Podstawy baz danych- projekt i implementacja systemu bazodanowego.

Adam Bera

Łukasz Kowalski

System zarządzania konferencjami

1.Opis problemu

Projekt dotyczy systemu wspomagania działalności firmy organizującej konferencję.

2.Spis aktorów i ich możliwości w systemie.

A. Organizator
- dodawanie i edycja konferencji,
- dodawanie i edycja warsztatów,
- generowanie raportów,
- ustalenie zniżki dla studentów,
- dostęp do informacji o klientach
- dodanie informacji o zatwierdzonych płatnościach do systemu,
- kontakt z firmą rejestrującą się w sprawie wprowadzenia dodatkowych danych,
- dodanie danych osobowych klientów do systemu

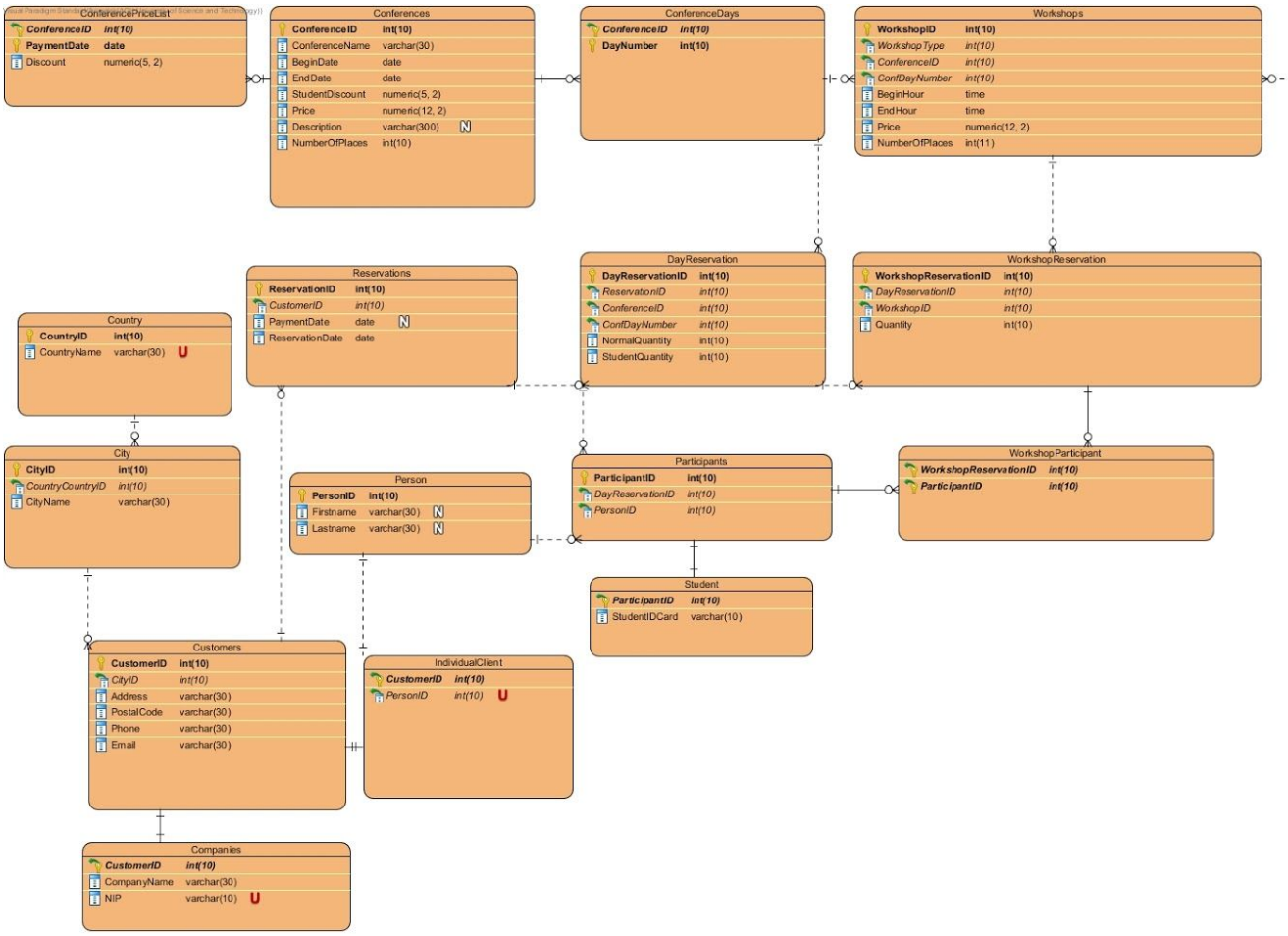
B. Administrator
- posiada wszystkie uprawnienia bazy, zajmuje się jej utrzymaniem i ewentualnym wdrażaniem nowych funkcjonalności,

D. Klient indywidualny
- wyświetlanie informacji na temat dostępnych konferencji i warsztatów,
- dodanie swoich danych osobowych,
- rejestracja indywidualna na konferencje,
- anulowanie rejestracji

E. Klient - firma
- wyświetlanie informacji na temat dostępnych konferencji i warsztatów,
- rezerwacja dowolnej ilości miejsc na konferencje,
- dodanie danych osobowych pracowników firmy do systemu,
- anulowanie rejestracji

F. Uczestnik konferencji
- wyświetlanie informacji o warsztatach, na które jest zapisany
- wyświetlanie informacji o konferencji, w której uczestnicy

3. Schemat bazy danych



*Uwaga do schematu- w rzeczywistości kolumny takie jak Price, gdzie typem jest numeric(5,2) mają typ money.

4. Create Tables

Tabela City

Tabela przechowująca miasta i służąca jako słownik miast dla tabeli Customers.
Posiada klucz główny **CityID**, nazwe miasta **CityName** oraz klucz obcy **CountryCountryID.**

```
CREATE TABLE City (
 CityID           int IDENTITY NOT NULL ,
 CountryCountryID int NOT NULL,
 CityName         varchar(30) NOT NULL,
 PRIMARY KEY (CityID));
```

Tabela Companies

Tabela przechowująca dane klienta firmowego.
Posiada klucz główny **CustomerID**, nazwe firmy **CompanyName,** numer **NIP.**

```
CREATE TABLE Companies (
 CustomerID  int NOT NULL,
 CompanyName varchar(30) NOT NULL,
 NIP varchar(10) NOT NULL UNIQUE,
 PRIMARY KEY (CustomerID));
```

Tabela ConferenceDays

Tabela łącząca tabele **Conferences** z **DayReservation**
Posiada klucz główny składający się z **ConferenceID** i **DayNumber. ConferenceID** jest też kluczem obcym

```
CREATE TABLE ConferenceDays (
 ConferenceID int NOT NULL,
 DayNumber    int NOT NULL,
 PRIMARY KEY (ConferenceID,
 DayNumber));
```

Tabela ConferencePriceList

Tabela przechowująca cene, będąca słownikiem tabeli **Conferences**.
Posiada klucz główny składający się z klucza obcego **ConferenceID** oraz z **PaymentDate.**
Zawiera rekordy ze zniżką **Discount**, która obowiązuje przed danym **PaymentDate.**

```
CREATE TABLE ConferencePriceList (
 ConferenceID int NOT NULL,
 PaymentDate  date NOT NULL,
 Discount     numeric(5, 2)  NULL default 0,
```

```
 PRIMARY KEY (ConferenceID,
 PaymentDate));
```

Tabela Conferences

Tabela przechowująca dane konferencji.
Posiada klucz główny **ConferenceID**, nazwe konferencji **ConferenceName,** date rozpoczęcia **BeginDate**, date końca **EndDate**, zniżke studencką **StudentDiscount,** cene konferencji **Price**, opis **Description** oraz maksymalną liczbe miejsc **NumberOfPlaces.**

```
CREATE TABLE Conferences (
 ConferenceID    int IDENTITY  NOT NULL,
 ConferenceName  varchar(30) NOT NULL,
 BeginDate       date NOT NULL,
 EndDate         date NOT NULL,
 StudentDiscount numeric(5, 2)  NULL default 0,
 Price           money NOT NULL,
 Description     varchar(255),
 NumberOfPlaces  int NOT NULL,
 PRIMARY KEY (ConferenceID));
```

Tabela Country

Tabela przechowująca Państwa, będąca słownikiem dla **City**
Posiada klucz główny **CountryID** i nazwe kraju **CountryName.**

```
CREATE TABLE Country (
 CountryID   int IDENTITY (1,1) NOT NULL ,
 CountryName varchar(30) NOT NULL UNIQUE,
 PRIMARY KEY (CountryID));
```

Tabela Customers

Tabela przechowująca dane klienta.
Posiada klucz główny **CustomerID**, klucz obcy **CityID**, adres **Address**, kod pocztowy **PostalCode**, numer telefonu **Phone**, e-mail **Email**.

```
CREATE TABLE Customers (
 CustomerID int IDENTITY  NOT NULL ,
 CityID     int NOT NULL,
 Address    varchar(30) NOT NULL,
 PostalCode varchar(30) NOT NULL,
 Phone      varchar(30) NOT NULL,
 Email      varchar(30) NOT NULL,
 PRIMARY KEY (CustomerID));
```

Tabela DayReservation

Tabela przechowująca dane na temat zarezerwowanego dnia konferencji.
Posiada klucz główny **DayReservationID**, klucze obce **ReservationID, ConferenceID, ConfDayNumber**, ilość biletów normalnych **NormalQuantity**, ilość biletów studenckich **StudentQuantity**.

```
CREATE TABLE DayReservation (
 DayReservationID int IDENTITY  NOT NULL ,
 ReservationID    int NOT NULL,
 ConferenceID     int NOT NULL,
 ConfDayNumber    int NOT NULL,
 NormalQuantity   int NOT NULL,
 StudentQuantity  int NOT NULL,
 PRIMARY KEY (DayReservationID));
```

Tabela IndividualClient

Tabela łącząca tebele **Person** z tabelą **Customers** dla klientów indywidualnych.
Posiada klucz główny **CustomerID** oraz klucz obcy **PersonID.**

```
CREATE TABLE IndividualClient (
 CustomerID int NOT NULL,
 PersonID   int NOT NULL UNIQUE,
 PRIMARY KEY (CustomerID));
```

Tabela Participants

Tabela przechowująca uczestników konferencji.
Posiada klucz główny **ParticipantID**, klucz obcy dla dnia rezerwacji **DayReservationID** oraz klucz obcy dla danych imiennych **PersonID.**

```
CREATE TABLE Participants (
 ParticipantID    int IDENTITY  NOT NULL ,
 DayReservationID int NOT NULL,
 PersonID         int NOT NULL,
 PRIMARY KEY (ParticipantID));
```

Tabela Person

Tabela przechowująca dane imienne dla uczestnika.
Posiada klucz główny **PersonID**, imię uczestnika **Firstname** oraz nazwisko **Lastname**.

```
CREATE TABLE Person (
 PersonID  int IDENTITY  NOT NULL ,
 Firstname varchar(30),
 Lastname  varchar(30),
 PRIMARY KEY (PersonID));
```

Tabela Reservations

Tabela przechowująca dane rezerwacji.
Posiada klucz główny **ReservationID,** klucz obcy **CustomerID**, date zapłaty za bilety **PaymentDate** oraz date rezerwacji **ReservationDate**.

```
CREATE TABLE Reservations (
 ReservationID   int IDENTITY  NOT NULL ,
 CustomerID      int NOT NULL,
 PaymentDate     date,
 ReservationDate date NOT NULL,
 PRIMARY KEY (ReservationID));
```

Tabela Student

Tabela będąca słownikiem dla uczestnika.
Posiada klucz główny **ParticipantID** oraz numer identyfikacyjny studenta **StudentIDCard**.

```
CREATE TABLE Student (
 ParticipantID int NOT NULL,
 StudentIDCard varchar(10) NOT NULL,
 PRIMARY KEY (ParticipantID));
```

Tabela WorkshopDict

Tabela szczegółowe dane dla warsztatów, będąca słownikiem dla tabeli **Workshops**.
Posiada klucz główny **WorkshopTypeID,** nazwę warsztatu **WorkshopName**, opis warsztatu **Description**, cene biletu za warsztat **Price** oraz maksymalną ilość miejsc **NumberOfPlaces**.

```
CREATE TABLE WorkshopDict (
 WorkshopTypeID int IDENTITY  NOT NULL ,
 WorkshopName   varchar(30) NOT NULL,
 Description    varchar(255),
 Price          money  NULL default 0,
 NumberOfPlaces int NOT NULL,
 PRIMARY KEY (WorkshopTypeID));
```

Tabela WorkshopParticipant

Tabela łącząca tabele **WorkshopReservation** oraz **Participants**.
Posiada klucz główny składający się z dwóch kluczów obcych **WorkshopReservationID** oraz **ParticipantID**.

```
CREATE TABLE WorkshopParticipant (
 WorkshopReservationID int NOT NULL,
 ParticipantID         int NOT NULL,
 PRIMARY KEY (WorkshopReservationID,
 ParticipantID));
```

Tabela WorkshopReservation

Tabela przechowująca dane na temat rezerwacji warsztatów.
Posiada klucz główny **WorkshopReservationID**, klucz obcy **DayReservationID**, klucz obcy **WorkshopID**, oraz ilość zarezerwowanych miejsc **Quantity**.

```
CREATE TABLE WorkshopReservation (
 WorkshopReservationID int IDENTITY  NOT NULL ,
 DayReservationID      int NOT NULL,
 WorkshopID            int NOT NULL,
 Quantity              int NOT NULL,
 PRIMARY KEY (WorkshopReservationID));
```

Tabela Workshops

Tabela przechowująca szczegółowe dane o warsztacie.
Posiada klucz główny **WorkshopID**, klucze obce **WorkshopType, ConferenceID, ConfDayNumber**, godzine rozpoczęcia **BeginHour** oraz zakończenia **EndHour**, cene za warsztat **Price** i ilość miejsc **NumberOfPlaces**.

```
CREATE TABLE Workshops (
 WorkshopID     int IDENTITY NOT NULL ,
 WorkshopType   int NOT NULL,
 ConferenceID   int NOT NULL,
 ConfDayNumber  int NOT NULL,
 BeginHour      time NOT NULL,
 EndHour        time NOT NULL,
 Price          money  NULL default 0,
 NumberOfPlaces int NOT NULL,
 PRIMARY KEY (WorkshopID));
```

5.Foreign Keys - klucze obce

Klucz obcy między tabelami **IndividualClient** oraz **Customers**.

```
ALTER TABLE IndividualClient ADD CONSTRAINT FKIndividual1
FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID) ;
```

Klucz obcy między tabelami **IndividualaClient** oraz **Person**.

```
ALTER TABLE IndividualClient ADD CONSTRAINT FKIndividual2
FOREIGN KEY (PersonID) REFERENCES Person (PersonID);
```

Klucz obcy między tabelami **Student** oraz **Participants**

```
ALTER TABLE Student ADD CONSTRAINT FKStudent
FOREIGN KEY (ParticipantID) REFERENCES Participants (ParticipantID)
ON DELETE CASCADE;
```

Klucz obcy między tabelami **WorkshopParticipant** oraz **WorkshopReservation**.

```
ALTER TABLE WorkshopParticipant ADD CONSTRAINT FKWorkshopPaWorksh
FOREIGN KEY (WorkshopReservationID) REFERENCES WorkshopReservation
(WorkshopReservationID) ON DELETE CASCADE;
```

Klucz obcy między tabelami **WorkshopParticipant** oraz **Participants**.

```
ALTER TABLE WorkshopParticipant ADD CONSTRAINT FKWorkshopPaPart
FOREIGN KEY (ParticipantID) REFERENCES Participants (ParticipantID)
ON DELETE CASCADE;
```

Klucz obcy między tabelami **Participants** oraz **DayReservation**.

```
ALTER TABLE Participants ADD CONSTRAINT FKParticipantDay
FOREIGN KEY (DayReservationID) REFERENCES DayReservation
(DayReservationID) ON DELETE CASCADE;
```

Klucz obcy między tabelami **Participants** oraz **Person**.

```
ALTER TABLE Participants ADD CONSTRAINT FKParticipantPerson
FOREIGN KEY (PersonID) REFERENCES Person (PersonID) ;
```

Klucz obcy między tabelami **WorkshopReservation** oraz **Workshops**.

```
ALTER TABLE WorkshopReservation ADD CONSTRAINT FKWorkshopRe1
```

```
FOREIGN KEY (WorkshopID) REFERENCES Workshops (WorkshopID) ON DELETE
CASCADE;
```

Klucz obcy między tabelami **DayReservation** oraz **ConferenceDays**.

```
ALTER TABLE DayReservation ADD CONSTRAINT FKDayReserva
FOREIGN KEY (ConferenceID, ConfDayNumber) REFERENCES ConferenceDays
(ConferenceID, DayNumber) ON DELETE CASCADE;
```

Klucz obcy między tabelami **WorkshopReservation** oraz **DayReservation**.

```
ALTER TABLE WorkshopReservation ADD CONSTRAINT FKWorkshopRe2
FOREIGN KEY (DayReservationID) REFERENCES DayReservation
(DayReservationID);
```

Klucz obcy między tabelami **DayReservation** a **Resevations.**

```
ALTER TABLE DayReservation ADD CONSTRAINT FKDayReserva1
FOREIGN KEY (ReservationID) REFERENCES Reservations (ReservationID)
ON DELETE CASCADE;
```

Klucz obcy między tabelami **Reservations** a **Customers.**

```
ALTER TABLE Reservations ADD CONSTRAINT FKReservatio
FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID);
```

Klucz obcy między tabelami **Customers** a **City.**

```
ALTER TABLE Customers ADD CONSTRAINT FKCustomers
FOREIGN KEY (CityID) REFERENCES City (CityID);
```

Klucz obcy między tabelami **City** a **Country.**

```
ALTER TABLE City ADD CONSTRAINT FKCity
FOREIGN KEY (CountryCountryID) REFERENCES Country (CountryID);
```

Klucz obcy między tabelami **Workshops** a **WorkshopDict.**

```
ALTER TABLE Workshops ADD CONSTRAINT FKWorkshops1
FOREIGN KEY (WorkshopType) REFERENCES WorkshopDict (WorkshopTypeID);
```

Klucz obcy między tabelami **Companies** a **Customers.**

```
ALTER TABLE Companies ADD CONSTRAINT FKCompanies1
FOREIGN KEY (CustomerID) REFERENCES Customers (CustomerID);
```

Klucz obcy między tabelami **ConferencePriceList** a **Conferences.**

```
ALTER TABLE ConferencePriceList ADD CONSTRAINT FKConference1
FOREIGN KEY (ConferenceID) REFERENCES Conferences (ConferenceID) ON
DELETE CASCADE;
```

Klucz obcy między tabelami **Workshops** a **ConferenceDays.**

```
ALTER TABLE Workshops ADD CONSTRAINT FKWorkshops2
FOREIGN KEY (ConferenceID, ConfDayNumber) REFERENCES ConferenceDays
(ConferenceID, DayNumber);
```

Klucz obcy między **ConferenceDays** a **Conferences**

```
ALTER TABLE ConferenceDays ADD CONSTRAINT FKConference2
FOREIGN KEY (ConferenceID) REFERENCES Conferences (ConferenceID);
```

6. Constraints- ograniczenia default i check

Sprawdzenie czy NIP jest liczbą i czy ma długość 10.

```
alter table Companies
add constraint NIPnumeric
 check ( (isnumeric(NIP)=1) and (len(NIP)=10) )
```

Numer dnia musi być dodatni.

```
alter table ConferenceDays
add constraint positiveDayNumb
 check (DayNumber >0)
```

Zniżka musi zawierać się między 0, a 1.

```
alter table ConferencePriceList
add constraint discountBetween
check (Discount <=1 and Discount >=0)
```

Data rozpoczęcia musi być nie większa od daty zakończenia.

```
alter table Conferences
add constraint dateCorrectness
check (BeginDate <= EndDate)
```

Zniżka studencka musi zawierać się między 0, a 1.

```
alter table Conferences
add constraint studentDiscountBetween
check(StudentDiscount <=1 and StudentDiscount >=0)
```

Ilość miejsc musi być dodatnia.

```
alter table Conferences
add constraint numberOfPlacesPositive
check (NumberOfPlaces > 0)
```

E-mail musi zawierać w sobie znak @ oraz . po iluś znakach po @.

```
alter table Customers
add constraint customerEmail
check (Email like '%@%.%' )
```

E mail- musi być unikalny.

```
alter table Customers
add unique(Email)
```

NIP nie może się powtarzać.

```
alter table Companies
add unique(NIP)
```

Numer telefonu nie może się powtarzać.

```
alter table Customers
add unique(Phone)
```

Numer telefonu musi składać się z dziewięciu cyfr z zakresu 0-9.

```
alter table Customers
add constraint customerPhone
check (Phone like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
```

Numer dnia konferencji musi być dodatni.

```
alter table DayReservation
add constraint reservationPositiveDayNumber
check(ConfDayNumber > 0)
```

Liczba biletów normalnych nie może być ujemna.

```
alter table DayReservation
add constraint normalQuantityPositive
check ( NormalQuantity >=0)
```

Liczba biletów studenckich nie może być ujemna.

```
alter table DayReservation
add constraint studentQuantityPositive
check ( StudentQuantity >=0)
```

Data zapłaty musi być późniejsza od daty rezerwacji.

```
alter table Reservations
add constraint paymentReservationDateCorrectness
check ( PaymentDate >= ReservationDate)
```

Ilość miejsc nie może być ujemna.

```
alter table WorkshopDict
add constraint workshopPlacesPositive
check ( NumberOfPlaces >0 )
```

Ilość biletów nie może być ujemna.

```
alter table WorkshopReservation
add constraint workshopReservationQuantityPositive
check ( Quantity >0 )
```

Ilość miejsc nie może być ujemna.

```
alter table Workshops
add constraint workshopInstancePlacesPositive
check ( NumberOfPlaces >0 )
```

7.Widoki

Nadchodzące konferencje.

```
CREATE VIEW [dbo].[upcomingConferences]
AS
SELECT        ConferenceID, ConferenceName, Description, BeginDate,
EndDate, Price, NumberOfPlaces, StudentDiscount
FROM            dbo.Conferences
WHERE         (BeginDate > GETDATE())
```

Nadchodzące warsztaty.

```
CREATE VIEW [dbo].[upcomingWorkshops]
AS
SELECT        dbo.Workshops.WorkshopID,
dbo.WorkshopDict.WorkshopName, dbo.WorkshopDict.Description,
dbo.Conferences.ConferenceName, dbo.Workshops.ConfDayNumber,
dbo.Workshops.BeginHour, dbo.Workshops.EndHour,
                         dbo.Workshops.Price,
dbo.Workshops.NumberOfPlaces, dbo.Workshops.NumberOfPlaces -
SUM(dbo.WorkshopReservation.Quantity) AS [Places Left]
FROM            dbo.ConferenceDays INNER JOIN
                         dbo.Conferences ON
dbo.ConferenceDays.ConferenceID = dbo.Conferences.ConferenceID
INNER JOIN
                         dbo.Workshops ON
dbo.ConferenceDays.ConferenceID = dbo.Workshops.ConferenceID AND
dbo.ConferenceDays.DayNumber = dbo.Workshops.ConfDayNumber INNER
JOIN
                         dbo.WorkshopDict ON
dbo.Workshops.WorkshopType = dbo.WorkshopDict.WorkshopTypeID INNER
JOIN
                         dbo.WorkshopReservation ON
dbo.Workshops.WorkshopID = dbo.WorkshopReservation.WorkshopID
WHERE         (dbo.Conferences.BeginDate > GETDATE())
GROUP BY dbo.Workshops.WorkshopID, dbo.WorkshopDict.WorkshopName,
dbo.WorkshopDict.Description, dbo.Conferences.ConferenceName,
dbo.Workshops.ConfDayNumber, dbo.Workshops.BeginHour,
dbo.Workshops.EndHour,
                         dbo.Workshops.Price,
dbo.Workshops.NumberOfPlaces
```

Progi cenowe konferencji

```
CREATE VIEW [dbo].[conferencePrices]
AS
SELECT        TOP (100) PERCENT dbo.Conferences.ConferenceID,
dbo.Conferences.ConferenceName, dbo.ConferencePriceList.PaymentDate
AS [Pay Before], dbo.Conferences.Price * (1 -
dbo.ConferencePriceList.Discount) AS Price
FROM            dbo.ConferencePriceList INNER JOIN
                         dbo.Conferences ON
dbo.ConferencePriceList.ConferenceID =
dbo.Conferences.ConferenceID
ORDER BY dbo.Conferences.ConferenceID
```

Najbardziej popularne warsztaty.

```
CREATE VIEW [dbo].[mostPopularWorkshops]
AS
SELECT        TOP (100) PERCENT dbo.WorkshopDict.WorkshopTypeID,
dbo.WorkshopDict.WorkshopName, dbo.WorkshopDict.Description,
SUM(dbo.WorkshopReservation.Quantity) AS PlacesSum
FROM            dbo.WorkshopDict INNER JOIN
                         dbo.Workshops ON
dbo.WorkshopDict.WorkshopTypeID = dbo.Workshops.WorkshopType INNER
JOIN
                         dbo.WorkshopReservation ON
dbo.Workshops.WorkshopID = dbo.WorkshopReservation.WorkshopID
GROUP BY dbo.WorkshopDict.WorkshopTypeID,
dbo.WorkshopDict.WorkshopName, dbo.WorkshopDict.Description
ORDER BY PlacesSum DESC
```

Najbardziej popularne konferencje.

```
CREATE VIEW [dbo].[mostPopularConferences]
AS
SELECT        TOP (100) PERCENT dbo.Conferences.ConferenceID,
dbo.Conferences.ConferenceName, dbo.Conferences.Description,
SUM(dbo.DayReservation.NormalQuantity) +
SUM(dbo.DayReservation.StudentQuantity)
                          AS PlacesSum
FROM            dbo.ConferenceDays INNER JOIN
                         dbo.Conferences ON
dbo.ConferenceDays.ConferenceID = dbo.Conferences.ConferenceID
INNER JOIN
                         dbo.DayReservation ON
dbo.ConferenceDays.ConferenceID = dbo.DayReservation.ConferenceID
AND dbo.ConferenceDays.DayNumber = dbo.DayReservation.ConfDayNumber
GROUP BY dbo.Conferences.ConferenceID,
dbo.Conferences.ConferenceName, dbo.Conferences.Description
```

```
ORDER BY PlacesSum DESC
```

Najlepsi klienci firmowi.

```
CREATE VIEW [dbo].[topCompanyClients]
AS
SELECT        TOP (100) PERCENT dbo.Companies.CustomerID,
dbo.Companies.CompanyName, SUM(dbo.DayReservation.NormalQuantity) +
SUM(dbo.DayReservation.StudentQuantity) AS [Reserved And Paid
Places]
FROM            dbo.Companies INNER JOIN
                         dbo.Customers ON dbo.Companies.CustomerID =
dbo.Customers.CustomerID INNER JOIN
                         dbo.Reservations ON
dbo.Customers.CustomerID = dbo.Reservations.CustomerID INNER JOIN
                         dbo.DayReservation ON
dbo.Reservations.ReservationID = dbo.DayReservation.ReservationID
WHERE        (dbo.Reservations.PaymentDate IS NOT NULL)
GROUP BY dbo.Companies.CustomerID, dbo.Companies.CompanyName
ORDER BY [Reserved And Paid Places] DESC
```

Najlepsi klienci indywidualni.

```
CREATE VIEW [dbo].[topIndividualClients]
AS
SELECT        TOP (100) PERCENT dbo.Customers.CustomerID,
dbo.Person.Firstname + ' ' + dbo.Person.Lastname AS [Client Name],
COUNT(dbo.Reservations.ReservationID) AS [Reserved And Paid Places]
FROM            dbo.Customers INNER JOIN
                         dbo.IndividualClient ON
dbo.Customers.CustomerID = dbo.IndividualClient.CustomerID INNER
JOIN
                         dbo.Person ON dbo.IndividualClient.PersonID
= dbo.Person.PersonID INNER JOIN
                         dbo.Reservations ON
dbo.Customers.CustomerID = dbo.Reservations.CustomerID
WHERE        (dbo.Reservations.PaymentDate IS NOT NULL)
GROUP BY dbo.Customers.CustomerID, dbo.Person.Firstname + ' ' +
dbo.Person.Lastname
ORDER BY [Reserved And Paid Places] DESC
```

Nieopłacone rezerwacje firm.

```
CREATE VIEW [dbo].[unpaidCompanyReservations]
AS
```

```
SELECT        dbo.Companies.CustomerID, dbo.Companies.CompanyName,
dbo.Reservations.ReservationID, dbo.Reservations.ReservationDate,
dbo.Customers.Phone, dbo.Customers.Email
FROM            dbo.Companies INNER JOIN
                         dbo.Customers ON dbo.Companies.CustomerID =
dbo.Customers.CustomerID INNER JOIN
                         dbo.Reservations ON
dbo.Customers.CustomerID = dbo.Reservations.CustomerID
WHERE        (dbo.Reservations.PaymentDate IS NULL)
```

Nieopłacone rezerwacje indywidualne.

```
CREATE VIEW [dbo].[unpaidIndividualReservations]
AS
SELECT        dbo.Customers.CustomerID, dbo.Person.Firstname + ' '
+ dbo.Person.Lastname AS ClientName,
dbo.Reservations.ReservationID, dbo.Reservations.ReservationDate,
dbo.Customers.Phone, dbo.Customers.Email
FROM            dbo.Customers INNER JOIN
                         dbo.IndividualClient ON
dbo.Customers.CustomerID = dbo.IndividualClient.CustomerID INNER
JOIN
                         dbo.Person ON dbo.IndividualClient.PersonID
= dbo.Person.PersonID INNER JOIN
                         dbo.Reservations ON
dbo.Customers.CustomerID = dbo.Reservations.CustomerID
WHERE        (dbo.Reservations.PaymentDate IS NULL)
```

Niewypełnione rezerwacje (brak imion i nazwisk uczestnikow) firmowe.

```
CREATE VIEW [dbo].[unfilledCompanyReservations]
AS
SELECT        dbo.Reservations.ReservationID,
dbo.Customers.CustomerID, dbo.Companies.CompanyName,
dbo.Customers.Phone, dbo.Customers.Email
FROM            dbo.Reservations INNER JOIN
                         dbo.Customers ON
dbo.Reservations.CustomerID = dbo.Customers.CustomerID INNER JOIN
                         dbo.Companies ON dbo.Customers.CustomerID =
dbo.Companies.CustomerID
WHERE        (dbo.Reservations.ReservationID IN
                             (SELECT
Reservations_1.ReservationID
                               FROM            dbo.Reservations AS
Reservations_1 INNER JOIN

dbo.DayReservation AS DayReservation_1 ON
```

```
Reservations_1.ReservationID = DayReservation_1.ReservationID INNER
JOIN

dbo.Participants AS Participants_1 ON
DayReservation_1.DayReservationID = Participants_1.DayReservationID
INNER JOIN
                                                        dbo.Person
AS Person_1 ON Person_1.PersonID = Participants_1.PersonID
                              WHERE        (Person_1.Firstname IS
NULL) AND (Person_1.Lastname IS NULL)))
```

Wolne miejsca na dni konferencji.

```
CREATE VIEW [dbo].[placesLeftPerConferenceDay]
AS
SELECT        dbo.Conferences.ConferenceID,
dbo.Conferences.ConferenceName, dbo.ConferenceDays.DayNumber,
dbo.Conferences.NumberOfPlaces -
SUM(dbo.DayReservation.NormalQuantity) -
SUM(dbo.DayReservation.StudentQuantity)
                        AS [Places Left]
FROM            dbo.ConferenceDays INNER JOIN
                        dbo.Conferences ON
dbo.ConferenceDays.ConferenceID = dbo.Conferences.ConferenceID
INNER JOIN
                        dbo.DayReservation ON
dbo.ConferenceDays.ConferenceID = dbo.DayReservation.ConferenceID
AND dbo.ConferenceDays.DayNumber =
dbo.DayReservation.ConfDayNumber
GROUP BY dbo.Conferences.ConferenceID,
dbo.Conferences.ConferenceName, dbo.ConferenceDays.DayNumber,
dbo.Conferences.NumberOfPlaces
```

8.Procedury

Procedura tworzy krotke kraju w tabeli Country, jeśli istnieje już kraj o podanej nazwie, to zwraca jego ID zamiast tworzonego.

```
CREATE PROCEDURE [dbo].[procCreateCountry]
@CountryName varchar(30),
@CountryID int OUTPUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
  BEGIN TRANSACTION
   SET  @CountryID = ( SELECT CountryID
        FROM Country
        WHERE Country.CountryName = @CountryName)
   IF(@CountryID IS NULL)
   BEGIN
    INSERT INTO Country(CountryName)
    VALUES (@CountryName);
    SET @CountryID =@@IDENTITY;
   END
  COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
  ROLLBACK TRANSACTION
  DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy wstawianiu kraju' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
  THROW 52000,@msg,1;
 END CATCH

END
GO
```

Procedura tworzy krotkę w tabeli City reprezetnującą miasto w podanym kraju, jeśli istnieje już taka para miasto-kraj to zwraca ID tego miasta a nie tworzonego.

```
CREATE PROCEDURE [dbo].[procCreateCity]
@CityName varchar(30),
@CountryName varchar(30),
@CityID int OUTPUT
AS
BEGIN
```

```
SET NOCOUNT ON;
BEGIN TRY
  BEGIN TRANSACTION
   SET @CityID = NULL
   BEGIN
    DECLARE @CountryID int
    EXEC procCreateCountry
     @CountryName,
     @CountryID = @CountryID OUT
    SET @CityID = (SELECT CityID
         FROM City
         WHERE City.CityName= @CityName and City.CountryCountryID=
@CountryID)
    IF(@CityID IS NULL)
    BEGIN
     INSERT INTO City(CityName,CountryCountryID)
     VALUES(@CityName,@CountryID);
     SET @CityID = @@IDENTITY;
    END
   END
  COMMIT TRANSACTION
 END TRY
 BEGIN CATCH
  ROLLBACK TRANSACTION
  DECLARE @msg NVARCHAR(2048)= 'B��d przy wstawianiu miasta' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
  THROW 52000,@msg,1;
 END CATCH

END
GO
```

Procedura wstawia krotkę do tabeli Customers, jest pomocnicza procedurą przy wstwianiu klienta indywidualnego lub firmowego.

```
CREATE PROCEDURE [dbo].[procCreateCustomer]
@Email varchar(30),
@Phone varchar(30),
@PostalCode varchar(30),
@Address varchar(30),
@CityName varchar(30),
@CountryName varchar(30),
@CustomerID int OUTPUT
AS
BEGIN
SET NOCOUNT ON;
```

```
BEGIN TRY
 DECLARE @CityID int
 Exec procCreateCity
  @CityName,
  @CountryName,
  @CityID= @CityID OUTPUT
 INSERT INTO Customers(Email,Phone,PostalCode,Address,CityID)
 VALUES (@Email,@Phone,@PostalCode,@Address,@CityID);
 SET @CustomerID =@@IDENTITY
 END TRY
 BEGIN CATCH
  DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy wstawianiu klienta' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
  THROW 52000,@msg,1;
 END CATCH

END
GO
```

Procedura tworzy krotkę w tabeli Companies reprezentującą klienta firmowego oraz tworzy związaną z nią krotkę w tabeli Customers.

```
CREATE PROCEDURE [dbo].[procCreateCompany]
@CompanyName varchar(30),
@NIP varchar(10),
@Email varchar(30),
@Phone varchar(30),
@PostalCode varchar(30),
@Address varchar(30),
@CityName varchar(30),
@CountryName varchar(30),
@CustomerID int OUTPUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  EXEC procCreateCustomer
   @Email,
   @Phone,
   @PostalCode,
   @Address,
   @CityName,
   @CountryName,
   @CustomerID=@CustomerID OUTPUT
```

```
  INSERT INTO Companies(CustomerID,CompanyName,NIP)
  VALUES (@CustomerID,@CompanyName,@NIP);
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy wstawianiu firmy ' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli Person. jest używana jako pomocnicza przy tworzeniu klienta indywidualnego czy dodawania uczestników do rezerwacji konferencji dla firmy.

```
CREATE PROCEDURE [dbo].[procCreatePerson]
@Firstname varchar(30),
@Lastname varchar(30),
@PersonID int OUTPUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 INSERT INTO Person(Firstname,Lastname)
 VALUES (@Firstname,@Lastname);
 SET @PersonID= @@IDENTITY
END TRY
BEGIN CATCH
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy wstawianiu danych
osobowych' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli IndiviudalClient reprezentującą klienta indywiudalnego oraz tworzy związaną z nią krotkę w tabeli Customers i Person.

```
CREATE PROCEDURE [dbo].[procCreateIndividual]
@Firstname varchar(30),
@Lastname varchar(10),
@Email varchar(30),
@Phone varchar(30),
@PostalCode varchar(30),
@Address varchar(30),
```

```
@CityName varchar(30),
@CountryName varchar(30),
@CustomerID int OUTPUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  DECLARE @PersonID int

  EXEC procCreateCustomer
   @Email,
   @Phone,
   @PostalCode,
   @Address,
   @CityName,
   @CountryName,
   @CustomerID=@CustomerID OUTPUT

  EXEC procCreatePerson
   @Firstname,
   @Lastname,
   @PersonID = @PersonId OUTPUT

  INSERT INTO IndividualClient(CustomerID,PersonID)
  VALUES (@CustomerID,@PersonID);
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy wstawianiu klienta
indywidualnego ' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli Conference reprezentującą konferencję oraz dodaje do tabeli ConferenceDay odpowiednia ilosc wpisów.

```
CREATE PROCEDURE [dbo].[procCreateConference]
@ConferenceName varchar(30),
@BegDate varchar(10),
@EnDate varchar(10),
@StudentDiscount numeric(5,2),
@Price money,
```

```
@Description varchar(255),
@NumberOfPlaces int,
@ConferenceID int OUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION

 BEGIN TRY
  DECLARE @BeginDate date = convert (date, @BegDate)
  DECLARE @EndDate date = convert (date, @EnDate)
 END TRY
 BEGIN CATCH
 ;THROW 52000,
   'Niepoprawny format daty wprowadzonej konferencji',1;
 END CATCH

 DECLARE @Length int = DATEDIFF(day,@BeginDate,@EndDtae) +1;

  IF(@BeginDate < GETDATE())
  BEGIN
   ;THROW 52000,
   'Niepoprawna Data konferencji, jej pocz◆tek jest przed dniem
dzisiejszym',1;
  END

  INSERT INTO
Conferences(ConferenceName,BeginDate,EndDate,StudentDiscount,Price
,Description,NumberOfPlaces)
  VALUES
(@ConferenceName,@BeginDate,@EndDate,@StudentDiscount,@Price,@Desc
ription,@NumberOfPlaces);
  SET @ConferenceID = @@IDENTITY;

  DECLARE @DayNumber int = 1
  WHILE @DayNumber <= @Length
  BEGIN
   INSERT INTO ConferenceDays (ConferenceID,DayNumber)
   VALUES (@ConferenceID,@DayNumber);
   SET @DayNumber =@DayNumber +1;
  END
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
```

```
 DECLARE @msg NVARCHAR(2048)= 'B��d przy wstawianiu konferencji ' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli ConferencePriceList odpowiadającą progowi cenowemu dla podanej konferencji.

```
Create PROCEDURE [dbo].[procCreateConferencePrice]
@ConferenceID int,
@PayDate varchar(10),
@Discount numeric(5,2)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY

 BEGIN TRY
  DECLARE @PaymentDate date = convert (date, @PayDate)
 END TRY
 BEGIN CATCH
 ;THROW 52000,
   'Niepoprawny format daty wprowadzonej konferencji',1;
 END CATCH

 BEGIN TRANSACTION
  IF(@PaymentDate < GETDate())
  BEGIN
   ;THROW 52000,
   'Niepoprawna data, wcze�niejsza ni� dzie� dzisiejszy',1;
  END

  IF(@PaymentDate >( SELECT BeginDate
       FROM  Conferences
       WHERE Conferences.ConferenceID =@ConferenceID))
  BEGIN
   ;THROW 52000,
   'Niepoprawna data, p�niejsza ni� dzie� rozpocz�cia
konferencji',1;
  END

  IF(@PaymentDate IN ( SELECT PaymentDate
       FROM ConferencePriceList
```

```
        WHERE ConferenceID=@ConferenceID AND PaymentDate=
@PaymentDate))
  BEGIN
   ;THROW 52000,
   'Niepoprawna data, istnieje ju pr g dla tej daty',1;
  END
  INSERT INTO
ConferencePriceList(ConferenceID,PaymentDate,Discount)
  VALUES (@ConferenceID,@PaymentDate,@Discount);

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B d przy wstawianiu progu cenowego
' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
```

Procedura tworzy krotkę w tabeli WorkshopDict reprezentującą podany typ warsztatów.

```
CREATE PROCEDURE [dbo].[procCreateWorkshopDict]
@WorkshopName varchar(30),
@Description varchar(255),
@Price money,
@NumberOfPlaces int,
@WorkshopTypeID int OUT
AS
BEGIN
SET NOCOUNT ON;
INSERT INTO
WorkshopDict(WorkshopName,Description,Price,NumberOfPlaces)
VALUES (@WorkshopName,@Description,@Price,@NumberOfPlaces);
SET @WorkshopTypeID= @@IDENTITY;
END
```

Procedura tworzy krotkę w tabeli Workshops, która reprezentuje pojedynczy warsztat, w podanej konferencji oraz godzinach rozpoczęcia.

```
CREATE PROCEDURE [dbo].[procCreateWorkshop]
@WorkshopType int,
@ConferenceID int,
@ConfDayNumber int,
@BeginHr varchar(5),
```

```
@EndHr varchar (5),
@WorkshopID int out
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY

BEGIN TRY
 DECLARE @BeginHour time(7) = convert(time,@BeginHr)
 DECLARE @EndHour time(7) = convert(time,@EndHr)
END TRY
BEGIN CATCH
;THROW 52000,
   'Podano godzin◆ w z◆ym formacie, format to HH:MM',1;
END CATCH

 BEGIN TRANSACTION
  IF( (SELECT  NumberOfPlaces
   FROM WorkshopDict
   WHERE WorkshopDict.WorkshopTypeID = @WorkshopType) >
   (SELECT NumberOfPlaces
   FROM Conferences
   WHERE Conferences.ConferenceID = @ConferenceID))
  BEGIN
   ;THROW 52000,
   'Warsztat nie mo◆e mie◆ wi◆cej miejsc ni◆ pojedynczy dzie◆
konferencji',1;
  END
  DECLARE @Price  money = (SELECT Price FROM WorkshopDict WHERE
WorkshopTypeID = @WorkshopType);
  DECLARE @NumberOfPlaces int = (SELECT NumberOfPlaces FROM
WorkshopDict WHERE WorkshopTypeID= @WorkshopType);
  INSERT INTO
Workshops(WorkshopType,ConferenceID,ConfDayNumber,BeginHour,EndHou
r,Price,NumberOfPlaces)
  VALUES
(@WorkshopType,@ConferenceID,@ConfDayNumber,@BeginHour,@EndHour,@P
rice,@NumberOfPlaces)
  SET @WorkshopID = @@IDENTITY;
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy wstawianiu Warszatu ' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
```

```
END CATCH
END

GO
```

Procedura tworzy krotkę w tabeli Reservations odpowiadającą rezerwacji dla podanego klienta.

```
create PROCEDURE [dbo].[procCreateReservation]
@CustomerID int,
@ReservationID int OUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  DECLARE @Date date = GETDATE()
  INSERT INTO Reservations (CustomerID,ReservationDate)
  VALUES (@CustomerID,@Date);

  SET @ReservationID = @@IDENTITY
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy tworzeniu rezerwacji' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END

GO
```

Procedura tworzy krotkę w tabeli DayReservation, oznacza rezerwacje podanej liczby uczestników na dany dzień zarezerwowanej konferencji. Jest używany przy rezerwacji klienta firmowego lub jako pomocnicza procedura przy rezerwacji klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[procCreateDayReservation]
@ReservationID int,
@ConferenceID int,
@ConfDayNumber int,
@NormalQuantity int =0,
@StudentQuantity int =0,
@DayReservationID int OUT
AS
```

```
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  IF( (SELECT count(*)
   FROM DayReservation
   WHERE ReservationID = @ReservationID
   AND ConferenceID = @ConferenceID
   AND ConfDayNumber = @ConfDayNumber) >0 )
  BEGIN
   ;THROW 52000,
   'Na ten dzie dokonano ju rezerwacji w ramach rezerwacji
konferencji',1;
  END

  IF( @NormalQuantity  + @StudentQuantity =0 )
  BEGIN
   ;THROW 52000,
   'Nie mo na rezerwowa 0 miejsc',1;
  END

  IF( dbo.funcConfDayFreePlaces(@ConferenceID, @ConfDayNumber) <
@StudentQuantity + @NormalQuantity)
  BEGIN
   ;THROW 52000,
   'Za ma o miejsc na konferencji',1;
  END
  INSERT INTO DayReservation
(ReservationID,ConferenceID,ConfDayNumber,NormalQuantity,StudentQu
antity)
  VALUES
(@ReservationID,@ConferenceID,@ConfDayNumber,@NormalQuantity,@Stud
entQuantity);
  SET @DayReservationID = @@IDENTITY;
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B d przy tworzeniu rezerwacji dnia'
+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli DayReservation dla klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[procCreateDayReservationIndiv]
@ReservationID int,
@ConferenceID int,
@ConfDayNumber int,
@StudentIDCard varchar(10)=null,
@DayReservationID int OUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION

  DECLARE @StudentQuantity int =0
  DECLARE @NormalQuantity int =1
  IF(@StudentIDCard IS NOT NULL)
  BEGIN
   SET @StudentQuantity = 1
   SET @NormalQuantity =0
  END
  DECLARE @PersonID int = ( SELECT Person.PersonID
         FROM Reservations INNER JOIN IndividualClient
         ON IndividualClient.CustomerID = Reservations.CustomerID
         INNER JOIN  Person ON Person.PersonID =
IndividualClient.PersonID)

  EXEC procCreateDayReservation
  @ReservationID,
@ConferenceID,@ConfDayNumber,@NormalQuantity,@StudentQuantity,
  @DayReservationID= @DayReservationID OUT

  INSERT INTO Participants(DayReservationID,PersonID)
  VALUES (@DayReservationID,@PersonID)
  DECLARE @ParticipantID int = @@IDENTITY

  IF(@StudentIDCard IS NOT NULL)
  BEGIN
   INSERT INTO Student(ParticipantID,StudentIDCard)
   VALUES (@ParticipantID,@StudentIDCard)
  END
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
```

```
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy tworzeniu rezerwacji dnia'
+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli WorkshopReservation oznaczającą rezerwację podanego warsztatu na dany zarezerwowany dzień konferencji. Jest używana przy rezerwowaniu dla firm lub jako pomocnicza procedura przy rezerwacji dla klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[procCreateWorkshopReservation]
@DayReservationID int,
@WorkshopID int,
@Quantity int ,
@WorkshopReservationID int OUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  IF( (SELECT count(*)
   FROM WorkshopReservation
   WHERE DayReservationID = @DayReservationID
   AND WorkshopId = @WorkshopID) >0 )
  BEGIN
   ;THROW 52000,
   'Na ten warsztat dokonano ju◆ rezerwacji w ramach rezerwacji',1;
  END

  IF( @Quantity =0 )
  BEGIN
   ;THROW 52000,
   'Nie mo◆na rezerwowa◆ 0 miejsc',1;
  END

  IF( dbo.funcWorkshopFreePlaces(@WorkshopID) < @Quantity)
  BEGIN
   ;THROW 52000,
   'Za ma◆o miejsc na konferencji',1;
  END

  INSERT INTO WorkshopReservation
(DayReservationID,WorkshopID,Quantity)
  VALUES (@DayReservationID,@WorkshopID,@Quantity);
  SET @WorkshopReservationID = @@IDENTITY;
```

```
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy tworzeniu rezerwacji
warsztatu' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli WorkshopReservation jako rezerwację warsztatu dla klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[procCreateWorkshopReservationIndiv]
@DayReservationID int,
@WorkshopID int,
@WorkshopReservationID int OUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  DECLARE @ParticipantID int =
  (SELECT ParticipantID
  FROM Participants
  WHERE Participants.DayReservationID = @DayReservationID)

  EXEC procCreateWorkshopReservation
  @DayReservationId,@WorkshopID,1,
  @WorkshopReservationID = @WorkshopReservationID OuT

  EXEC procCreateWorkshopParticipant
  @ParticipantID,
  @WorkshopReservationID


 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy rezerwacji warsztatu dla
pojedynczego klienta' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
```

```
GO
```

Procedura tworzy krotkę w tabeli WorkshopParticipant, która przedstawia dodanie uczestnika na rezerwacje warsztatu.

```
CREATE PROCEDURE [dbo].[procCreateWorkshopParticipant]
@ParticipantID int,
@WorkshopReservationID int
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  INSERT INTO
WorkshopParticipant(WorkshopReservationID,ParticipantID)
  VALUES (@WorkshopReservationID,@ParticipantID)

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy dodawaniu uczestnika do
rezerwacji warsztatu' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura w zależności od przekazanych danych, uaktualnia dane o studencie rezerwowanym przez firmę, tworzy zwykłego uczestnika konferencji lub tworzy ‘blank studenta’, który ma podaną legitymację ale nie dane osobowe.

```
CREATE PROCEDURE [dbo].[procCreateUpdateParticipantCompany]
@DayReservationID int,
@Firstname varchar(30),
@Lastname varchar(30),
@StudentIDCard varchar(10),
@ParticipantID int OUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  DECLARE @PersonID int
  IF(@FirstName IS NOT NULL AND @Lastname IS NOT NULL
  AND @StudentIDCard IS NOT NULL) -- uzupelniam dane dla studenta
```

```
  BEGIN

   SET @ParticipantID = (SELECT Student.ParticipantID
         FROM Student INNER JOIN Participants
         ON Student.ParticipantId = Participants.ParticipantID
         WHERE Student.StudentIDCard = @StudentIDCard
         AND DayReservationID= @DayReservationID)
   IF (@ParticipantID IS NULL)
   BEGIN
    ;THROW 52000,
    'W podanej rezerwacji dnia konferencji nie ma studenta o takiej
legitymacji',1;
   END
   SET @PersonID = (SELECT PersonID
       FROM Participants
       WHERE ParticipantID=@ParticipantID)
   UPDATE Person SET
    Firstname= @Firstname,
    Lastname = @Lastname
   WHERE Person.PersonID = @PersonID

  END

  IF(@Firstname IS NULL AND @Lastname IS NULL
  AND @StudentIDCard IS NOT NULL) -- tworze blank studenta
  BEGIN

   EXEC procCreatePerson
    @Firstname,@Lastname,
    @PersonId= @PersonID OUT

   INSERT INTO Participants(DayReservationID,PersonID)
   VALUES (@DayReservationID,@PersonID)
   SET @ParticipantID = @@IDENTITY

   INSERT INTO Student(ParticipantID,StudentIDCard)
   VALUES (@ParticipantID,@StudentIDCard)
  END

  IF (@Firstname IS NOT NULL AND @Lastname IS NOT NULL
  AND @StudentIDCard IS NULL) -- tworze zwyklego uczestnika
  BEGIN

   EXEC procCreatePerson
    @Firstname,@Lastname,
    @PersonId= @PersonID OUT
```

```
   INSERT INTO Participants(DayReservationID,PersonID)
   VALUES (@DayReservationID,@PersonID)
   SET @ParticipantID = @@IDENTITY
  END

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy tworzeniu uczestnika dnia
konferencji' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura tworzy krotkę w tabeli Students, Participants i Person reprezentującą studenta bez podanych danych osobowych.

```
CREATE PROCEDURE [dbo].[procCreateBlankStudent]
@DayReservationID int,
@StudentIDCard varchar(10),
@ParticipantID int OUT
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  EXEC procCreateUpdateParticipantCompany
  @DayReservationID,null,null,@StudentIDCard,
  @ParticipantID = @ParticipantID OUT

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy tworzeniu blank studenta'
+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura uaktualnia dane o dacie zapłaty za rezerwację.

```
CREATE PROCEDURE [dbo].[procPayForReservation]
@ReservationID int
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION

 IF( (SELECT PaymentDate
  FROM Reservations
  WHERE ReservationID = @ReservationID) IS NOT NULL)
 BEGIN
  ;THROW 52000,
  'Rezerwacja juz jest oplacona',1;
 END

 UPDATE Reservations
 SET PaymentDate=GETDATE()
 WHERE ReservationID = @ReservationID

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy op◆acaniu rezerwacji' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura usuwa wszystkie rezerwacje, od których złożenia minął tydzień.

```
CREATE PROCEDURE [dbo].[procRemoveUnpaidReservations]
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION

 DELETE FROM Reservations
 WHERE PaymentDate IS NULL AND
DATEDIFF(day,ReservationDate,GETDATE()) >=7
```

```
 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy usuwaniu nieop◆aconych
rezerwacji' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura usuwa uczestnika o podanym ID z bazy.

```
CREATE PROCEDURE [dbo].[procRemoveParticipant]
@ParticipantID int
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  IF( (SELECT ParticipantID
   FROM Participants
   WHERE ParticipantID = @ParticipantID) IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie znaleziono takiego Participanta',1;
  END
  --usuwa sprzezonego Person, moze przeniesiemy do Triggerow jak co◆
  DECLARE @PersonID int  = ( SELECT Person.PersonID
        FROM Person INNER JOIN Participants
        ON Person.PersonID = Participants.PersonID
        WHERE ParticipantID = @ParticipantID)
  DELETE Participants
  WHERE Participants.ParticipantID = @ParticipantID
  IF ( @PersonID IS NOT NULL)
  BEGIN
   DELETE Person
   WHERE PersonID = @ParticipantID
  END

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy usuwaniu nieop◆aconych
rezerwacji' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
```

```
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura usuwa uczestnika warsztatu o podanym id.

```
CREATE PROCEDURE [dbo].[procRemoveWorkshopParticipant]
@ParticipantID int,
@WorkshopReservationID int
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  IF( (SELECT ParticipantID
   FROM WorkshopParticipant
   WHERE ParticipantID = @ParticipantID
   AND WorkshopReservationID = @WorkshopReservationID) IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie znaleziono takiego uczestnika warsztatu',1;
  END

  DELETE WorkshopParticipant
  WHERE ParticipantID = @ParticipantID AND WorkshopReservationID =
@WorkshopReservationID


 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy usuwaniu uczestnika
warsztatu' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura usuwa rezerwację o podanym ID.

```
CREATE PROCEDURE [dbo].[procRemoveReservation]
@ReservationID int
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION


  IF ( @ReservationID IS NOT NULL AND ( SELECT PaymentDate
    FROM Reservations
    WHERE ReservationID = @ReservationID) IS NOT NULL)
  BEGIN
   ;THROW 52000,
   'Nie mozna usunac oplaconej rezerwacji',1;
  END

  IF ( ( SELECT ReservationID
    FROM Reservations
    WHERE ReservationID = @ReservationID) IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie mozna odnalezc rezerwacji o podanym ID',1;
  END

  DELETE FROM  Reservations
  WHERE ReservationID= @ReservationID

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy usuwaniu  rezerwacji' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura usuwa rezerwacje dnia konferencji.

```
CREATE PROCEDURE [dbo].[procRemoveDayReservation]
@DayReservationID int
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION

  DECLARE @ReservationID int = ( SELECT ReservationID
          FROM DayReservation
          WHERE DayReservationID= @DayReservationID)

  IF ( @ReservationID IS NOT NULL AND ( SELECT PaymentDate
    FROM Reservations
    WHERE ReservationID = @ReservationID) IS NOT NULL)
  BEGIN
   ;THROW 52000,
   'Nie mozna usunac rezerwacji oplaconego dnia',1;
  END

  IF ( @ReservationID IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie mozna odnalezc dniia rezerwacji o podanym ID',1;
  END

  DELETE FROM  DayReservation
  WHERE DayReservationID= @DayReservationID

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy usuwaniu rezerwacji dnia'
+ CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura usuwa rezerwację warsztatu.

```
CREATE PROCEDURE [dbo].[procRemoveWorkshopReservation]
@WorkshopReservationID int
AS
BEGIN
```

```
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION

  DECLARE @DayReservationID  int = ( SELECT DayReservationID
          FROM WorkshopReservation
          WHERE WorkshopReservationID= @WorkshopReservationID)

  DECLARE @ReservationID int = ( SELECT ReservationID
          FROM DayReservation
          WHERE DayReservationID= @DayReservationID)

  IF ( @ReservationID IS NOT NULL AND ( SELECT PaymentDate
    FROM Reservations
    WHERE ReservationID = @ReservationID) IS NOT NULL)
  BEGIN
   ;THROW 52000,
   'Nie mozna usunac warsztatu oplaconej rezerwacji',1;
  END


  IF ( ( SELECT WorkshopReservationID
    FROM WorkshopReservation
    WHERE WorkshopReservationID = @WorkshopReservationID) IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie mozna odnalezc warsztatu o podanym ID',1;
  END

  DELETE FROM  WorkshopReservation
  WHERE WorkshopReservationID= @WorkshopReservationID

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy usuwaniu rezerwacji
warsztatu' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura uaktualnia dane klienta, jest używana jako pomocnicza przy uaktualnianiu danych firmy lub klienta indywidualnego.

```
CREATE PROCEDURE [dbo].[procUpdateCustomer]
@CustomerID int,
@CityName varchar(30),
@CountryName varchar(30),
@Address varchar(30),
@PostalCode varchar(30),
@Phone varchar(30),
@Email varchar(30)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  IF( (SELECt CustomerID
   FROM Customers
   WHERE CustomerID = @CustomerID) IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie znaleziono klienta o podanym ID',1;
  END

 DECLARE @CityID int
 EXEC procCreateCity
 @CityName,@CountryName,
 @CityID=@CityID

  UPDATE Customers
  SET CityID= @CityID,
  Address= @Address,
  PostalCode = @PostalCode,
  Phone = @Phone,
  Email = @Email
  WHERE CustomerID = @CustomerID

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy uaktualnianiu klienta' +
CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura uaktualnia dane klienta firmowego.

```
CREATE PROCEDURE [dbo].[procUpdateCompanyCustomer]
@CustomerID int,
@CityName varchar(30),
@CountryName varchar(30),
@Address varchar(30),
@PostalCode varchar(30),
@Phone varchar(30),
@Email varchar(30),
@CompanyName varchar(30),
@NIP varchar(30)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  IF( (SELECt CustomerID
   FROM Customers
   WHERE CustomerID = @CustomerID) IS NULL
   OR (SELECT CustomerID
    FROM Companies
    WHERE CustomerID = @CustomerID) IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie znaleziono klienta o podanym ID',1;
  END
  EXEC procUpdateCustomer

@CustomerID,@CityName,@CountryName,@Address,@PostalCode,@Phone,@Em
ail

  UPDATE Companies
  SET CompanyName=@CompanyName,
  NiP=@NIP
  WHERE CustomerID = @CustomerID

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy uaktualnianiu klienta
firmowego' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

Procedura uaktualnia dane klienta indywidualnego

```
CREATE PROCEDURE [dbo].[procUpdateIndivCustomer]
@CustomerID int,
@CityName varchar(30),
@CountryName varchar(30),
@Address varchar(30),
@PostalCode varchar(30),
@Phone varchar(30),
@Email varchar(30),
@Firstname varchar(30),
@Lastname varchar(30)
AS
BEGIN
SET NOCOUNT ON;
BEGIN TRY
 BEGIN TRANSACTION
  IF( (SELECt CustomerID
   FROM Customers
   WHERE CustomerID = @CustomerID) IS NULL
   OR (SELECT CustomerID
    FROM IndividualClient
    WHERE CustomerID = @CustomerID) IS NULL)
  BEGIN
   ;THROW 52000,
   'Nie znaleziono klienta o podanym ID',1;
  END
  DECLARE @PersonID int = (SELECT PersonID
        FROM IndividualClient
        WHERE CustomerID= @CustomerID)
  EXEC procUpdateCustomer

@CustomerID,@CityName,@CountryName,@Address,@PostalCode,@Phone,@Em
ail


  UPDATE Person
  SET Firstname=@Firstname,
  Lastname=@Lastname
  WHERE PersonID= @PersonID

 COMMIT TRANSACTION
END TRY
BEGIN CATCH
 ROLLBACK TRANSACTION
```

```
 DECLARE @msg NVARCHAR(2048)= 'B◆◆d przy uaktualnianiu klienta
indywidualnego' + CHAR(13) + CHAR(10) + ERROR_MESSAGE();
 THROW 52000,@msg,1;
END CATCH
END
GO
```

9.Funkcje

Funkcje:

Zwraca ilość wolnych miejsc na dzień konferencji.

```
CREATE FUNCTION [dbo].[funcConfDayFreePlaces]
( @ConferenceID int ,@ConfDayNumber int)
RETURNS int
AS
BEGIN
DECLARE @NumberOfPlaces int=
dbo.funcConferenceNumberOfPlaces(@ConferenceID);

DECLARE @UsedPlaces int =
dbo.funcConferenceDayTakenPlaces(@ConferenceID,@ConfDayNumber);

RETURN @NumberOfPlaces -@UsedPlaces
END
```

Zwraca ilość maksymalną miejsc dla danej konferencji.

```
CREATE FUNCTION [dbo].[funcConferenceNumberOfPlaces]
( @ConferenceID int )
RETURNS INT
AS
BEGIN
RETURN ( ISNULL( (SELECT NumberOfPlaces
   FROM Conferences
   WHERE ConferenceID =@ConferenceID),0) )
END
```

Zwraca ilość zajętych miejsc dla danego dnia konferencji.

```
CREATE FUNCTION [dbo].[funcConferenceDayTakenPlaces]
( @ConferenceID int, @ConfDayNumber int )
```

```
RETURNS INT
AS
BEGIN
RETURN ( ISNULL( (SELECT sum(NormalQuantity + StudentQuantity)
   FROM DayReservation
   WHERE ConferenceID = @ConferenceID
   AND ConfDayNumber = @ConfDayNumber  ),0))
END
```

Zwraca ilość wolnych miejsc dla warsztatu.

```
CREATE FUNCTION [dbo].[funcWorkshopFreePlaces]
( @WorkshopID int)
RETURNS int
AS
BEGIN
DECLARE @NumberOfPlaces int=
dbo.funcWorkshopNumberOfPlaces(@WorkshopID);

DECLARE @UsedPlaces int = dbo.funcWorkshopTakenPlaces(@WorkshopID);

RETURN @NumberOfPlaces -@UsedPlaces
END
```

Zwraca maksymalną ilość miejsc dla danego warsztatu.

```
CREATE FUNCTION [dbo].[funcWorkshopNumberOfPlaces]
( @WorkshopID int )
RETURNS INT
AS
BEGIN
RETURN ( ISNULL( (SELECT NumberOfPlaces
   FROM Workshops
   WHERE WorkshopID =@WorkshopID),0) )
END
```

Zwraca ilość zajętych miejsc dla danego warsztatu

```
CREATE FUNCTION [dbo].[funcWorkshopTakenPlaces]
( @WorkshopID int )
RETURNS INT
AS
BEGIN
RETURN ( ISNULL( (SELECT sum(Quantity)
   FROM WorkshopReservation
```

```
   WHERE WorkshopID= @WorkshopID  ),0))
END
```

Zwraca całkowity koszt rezerwacji.

```
CREATE FUNCTION [dbo].[funcPayment]
( @ReservationID int )
returns money
as
begin
declare @normalprice money =
(SELECT dbo.funcConferencePrice(C.ConferenceID) *
(1 - dbo.funcConferencePriceDiscount(C.ConferenceID,
R.PaymentDate))
FROM Reservations as R
JOIN Conferences as C
ON C.ConferenceID =
dbo.funcConferenceIDbyReservation(@ReservationID)
WHERE R.ReservationID = @reservationID)

declare @studentDiscount numeric(5,2) =
(select studentdiscount
from Conferences as c
where
dbo.funcConferenceIDbyReservation(@ReservationID)=c.ConferenceID)

DECLARE @conferenceCost MONEY =
(Select dbo.funcNormalQuantity(@ReservationID) * @normalprice +
dbo.funcStudentQuantity(@ReservationID) * @normalprice * (1 -
@studentDiscount)
From [DayReservation]
WHERE ReservationID = @reservationID)

DECLARE @workshopCost MONEY =
(Select SUM(quantity * wd.price)
FROM WorkshopReservation as WR
join DayReservation as DR on DR.DayReservationID =
WR.DayReservationID
JOIN Workshops as W ON W.WorkshopID = WR.WorkshopID
join WorkshopDict as WD on WD.WorkshopTypeID=W.WorkshopType
join Reservations as r on r.ReservationID=DR.ReservationID
where r.ReservationID=@ReservationID)
```

```
return (isnull(@conferenceCost,0) + isnull(@workshopCost,0))
end
```

Zwraca ilość zarezerwowanych normalnych biletów.

```
create function [dbo].[funcNormalQuantity]
(@ReservationID int)
returns int
as
begin
return (isnull((select SUM(normalquantity)
   from DayReservation
   where ReservationID=@ReservationID),0))
end
```

Zwraca ilość zarezerwowanych studenckich biletów.

```
create function [dbo].[funcStudentQuantity]
(@ReservationID int)
returns int
as
begin
return (isnull((select SUM(studentquantity)
   from DayReservation
   where ReservationID=@ReservationID),0))
end
```

Zwraca zniżke dla podanej konferencji.

```
create function [dbo].[funcConferencePriceDiscount]
(
@ConferenceID int,
@PaymentDate date
)
returns numeric(5,2)
as
begin
return (isnull((select a.discount
   from (  select discount, MIN(PaymentDate) as minimal
     from ConferencePriceList
     where ConferenceID=@ConferenceID and @PaymentDate>PaymentDate
```

```
     group by Discount) as a),0))

end
```

Zwraca liczbę uczestników na konferencji.

```
CREATE FUNCTION dbo.funcConferenceParticipants(@ConferenceID int)
RETURNS int
AS
BEGIN
RETURN
(SELECT COUNT(p.ParticipantID)
FROM dbo.DayReservation AS dr
JOIN dbo.Participants AS p ON
p.DayReservationID=dr.DayReservationID
WHERE dr.ConferenceID=@ConferenceID)
END
```

Zwraca nazwe firmy do której należy uczestnik, lub nulla jeśli nie należy do żadnej firmy.

```
CREATE FUNCTION dbo.funcParticipantCompanyName(@ParticipantID int)
RETURNS varchar(30)
AS
BEGIN
RETURN
ISNULL((SELECT c.CompanyName
FROM dbo.Companies AS c
JOIN dbo.Customers AS cu ON cu.CustomerID=c.CustomerID
JOIN dbo.Reservations AS r ON c.CustomerID=r.CustomerID
JOIN dbo.DayReservation AS dr ON r.ReservationID=dr.ReservationID
JOIN dbo.Participants AS p ON
dr.DayReservationID=p.DayReservationID
WHERE p.ParticipantID=@ParticipantID)
,NULL)
END
```

Zwraca cenę konferencji dla id konferencji.

```
create function [dbo].[funcConferencePrice]
(@ConferenceID int)
returns money
```

```
as
begin
return (isnull((select price
   from Conferences
   where ConferenceID=@ConferenceID),0))
end
```

Zwraca id konferencji dla id rezerwacji.

```
create function [dbo].[funcConferenceIDbyReservation]
(@ReservationID int)
returns int
as
begin
return ( ISNULL( (SELECT DR.ConferenceID
   FROM Reservations as R
   join DayReservation as DR on R.ReservationID=DR.ReservationID
   WHERE R.ReservationID= @ReservationID  ),0))
end
```

Funkcje z widokami:

Pokazuje nieopłacone rezerwacje dla danego klienta.

```
CREATE FUNCTION [dbo].[funcNotPaidReservationsForCustomer]
(
@CustomerID int
)
RETURNS TABLE
AS
RETURN
(
SELECT ReservationID, ConferenceName, BeginDate, EndDate
FROM dbo.Reservations
INNER JOIN dbo.Conferences
ON Conferences.ConferenceID =
dbo.funcConferenceIDbyReservation(ReservationID)
INNER JOIN dbo.Customers
ON customers.CustomerID = Reservations.CustomerID
WHERE customers.CustomerID = @CustomerID AND PaymentDate IS NULL
)
```

Dla danej konferencji podaje liste uczestników.

```
CREATE FUNCTION [dbo].[funcParticipantsInConference]
(
@ConferenceID int
)
RETURNS TABLE
AS
RETURN
(
SELECT Firstname, Lastname
FROM Conferences as c
join DayReservation as dr on c.ConferenceID=dr.ConferenceID
join Participants as p on p.DayReservationID=dr.DayReservationID
join Person as pe on pe.PersonID=p.PersonID
where c.ConferenceID=@ConferenceID
)
```

Dla warsztatu podaje liste uczestników.

```
CREATE FUNCTION [dbo].[funcParticipantsInWorkshop]
(
@WorkshopID int
)
RETURNS TABLE
AS
RETURN
(
SELECT Firstname, Lastname
from Workshops
inner join WorkshopReservation
on workshops.WorkshopID=WorkshopReservation.WorkshopID
inner join WorkshopParticipant
on
WorkshopParticipant.WorkshopReservationID=WorkshopReservation.Work
shopReservationID
INNER JOIN Participants
ON Participants.ParticipantID = WorkshopParticipant.ParticipantID
INNER JOIN Person
ON Person.PersonID = Participants.PersonID
where Workshops.WorkshopID=@WorkshopID
)
```

Pokazuje aktualne rezerwacje dla danego klienta.

```
CREATE FUNCTION [dbo].[funcActualReservationsForCustomer]
(
@CustomerID int
)
RETURNS TABLE
AS
RETURN
(
SELECT ReservationDate, PaymentDate, ConferenceName, BeginDate
FROM Reservations as r
INNER JOIN customers as c
ON C.CustomerID = R.CustomerID
INNER JOIN Conferences as co
ON Co.ConferenceID =
dbo.funcConferenceIDbyReservation(r.ReservationID)
WHERE (DATEDIFF(day, BeginDate, R.ReservationDate) > 0)
AND (DATEDIFF(DAY, BeginDate, GETDATE()) > 0) and
c.CustomerID=@CustomerID
)
```

Pokazuje aktualne konferencje dla danej osoby.

```
CREATE FUNCTION [dbo].[funcActualConferencesForPerson]
(
@PersonID int
)
RETURNS TABLE
AS
RETURN
(
SELECT ConferenceName,BeginDate,EndDate
FROM dbo.Conferences
INNER JOIN Reservations
ON dbo.funcConferenceIDbyReservation(Reservations.ReservationID) =
Conferences.ConferenceID
INNER JOIN DayReservation
ON DayReservation.ReservationID = Reservations.ReservationID
INNER JOIN Participants
ON Participants.DayReservationID =DayReservation.DayReservationID
INNER JOIN Person
ON Person.PersonID = Participants.PersonID
AND Person.PersonID = @PersonID
WHERE (DATEDIFF(day, BeginDate, Reservations.ReservationDate) > 0)
AND (DATEDIFF(DAY, BeginDate, GETDATE()) > 0)
)
```

Pokazuje aktualne warsztaty dla danej osoby.

```
CREATE FUNCTION fp_ActualWorkshopsForPerson
(
@PersonID int
)
RETURNS TABLE
AS
RETURN
(
SELECT
WorkshopName,workshopdict.Description,BeginDate,BeginHour,EndHour
FROM WorkshopDict
INNER JOIN Workshops
ON Workshops.WorkshopType = WorkshopDict.WorkshopTypeID
INNER JOIN Conferences
ON Conferences.ConferenceID = Workshops.ConferenceID
INNER JOIN WorkshopReservation
ON WorkshopReservation.WorkshopID =Workshops.WorkshopID
INNER JOIN DayReservation
ON DayReservation.DayReservationID
=DayReservation.DayReservationID
INNER JOIN Participants
ON Participants.DayReservationID =DayReservation.DayReservationID
INNER JOIN Person
ON Person.PersonID = Participants.PersonID
AND Participants.PersonID = @PersonID
)
```

Pokazuje uczestników dla dnia konferencji.

```
CREATE FUNCTION [dbo].[funcParticipantsInConferenceDay]
(
@ConferenceDayID int
)
RETURNS TABLE
AS
RETURN
(
SELECT Firstname, Lastname
FROM Person
INNER JOIN Participants
ON Person.PersonID = Participants.PersonID
INNER JOIN DayReservation
on DayReservation.DayReservationID=Participants.DayReservationID
```

```
AND DayReservation.DayReservationID = @ConferenceDayID
)
```

Dla danego dnia pokazuje konferencje.

```
CREATE FUNCTION dbo.funcConferencesPerDay( @Date date )
RETURNS TABLE
AS
RETURN
(SELECT * FROM Conferences
WHERE BeginDate <= @Date AND EndDate >= @Date )
--Dla danego okresu czasu pokazuje liste konferencji
CREATE FUNCTION dbo.funcConferencesPerTimeFrame( @BeginDate date,
@EndDate date )
RETURNS TABLE
AS
RETURN
(SELECT * FROM conferences
WHERE BeginDate >= @BeginDate AND EndDate <= @EndDate )
```

Dni konferencji dla danej osoby.

```
CREATE FUNCTION funcActualConferenceDaysForPerson
(
@PersonID int
)
RETURNS TABLE
AS
RETURN
(
SELECT distinct c.ConferenceID, ConferenceName,BeginDate, EndDate,
Description, DayNumber
FROM Conferences as c
join ConferenceDays as cd on c.ConferenceID=cd.ConferenceID
join DayReservation as dr on dr.ConferenceID=cd.ConferenceID and
dr.ConfDayNumber=cd.DayNumber
join Participants as p on p.DayReservationID=dr.DayReservationID
join Person as pe on pe.PersonID=p.PersonID
where pe.PersonID=@PersonID
)
```

Dni konferencji dla danej rezerwacji.

```
create function funcConferenceDaysForReservation(@ReservationID
int)
returns table
```

```
as
return
(select c.ConferenceName, cd.DayNumber, dr.NormalQuantity as
'liczba bilet◆w normalnych',
dr.StudentQuantity as 'liczba bilet◆w studenckich'
from Conferences as c
join ConferenceDays as cd on cd.ConferenceID=c.ConferenceID
join DayReservation as dr on dr.ConferenceID=cd.ConferenceID and
dr.ConfDayNumber=cd.DayNumber
where dr.ReservationID=@ReservationID)
```

Zarezerwowane warsztaty dla rezerwacji.

```
create function funcWorkshopsForReservation(@ReservationID int)
returns table
as
return
(select wd.WorkshopName, w.ConfDayNumber, wr.Quantity as 'liczba
zarezerwowanych miejsc'
from WorkshopDict as wd
join Workshops as w on w.WorkshopType=wd.WorkshopTypeID
join WorkshopReservation as wr on wr.WorkshopID=w.WorkshopID
join DayReservation as dr on
dr.DayReservationID=wr.DayReservationID
where dr.ReservationID=@ReservationID)
```

Dni konferencji dla konferencji.

```
CREATE FUNCTION funcConferenceDaysOfConference(@ConferenceID int)
RETURNS TABLE
AS
RETURN
(SELECT * FROM ConferenceDays WHERE ConferenceID = @ConferenceID)
--Warsztaty dla dnia konferencji
CREATE FUNCTION funcWorkshopOfConferenceDay(@ConferenceID int,
@ConferenceDayNumber int)
RETURNS TABLE
AS
RETURN
(SELECT w.ConferenceID, w.ConfDayNumber as 'Dzie◆ konferencji',
w.WorkshopID,
wd.WorkshopName, w.BeginHour, w.EndHour,
wd.Description,w.NumberOfPlaces
from Workshops as w
join WorkshopDict as wd on wd.WorkshopTypeID=w.WorkshopType
```

```
where w.ConferenceID=@ConferenceID and
w.ConfDayNumber=@ConferenceDayNumber)
```

Faktura.

```
create function funcInvoice(@ReservationID int)
returns table
as
return
(select 'Faktura' as Faktura
union all
select 'Dane klienta' as Dane
union all
select 'Company Name:'+CompanyName+'| NIP: '+NIP+'| Telefon: '+
Phone+'| E-mail: '+Email+'| Adres: '+CountryName+', '+CityName+',
'+
Address+', '+PostalCode
from Companies as co
join Customers as c on co.CustomerID=c.CustomerID
join City on c.CityID=City.CityID
join Country on City.CountryCountryID=Country.CountryID
join Reservations as r on r.CustomerID=c.CustomerID
where ReservationID=@ReservationID
union all
select 'Imie:'+p.Firstname+'| Nazwisko: '+p.Lastname+'| Telefon: '+
Phone+'| E-mail: '+Email+'| Adres: '+CountryName+', '+CityName+',
'+
Address+', '+PostalCode
from Person as p
join IndividualClient as ic on ic.PersonID=p.PersonID
join Customers as c on ic.CustomerID=c.CustomerID
join City on c.CityID=City.CityID
join Country on City.CountryCountryID=Country.CountryID
join Reservations as r on r.CustomerID=c.CustomerID
where ReservationID=@ReservationID
union all
select 'Rezerwacja dni konferencji, bilety normalne'
union all
select 'Nazwa konferencji: '+ConferenceName+'| Dzie◆:
'+STR(dr.ConfDayNumber)+
'| Ilo◆◆ bilet◆w normalnych: '+str(dr.NormalQuantity)+'| Cena za
bilet: '+
STR(Price)+'| Suma: '+
STR((dr.NormalQuantity*Price*(1-dbo.funcConferencePriceDiscount(c.
ConferenceID, r.PaymentDate))))
```

```
from Conferences as c
join DayReservation as dr on dr.ConferenceID=c.ConferenceID
JOIN dbo.Reservations AS r ON r.ReservationID=dr.ReservationID
where dr.ReservationID=@ReservationID
GROUP BY c.ConferenceID, c.ConferenceName, dr.ConfDayNumber,
dr.NormalQuantity, c.Price, r.PaymentDate
union all
select 'Rezerwacja dni konferencji, bilety studenckie'
union all
select 'Nazwa konferencji: '+ConferenceName+'| Dzie◆:
'+str(dr.ConfDayNumber)+
'| Ilo◆◆ bilet◆w studenckich: '+STR(dr.StudentQuantity)+'| Cena za
bilet: '+STR(Price)+
'| Zni◆ka studencka: '+STR(c.StudentDiscount)+'| Suma: '+
STR((dr.StudentQuantity*Price*(1-c.StudentDiscount)*(1-dbo.funcCon
ferencePriceDiscount(c.ConferenceID, r.PaymentDate))))
from Conferences as c
join DayReservation as dr on dr.ConferenceID=c.ConferenceID
JOIN dbo.Reservations AS r ON r.ReservationID=dr.ReservationID
where dr.ReservationID=@ReservationID
GROUP BY c.ConferenceID, c.ConferenceName, dr.ConfDayNumber,
dr.StudentQuantity, c.Price, c.StudentDiscount, r.PaymentDate
union all
select 'Rezerwacja warsztat◆w'
union all
select 'Nazwa warsztatu: '+WorkshopName+'| Dzie◆:
'+STR(dr.ConfDayNumber)+
'| Ilo◆◆ bilet◆w: '+STR(wr.Quantity)+'| Cena za bilet: '+
STR(wd.Price)+'| Suma: '+STR((wr.Quantity*wd.Price))
from WorkshopDict as wd
join Workshops as w on wd.WorkshopTypeID=w.WorkshopType
join WorkshopReservation as wr on wr.WorkshopID=w.WorkshopID
join DayReservation as dr on
dr.DayReservationID=wr.DayReservationID
where dr.ReservationID=@ReservationID
GROUP BY workshopname, dr.ConfDayNumber, wd.price, wr.Quantity
union all
select 'Podsumowanie'
union all
select 'Kwota do zap◆aty: '+STR(dbo.funcPayment(@ReservationID))
union all
select 'Data zap◆aty: '+cast(PaymentDate AS VARCHAR(30))
from Reservations
where ReservationID=@ReservationID)
```

10.Triggery

Sprawdza czy nie ma już rezerwacji tego samego dnia dla tej konferencji.

```
CREATE TRIGGER ReservationDayMulti
ON DayReservation
AFTER INSERT,UPDATE
AS
BEGIN
 IF ( EXISTS(
  SELECT * FROM inserted
   CROSS JOIN DayReservation
   WHERE inserted.ConfDayNumber = DayReservation.ConfDayNumber
   AND inserted.ConferenceID = DayReservation.ConferenceID
   AND inserted.ReservationID = DayReservation.ReservationID
   AND inserted.DayReservationID <>
DayReservation.DayReservationID ) )
 BEGIN
  ROLLBACK TRANSACTION
  ;THROW 52000,
  'W ramach tej rezerwacji juz rezerwowano ten dzie◆',1;
 END
END
GO
```

Sprawdza czy dodany próg cenowy nie jest bezsensowny ze wzgledu na data-znizka.

```
CREATE TRIGGER PriceListDate
ON ConferencePriceList
AFTER INSERT,UPDATE
```

```
AS
BEGIN
 IF ( EXISTS(
  SELECT * FROM inserted join ConferencePriceList conf
  on conf.ConferenceID = inserted.ConferenceID
  where (inserted.PaymentDate > conf.PaymentDate and
inserted.Discount > conf.Discount)
    ) )
 BEGIN
  ROLLBACK TRANSACTION
  ;THROW 52000,
  'W tej konferencji istnieje termin z wcze♦niejsz♦ dat♦ i mniejsz♦
zni♦k♦ ni♦ wprowadzany pr♦g. Wprowadzany pr♦g
  musi mie♦ mniejsz♦ zni♦k♦, ni♦ wszystkie poprzedzaj♦ce go.',1;
 END
END
GO

Sprawdza czy nie ma już rezerwacji tego warsztatu do tej samej
rezerwacji dnia.

CREATE TRIGGER WorkshopReservationMulti
ON WorkshopReservation
AFTER INSERT,UPDATE
AS
BEGIN
 IF ( EXISTS(
  SELECT * FROM inserted
   CROSS JOIN WorkshopReservation
   WHERE inserted.DayReservationID =
WorkshopReservation.DayReservationID
   AND inserted.WorkshopID = WorkshopReservation.WorkshopID
   AND inserted.WorkshopReservationID <>
WorkshopReservation.WorkshopReservationID ) )
 BEGIN
  ROLLBACK TRANSACTION
  ;THROW 52000,
  'W ramach rezerwacji tego dnia rezerwowano ju♦ ten warsztat',1;
 END
END
GO
```

Sprawdza czy rezerwowany warsztat odbywa się w tym samym dniu co dzień konferencji, z którym powiązana jest rezerwacja warsztatu.

```
CREATE TRIGGER WorkshopReservationToDayReservation
ON WorkshopReservation
AFTER INSERT,UPDATE
AS
BEGIN
 IF (NOT  EXISTS(
  SELECT * FROM inserted INNER JOIN DayReservation
  ON DayReservation.DayReservationID = inserted.DayReservationID
  INNER JOIN ConferenceDays
  ON DayReservation.ConferenceID = ConferenceDays.ConferenceID
  AND DayReservation.ConfDayNumber = ConferenceDays.DayNumber
  INNER JOIN Workshops
  ON Workshops.ConferenceID = ConferenceDays.ConferenceID
  AND Workshops.ConfDayNumber = ConferenceDays.DayNumber
  AND Workshops.WorkshopID = inserted.WorkshopID
  ) )
 BEGIN
  ROLLBACK TRANSACTION
  ;THROW 52000,
  'Rezerwowany warsztat nie nale◆y do dnia konferencji, kt◆rego
dotyczy rezerwacja',1;
 END
END
GO
```

Sprawdza czy osoba dodana do rezerwacji nie przekracza limitu osób.

```
CREATE TRIGGER ParticipantReservationLimit
ON Participants
AFTER INSERT,UPDATE
AS
BEGIN
  IF ( EXISTS ( SELECT * FROM inserted
    JOIN  Student
    ON Student.ParticipantID = inserted.ParticipantID))
  BEGIN --przypadek ze studentem
   IF ( (SELECT COUNT( *)
    FROM inserted JOIN Participants
    ON inserted.DayReservationID = Participants.DayReservationID)
    +1 > (SELECT StudentQuantity
      FROM DayReservation JOIN inserted
      on DayReservation.DayReservationID =
inserted.DayReservationID
         ))
   BEGIN
```

```
    ;THROW 52000,
    'Limit miejsc w rezerwacji dnia dla ilo◆◆ student◆w',1;
   END
  END

  IF (NOT EXISTS ( SELECT * FROM inserted
    JOIN  Student
    ON Student.ParticipantID = inserted.ParticipantID))
  BEGIN --przypadek z niestudentem
   IF ( (SELECT COUNT( *)
    FROM inserted JOIN Participants
    ON inserted.DayReservationID = Participants.DayReservationID)
    +1 > (SELECT NormalQuantity
      FROM DayReservation JOIN inserted
      on DayReservation.DayReservationID =
inserted.DayReservationID
        ))
   BEGIN
    ;THROW 52000,
    'Limit miejsc w rezerwacji dnia dla ilo◆◆ normalnych
bilet◆w',1;
   END
  END

END
GO
```

Sprawdza limit miejsc przy dodawaniu uczestnika warsztatu.

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

   IF ( (SELECT COUNT( *)
    FROM inserted JOIN WorkshopParticipant
    ON inserted.WorkshopReservationID =
WorkshopParticipant.WorkshopReservationID)
    +1 > (SELECT Quantity
      FROM WorkshopReservation JOIN inserted
      on WorkshopReservation.WorkshopReservationID=
inserted.WorkshopReservationID
        ))
   BEGIN
    ROLLBACK TRANSACTION
```

```
    ;THROW 52000,
    'Limit miejsc w rezerwacji warsztatu!',1;
   END
  END


GO
```

Sprawdza czy uczestnik nie próbuje zostać dodany na równoległe warsztaty.

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

  DECLARE @EndHour time (7)= (
   SELECT Endhour
   FROM inserted JOIN WorkshopReservation
   ON inserted.WorkshopReservationID =
WorkshopReservation.WorkshopReservationID
   JOIN Workshops
   ON Workshops.WorkshopID = WorkshopReservation.WorkshopID)

  DECLARE @BeginHour time(7)=  (
   SELECT BeginHour
   FROM inserted JOIN WorkshopReservation
   ON inserted.WorkshopReservationID =
WorkshopReservation.WorkshopReservationID
   JOIN Workshops
   ON Workshops.WorkshopID = WorkshopReservation.WorkshopID)

  IF ( EXISTS (
   SELECT *
   FROM inserted JOIN WorkshopReservation as PartRes
   ON inserted.WorkshopReservationID =
PartRes.WorkshopReservationID
   JOIN DayReservation
   ON DayReservation.DayReservationID = PartRes.DayReservationID
   JOIN WorkshopReservation as WorkRes
   ON DayReservation.DayReservationID = WorkRes.DayReservationID
   JOIN Workshops
   ON Workshops.WorkshopID = WorkRes.WorkshopID
   WHERE (BeginHour BETWEEN @BeginHour AND @EndHour )
   OR (EndHour BETWEEN @BeginHour AND @EndHour)
   AND (WorkRes.WorkshopReservationID <>
inserted.WorkshopReservationID )) )
```

```
  BEGIN
    ROLLBACK TRANSACTION
    ;THROW 52000,
    'Ten uczestnik jest zarezerwowany na warsztat w przedziale
    godzin warsztatu.',1;
  END
 END

GO
```

11.Uprawnienia

Administrator:

- posiada dostęp do każdego widoku, procedury i funkcji.

Organizator:

-posiada dostęp do każdego widoku.
- z procedur nie ma dostępu tylko do :
a)procCreateReservation
b)procCreateDayReservation
c)procCreaateDayReservationInd
d)procCreateWorkshopReservation
e)procCreateWorkshopReservationIndiv
-funkcje z widokami:
a)funcConferenceDaysOfConference
b)funcParticipantsInConference
c)funcParticipantsInConferenceDay
d)funcConferencePerTimeFrame
e)funcParticipantsInWorkshop
f)funcConferencePerDay
-funkcje:
posiada uprawnienia na wszystkie funkcje poza funcPayment.

Klient indywidualny:

-posiada dostep do widoków:
a)conferencePrices
b)placesLeftPerConference,
c)upcomingConferences,
d)upcomingWorkshops
-posiada dostęp do procedur:
a)procCreateIndividual,
b)procCreateDayReservationIndiv,

c)procCreateReservation,
e)procCreateWorkshopReservationIndiv
f)procRemoveReservation,
g)procRemoveDayaReservation,
h)procRemoveWorkshopReservation
-funkcje z widokami:
a)funcActualConferenceDaysForPerson
b)funcActualWorkshopsForPerson
c)funcConferenceDaysForReservation

d)funcActualReservationsForCustomer
f)funcConferencePerDay
g)funcInvoice
-funkcje:
a)funcConfDayFreePlaces
b)funcConferencePrice
c)funcConferencePriceDiscount
d)funcPayment

Klient firma:

-posiada dostep do widoków:
a)conferencePrices
b)placesLeftPerConference,
c)upcomingConferences,
d)upcomingWorkshops
-posiada dostęp do procedur:
a)procCreateCompany,
b)procCreateDayReservation,
c)procCreateReservation,
e)procCreateWorkshopReservation,
f)procRemoveReservation,
g)procRemoveDayaReservation,
h)procRemoveWorkshopReservation
i)RemoveParticipant,
j)RemoveWorkshopParticipant
-funkcje z widokami:
a)funcActualConferenceDaysForPerson
b)funcActualWorkshopsForPerson
c)funcConferenceDaysForReservation
d)funcActualReservationsForCustomer
f)funcConferencePerDay
g)funcInvoice
-funkcje:
a)funcConfDayFreePlaces
b)funcConferencePrice
c)funcConferencePriceDiscount

d)funcPayment

Uczestnik konferencji:

-nie posiada dostępu do żadnego widoku, ani procedury.

-funkcje z widokami:

a)funcConferencesPerDay

b)funcConferencesPerTimeFrame

c)funcActualWorkshopsForPerson.

-nie posiada uprawnień do reszty funkcji.