCHAR(100)	
Surname	VARCHAR(100)	
Email	VARCHAR(100)	

Indeksy

```
CREATE INDEX Client ON Clients (ClientID ASC)
```

```
CREATE INDEX Client_ConferenceBooking ON ConferenceBooking  
(Clients_ClientID ASC)
```

```
CREATE INDEX ConferenceBook ON ConferenceDayBooking (ConferenceDayBookingID  
ASC)
```

```
CREATE INDEX ConferenceDay ON ConferenceDayBooking  
(ConferenceDays_ConferenceDayID ASC)
```

```
CREATE INDEX NumberOfParticipantsInConferenceDay ON ConferenceDays  
(NumberOfParticipants ASC)
```

```
CREATE INDEX Conference ON ConferenceDays (Conferences_ConferenceID ASC)
```

```
CREATE INDEX Participant ON DayParticipants (DayParticipantID ASC)
```

```
CREATE INDEX ConferenceDayBook ON WorkshopBooking  
(ConferenceDayBooking_ConferenceDayBookingID ASC)
```

```
CREATE INDEX DayParticipant ON WorkshopParticipants  
(DayParticipants_DayParticipantID ASC)
```

```
CREATE INDEX NumberOfParticipantsInWorkshop ON Workshops  
(NumberOfParticipants ASC)
```

```
CREATE INDEX ConferenceDayID ON Workshops (ConferenceDays_ConferenceDayID  
ASC)
```

Widoki

View_MostPopularWorkshops

Wyświetla najpopularniejsze warsztaty

```
CREATE VIEW view_MostPopularWorkshops
AS
SELECT TOP 10 conf.name AS ConferenceName
    ,w.workshopid
    ,w.name
    ,isnull(sum(wb.NumberOfParticipants), 0) AS Popularnosc
FROM Workshops AS w
INNER JOIN conferencedays AS cd ON cd.conferencedayid =
w.ConferenceDays_ConferenceDayID
INNER JOIN conferences AS conf ON conf.conferenceid =
cd.Conferences_ConferenceID
INNER JOIN WorkshopBooking AS wb ON wb.Workshops_WorkshopID = w.WorkshopID
INNER JOIN WorkshopParticipants AS wp ON
wp.WorkshopBooking_WorkshopBookingID = wb.workshopbookingid
GROUP BY w.workshopid
    ,w.name
    ,conf.name
ORDER BY Popularnosc DESC
GO
```

View_MostPopularConferences

Wyświetla najpopularniejsze konferencje

```
CREATE VIEW view_MostPopularConferences
AS
SELECT TOP 10 c.ConferenceID
    , c.Name
    , isnull(sum(cdb.NumberOfParticipants), 0) AS popularity
FROM Conferences AS c
INNER JOIN ConferenceDays AS cd ON cd.Conferences_ConferenceID =
c.ConferenceID
INNER JOIN ConferenceDayBooking AS cdb ON
cdb.ConferenceDays_ConferenceDayID = cd.ConferenceDayID
INNER JOIN DayParticipants AS dp ON
dp.ConferenceDayBooking_ConferenceDayBookingID = cdb.ConferenceDayBookingID
GROUP BY c.ConferenceID
    , c.Name
ORDER BY popularity DESC
GO
```

View_MostPopularConferencesByStudents

Wyświetla najpopularniejsze konferencje wśród studentów

```
CREATE VIEW view_MostPopularConferencesByStudents
AS
SELECT TOP 10 c.ConferenceID
    ,c.Name
    ,isnull(sum(cdb.NumberOfParticipants), 0) AS popularity
FROM Conferences AS c
INNER JOIN ConferenceDays AS cd ON cd.Conferences_ConferenceID =
c.ConferenceID
INNER JOIN ConferenceDayBooking AS cdb ON
cdb.ConferenceDays_ConferenceDayID = cd.ConferenceDayID
INNER JOIN DayParticipants AS dp ON
dp.ConferenceDayBooking_ConferenceDayBookingID = cdb.ConferenceDayBookingID
    AND dp.StudentID IS NOT NULL
GROUP BY c.ConferenceID
    ,c.Name
ORDER BY popularity DESC
GO
```

View_MostPopularWorkshopsByStudents

Wyświetla najpopularniejsze warsztaty wśród studentów

```
CREATE VIEW view_MostPopularWorkshopsByStudents
AS
SELECT TOP 10 conf.name AS ConferenceName
, w.workshopid
, w.name
, isnull(sum(wb.NumberOfParticipants), 0) AS popularnosc
FROM Workshops AS w
INNER JOIN conferencedays AS cd ON cd.conferencedayid =
w.ConferenceDays_ConferenceDayID
INNER JOIN conferences AS conf ON conf.conferenceid =
cd.Conferences_ConferenceID
INNER JOIN WorkshopBooking AS wb ON wb.Workshops_WorkshopID = w.WorkshopID
INNER JOIN WorkshopParticipants AS wp ON
wp.WorkshopBooking_WorkshopBookingID = wb.workshopbookingid
INNER JOIN DayParticipants AS dp ON dp.DayParticipantID =
wp.DayParticipants_DayParticipantID
AND dp.StudentID IS NOT NULL
GROUP BY w.workshopid
, w.name
, conf.name
ORDER BY popularnosc DESC
GO
```

View_MostProfitableConference

Wyświetla najbardziej zyskowe konferencje

```
CREATE VIEW view_MostProfitableConference
AS
SELECT TOP 10 c.ConferenceID
    , c.Name
    , isnull(sum(p.Amount), 0) AS Profit
FROM Conferences AS c
INNER JOIN ConferenceBooking AS cb ON cb.Conferences_ConferenceID =
c.ConferenceID
INNER JOIN Payments AS p ON p.ConferenceBooking_ConferenceBookingID =
cb.ConferenceBookingID
GROUP BY c.ConferenceID
    , c.Name
ORDER BY Profit DESC
GO
```

View_MostProfitableWorkshops

Wyświetla najbardziej zyskowe warsztaty

```
CREATE VIEW view_MostProfitableWorkshops
AS
SELECT TOP 10 c.name AS NazwaKonferencji,
             w.name,
             w.cost
FROM Workshops AS w
INNER JOIN ConferenceDays AS cd ON cd.ConferenceDayID =
w.ConferenceDays_ConferenceDayID
INNER JOIN Conferences AS c ON c.ConferenceID = cd.Conferences_ConferenceID
ORDER BY w.cost DESC
```


View_WorkshopsFreePlaces

Wyświetla informacje o miejscach na poszczególne warsztaty

```
CREATE VIEW view_WorkshopsFreePlaces
AS
SELECT c.name AS ConferenceName
, w.workshopid
, w.name
, w.numberofparticipants AS Miejsca
, isnull(SUM(wb.NumberOfParticipants), 0) AS Zajete
, w.numberofparticipants - isnull(SUM(wb.NumberOfParticipants), 0) AS
WolneMiejsca
FROM workshops AS w
INNER JOIN WorkshopBooking AS wb ON wb.workshops_workshopid = w.workshopid
INNER JOIN WorkshopParticipants AS wp ON
wp.WorkshopBooking_WorkshopBookingID = wb.WorkshopBookingID
INNER JOIN conferencedays AS cd ON cd.conferencedayid =
w.conferencedays_conferencedayid
INNER JOIN conferences AS c ON c.conferenceid = cd.conferences_conferenceid
GROUP BY w.workshopid
, w.name
, w.numberofparticipants
, c.name
GO
```

View_ConferenceFreePlaces

Wyświetla informacje o miejscach na poszczególne konferencje

```
CREATE VIEW view_ConferenceFreePlaces
AS
SELECT c.ConferenceID
      , c.name
      , cd.DATE
      , isnull(sum(cdb.NumberOfParticipants), 0) AS Zajete
      , cd.NumberOfParticipants AS Miejsca
      , cd.NumberOfParticipants - isnull(sum(cdb.NumberOfParticipants), 0) AS
WolneMiejsca
FROM Conferences AS c
INNER JOIN ConferenceDays AS cd ON cd.Conferences_ConferenceID =
c.ConferenceID
INNER JOIN ConferenceDayBooking AS cdb ON
cdb.ConferenceDays_ConferenceDayID = cd.ConferenceDayID
INNER JOIN DayParticipants AS dp ON
dp.ConferenceDayBooking_ConferenceDayBookingID = cdb.ConferenceDayBookingID
GROUP BY c.conferenceid
      , c.Name
      , cd.DATE
      , cd.NumberOfParticipants
GO
```

View_AvailableConferenceDays

Wyświetla informacje o miejscach na konferencje mające wolne miejsca

```
CREATE VIEW view_AvailableConferenceDays
AS
SELECT cdp.*
FROM VIEW_ConferenceFreePlaces cdp
WHERE WolneMiejsca > 0
```

View_AvailableWorkshops

Wyświetla informacje o miejscach na warsztaty mające wolne miejsca

```
CREATE VIEW view_AvailableWorkshops
AS
SELECT w.*
FROM view_WorkshopsFreePlaces w
WHERE WolneMiejsca > 0
```

View_ClientsActivity

Wyświetla informacje o aktywności klientów, ich zamówieniach oraz ich wartości

```
CREATE VIEW view_ClientsActivity
AS
SELECT c.*
, isnull((
    SELECT COUNT(*)
    FROM conferencebooking AS cb
    WHERE cb.clients_clientid = c.ClientID
), 0) AS Bookings
, isnull((
    SELECT SUM(p.amount)
    FROM conferencebooking AS cb
    INNER JOIN payments AS p ON cb.ConferenceBookingID =
p.ConferenceBooking_ConferenceBookingID
    WHERE cb.clients_clientid = c.ClientID
), 0) AS Payments
FROM Clients AS c
GO
```

View_MostProfitableClients

Wyświetla informacje o klientach od których firma zarobiła najwięcej

```
CREATE VIEW view_MostProfitableClients
AS
SELECT TOP 10 c.*
FROM view_ClientsActivity c
ORDER BY Payments
```

Procedury

Dodające

Procedure_AddConferenceDay

Procedura dodaje dzień do konferencji.

```
CREATE PROCEDURE PROCEDURE_AddConferenceDay (  
    @ConferenceID INT,  
    @Date DATE,  
    @NumberOfParticipants INT  
)  
  
AS  
BEGIN  
    INSERT INTO ConferenceDays (  
        Conferences_ConferenceID,  
        DATE,  
        NumberOfParticipants  
    )  
    VALUES (  
        @ConferenceID,  
        @Date,  
        @NumberOfParticipants  
    );  
  
END
```

Procedure_AddWorkshop

Procedura dodaje warsztat do konferencji.

```
CREATE PROCEDURE PROCEDURE_AddWorkshop (  
    @ConferenceDayID INT,  
    @Name VARCHAR(100),  
    @StartTime DATE,  
    @EndTime DATE,  
    @Cost DECIMAL(9, 2),  
    @NumberOfParticipants INT  
)  
  
AS  
BEGIN  
    INSERT INTO Workshops (  
        ConferenceDays_ConferenceDayID,  
        Name,  
        StartTime,  
        EndTime,  
        Cost,  
        NumberOfParticipants  
    )  
    VALUES (  
        @ConferenceDayID,  
        @Name,  
        @StartTime,  
        @EndTime,  
        @Cost,  
        @NumberOfParticipants  
    );  
  
END
```


Procedure_AddConferenceCost

Procedura dodaje koszt konferencji na ustalone dni.

```
CREATE PROCEDURE PROCEDURE_AddConferenceCost (
    @ConferenceID INT,
    @Cost DECIMAL(9, 2),
    @DateFrom DATE,
    @DateTo DATE
)
AS
BEGIN
    INSERT INTO ConferenceCosts (
        Conferences_ConferenceID,
        Cost,
        DateFrom,
        DateTo
    )
    VALUES (
        @ConferenceID,
        @Cost,
        @DateFrom,
        @DateTo
    );
END
```

Procedure_AddConferenceBooking

Procedura dodaje rezerwację klienta na konferencję.

```
CREATE PROCEDURE PROCEDURE_AddConferenceBooking (
    @ConferenceID INT,
    @ClientID INT,
    @BookingDate DATE
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM Conferences
        WHERE @ConferenceID = ConferenceID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono konferencji', 1
    END
    ELSE
    BEGIN
        IF @BookingDate IS NULL
        BEGIN
            INSERT INTO ConferenceBooking (
                Conferences_ConferenceID,
                Clients_ClientID
            )
            VALUES (
                @ConferenceID,
                @ClientID
            );
        END
        ELSE
        BEGIN
            INSERT INTO ConferenceBooking (
                Conferences_ConferenceID,
                BookingDate,
                Clients_ClientID
            )
            VALUES (
                @ConferenceID,
                @BookingDate,
                @ClientID
            );
        END
    END
END
END
```

Procedure_AddConferenceDayBooking

Procedura dodaje rezerwację klienta na określony dzień konferencji.

```
CREATE PROCEDURE PROCEDURE_AddConferenceDayBooking (
    @ConferenceDayID INT,
    @ConferenceBookingID INT,
    @NumberOfParticipants INT,
    @NumberOfStudents INT
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM ConferenceDays
        WHERE @ConferenceDayID = ConferenceDayID
    )
    OR NOT EXISTS (
        SELECT *
        FROM ConferenceBooking
        WHERE @ConferenceBookingID = ConferenceBookingID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono ConferenceDayID lub ConferenceBookingID', 1
    END
    ELSE
    BEGIN
        IF @NumberOfStudents IS NULL
        BEGIN
            INSERT INTO ConferenceDayBooking (
                ConferenceDays_ConferenceDayID,
                ConferenceBooking_ConferenceBookingID,
                NumberOfParticipants
            )
            VALUES (
                @ConferenceDayID,
                @ConferenceBookingID,
                @NumberOfParticipants
            );
        END
        ELSE
        BEGIN
            INSERT INTO ConferenceDayBooking (
                ConferenceDays_ConferenceDayID,
                ConferenceBooking_ConferenceBookingID,
                NumberOfParticipants,
                NumberOfStudents
            )
            VALUES (
                @ConferenceDayID,
                @ConferenceBookingID,
                @NumberOfParticipants,
                @NumberOfStudents
            );
        END
    END
END
END
```

Procedure_AddWorkshopBooking

Procedura dodaje rezerwację klienta na warsztat.

```
CREATE PROCEDURE PROCEDURE_AddWorkshopBooking (
    @WorkshopID INT,
    @ConferenceDayBookingID INT,
    @BookingDate DATE,
    @NumberOfParticipants INT
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM Workshops
        WHERE @WorkshopID = WorkshopID
    )
    OR NOT EXISTS (
        SELECT *
        FROM ConferenceDayBooking
        WHERE @ConferenceDayBookingID = ConferenceDayBookingID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono WorkshopID lub ConferenceDayBookingID', 1
    END
    ELSE
    BEGIN
        IF @BookingDate IS NULL
        BEGIN
            INSERT INTO WorkshopBooking (
                Workshops_WorkshopID,
                ConferenceDayBooking_ConferenceDayBookingID,
                NumberOfParticipants
            )
            VALUES (
                @WorkshopID,
                @ConferenceDayBookingID,
                @NumberOfParticipants
            );
        END
        ELSE
        BEGIN
            INSERT INTO WorkshopBooking (
                Workshops_WorkshopID,
                ConferenceDayBooking_ConferenceDayBookingID,
                BookingDate,
                NumberOfParticipants
            )
            VALUES (
                @WorkshopID,
                @ConferenceDayBookingID,
                @BookingDate,
                @NumberOfParticipants
            );
        END
    END
END
```

Procedure_AddDayParticipant

Procedura dodaje osobę do rezerwacji dnia konferencji.

```
CREATE PROCEDURE PROCEDURE_AddDayParticipant (
    @ConferenceDayBookingID INT,
    @ParticipantID INT,
    @StudentID VARCHAR(6)
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM ConferenceDayBooking
        WHERE @ConferenceDayBookingID = ConferenceDayBookingID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono ConferenceDayBookingID', 1
    END
    ELSE
    BEGIN
        IF NOT EXISTS (
            SELECT *
            FROM Participants
            WHERE @ParticipantID = ParticipantID
        )
        BEGIN
            THROW 50000, 'Nie znaleziono ParticipantID', 1
        END
        ELSE
        BEGIN
            INSERT INTO DayParticipants (
                ConferenceDayBooking_ConferenceDayBookingID,
                Participants_ParticipantID,
                StudentID
            )
            VALUES (
                @ConferenceDayBookingID,
                @ParticipantID,
                @StudentID
            );
        END
    END
END
```

Procedure_AddWorkshopParticipant

Procedura dodaje osobę do rezerwacji warsztatu.

```
CREATE PROCEDURE PROCEDURE_AddWorkshopParticipant (
    @WorkshopBookingID INT,
    @DayParticipantID INT
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM WorkshopBooking
        WHERE @WorkshopBookingID = WorkshopBookingID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono WorkshopBookingID', 1
    END
    ELSE
    BEGIN
        IF NOT EXISTS (
            SELECT *
            FROM Participants
            WHERE @DayParticipantID = ParticipantID
        )
        BEGIN
            THROW 50000, 'Nie znaleziono ParticipantID', 1
        END
        ELSE
        BEGIN
            INSERT INTO WorkshopParticipants (
                WorkshopBooking_WorkshopBookingID,
                DayParticipants_DayParticipantID
            )
            VALUES (
                @WorkshopBookingID,
                @DayParticipantID
            );
        END
    END
END
END
```

Procedure_AddParticipant

Procedura dodaje uczestnika.

```
CREATE PROCEDURE PROCEDURE_AddParticipant (
    @FirstName VARCHAR(40),
    @LastName VARCHAR(40),
    @Email VARCHAR(100),
    @Street VARCHAR(40),
    @City VARCHAR(40),
    @PostalCode VARCHAR(10),
    @Country VARCHAR(40)
)
AS
BEGIN
    INSERT INTO Participants (
        FirstName,
        LastName,
        Email,
        Street,
        City,
        PostalCode,
        County
    )
    VALUES (
        @FirstName,
        @LastName,
        @Email,
        @Street,
        @City,
        @PostalCode,
        @Country
    );
END
```

Procedure_AddClient

Procedura dodaje klienta.

```
CREATE PROCEDURE PROCEDURE_AddClient (  
    @IsCompany BIT,  
    @Name VARCHAR(100),  
    @Surname VARCHAR(100),  
    @Email VARCHAR(100)  
)  
AS  
BEGIN  
    INSERT INTO Clients (  
        IsCompany,  
        Name,  
        Surname,  
        Email  
    )  
VALUES (  
    @IsCompany,  
    @Name,  
    @Surname,  
    @Email  
    );  
END
```


Aktualizujące

Procedure_RemoveConference

Procedura usuwa konferencję.

```
CREATE PROCEDURE PROCEDURE_RemoveConference (@ConferenceID INT)
AS
IF NOT EXISTS (
    SELECT *
    FROM Conferences
    WHERE @ConferenceID = ConferenceID
)
BEGIN
    THROW 50000, 'Nie znaleziono konferencji', 1
END
ELSE
BEGIN
    DELETE
    FROM Conferences
    WHERE Conferences.ConferenceID = @ConferenceID;
END
```

Procedure_UpdateConferenceDetails

Procedura aktualizuje dane konferencji.

```
CREATE PROCEDURE PROCEDURE_UpdateConferenceDetails (
    @ConferenceID INT,
    @Name VARCHAR(100),
    @Place VARCHAR(100),
    @DiscountForStudents FLOAT,
    @Description VARCHAR(200)
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM Conferences
        WHERE @ConferenceID = ConferenceID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono konferencji', 1
    END
    ELSE
    BEGIN
        IF @Name IS NOT NULL
        BEGIN
            UPDATE Conferences
            SET Name = @Name
            WHERE ConferenceID = @ConferenceID;
        END
        ELSE
        BEGIN
            THROW 50000, 'ConferenceID is null', 1
        END

        IF @DiscountForStudents IS NOT NULL
        BEGIN
            UPDATE Conferences
            SET DiscountForStudents = @DiscountForStudents
            WHERE ConferenceID = @ConferenceID;
        END
        ELSE
        BEGIN
            THROW 50000, 'DiscountForStudents is null', 1
        END

        IF @Place IS NOT NULL
        BEGIN
            UPDATE Conferences
            SET Place = @Place
            WHERE ConferenceID = @ConferenceID;
        END
        ELSE
        BEGIN
            THROW 50000, 'Place is null', 1
        END

        IF @Description IS NOT NULL
        BEGIN
            UPDATE Conferences
            SET Description = @Description
            WHERE ConferenceID = @ConferenceID;
        END
    END
END
```

```
ELSE
BEGIN
    THROW 50000, 'Description is null', 1
END
END
END
```

Procedure_UpdateWorkshopDetails

Procedura aktualizuje dane warsztatu.

```
CREATE PROCEDURE PROCEDURE_UpdateWorkshopDetails (
    @WorkshopID INT,
    @Name VARCHAR(100),
    @StartTime DATE,
    @EndTime DATE,
    @Cost DECIMAL(9, 2),
    @NumberOfParticipants INT
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM Workshops
        WHERE @WorkshopID = WorkshopID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono warsztatu', 1
    END
    ELSE
    BEGIN
        IF @Name IS NOT NULL
        BEGIN
            UPDATE Workshops
            SET Name = @Name
            WHERE WorkshopID = @WorkshopID;
        END
        ELSE
        BEGIN
            THROW 50000, 'WorkshopID is null', 1
        END

        IF @StartTime IS NOT NULL
        BEGIN
            UPDATE Workshops
            SET StartTime = @StartTime
            WHERE WorkshopID = @WorkshopID;
        END
        ELSE
        BEGIN
            THROW 50000, 'StartTime is null', 1
        END

        IF @EndTime IS NOT NULL
        BEGIN
            UPDATE Workshops
            SET EndTime = @EndTime
            WHERE WorkshopID = @WorkshopID;
        END
        ELSE
        BEGIN
            THROW 50000, 'EndTime is null', 1
        END

        IF @Cost IS NOT NULL
        BEGIN
            UPDATE Workshops
            SET Cost = @Cost
            WHERE WorkshopID = @WorkshopID;
```

```

END
ELSE
BEGIN
    THROW 50000, 'Cost is null', 1
END

IF @NumberOfParticipants IS NOT NULL
BEGIN
    UPDATE Workshops
    SET NumberOfParticipants = @NumberOfParticipants
    WHERE WorkshopID = @WorkshopID;
END
ELSE
BEGIN
    THROW 50000, 'NumberOfParticipants is null', 1
END
END
END

```

Procedure_CancelConferenceBooking

Procedura anuluje rezerwację klienta na konferencję.

```
CREATE PROCEDURE PROCEDURE_CancelConferenceBooking (@ConferenceBookingID
INT)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM ConferenceBooking
        WHERE @ConferenceBookingID = ConferenceBookingID
    )
    OR NOT (
        SELECT IsCanceled
        FROM ConferenceBooking
        WHERE @ConferenceBookingID = ConferenceBookingID
    ) = 0
    BEGIN
        THROW 50000, 'Nie znaleziono ConferenceBookingID lub rezerwacja została
już wcześniej anulowana', 1
    END
    ELSE
    BEGIN
        UPDATE ConferenceBooking
        SET IsCanceled = 1
        WHERE @ConferenceBookingID = ConferenceBookingID;
    END
    BEGIN
        UPDATE ConferenceDayBooking
        SET IsCancelled = 1
        WHERE @ConferenceBookingID = ConferenceBooking_ConferenceBookingID;
    END
END
```

Procedure_CancelConferenceDayBooking

Procedura anuluje rezerwację klienta na dzień konferencji.

```
CREATE PROCEDURE PROCEDURE_CancelConferenceDayBooking
(@ConferenceDayBookingID INT)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM ConferenceDayBooking
        WHERE @ConferenceDayBookingID = ConferenceDayBookingID
    )
    OR NOT (
        SELECT IsCancelled
        FROM ConferenceDayBooking
        WHERE @ConferenceDayBookingID = ConferenceDayBookingID
    ) = 0
    BEGIN
        THROW 50000, 'Nie znaleziono ConferenceDayBookingID lub rezerwacja
została już wcześniej anulowana', 1
    END
    ELSE
    BEGIN
        UPDATE ConferenceDayBooking
        SET IsCancelled = 1
        WHERE @ConferenceDayBookingID = ConferenceDayBookingID;
    END
END
```

Procedure_UpdateWorkshopNumberOfParticipants

Procedura aktualizuje ilość miejsc na warsztacie.

```
CREATE PROCEDURE PROCEDURE_UpdateWorkshopNumberOfParticipants (
    @WorkshopID INT,
    @NumberOfParticipants INT
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM Workshops
        WHERE @WorkshopID = WorkshopID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono WorkshopID', 1
    END
    ELSE
    BEGIN
        UPDATE Workshops
        SET NumberOfParticipants = @NumberOfParticipants
        WHERE @WorkshopID = WorkshopID;
    END
END
```


Procedure_UpdateConferenceDayNumberOfParticipants

Procedura aktualizuje ilość miejsc w dniu konferencji.

```
CREATE PROCEDURE PROCEDURE_UpdateConferenceDayNumberOfParticipants (
    @ConferenceDayID INT,
    @NumberOfParticipants INT
)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM ConferenceDays
        WHERE @ConferenceDayID = ConferenceDayID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono ConferenceDayID', 1
    END
    ELSE
    BEGIN
        UPDATE ConferenceDays
        SET NumberOfParticipants = @NumberOfParticipants
        WHERE @ConferenceDayID = ConferenceDayID;
    END
END
```

Procedure_CancelConferenceBookingWithoutPayingAfterSevenDays

Procedura anuluje niedokonane płatności w przeciągu 7 dni.

```
CREATE PROCEDURE
PROCEDURE_CancelConferenceBookingWithoutPayingAfterSevenDays
AS
BEGIN
    UPDATE ConferenceBooking
    SET IsCanceled = 1
    FROM (
        SELECT *
        FROM ConferenceBooking
        LEFT JOIN Payments ON ConferenceBookingID =
ConferenceBooking_ConferenceBookingID
        WHERE IsCanceled = 0
        AND PaymentID IS NULL
        AND DATEDIFF(DAY, BookingDate, getdate()) > 7
    ) AS a
    WHERE ConferenceBooking.ConferenceBookingID = a.ConferenceBookingID
END

BEGIN
    UPDATE ConferenceDayBooking
    SET IsCancelled = 1
    FROM (
        SELECT ConferenceDayBookingID AS ID
        FROM ConferenceBooking
        INNER JOIN ConferenceDayBooking ON ConferenceBookingID =
ConferenceBooking_ConferenceBookingID
        WHERE ConferenceBooking.IsCanceled = 1
        AND ConferenceDayBooking.IsCancelled = 0
    ) AS b
    WHERE ConferenceDayBookingID = b.ID
END
```

Wyświetlające

Procedure_ShowConferenceDaysAmountOfParticipants

Procedura wyświetli ilość uczestników określonego dnia konferencji.

```
CREATE PROCEDURE PROCEDURE_ShowConferenceDaysAmountOfParticipants
(@ConferenceID INT)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM Conferences
        WHERE @ConferenceID = ConferenceID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono ConferenceID', 1
    END
    ELSE
    BEGIN
        SELECT ConferenceDayID,
            DATE,
            SUM(ConferenceDayBooking.NumberOfParticipants) AS Participants,
            SUM(NumberOfStudents) AS Students
        FROM ConferenceDays
        INNER JOIN ConferenceDayBooking ON Conferences_ConferenceID =
ConferenceDays_ConferenceDayID
        WHERE Conferences_ConferenceID = @ConferenceID
        AND IsCancelled = 0
        GROUP BY ConferenceDayID,
            DATE
    END
END
```

Procedure_ShowListOfEventsOfConference

Procedura wyświetli ilość wydarzeń określonego dnia konferencji

```
CREATE PROCEDURE PROCEDURE_ShowListOfEventsOfConference (@ConferenceDayID
INT)
AS
BEGIN
    IF NOT EXISTS (
        SELECT *
        FROM ConferenceDays
        WHERE @ConferenceDayID = ConferenceDayID
    )
    BEGIN
        THROW 50000, 'Nie znaleziono ConferenceDayID', 1
    END
    ELSE
    BEGIN
        SELECT WorkshopID,
            Name,
            StartTime,
            EndTime,
            Cost,
            NumberOfParticipants
        FROM Workshops
        WHERE ConferenceDays_ConferenceDayID = @ConferenceDayID
    END
END
```

Funkcje

Function_FreeDayPlaces

Zwraca ilość wolnych miejsc na podany dzień konferencji

```
CREATE FUNCTION FUNCTION_FreeDayPlaces (@ConferenceDayID INT)
RETURNS INT
AS
BEGIN
    RETURN (
        SELECT cd.numberofparticipants - isnull(SUM(cdb.NumberOfParticipants),
0)
        FROM conferencedays AS cd
        LEFT JOIN ConferenceDayBooking AS cdb ON cd.conferencedayid =
cdb.conferencedays_conferencedayid
        WHERE cd.conferencedayid = @ConferenceDayID
        GROUP BY cd.conferencedayid
        ,cd.numberofparticipants
    );
END;
GO
```

Function_FreeWorkshopPlaces

Zwraca ilość wolnych miejsc na podany warsztat

```
CREATE FUNCTION FUNCTION_FreeWorkshopPlaces (@WorkShopID INT)
RETURNS INT
AS
BEGIN
    RETURN (
        SELECT w.numberofparticipants - isnull(SUM(wb.NumberOfParticipants), 0)
        FROM Workshops AS w
        LEFT JOIN WorkshopBooking AS wb ON wb.Workshops_WorkshopID =
w.WorkshopID
        WHERE w.WorkshopID = @WorkShopID
        GROUP BY w.WorkshopID
        ,w.numberofparticipants
    );
END;
GO
```

Function_BookingFreeStudentPlaces

Zwraca ilość wolnych miejsc dla studentów

```
CREATE FUNCTION FUNCTION_BookingFreeStudentPlaces (@ConferenceDayBookingID
INT)
RETURNS INT
AS
BEGIN
    RETURN (
        SELECT cdb.NumberOfStudents -
COUNT(dp.conferencedaybooking_conferencedaybookingid)
        FROM ConferenceDayBooking cdb
        LEFT JOIN dayparticipants dp ON cdb.conferencedaybookingid =
dp.ConferenceDayBooking_ConferenceDayBookingID
        AND dp.studentid IS NOT NULL
        WHERE cdb.ConferenceDayBookingID = @ConferenceDayBookingID
        GROUP BY cdb.ConferenceDayBookingID
        , cdb.NumberOfStudents
    );
END;
GO
```

Function_DaysOfConference

Dla podanej konferencji zwraca dni, w których dana konferencja się odbywa

```
CREATE FUNCTION FUNCTION_DaysOfConference (@ConferenceID INT)
RETURNS @days TABLE (ConferenceDayID INT)
AS
BEGIN
    INSERT INTO @days
    SELECT cd.ConferenceDayID
    FROM ConferenceDays AS cd
    WHERE cd.Conferences_ConferenceID = @ConferenceID

    RETURN;
END
```


Function_ConferenceDayParticipants

Dla danego dnia konferencji zwraca tabelę z danymi uczestników określonego dnia tej konferencji

```
CREATE FUNCTION FUNCTION_ConferenceDayParticipants (@ConferenceDayID INT)
RETURNS @Participants TABLE (
    firstname VARCHAR(40)
    ,lastname VARCHAR(40)
    ,email VARCHAR(100)
    ,county VARCHAR(40)
    ,city VARCHAR(40)
    ,street VARCHAR(40)
    ,postalcode VARCHAR(10)
)
AS
BEGIN
    INSERT INTO @Participants
    SELECT p.firstname
        ,p.lastname
        ,p.email
        ,p.county
        ,p.city
        ,p.Street
        ,p.PostalCode
    FROM Participants AS p
    INNER JOIN DayParticipants AS dp ON dp.Participants_ParticipantID =
p.ParticipantID
    INNER JOIN ConferenceDayBooking AS cdb ON cdb.ConferenceDayBookingID =
dp.ConferenceDayBooking_ConferenceDayBookingID
    WHERE cdb.ConferenceDays_ConferenceDayID = @ConferenceDayID
        AND cdb.IsCancelled = 0

    RETURN
END
GO
```

Function_WorkshopsPerConference

Dla danej konferencji zwraca tabelę z informacją o warsztatach odbywających się w ramach tej konferencji

```
CREATE FUNCTION FUNCTION_WorkshopsPerConference (@ConferenceID INT)
RETURNS @Workshop TABLE (
    workshopID INT
    ,name VARCHAR(100)
    ,starttime DATE
    ,endtime DATE
    ,cost DECIMAL(9, 2)
    ,numberofparticipants INT
)
AS
BEGIN
    INSERT INTO @Workshop
    SELECT w.workshopid
        ,w.name
        ,w.StartTime
        ,w.EndTime
        ,w.cost
        ,w.NumberOfParticipants
    FROM Workshops AS w
    INNER JOIN ConferenceDays AS cd ON cd.ConferenceDayID =
w.ConferenceDays_ConferenceDayID
    WHERE cd.Conferences_ConferenceID = @ConferenceID

    RETURN
END
GO
```

Function_WorkshopDate

Dla ID warsztatu zwraca tabelę z informacjami o danym warsztacie

```
CREATE FUNCTION FUNCTION_WorkshopDate (@WorkshopID INT)
RETURNS @WShop TABLE (
    workshopID INT
    ,name VARCHAR(100)
    ,starttime DATE
    ,endtime DATE
)
AS
BEGIN
    INSERT INTO @WShop
    SELECT workshopid
        ,name
        ,StartTime
        ,EndTime
    FROM Workshops
    WHERE WorkshopID = @WorkshopID

    RETURN
END
GO
```

Function_BookingDaysCost

Zwraca wartość sprzedanych miejsc dla konferencji bez warsztatów

```
CREATE FUNCTION FUNCTION_BookingDaysCost (@ConferenceBookID INT)
RETURNS MONEY
AS
BEGIN
    RETURN (
        SELECT isnull(SUM((cdb.NumberOfParticipants) * cc.cost +
cdb.NumberOfStudents * cc.cost * (1 - c.discountforstudents)), 0)
        FROM ConferenceDayBooking cdb
        INNER JOIN ConferenceBooking cb ON
cdb.ConferenceBooking_ConferenceBookingID = cb.ConferenceBookingID
        INNER JOIN conferences AS c ON cb.conferences_conferenceid =
c.conferenceid
        INNER JOIN conferencecosts cc ON c.conferenceid =
cc.conferences_conferenceid
        WHERE cb.ConferenceBookingID = @ConferenceBookID
        GROUP BY cb.ConferenceBookingID
    );
END;
```

Function_BookingWorkshopCost

Zwraca wartość sprzedanych miejsc dla warsztatów danej konferencji

```
CREATE FUNCTION FUNCTION_BookingWorkshopCost (@ConferenceBookID INT)
RETURNS DECIMAL(9, 2)
AS
BEGIN
    RETURN (
        SELECT isnull(SUM(wb.numberofparticipants * w.cost), 0)
        FROM ConferenceBooking cb
        LEFT JOIN conferencedaybooking cdb ON cb.conferencebookingid =
cdb.ConferenceBooking_ConferenceBookingID
        LEFT JOIN workshopbooking wb ON cdb.conferencedaybookingid =
wb.conferencedaybooking_conferencedaybookingid
        LEFT JOIN workshops w ON wb.workshops_workshopid = w.workshopid
        WHERE cb.conferencebookingid = @ConferenceBookID
        GROUP BY cb.conferencebookingid
    );
END;
GO
```

Function_TotalBookingCost

Zwraca wartość sprzedanych miejsc dla danej konferencji włącznie z warsztatami

```
CREATE FUNCTION FUNCTION_TotalBookingCost (@ConferenceBookID INT)
RETURNS DECIMAL(9, 2)
AS
BEGIN
    RETURN (
        SELECT dbo.FUNCTION_BookingDaysCost(bs.conferencebookingid) +
        dbo.FUNCTION_BookingWorkshopCost(bs.conferencebookingid)
        FROM conferencebooking bs
        WHERE bs.conferencebookingid = @ConferenceBookID
    );
END
```

Function_WorkshopListForParticipant

Zwraca tabelę z informacją o warsztatach w jakich uczestniczyła podana osoba

```
CREATE FUNCTION FUNCTION_WorkshopListForParticipant (@Participant INT)
RETURNS @table TABLE (
    workshopid INT
    ,Name VARCHAR(100)
)
AS
BEGIN
    INSERT INTO @table
    SELECT w.workshopid
        ,w.name
    FROM participants AS p
    INNER JOIN dayparticipants AS dp ON p.participantid =
dp.participants_participantid
    INNER JOIN workshopparticipants AS wp ON dp.dayparticipantid =
wp.dayparticipants_dayparticipantid
    INNER JOIN WorkshopBooking AS wb ON wp.WorkshopBooking_WorkshopBookingID =
wb.workshopbookingid
    INNER JOIN workshops AS w ON wb.workshops_workshopid = w.workshopid
    WHERE p.participantid = @Participant

    RETURN
END
GO
```

Function_ConferencesDaysListForParticipant

Zwraca tabelę z informacją o konferencjach w jakich uczestniczyła podana osoba

```
CREATE FUNCTION FUNCTION_ConferencesDaysListForParticipant (@Participant
INT)
RETURNS @table TABLE (
    name VARCHAR(100)
    ,place VARCHAR(100)
    ,DATE DATE
)
AS
BEGIN
    INSERT INTO @table
    SELECT c.name
        ,c.place
        ,cd.DATE
    FROM participants AS p
    INNER JOIN dayparticipants AS dp ON p.participantid =
dp.participants_participantid
    INNER JOIN ConferenceDayBooking AS cdb ON cdb.ConferenceDayBookingID =
dp.ConferenceDayBooking_ConferenceDayBookingID
    INNER JOIN ConferenceDays AS cd ON cd.ConferenceDayID =
cdb.ConferenceDays_ConferenceDayID
    INNER JOIN Conferences AS c ON c.ConferenceID =
cd.Conferences_ConferenceID
    WHERE p.participantid = @Participant

    RETURN
END
GO
```


Function_ClientsOrdersList

Zwraca listę zamówień klientów

```
CREATE FUNCTION FUNCTION_ClientsOrdersList (@ClientID INT)
RETURNS @table TABLE (
    conferenceid INT
    ,name VARCHAR(100)
    ,place VARCHAR(100)
)
AS
BEGIN
    INSERT INTO @table
    SELECT c.ConferenceID
        ,c.name
        ,c.place
    FROM Conferences AS c
    INNER JOIN ConferenceBooking AS cb ON cb.Conferences_ConferenceID =
c.ConferenceID
    INNER JOIN Clients AS cl ON cl.ClientID = cb.Clients_ClientID
    WHERE cl.ClientID = @ClientID

    RETURN
END
GO
```

Triggery

Trigger_TooFewFreePlacesForDayBooking

Sprawdza, czy jest wystarczająca liczba miejsc w dniu konferencji.

```
CREATE TRIGGER TRIGGER_TooFewFreePlacesForDayBooking ON
ConferenceDayBooking
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        WHERE dbo.FUNCTION_FreeDayPlaces(a.ConferenceDays_ConferenceDayID) < 0
    )
    BEGIN
        THROW 50000, 'Brak wystarczajacej liczby miejsc w dniu konferencji', 1;
    END
END
```

Trigger_TooFewFreePlacesForWorkshopBooking

Sprawdza, czy jest wystarczająca liczba miejsc na warsztacie.

```
CREATE TRIGGER TRIGGER_TooFewFreePlacesForWorkshopBooking ON
WorkshopBooking
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        WHERE dbo.FUNCTION_FreeWorkshopPlaces(a.Workshops_WorkshopID) < 0
    )
    BEGIN
        THROW 50000, 'Brak wystarczającej liczby miejsc w warsztacie', 1;
    END
END
```

Trigger_LessPlacesForDayThanForWorkshop

Blokuje rezerwację na warsztat, jeżeli klient zarezerwował mniej miejsc na dzień niż warsztat.

```
CREATE TRIGGER TRIGGER_LessPlacesForDayThanForWorkshop ON
ConferenceDayBooking
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        INNER JOIN WorkshopBooking AS b ON
b.ConferenceDayBooking_ConferenceDayBookingID = a.ConferenceDayBookingID
        WHERE a.NumberOfParticipants < b.NumberOfParticipants
    )
    BEGIN
        THROW 50000,
        'Klient zarezerwował mniej miejsc na dzień niż na warsztat',
        1
    END
END
```

Trigger_NotEnoughBookedPlacesForDay

Blokuje zapis uczestnika na dzień konferencji, jeżeli wszystkie miejsca od klienta są już zajęte.

```
CREATE TRIGGER TRIGGER_NotEnoughBookedPlacesForDay ON DayParticipants
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        WHERE (
            a.StudentID IS NULL
            AND
            dbo.FUNCTION_FreeDayPlacesForStudents(a.ConferenceDayBooking_ConferenceDayB
ookingID) < 0
        )
        OR (
            a.StudentID IS NULL
            AND
            dbo.FUNCTION_FreeDayPlacesForParticipants(a.ConferenceDayBooking_Conference
DayBookingID) < 0
        )
    )
    BEGIN
        THROW 50000, 'Wszystkie miejsca klienta zostały już zarezerwowane', 1;
    END
END
```

Trigger_NotEnoughBookedPlacesForWorkshop

Blokuje zapis uczestnika na warsztat, jeżeli wszystkie zarezerwowane miejsca są już zajęte.

```
CREATE TRIGGER TRIGGER_NotEnoughBookedPlacesForWorkshop ON
WorkshopParticipants
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        WHERE
            dbo.FUNCTION_FreeWorkshopPlaces(a.WorkshopBooking_WorkshopBookingID) < 0
    )
    BEGIN
        THROW 50000, 'Wszystkie zarezerwowane miejsca są już zajęte', 1;
    END
END
```

Trigger_TooFewPlacesAfterDecreasingDayCapacity

Sprawdza, czy po zmniejszeniu liczby miejsc na dzień konferencji zarezerwowane miejsca mieszczą się w nowym limicie.

```
CREATE TRIGGER TRIGGER_TooFewPlacesAfterDecreasingDayCapacity ON
ConferenceDays
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        LEFT JOIN ConferenceDayBooking AS cdb ON
cdb.ConferenceDays_ConferenceDayID = a.ConferenceDayID
        GROUP BY a.ConferenceDayID,
        a.NumberOfParticipants
        HAVING a.NumberOfParticipants < SUM(cdb.NumberOfParticipants) +
SUM(cdb.NumberOfStudents)
    )
    BEGIN
        THROW 50000, 'Po zmniejszeniu liczby miejsc na dzień konferencji
zarezerwowane miejsca nie mieszczą się w nowym limicie', 1;
    END
END
```

Trigger_TooFewPlacesAfterDecreasingWorkshopCapacity

Sprawdza, czy po zmniejszeniu liczby miejsc na warsztat zarezerwowane miejsca mieszczą się w nowym limicie.

```
CREATE TRIGGER TRIGGER_TooFewPlacesAfterDecreasingWorkshopCapacity ON
Workshops
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        LEFT JOIN WorkshopBooking AS wb ON wb.Workshops_WorkshopID =
a.WorkshopID
        GROUP BY a.WorkshopID,
        a.NumberOfParticipants
        HAVING a.NumberOfParticipants < SUM(wb.NumberOfParticipants)
    )
    BEGIN
        THROW 50000, 'Po zmniejszeniu liczby miejsc na warsztat zarezerwowane
miejsca nie mieszczą się w nowym limicie', 1;
    END
END
```


Trigger_BookingDayInDifferentConference

Sprawdza, czy rezerwowany jest dzień z konferencji odpowiadającej rezerwacji na konferencję.

```
CREATE TRIGGER TRIGGER_BookingDayInDifferentConference ON
ConferenceDayBooking
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        INNER JOIN ConferenceDays AS cd ON cd.ConferenceDayID =
a.ConferenceDays_ConferenceDayID
        INNER JOIN Conferences AS c1 ON c1.ConferenceID =
cd.Conferences_ConferenceID
        INNER JOIN ConferenceBooking AS cb ON cb.ConferenceBookingID =
a.ConferenceBooking_ConferenceBookingID
        INNER JOIN Conferences AS c2 ON c2.ConferenceID =
cb.Conferences_ConferenceID
        WHERE c1.ConferenceID != c2.ConferenceID
    )
    BEGIN
        THROW 50000, 'Klient próbuje przepisać do konferencji rezerwację dnia z
innej konferencji', 1;
    END
END
```

Trigger_BookingDayAlreadyExists

Sprawdza, czy rezerwacja danego dnia konferencji już istnieje.

```
CREATE TRIGGER TRIGGER_BookingDayAlreadyExists ON ConferenceDayBooking
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        LEFT JOIN ConferenceDayBooking AS cbd ON
a.ConferenceBooking_ConferenceBookingID =
cbd.ConferenceBooking_ConferenceBookingID
        AND a.ConferenceDays_ConferenceDayID =
cbd.ConferenceDays_ConferenceDayID
        WHERE a.ConferenceBooking_ConferenceBookingID !=
cbd.ConferenceBooking_ConferenceBookingID
    )
    BEGIN
        THROW 50000, 'Rezerwacja danego dnia konferencji już istnieje', 1;
    END
END
```

Trigger_BookingWorkshopInDifferentDay

Sprawdza, czy rezerwowany jest warsztat z dnia odpowiadającemu rezerwacji.

```
CREATE TRIGGER TRIGGER_BookingWorkshopInDifferentDay ON WorkshopBooking
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        INNER JOIN Workshops AS w ON w.WorkshopID = a.Workshops_WorkshopID
        INNER JOIN ConferenceDays AS cd1 ON cd1.ConferenceDayID =
w.ConferenceDays_ConferenceDayID
        INNER JOIN ConferenceDayBooking AS cdb ON cdb.ConferenceDayBookingID =
a.ConferenceDayBooking_ConferenceDayBookingID
        INNER JOIN ConferenceDays AS cd2 ON cd2.ConferenceDayID =
cdb.ConferenceDays_ConferenceDayID
        WHERE cd1.Conferences_ConferenceID != cd2.Conferences_ConferenceID
    )
    BEGIN
        THROW 50000, 'Klient próbuje zapisać się do warsztatu z innego dnia niż
jego rezerwacja', 1;
    END
END
```

Trigger_ArePriceThresholdsMonotonous

Sprawdza, czy progi cenowe konferencji są ułożone w porządku rosnącym w stosunku do czasu pozostałego do konferencji.

```
CREATE TRIGGER TRIGGER_ArePriceThresholdsMonotonous ON ConferenceCosts
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Cost DECIMAL(9, 2) = (
        SELECT a.Cost
        FROM inserted AS a
    );

    IF EXISTS (
        SELECT *
        FROM inserted AS a
        LEFT JOIN ConferenceCosts AS cc ON a.Conferences_ConferenceID =
cc.Conferences_ConferenceID
        WHERE (
            (
                cc.DateFrom < a.DateFrom
                AND a.DateFrom < cc.dateto
            )
            OR (
                a.DateFrom < cc.DateFrom
                AND cc.DateTo < a.DateTo
            )
            OR (
                cc.DateFrom >= a.DateFrom
                AND a.DateTo >= cc.DateTo
            )
            OR (
                a.DateFrom >= cc.DateFrom
                AND cc.DateTo >= a.DateTo
            )
        )
        AND cc.ConferenceCostID != a.ConferenceCostID
    )
    BEGIN
        THROW 50000,
        'Koszt pokrywa się z istniejącymi kosztami',
        1
    END
    ELSE
    BEGIN
        DECLARE @PreviousCost DECIMAL(9, 2) = (
            SELECT TOP 1 a.Cost
            FROM inserted AS a
            INNER JOIN ConferenceCosts AS cc ON cc.ConferenceCostID =
a.ConferenceCostID
            WHERE cc.Conferences_ConferenceID = a.Conferences_ConferenceID
            AND cc.DateTo < a.DateFrom
            ORDER BY cc.DateFrom DESC
        )
        DECLARE @NextCost DECIMAL(9, 2) = (
            SELECT TOP 1 a.Cost
            FROM inserted AS a
```

```

        INNER JOIN ConferenceCosts AS cc ON cc.ConferenceCostID =
a.ConferenceCostID
        WHERE cc.Conferences_ConferenceID = a.Conferences_ConferenceID
        AND cc.DateFrom > a.DateTo
        ORDER BY cc.DateFrom
    )

    IF (
        (
            @PreviousCost IS NOT NULL
            AND @PreviousCost >= @Cost
        )
        OR (
            @NextCost IS NOT NULL
            AND @NextCost <= @Cost
        )
    )
    BEGIN
        THROW 50000, 'Cena nie jest w poprawnej kolejności z poprzednimi
(PreviousCost = %, NextCost = %.,
        @PreviousCost,
        @NextCost', 1;
    END
END
END

```