PODSTAWY BAZ DANYCH

Dokumentacja projektu – system zarządzania konferencjami

Jakub Raban, Paweł Długosz

Spis treści

Opis problemu

Projekt dotyczy systemu wspomagania działalności firmy organizującej konferencje:

Ogólne informacje

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.

Warsztaty

Ponadto z konferencją związane są warsztaty, na które uczestnicy także mogą się zarejestrować - muszą być jednak zarejestrowani tego dnia na konferencję, aby móc w nich uczestniczyć. Kilka warsztatów może trwać równocześnie, ale uczestnik nie może zarejestrować się na więcej niż jeden warsztat, który trwa w tym samym czasie. Jest także ograniczona ilość miejsc na każdy warsztat i na każdy dzień konferencji. Część warsztatów może być płatna, a część jest darmowa.

Opłaty

Opłata za udział w konferencji zależy nie tylko od wykupionych usług, ale także od terminu ich wykupienia - jest kilka progów ceny (progi ceny dotyczą tylko udziału w konferencji, cena warsztatów jest stała) i im bliżej rozpoczęcia konferencji, tym cena jest wyższa (jest także zniżka procentowa dla studentów i wtedy przy rejestracji trzeba podać nr legitymacji studenckiej). Na zapłatę klienci mają tydzień od rejestracji na konferencję - jeśli do tego czasu nie pojawi się opłata, rejestracja jest anulowana.

Raporty

Dla organizatora najbardziej istotne są listy osobowe uczestników na każdy dzień konferencji i na każdy warsztat, a także informacje o płatnościach klientów. Ponadto organizator chciałby mieć informację o klientach, którzy najczęściej korzystają z jego usług.

Specyfika firmy

Firma organizuje średnio 2 konferencje w miesiącu, każda z nich trwa zwykle 2-3 dni, w tym średnio w każdym dniu są 4 warsztaty. Na każdą konferencję średnio rejestruje się 200 osób. Stworzona baza danych powinna zostać wypełniona w takim stopniu, aby odpowiadała 3-letniej działalności firmy.

Aktorzy

Administrator

Osoba odpowiedzialna za stworzenie bazy danych oraz zarządzanie nią. Ma pełny dostęp do danych oraz kodu źródłowego. Funkcje:

- Wgląd do kodu źródłowego
- Możliwość modyfikacji bazy danych
- Dostęp do wszystkich danych
- Możliwość przyznawania i odbierania uprawnień

Organizator konferencji

Osoba wyznaczona przez firmę do organizacji konkretnej konferencji. Funkcje:

- Generowanie listy uczestników konferencji oraz warsztatów
- Dostęp do statystyk dotyczących ilości sprzedanych biletów

Pracownik firmy organizującej konferencję

Każdy pracownik z firmy organizującej konferencję. Ma uprawnienia do przetwarzania niewrażliwych danych z bazy danych. Funkcje:

- Dodawanie nowych uczestników do konferencji oraz do warsztatów
- Dostęp do list uczestników konferencji oraz warsztatów
- Wgląd w opłaty wnoszone przez klientów

Klient indywidualny

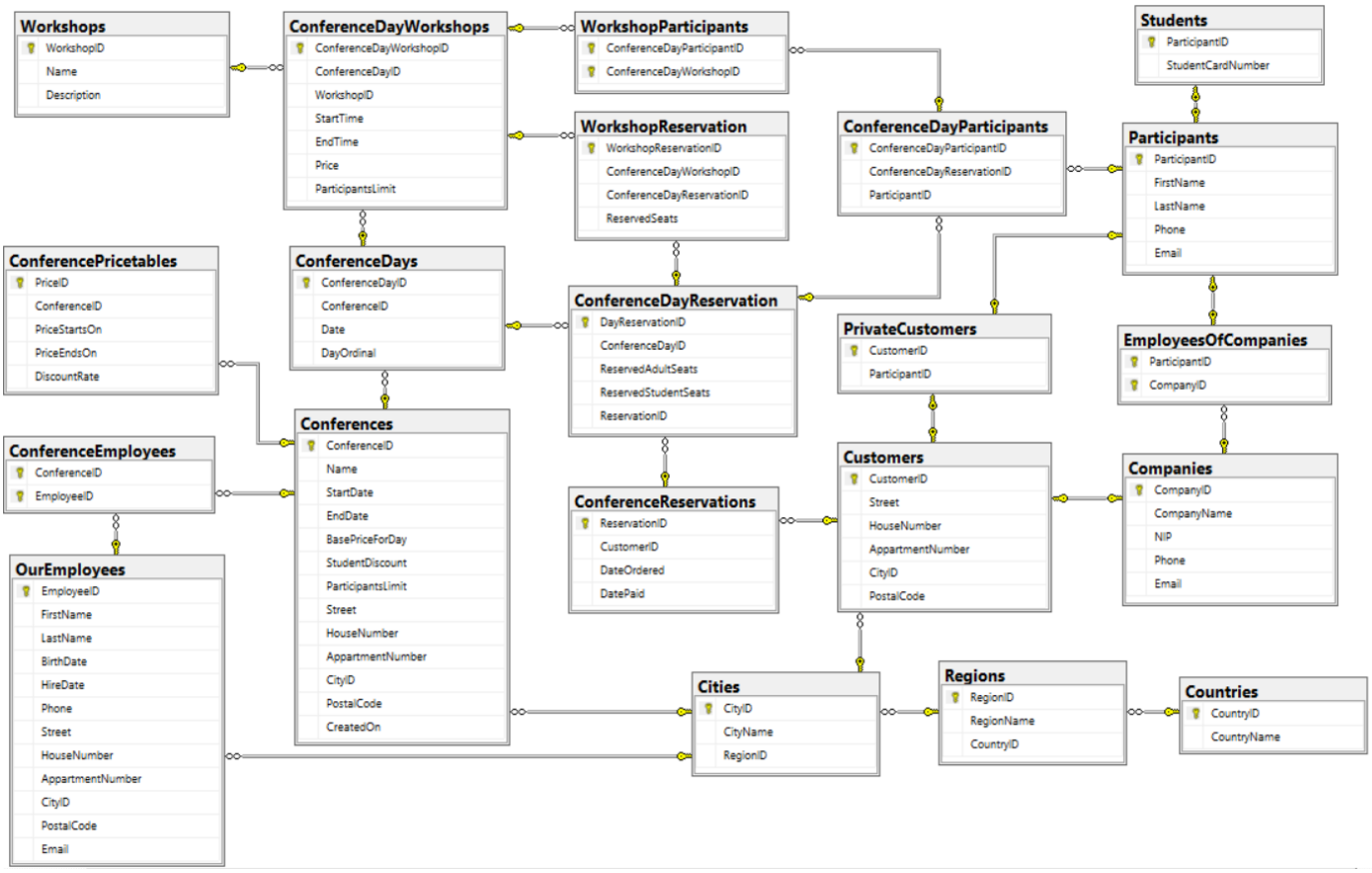
Osoba prywatna chcąca wziąć udział w konferencji. Funkcje:

- Rejestracja on-line
- Podawanie, uzupełnianie i prośba o aktualizacje danych
- Rezerwacja pojedynczego miejsca na konferencję i jej warsztaty
- Przeglądanie historii

Klient firmowy

- Rejestracja on-line
- Podawanie, uzupełnianie i prośba o aktualizacje danych
- Rezerwacja wielu miejsc na konferencję i warsztat
- Uzupełnienie danych uczestników
- Przeglądanie historii

Schemat bazy



Opisy tabel, warunki integralnościowe, constrainty, triggery i indeksy

Cities

Tabela z miastami oraz identyfikatorami regionów w których się znajdują.

```
create table Cities
(
  CityID   int identity (0, 1)
    primary key,
  CityName varchar(80) not null,
  RegionID int         not null
    constraint FK__Cities__RegionID__7CD98669
      references Regions
)
go
```

Companies

Firmy wraz z charakterystycznymi dla firmy danymi (nazwa, NIP)

```
create table Companies
(
  CompanyID   int         not null
    constraint PK__Companie__2D971C4C89B49313
      primary key
    constraint FK__Companies__Compa__1D7B6025
      references Customers,
  CompanyName nvarchar(150),
  NIP         char(10)    not null
    constraint UQ__Companie__C7DEC3C65B7646BC
      unique
    constraint CHK_NIP
      check (NOT [NIP] like '%[^0-9]%'),
  Phone       varchar(15) not null
    constraint UQ__Companie__5C7E359E2281BCA6
      unique
    constraint PhoneFormat
      check (NOT [phone] like '%[^0-9]%'),
  Email       varchar(100)
    constraint UQ_EMAIL
      unique
    constraint EmailFormat2
      check ([Email] like '%_@__%.__%')
)
go
```

ConferenceDayReservation

Tabela zawierająca informacje o tym, na który dzień jest zapisana, jej rozmiarze (ilości miejsc normalnych i studenckich). Zawiera triggera, który dodaje puste krotki (przeznaczone do uzupełnienia) do tabeli Participants.

```
create table ConferenceDayReservation
(
  DayReservationID     int identity (0, 1)
    constraint PK__Conferen__5572EBDB96C393BD
      primary key,
  ConferenceDayID      int not null
    constraint FK__Conferenc__Confe__41B8C09B
      references ConferenceDays
    constraint CHK_SIZE_OK
      check ([dbo].[ReservedSeatsPerConferenceDay]([ConferenceDayID]) <=
[dbo].[ConferenceSize]([ConferenceDayID])),
  ReservedAdultSeats   int not null,
  ReservedStudentSeats int not null,
  ReservationID        int not null
    constraint FK__Conferenc__Reser__66EA454A
```

```
      references ConferenceReservations,
  constraint CHK_RESERVATION
    check ([ReservedAdultSeats] >= 0 AND [ReservedStudentSeats] >= 0 AND
           ([ReservedAdultSeats] + [ReservedStudentSeats]) > 0),
  constraint CHK_RESERVED_BY_COMPANY_OR_ONE
    check ([dbo].[IsReservationByCompany]([ReservationID]) = 1 OR ([ReservedAdultSeats]
+ [ReservedStudentSeats]) = 1)
)
go

CREATE trigger [dbo].[InsertParticipantsForReservation] on
[dbo].[ConferenceDayReservation]
for insert
as
declare @AdultPointer int = 1,
  @StudentPointer int = 1,
  @NumberOfAdults int = (select ReservedAdultSeats from inserted),
  @NumberOfStudents int = (Select ReservedStudentSeats from inserted),
  @NewParticipantID int;
while @AdultPointer <= @NumberOfAdults begin
 insert into Participants default values
 set @NewParticipantID = (select max(ParticipantID) from Participants)
 insert into ConferenceDayParticipants (ConferenceDayReservationID, ParticipantID)
 values ((select DayReservationID from inserted), @NewParticipantID)
 set @AdultPointer = @AdultPointer + 1
end
while @StudentPointer <= @NumberOfStudents begin
 insert into Participants default values
 set @NewParticipantID = (select max(ParticipantID) from Participants)
 insert into Students (ParticipantID) values (@NewParticipantID)
 insert into ConferenceDayParticipants (ConferenceDayReservationID, ParticipantID)
 values ((select DayReservationID from inserted), @NewParticipantID)
 set @StudentPointer = @StudentPointer + 1
end
go
```

ConferenceDays

Znajdują się w niej poszczególne dni każdej konferencji wraz z informacją o dacie i kolejności dnia w całej konferencji.

```
create table ConferenceDays
(
  ConferenceDayID int identity (0, 1)
    constraint PK__Conferen__E57A6462D2FB2DE1
      primary key,
  ConferenceID    int  not null
    constraint FK__Conferenc__Confe__3EDC53F0
      references Conferences,
  Date            date not null,
  DayOrdinal      smallint
    constraint CHK_DAY_ORDINAL
      check ([dayordinal] > 0),
  constraint CHK_DAY_ORD_UNIQ
    unique (ConferenceID, DayOrdinal)
)
go
```

ConferenceDayWorkshops

Zawiera informacje o warsztatach w danym dniu. Zawiera klucz obcy do tabeli słownikowej Workshops oraz do ConferenceDays. Ma informacje o dacie, czasie, cenie i limicie miejsc.

```
create table ConferenceDayWorkshops
(
  ConferenceDayWorkshopID int identity (0, 1)
    constraint PK__Conferen__714B153C1B305831
```

```
    primary key,
  ConferenceDayID         int  not null
    constraint FK__Conferenc__Confe__4E1E9780
      references ConferenceDays,
  WorkshopID              int  not null
    constraint FK__Conferenc__Works__4F12BBB9
      references Workshops,
  StartTime               time not null,
  EndTime                 time not null,
  Price                   money
    constraint DEF_WKSH_PRICE default 0,
  ParticipantsLimit       int,
  constraint CHK_SIZES_NON_NEGATIVE
    check ([ParticipantsLimit] > 0 AND [price] >= 0),
  constraint CHK_SIZE_SMALLER_THAN_CONF
    check ([dbo].[ConferenceSize]([ConferenceDayID]) >= [ParticipantsLimit]),
  constraint CHK_TIME
    check ([StartTime] < [EndTime])
)
go
```

ConferenceEmployees

Przyporządkowuje pracowników firmy do obsługi danej konferencji.

```
create table ConferenceEmployees
(
  ConferenceID int not null
    constraint FK__Conferenc__Confe__4865BE2A
      references Conferences,
  EmployeeID   int not null
    constraint FK__Conferenc__Emplo__4959E263
      references OurEmployees,
  primary key (ConferenceID, EmployeeID)
)
go
```

ConferencePricetables

Zawiera informację o progach cenowych każdej z konferencji – ich początki i końce oraz stopień zniżki.

```
create table ConferencePricetables
(
  PriceID       int identity (0, 1)
    primary key,
  ConferenceID  int  not null
    references Conferences,
  PriceStartsOn date not null,
  PriceEndsOn   date not null,
  DiscountRate  real not null
    check ([DiscountRate] >= 0 AND [DiscountRate] <= 1)
)
go
```

ConferenceDayParticipants

Przyporządkowuje uczestników do danej rezerwacji dnia konferencji. Zawiera triggera usuwającego pusty rekord wstawiony wcześniej do tabeli Participant jeśli dodany za pomocą procedury FillReservation uczestnik już istniał w bazie danych.

```
create table ConferenceDayParticipants
(
  ConferenceDayParticipantID int identity (0, 1)
    primary key,
  ConferenceDayReservationID int not null
```

```
    constraint FK__Conferenc__Confe__6CA31EA0
      references ConferenceDayReservation,
  ParticipantID             int not null
    constraint FK__Conferenc__Parti__6ABAD62E
      references Participants
)
go

CREATE TRIGGER DeleteEmptyRecordWhenFoundParticipant ON dbo.ConferenceDayParticipants
AFTER UPDATE
AS
DECLARE @p int
SELECT @p = ParticipantId FROM Deleted
DELETE FROM dbo.Participants
WHERE ParticipantID = @p AND LastName IS NULL
go
```

ConferenceReservations

Zawiera informacje o dacie zamówienia konferencji, płatności i zamawiającym kliencie.

```
create table ConferenceReservations
(
  ReservationID int identity (0, 1)
    primary key,
  CustomerID    int                                          not null
    constraint FK__Conferenc__Custo__65F62111
      references Customers,
  DateOrdered   date
    constraint DEF_ORDER_DATE default CONVERT([date], getdate()) not null,
  DatePaid      date,
  constraint CHK_PAID_SEVEN_DAYS_OR_FASTER
    check (datediff(day, [DateOrdered], [DatePaid]) <= 7)
)
go
```

Conferences

Zawiera informacje o konferencjach – czasie trwania, limicie miejsc, zniżce dla studenta oraz miejscu gdzie się odbywa. Zawiera dwa triggery – pierwszy tworzy wszystkie rekordy dni konferencji odpowiadające tej konferencji, drugi dodaje bazowy próg zniżkowy 100% na przeddzień daty dodania konferencji do bazy danych, dając podstawy do prawidłowego działania constraintów sprawdzających czy kolejne progi cenowe mają malejące zniżki.

```
create table Conferences
(
  ConferenceID      int identity (0, 1)
    primary key,
  Name              varchar(200) not null,
  StartDate         date         not null,
  EndDate           date         not null,
  BasePriceForDay   money
    constraint CHK_CONF_PRICE
      check ([BasePriceForDay] >= 0),
  StudentDiscount   real
    constraint DEF_BASEPRICE default 0
    constraint CK__Conferenc__Stude__336AA144
      check ([StudentDiscount] >= 0 AND [StudentDiscount] <= 1),
  ParticipantsLimit int
    constraint CHK_CONF_PARTICIP
      check ([ParticipantsLimit] > 0),
  Street            varchar(74),
  HouseNumber       varchar(5),
  AppartmentNumber  int,
  CityID            int
    references Cities,
```

```
  PostalCode        char(6)
    check ([PostalCode] like '[0-9][0-9]-[0-9][0-9][0-9]'),
  constraint CHK_CONF_DATES
    check (datediff(day, [StartDate], [EndDate]) >= 0)
)
go

CREATE TRIGGER BasicDiscountForConference ON Conferences AFTER INSERT
AS
INSERT INTO dbo.ConferencePricetables
(
    ConferenceID,
    PriceStartsOn,
    PriceEndsOn,
    DiscountRate
)
VALUES
(   (SELECT ConferenceID FROM Inserted),         -- ConferenceID - int
    (SELECT DATEADD(DAY, -1, CreatedOn) FROM Inserted), -- PriceStartsOn - date
    (SELECT DATEADD(DAY, -1, CreatedOn) FROM Inserted), -- PriceEndsOn - date
    1       -- DiscountRate - real
    )
go

create trigger InsertDaysForNewConference on Conferences
for insert
as

DECLARE @MinDate DATE,
        @MaxDate DATE,
  @DatePointer date,
  @DayOrdinal int = 1;

set @MinDate = (select startdate from inserted);
set @MaxDate = (select enddate from inserted);
set @DatePointer = @MinDate;

while @DatePointer <= @MaxDate begin
 insert into ConferenceDays (ConferenceID, Date, DayOrdinal)
 values (
  (select ConferenceID from inserted),
  @DatePointer,
  @DayOrdinal
 )
 set @DatePointer = DATEADD(day, 1, @DatePointer);
 set @DayOrdinal = @DayOrdinal + 1;
end;
go
```

Countries

Tabela z nazwami państw.

```
create table Countries
(
  CountryID   int identity (0, 1)
    primary key,
  CountryName varchar(80) not null
    constraint UQ_COUNTRY
      unique
)
go
```

Customers

Tabela z danymi adresowymi wszystkich klientów (prywatnych i firmowych), do której później odnoszą się relacje Company oraz PrivateCustomer.

```
create table Customers
(
  CustomerID       int identity (0, 1)
    primary key,
  Street           nvarchar(74),
  HouseNumber      nvarchar(5),
  AppartmentNumber int,
  CityID           int
    references Cities,
  PostalCode       char(6)
    check ([PostalCode] like '[0-9][0-9]-[0-9][0-9][0-9]')
)
go
```

EmployeesOfCompanies

Tabela wiążąca uczestników konferencji z ich firmami.

```
create table EmployeesOfCompanies
(
  ParticipantID int not null
    constraint IX_EmployeesOfCompanies
      unique
    constraint FK__Employees__Parti__2BC97F7C
      references Participants,
  CompanyID     int not null
    constraint FK__Employees__Compa__2CBDA3B5
      references Companies,
  primary key (ParticipantID, CompanyID)
)
go
```

OurEmployees

Tabela zawierająca dane pracowników firmy.

```
create table OurEmployees
(
  EmployeeID       int identity (0, 1)
    primary key,
  FirstName        varchar(30) not null,
  LastName         varchar(50) not null,
  BirthDate        date,
  HireDate         date,
  Phone            varchar(15) not null
    constraint CHK_EMP_PHONE_UNIQ
      unique
    constraint CHK_EMP_PHONE
      check (NOT [Phone] like '%[^0-9]%'),
  Street           varchar(74),
  HouseNumber      varchar(5),
  AppartmentNumber int,
  CityID           int
    references Cities,
  PostalCode       char(6)
    check ([PostalCode] like '[0-9][0-9]-[0-9][0-9][0-9]'),
  Email            varchar(100)
    constraint CHK_EMP_EMAIL_UNIQ
      unique
    check ([Email] like '%_@_%._%')
)
go
```

Participants

Tabela zawierająca imiona, nazwiska oraz dane kontaktowe uczestników.

```
create table Participants
(
  ParticipantID int identity (0, 1)
    constraint PK__Particip__7227997EFB4A4EAE
      primary key,
  FirstName     varchar(30),
  LastName      varchar(50),
  Phone         varchar(15)
    constraint PhoneFormat2
      check (NOT [phone] like '%[^0-9]%'),
  Email         varchar(100)
    constraint EmailFormat
      check ([Email] like '%_@__%.__%')
)
go

create unique index UQ_Email
  on Participants (Phone)
  where [Phone] IS NOT NULL
go
```

PrivateCustomers

Tabela zawierająca klucze obce do tych uczestników, którzy są klientami prywatnymi zamawiającymi konferencję. Łączy wpisy z tabeli Participants (dane osobowe) z danymi z tabeli Customers (dane adresowe).

```
create table PrivateCustomers
(
  CustomerID    int not null
    primary key
    constraint FK__PrivateCu__Custo__0B27A5C0
      references Customers,
  ParticipantID int not null
    constraint UniqueParticipant
      unique
    constraint FK__PrivateCu__Parti__0C1BC9F9
      references Participants
)
go
```

Regions

Tabela z nazwami regionów oraz identyfikatorami państw, w których one się znajdują.

```
create table Regions
(
  RegionID   int identity (0, 1)
    primary key,
  RegionName varchar(80) not null
    constraint UQ_REGION
      unique,
  CountryID  int         not null
    constraint FK__Regions__Country__79FD19BE
      references Countries
)
go
```

Students

Tabela zawierająca informację o tym którzy z uczestników są studentami oraz ich numery indeksów.

```
create table Students
(
  ParticipantID     int not null
    constraint PK__Student__7227997E8D9AA260
      primary key
    constraint FK__Student__Partici__28ED12D1
      references Participants,
  StudentCardNumber varchar(10)
)
go
```

WorkshopParticipants

Tabela przyporządkowująca tych uczestników dnia konferencji, którzy uczestniczą w jakimś warsztacie do tegoż warsztatu.

```
create table WorkshopParticipants
(
  ConferenceDayParticipantID int not null
    constraint FK__WorkshopP__Confe__54CB950F
      references ConferenceDayParticipants,
  ConferenceDayWorkshopID    int not null
    constraint FK__WorkshopP__Confe__55BFB948
      references ConferenceDayWorkshops,
  primary key (ConferenceDayParticipantID, ConferenceDayWorkshopID),
  constraint CHK_FREE_SEATS
    check ([dbo].[EmptySeatsInWorkshopReservation]([ConferenceDayParticipantID],
[ConferenceDayWorkshopID]) >= 0),
  constraint CHK_PARTICIPANT_NOT_IN_ANOTHER_WORKSHOP
    check ([dbo].[HasParticipantCollidingWorkshops]([ConferenceDayWorkshopID],
[ConferenceDayParticipantID]) = 0)
)
go
```

WorkshopReservation

Tabela zawierająca dane o rezerwacji warsztatu – z której rezerwacji dnia pochodzi, ile miejsc jest zarezerwowanych i na którą konferencję.

```
create table WorkshopReservation
(
  WorkshopReservationID      int identity (0, 1)
    primary key,
  ConferenceDayWorkshopID    int not null
    constraint FK__WorkshopR__Confe__6F7F8B4B
      references ConferenceDayWorkshops
    constraint CHK_ENOUGH_FREE_SEATS
      check ([dbo].[ReservedSeatsForWorkshop]([ConferenceDayWorkshopID]) <=
             [dbo].[WorkshopSeatsLimit]([ConferenceDayWorkshopID])),
  ConferenceDayReservationID int not null
    constraint FK__WorkshopR__Confe__7073AF84
      references ConferenceDayReservation,
  ReservedSeats              int not null
    constraint CHK_WORKSHOP_RESERV_NON_NEGATIVE
      check ([ReservedSeats] > 0),
  constraint UQ_ONE_DAY_RESERV_ONE_WORKSH_RESERV
    unique (ConferenceDayWorkshopID, ConferenceDayReservationID),
  constraint CHK_RESERVING_WORKSHOP_AT_CORRECT_CONF_DAY
    check
([dbo].[WorkshopReservationOnDayReservationConference]([ConferenceDayReservationID],

[ConferenceDayWorkshopID]) = 0),
  constraint CHK_WORKSHOP_RESERV_NOT_GREATER_THAN_DAY_RESERV
    check ([ReservedSeats] <=
[dbo].[ConferenceDayReservationSize]([ConferenceDayReservationID]))
```

```
)
go
```

Workshops

Relacja słownikowa z nazwami i opisami wszystkich organizowanych warsztatów.

```
create table Workshops
(
  WorkshopID  int identity (0, 1)
    constraint PK__Workshop__7A008C2A4FD1EE19
      primary key,
  Name        varchar(200) not null
    constraint UQ_NAME
      unique,
  Description varchar(1000)
)
go
```

Widoki

ConferencePlan

Wszystkie konferencje wraz z zaplanowanymi w trakcie ich trwania warsztatami i ich rozpiętością czasową.

```
CREATE VIEW ConferencePlan AS
SELECT Conferences.Name AS 'Conference name',
    Date,
    ISNULL(dbo.Workshops.Name, '[No workshops at that day]') AS 'Workshop name',
    StartTime AS 'Start time',
    EndTime AS 'End time',
    ISNULL(Description, '---') AS 'Description'
FROM dbo.Conferences
left JOIN dbo.ConferenceDays ON ConferenceDays.ConferenceID = Conferences.ConferenceID
left JOIN dbo.ConferenceDayWorkshops ON ConferenceDayWorkshops.ConferenceDayID =
ConferenceDays.ConferenceDayID
LEFT JOIN dbo.Workshops ON Workshops.WorkshopID = ConferenceDayWorkshops.WorkshopID
go
```

ConferencesWithAvailablePlaces

Zawiera informacje o tym ile miejsc zostało zarezerwowanych, a ile jest wolnych na każdą z konferencji.

```
CREATE VIEW [dbo].[ConferencesWithAvailablePlaces]
AS
SELECT Conferences.ConferenceID, Name, DayOrdinal AS Day, ConferenceDays.Date AS
'Date', ParticipantsLimit AS 'Seat limit',
 ISNULL(SUM(ReservedAdultSeats), 0) AS 'Reserved adult seats',
 ISNULL(SUM(ReservedStudentSeats), 0) AS 'Reserved student seats',
 ISNULL(SUM(ReservedAdultSeats + ReservedStudentSeats), 0) AS 'Total seats reserved',
 ParticipantsLimit - ISNULL(SUM(ReservedAdultSeats + ReservedStudentSeats), 0) AS
'Available seats'
FROM Conferences
JOIN ConferenceDays
ON Conferences.ConferenceID = ConferenceDays.ConferenceID
LEFT JOIN ConferenceDayReservation
ON ConferenceDays.ConferenceDayID = ConferenceDayReservation.ConferenceDayID
WHERE Conferences.EndDate >= CONVERT(DATE, GETDATE())
GROUP BY Conferences.ConferenceID, Name, DayOrdinal, ParticipantsLimit,
ConferenceDays.Date
go
```

CustomerContactData

Zawiera dane kontaktowe wszystkich klientów z wyszczególnieniem czy jest to klient prywatny czy firma.

```
CREATE VIEW [dbo].[CustomerContactData] AS
SELECT CustomerID, CompanyName AS 'Name',
    'Company' AS 'Customer type',
    Street + ' ' + HouseNumber + ISNULL('/' + CAST(AppartmentNumber AS VARCHAR), '') +
', ' + PostalCode + ' ' + CityName + ', ' + RegionName + ', ' + CountryName AS
'Address',
    Phone,
    Email
FROM dbo.Customers
INNER JOIN dbo.Companies ON Companies.CompanyID = Customers.CustomerID
INNER JOIN dbo.Cities ON Cities.CityID = Customers.CityID
INNER JOIN dbo.Regions ON Regions.RegionID = Cities.RegionID
INNER JOIN dbo.Countries ON Countries.CountryID = Regions.CountryID
UNION
SELECT Customers.CustomerID, FirstName + ' ' + LastName AS 'Name',
    'Private Customer' AS 'Customer type',
    Street + ' ' + HouseNumber + ISNULL('/' + CAST(AppartmentNumber AS VARCHAR), '') +
', ' + PostalCode + ' ' + CityName + ', ' + RegionName + ', ' + CountryName AS
'Address',
    Phone,
    Email
FROM dbo.Customers
INNER JOIN dbo.PrivateCustomers ON PrivateCustomers.CustomerID = Customers.CustomerID
INNER JOIN dbo.Participants ON Participants.ParticipantID =
PrivateCustomers.ParticipantID
INNER JOIN dbo.Cities ON Cities.CityID = Customers.CityID
INNER JOIN dbo.Regions ON Regions.RegionID = Cities.RegionID
INNER JOIN dbo.Countries ON Countries.CountryID = Regions.CountryID
go
```

CustomerWithPaidReservations

Widok ‘najaktywniejszych klientów’ – pokazuje ile opłaconych zamówień mają dani klienci.

```
CREATE VIEW [dbo].[CustomersWithPaidReservations]
AS
SELECT CompanyName AS Customer, 'Company' AS 'Customer type',
dbo.GetNumberOfPaidReservationForCustomer(Companies.Email) AS PaidReservations
FROM Customers
JOIN Companies
ON Customers.CustomerID = Companies.CompanyID
UNION
SELECT (FirstName + ' ' + LastName) AS Customer,
'Private customer' AS 'Customer type',
dbo.GetNumberOfPaidReservationForCustomer(Participants.Email) AS PaidReservations
FROM Customers
JOIN PrivateCustomers
ON Customers.CustomerID = PrivateCustomers.CustomerID
JOIN Participants
ON PrivateCustomers.ParticipantID = Participants.ParticipantID
go
```

DayReservationData

Pokazuje informacje o rezerwacjach dnia konferencji wraz z danymi klienta.

```
CREATE VIEW DayReservationData AS
SELECT DISTINCT Name AS 'Name', [Customer type], Address, Phone, Email, DateOrdered,
DatePaid,
  [Conference name], Date, DayOrdinal,
  ReservedAdultSeats, ReservedStudentSeats
FROM DayWorkshopReservationData
go
```

DayWorkshopReservationData

Widok pomocniczy dla widoków DayReservationData, WorkshopReservationData

```
CREATE VIEW [dbo].[DayWorkshopReservationData] AS
SELECT CustomerContactData.Name AS 'Name', [Customer type], Address, Phone, Email,
DateOrdered, DatePaid,
  Conferences.Name AS 'Conference name', Date, DayOrdinal,
  cdr.ReservedAdultSeats, cdr.ReservedStudentSeats, Workshops.name AS 'Workshop name',
ConferenceDays.Date AS 'Workshop date',
  dbo.WorkshopReservation.ReservedSeats
FROM dbo.ConferenceReservations
INNER JOIN dbo.ConferenceDayReservation cdr ON ConferenceReservations.ReservationID =
cdr.ReservationID
INNER JOIN dbo.CustomerContactData ON ConferenceReservations.CustomerID =
CustomerContactData.CustomerID
INNER JOIN dbo.ConferenceDays ON ConferenceDays.ConferenceDayID = cdr.ConferenceDayID
INNER JOIN dbo.Conferences ON Conferences.ConferenceID = ConferenceDays.ConferenceID
LEFT OUTER JOIN dbo.WorkshopReservation ON
WorkshopReservation.ConferenceDayReservationID = cdr.DayReservationID
LEFT OUTER JOIN dbo.ConferenceDayWorkshops ON
ConferenceDayWorkshops.ConferenceDayWorkshopID =
dbo.WorkshopReservation.ConferenceDayWorkshopID
LEFT OUTER JOIN dbo.Workshops ON Workshops.WorkshopID =
ConferenceDayWorkshops.WorkshopID
go
```

EmployeesInDuty

Pokazuje którzy pracownicy którą konferencję obsługują.

```
CREATE VIEW EmployeesInDuty AS
SELECT FirstName + ' ' + LastName AS 'Employee', Name
FROM dbo.ConferenceEmployees
INNER JOIN dbo.OurEmployees ON OurEmployees.EmployeeID = ConferenceEmployees.EmployeeID
INNER JOIN dbo.Conferences ON Conferences.ConferenceID =
ConferenceEmployees.ConferenceID
go
```

FrequentCustomers

Pokazuje 10 najczęściej rezerwujących klientów.

```
CREATE VIEW [dbo].[FrequentCustomers] AS
SELECT TOP 10 ISNULL(companyname,'') + ISNULL(firstname + ' ','') + ISNULL(lastname,'')
AS 'Customer Name',
     COUNT(reservationid) AS 'Number of paid reservations',
     ISNULL(Participants.Email,'') + ISNULL(dbo.Companies.Email,'') AS 'E-mail',
     ISNULL(dbo.Participants.Phone,'') + ISNULL(dbo.Companies.Phone,'') AS 'Phone'
FROM Customers
INNER JOIN ConferenceReservations
 ON Customers.CustomerID = ConferenceReservations.CustomerID
LEFT JOIN PrivateCustomers
 ON Customers.CustomerID = PrivateCustomers.CustomerID
LEFT JOIN Participants
 ON PrivateCustomers.ParticipantID = Participants.ParticipantID
LEFT JOIN Companies
 ON Customers.CustomerID = Companies.CompanyID
WHERE DatePaid IS NOT NULL
GROUP BY firstname, lastname, companyname, dbo.Participants.Email, dbo.Companies.Email,
dbo.Companies.Phone, dbo.Participants.Phone
ORDER by count(ReservationID) desc
go
```

OrdersToBeDeleted

Pokazuje zamówienia nieopłacone w terminie

```
CREATE VIEW OrdersToBeDeleted AS
SELECT * FROM dbo.UnpaidReservations
WHERE DATEDIFF(DAY, DateOrdered, CONVERT(DATE, GETDATE())) > 7
go
```

OrganisingCities

Miasta w których odbywają się konferencje wraz z ilością zorganizowanych tam konferencji.

```
CREATE VIEW OrganisingCities AS
SELECT CityName, RegionName, CountryName, COUNT(ConferenceID) AS 'Number of organised
conferences'
FROM dbo.Cities
INNER JOIN dbo.Regions ON Regions.RegionID = Cities.RegionID
INNER JOIN dbo.Countries ON Countries.CountryID = Regions.CountryID
INNER JOIN dbo.Conferences ON Conferences.CityID = Cities.CityID
GROUP BY CityName, RegionName, CountryName
go
```

ParticipantData

Dane osobowe wszystkich uczestników.

```
CREATE VIEW ParticipantData AS
SELECT Conferences.Name AS 'Conference name', ConferenceDays.Date AS 'Date', FirstName
+ ' ' + LastName AS 'Name', Phone, Email
FROM dbo.ConferenceDayParticipants
INNER JOIN dbo.Participants ON Participants.ParticipantID =
ConferenceDayParticipants.ParticipantID
INNER JOIN dbo.ConferenceDayReservation ON ConferenceDayReservation.DayReservationID =
ConferenceDayParticipants.ConferenceDayReservationID
INNER JOIN dbo.ConferenceDays ON ConferenceDays.ConferenceDayID =
ConferenceDayReservation.ConferenceDayID
INNER JOIN dbo.Conferences ON Conferences.ConferenceID = ConferenceDays.ConferenceID
WHERE LastName IS NOT NULL AND EndDate >= CONVERT(DATE, GETDATE())
go
```

ParticipantIdentificators

Identyfikatory dla uczestników konferencji.

```
CREATE VIEW [dbo].[ParticipantIdentificators]
AS
SELECT Name AS 'Conference Name', Date, FirstName, LastName, CompanyName,
StudentCardNumber
FROM Conferences
JOIN ConferenceDays
ON Conferences.ConferenceID = ConferenceDays.ConferenceID
JOIN ConferenceDayReservation
ON ConferenceDays.ConferenceDayID = ConferenceDayReservation.ConferenceDayID
JOIN ConferenceDayParticipants
ON ConferenceDayReservation.DayReservationID =
ConferenceDayParticipants.ConferenceDayReservationID
JOIN Participants
ON ConferenceDayParticipants.ParticipantID = Participants.ParticipantID
LEFT JOIN dbo.Students
ON Students.ParticipantID = Participants.ParticipantID
LEFT JOIN EmployeesOfCompanies
ON Participants.ParticipantID = EmployeesOfCompanies.ParticipantID
LEFT JOIN Companies
ON EmployeesOfCompanies.CompanyID = Companies.CompanyID
WHERE LastName IS NOT NULL AND date >= CONVERT(DATE, GETDATE())
go
```

Payments

Dane o płatnościach dokonywanych przez klientów.

```
CREATE view [dbo].[Payments] as
select ReservationID, CompanyName, DateOrdered, DatePaid,
 dbo.CalculatePriceForReservation( Companies.Email, DateOrdered) as Price
from ConferenceReservations
join Customers
on ConferenceReservations.CustomerID = Customers.CustomerID
join Companies
on Customers.CustomerID = Companies.CompanyID
union
select ReservationID, (FirstName + ' ' + LastName), DateOrdered, DatePaid,
 dbo.CalculatePriceForReservation( Participants.Email, DateOrdered) as Price
from ConferenceReservations
join Customers
on ConferenceReservations.CustomerID = Customers.CustomerID
join PrivateCustomers
on Customers.CustomerID = PrivateCustomers.CustomerID
join Participants
on PrivateCustomers.ParticipantID = Participants.ParticipantID
go
```

TwoWeekOldReservationsWithoutAllParticipants

Widok pokazujący klientów do których muszą zadzwonić pracownicy w związku z nieuzupełnionymi rezerwacjami.

```
CREATE view TwoWeekOldReservationsWithoutAllParticipants as
select cdp.ConferenceDayReservationID as 'Conference Day Reservation ID',
    (select 2 from (select cdpp.ConferenceDayReservationID as x1,
count(cdpp.ParticipantID) as y1
        from ConferenceDayParticipants cdpp
        inner join Participants p
       on P.ParticipantID = cdpp.ParticipantID
        left join Students
       on p.ParticipantID = students.ParticipantID
        where LastName is null and students.ParticipantID is null
        group by cdpp.ConferenceDayReservationID) as t where t.x1 =
cdp.ConferenceDayReservationID) as 'Adult Seats Left',
    (select 2 from (select cdpp.ConferenceDayReservationID as x2,
count(cdpp.ParticipantID) as y2
        from ConferenceDayParticipants cdpp
        inner join Participants p
       on P.ParticipantID = cdpp.ParticipantID
        inner join Students
       on p.ParticipantID = students.ParticipantID
        where LastName is null
        group by cdpp.ConferenceDayReservationID) as t where t.x2 =
cdp.ConferenceDayReservationID) as 'Student seats left',
    c.Phone
from ConferenceDayParticipants cdp
inner join ConferenceDayReservation cdr
 on cdp.ConferenceDayReservationID = cdr.DayReservationID
inner join ConferenceReservations cr
 on cdr.ReservationID = cr.ReservationID
inner join Customers cust
 on cr.CustomerID = cust.CustomerID
inner join Companies c
 on c.CompanyID = cust.CustomerID
where datediff(day, cr.dateordered, convert(date, getdate())) > 14 and (
(select 2 from (select cdpp.ConferenceDayReservationID as x2, count(cdpp.ParticipantID)
as y2
        from ConferenceDayParticipants cdpp
        inner join Participants p
```

```
       on P.ParticipantID = cdpp.ParticipantID
        inner join Students
       on p.ParticipantID = students.ParticipantID
        where LastName is null
        group by cdpp.ConferenceDayReservationID) as t where t.x2 =
cdp.ConferenceDayReservationID) > 0
        or
(select 2 from (select cdpp.ConferenceDayReservationID as x1, count(cdpp.ParticipantID)
as y1
        from ConferenceDayParticipants cdpp
        inner join Participants p
       on P.ParticipantID = cdpp.ParticipantID
        left join Students
       on p.ParticipantID = students.ParticipantID
        where LastName is null and students.ParticipantID is null
        group by cdpp.ConferenceDayReservationID) as t where t.x1 =
cdp.ConferenceDayReservationID) > 0)
go
```

UnpaidReservations

Pokazuje listę zamówień jeszcze nieopłaconych przez klientów.

```
create view UnpaidReservations
as
select ReservationID, CompanyName, DateOrdered
from ConferenceReservations
join Customers
on ConferenceReservations.CustomerID = Customers.CustomerID
join Companies
on Customers.CustomerID = Companies.CompanyID
where DatePaid is null
union
select ReservationID, (FirstName + ' ' + LastName), DateOrdered
from ConferenceReservations
join Customers
on ConferenceReservations.CustomerID = Customers.CustomerID
join PrivateCustomers
on Customers.CustomerID = PrivateCustomers.CustomerID
join Participants
on PrivateCustomers.ParticipantID = Participants.ParticipantID
where DatePaid is null
go
```

WorkshopReservationData

Widok pokazujący nazwę klienta, typ (prywatny/firmowy),

```
CREATE VIEW WorkshopReservationData AS
SELECT Name AS 'Name', [Customer type], Address, Phone, Email, DateOrdered, DatePaid,
    [Conference name], Date, DayOrdinal,
    [Workshop name], [Workshop date],
    [ReservedSeats]
FROM DayWorkshopReservationData
WHERE [Workshop name] IS NOT null
go
```

WorkshopsParticipantsList

Pokazuje listy osobowe uczestników zapisanych na warsztaty

```
CREATE VIEW [dbo].[WorkshopsParticipantsList]
AS
SELECT Conferences.Name AS 'Conference name', dbo.Workshops.Name AS 'Workshop name',
dbo.ConferenceDays.Date, dbo.ConferenceDayWorkshops.StartTime,
  dbo.ConferenceDayWorkshops.EndTime,
```

```
  FirstName, LastName, StudentCardNumber, CompanyName
FROM dbo.WorkshopParticipants
INNER JOIN dbo.ConferenceDayWorkshops ON ConferenceDayWorkshops.ConferenceDayWorkshopID
= WorkshopParticipants.ConferenceDayWorkshopID
INNER JOIN dbo.Workshops ON Workshops.WorkshopID = ConferenceDayWorkshops.WorkshopID
INNER JOIN dbo.ConferenceDays ON ConferenceDays.ConferenceDayID =
ConferenceDayWorkshops.ConferenceDayID
INNER JOIN dbo.Conferences ON Conferences.ConferenceID = ConferenceDays.ConferenceID
INNER JOIN dbo.ConferenceDayParticipants ON
ConferenceDayParticipants.ConferenceDayParticipantID =
WorkshopParticipants.ConferenceDayParticipantID
INNER JOIN dbo.Participants ON Participants.ParticipantID =
ConferenceDayParticipants.ParticipantID
LEFT JOIN dbo.Students ON Students.ParticipantID = Participants.ParticipantID
LEFT JOIN dbo.EmployeesOfCompanies ON EmployeesOfCompanies.ParticipantID =
Participants.ParticipantID
LEFT JOIN dbo.Companies ON Companies.CompanyID = EmployeesOfCompanies.CompanyID
go
```

WorkshopsWithAvailablePlaces

Pokazuje informację o ilości miejsc zarezerwowanych na każdy z warsztatów i ilości wolnych miejsc.

```
CREATE VIEW [dbo].[WorkshopsWithAvailablePlaces]
AS
SELECT Conferences.Name AS 'Conference Name',
 Date, Workshops.Name AS 'Workshop Name',  StartTime, EndTime, Price,
 ConferenceDayWorkshops.ParticipantsLimit AS 'Seats limit',
ISNULL(SUM(ReservedSeats),0) AS 'Reserved seats',
 ConferenceDayWorkshops.ParticipantsLimit - ISNULL(SUM(ReservedSeats),0) AS 'Available
Places',
 Description
FROM Conferences
JOIN ConferenceDays
ON Conferences.ConferenceID = ConferenceDays.ConferenceID
JOIN ConferenceDayWorkshops
ON ConferenceDays.ConferenceDayID = ConferenceDayWorkshops.ConferenceDayID
JOIN Workshops
ON ConferenceDayWorkshops.WorkshopID = Workshops.WorkshopID
LEFT JOIN WorkshopReservation
ON ConferenceDayWorkshops.ConferenceDayWorkshopID =
WorkshopReservation.ConferenceDayWorkshopID
GROUP BY Conferences.ConferenceID, Conferences.Name, Date, StartDate, StartTime,
 EndTime, Workshops.Name, Price, ConferenceDayWorkshops.ParticipantsLimit, Description
go
```

Funkcje

BaseDayPrices

Zwraca tabelę z cenami bazowymi konferencji dla wszystkich konferencji z tej rezerwacji.

```
CREATE FUNCTION [dbo].[BaseDayPrices] (@ReservationID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT DISTINCT c.ConferenceID, c.BasePriceForDay
 FROM ConferenceDayReservation cdr
 JOIN ConferenceDays cd
 ON cd.ConferenceDayID = cdr.ConferenceDayID
 JOIN Conferences c
 ON cd.ConferenceID = c.ConferenceID
 WHERE cdr.ReservationID = @ReservationID
);
go
```

CalculatePriceForReservation

Wyświetla sumaryczną cenę za dane zamówienie

```
CREATE function [dbo].[CalculatePriceForReservation] (@Email varchar(100), @DateOrdered
date)
returns money
as
begin
 declare @CustomerID int,
   @ReservationID int,
   @ConferenceID int,
   @ReservedAdults int,
   @ReservedStudents int,
   @PriceForWorkshops money,
   @TimeDiscount real,
   @PriceForDay money,
   @StudentDiscount real;
 exec @CustomerID = FindCustomerByEmail @Email;
 set @ReservationID = (select ReservationID
       from ConferenceReservations
       where CustomerID = @CustomerID and DateOrdered = @DateOrdered);
 set @ConferenceID = (select ConferenceID
      from ConferenceDays cd
      join ConferenceDayReservation cdr
      on cd.ConferenceDayID = cdr.ConferenceDayID
      where cdr.ReservationID = @ReservationID
      group by ConferenceID);
 set @ReservedAdults = (select sum(ReservedAdultSeats)
        from ConferenceDayReservation
        where ReservationID = @ReservationID);
 set @ReservedStudents = (select sum(ReservedStudentSeats)
       from ConferenceDayReservation
       where ReservationID = @ReservationID);
 if not exists (select *
      from WorkshopReservation
      where ConferenceDayReservationID in (select DayReservationID
            from ConferenceDayReservation
            where ReservationID = @ReservationID))
  set @PriceForWorkshops = 0
 else
  set @PriceForWorkshops = (select sum(cdw.Price * wr.ReservedSeats)
         from WorkshopReservation wr
         join ConferenceDayWorkshops cdw
         on wr.ConferenceDayWorkshopID = cdw.ConferenceDayWorkshopID
         where wr.ConferenceDayReservationID in (select DayReservationID
                  from ConferenceDayReservation
                  where ReservationID = @ReservationID))
 if not exists (select DiscountRate
      from ConferencePricetables cp
         where cp.ConferenceID = @ConferenceID and @DateOrdered between
cp.PriceStartsOn and cp.PriceEndsOn)
  set @TimeDiscount = 0;
 else
  set @TimeDiscount = (select DiscountRate
      from ConferencePricetables cp
      where cp.ConferenceID = @ConferenceID and @DateOrdered between cp.PriceStartsOn
and cp.PriceEndsOn)
 set @PriceForDay = (select (1 - @TimeDiscount) * BasePriceForDay
      from Conferences
      where ConferenceID = @ConferenceID);
 set @StudentDiscount = (select StudentDiscount
       from Conferences
       where ConferenceID = @ConferenceID);
 return @PriceForWorkshops + (1 - @StudentDiscount) * @PriceForDay * @ReservedStudents
+ @PriceForDay * @ReservedAdults
```

```
end
go
```

ConferenceDayReservationSize

Zwraca rozmiar rezerwacji dla danego ID

```
CREATE FUNCTION ConferenceDayReservationSize(@ConferenceDayReservationID int)
RETURNS INT
AS
BEGIN
 DECLARE @Size INT = (SELECT SUM(c.ReservedAdultSeats )+ SUM(c.ReservedAdultSeats)
      FROM dbo.ConferenceDayReservation c
      WHERE c.DayReservationID = @ConferenceDayReservationID)
 RETURN @Size
END
go
```

ConferenceOrderedAfterCreated

Zwraca dodatnią wartość jeśli rezerwacja nastąpiła po dacie utworzenia konferencji

```
CREATE FUNCTION ConferenceOrderedAfterCreated (@ConferenceDayID INT, @ReservationID
INT)
RETURNS INT
BEGIN
 DECLARE @ConfCreated DATE = (SELECT Conferences.CreatedOn FROM dbo.ConferenceDays
        INNER JOIN Conferences ON Conferences.ConferenceID =
ConferenceDays.ConferenceID
        WHERE ConferenceDays.ConferenceDayID = @ConferenceDayID)
 DECLARE @OrderDate DATE = (SELECT DateOrdered FROM dbo.ConferenceReservations
        WHERE ReservationID = @ReservationID)
 RETURN DATEDIFF(DAY, @ConfCreated, @OrderDate)
END
go
```

ConferenceSize

Maksymalna ilość uczestników dla danego ID konferencji.

```
CREATE FUNCTION ConferenceSize(@ConferenceDayID INT)
RETURNS int
BEGIN
 DECLARE @size int = (SELECT ParticipantsLimit FROM ConferenceDays
 INNER JOIN conferences ON Conferences.ConferenceID = ConferenceDays.ConferenceID
 WHERE ConferenceDayID = @ConferenceDayID)
 RETURN @size
END
go
```

DayReservationTotalSeats

Zwraca rozmiar rezerwacji dla danego ID

```
CREATE FUNCTION DayReservationTotalSeats(@ConferenceDayReservationID INT)
RETURNS INT
BEGIN
 DECLARE @result INT
 SELECT @result = SUM(ReservedAdultSeats) + SUM(ReservedStudentSeats)
 FROM dbo.ConferenceDayReservation cdr
 WHERE cdr.DayReservationID = @ConferenceDayReservationID
 RETURN @result
end
go
```

DiscountForConference

Zwraca zniżkę za date rezerwacji dla danego ID konferencji i dla danej daty.

```
CREATE function [dbo].[DiscountForConference](@DateOrdered date, @ConferenceID int)
returns real
as
begin
 declare @TimeDiscount real;
 if not exists (select DiscountRate
       from ConferencePricetables cp
          where cp.ConferenceID = @ConferenceID and @DateOrdered between
cp.PriceStartsOn and cp.PriceEndsOn)
  set @TimeDiscount = 0
 else
  set @TimeDiscount =  (select DiscountRate
         from ConferencePricetables cp
         where cp.ConferenceID = @ConferenceID and @DateOrdered between
cp.PriceStartsOn and cp.PriceEndsOn)
 return @TimeDiscount
end
go
```

DiscountForReservations

Zwraca tabelę zniżek za datę rezerwacji konferencji dla danego ID rezerwacji i danej daty.

```
CREATE function [dbo].[DiscountForReservations] (@DateOrdered date, @ReservationID int)
returns table
as
return
(
    SELECT distinct c.ConferenceID, dbo.DiscountForConference(@DateOrdered,
c.ConferenceID) as Discount
 from ConferenceDayReservation cdr
 join ConferenceDays cd
 on cd.ConferenceDayID = cdr.ConferenceDayID
 join Conferences c
 on cd.ConferenceID = c.ConferenceID
 left join ConferencePricetables cp
 on c.ConferenceID = cp.ConferenceID
 where cdr.ReservationID = @ReservationID
 group by c.ConferenceID
);
go
```

EmptySeatsInWorkshopReservation

Zwraca ilość wolnych miejsc w rezerwacji warsztatu lub -1 jeśli dla danego uczestnika w rezerwacji dnia konferencji z którą jest powiązany nie ma rezerwacji warszatatu.

```
CREATE FUNCTION [dbo].[EmptySeatsInWorkshopReservation] (@DayParticipantID INT,
@ConferenceDayWorkshopID int)
RETURNS INT
BEGIN
 DECLARE @SeatsReserved INT, @SeatsOccupied INT, @DayReservationID INT
    SELECT @DayReservationID = ConferenceDayReservationID
 FROM dbo.ConferenceDayParticipants
 WHERE ConferenceDayParticipantID = @DayParticipantID
 SELECT @SeatsReserved = ReservedSeats
 FROM dbo.WorkshopReservation
 WHERE ConferenceDayReservationID = @DayReservationID AND ConferenceDayWorkshopID =
@ConferenceDayWorkshopID
 SELECT @SeatsOccupied = COUNT(*)
 FROM dbo.WorkshopParticipants
 INNER JOIN dbo.ConferenceDayParticipants ON
ConferenceDayParticipants.ConferenceDayParticipantID =
```

```
WorkshopParticipants.ConferenceDayParticipantID
 WHERE ConferenceDayWorkshopID = @ConferenceDayWorkshopID AND
ConferenceDayReservationID = @DayReservationID
 IF @SeatsReserved IS NULL RETURN -1
 RETURN @SeatsReserved - @SeatsOccupied
END
go
```

FindCompanyByEmail, ~Phone, ~NIP, ~Name

Szukają firmy po odpowiednich danych

```
CREATE function [dbo].[FindCompanyByEmail] (@Email varchar(100))
returns int
as
begin
 return (select CompanyID from Companies where Email = @Email)
end
go

create function FindCompanyByName (@Name varchar(150))
returns int
as
begin
 return (select CompanyID from Companies where CompanyName = @Name)
end
go

create function FindCompanyByNIP (@NIP char(10))
returns int
as
begin
 return (select CompanyID from Companies where NIP = @NIP)
end
go

create function FindCompanyByPhone (@Phone varchar(15))
returns int
as
begin
 return (select CompanyID from Companies where Phone = @Phone)
end
go
```

FindCustomerByEmail

Zwraca ID uczestnika po adresie e-mail.

```
CREATE function [dbo].[FindCustomerByEmail] (@Email varchar(100))
returns int
as
begin
 declare @CompanyID int;
 exec @CompanyID = FindCompanyByEmail @Email;
 if @CompanyID is null
 begin
  declare @ParticipantID int;
  EXEC @ParticipantID = FindParticipantByEmail @Email;
  return (select CustomerID from PrivateCustomers where ParticipantID = @ParticipantID)
 end
 return @CompanyID
end
go
```

FindParticipantByEmail, ~Name

Zwraca ID uczestnika po odpowiednich danych

```
CREATE function [dbo].[FindParticipantByEmail] (@Email varchar(100))
returns int
as
begin
 return (select ParticipantID from Participants where Email = @Email)
end
go

create function FindParticipantByName (@FirstName varchar(30), @LastName varchar(50))
returns int
as
begin
 return (select ParticipantID from Participants where FirstName = @FirstName and
LastName = @LastName)
end
go
```

FindWorkshop

Znajduje warsztat z tabeli słownikowej po nazwie.

```
CREATE function FindWorkshop (@Name VARCHAR(200))
RETURNS INT
BEGIN
 DECLARE @ID INT = (SELECT WorkshopID FROM dbo.Workshops WHERE Name = @Name)
 RETURN @ID
END
go
```

FindConferenceStartDate

Zwraca datę rozpoczęcia konferencji o danym ID.

```
create function GetConferenceStartDate(@ConferenceID int)
returns date
as
begin
 return (select StartDate from Conferences where ConferenceID = @ConferenceID)
end
go
```

GetLatestDiscount

Zwraca najniższy próg cenowy dla danego ID konferencji

```
CREATE function [dbo].[GetLatestDiscount](@ConferenceID int)
returns real
as
begin
 declare @discount real;
 if exists (select *
      from ConferencePricetables
      where ConferenceID = @ConferenceID)
  set @discount = (select min(DiscountRate)
      from ConferencePricetables
      where ConferenceID = @ConferenceID)
 else
  set @discount = 1
 return @discount
end
go
```

GetNumberOfPaidReservationForCustomer

Zwraca liczbę opłaconych rezerwacji dla klienta o danym e-mailu.

```
CREATE function [dbo].[GetNumberOfPaidReservationForCustomer](@Email varchar(100))
returns int
begin
 declare @CustomerID int;
 exec @CustomerID = dbo.FindCustomerByEmail @Email;
 return (select count(*)
   from ConferenceReservations
   where CustomerID = @CustomerID and DatePaid is not null)
end
go
```

HasParticipantCollidingWorkshops

Informuje czy uczestnik jest w dwóch nachodzących na siebie czasowo warsztatach.

```
CREATE FUNCTION [dbo].[HasParticipantCollidingWorkshops](@NewWorkshopID INT,
@ConferenceDayParticipantID int)
RETURNS BIT
BEGIN
 DECLARE @Times TABLE (
  TimeID INT PRIMARY KEY IDENTITY(0,1),
  StartTime TIME,
  EndTime time
 )
 INSERT INTO @Times
 SELECT StartTime, EndTime
 FROM dbo.WorkshopParticipants
 INNER JOIN dbo.ConferenceDayWorkshops ON
ConferenceDayWorkshops.ConferenceDayWorkshopID =
WorkshopParticipants.ConferenceDayWorkshopID
 WHERE ConferenceDayID = (SELECT ConferenceDayID
       FROM dbo.ConferenceDayWorkshops
       WHERE ConferenceDayWorkshopID = @NewWorkshopID)
 AND ConferenceDayParticipantID = @ConferenceDayParticipantID;

 DECLARE @has BIT = (SELECT COUNT(*) from
 (SELECT a.TimeID
 FROM @Times AS a
 INNER JOIN @Times AS b ON ((a.StartTime BETWEEN b.StartTime AND b.EndTime) OR
          (a.EndTime BETWEEN b.StartTime AND b.EndTime) OR
          (a.StartTime < b.StartTime AND a.EndTime > b.EndTime))
         AND (a.TimeID != b.TimeID)) AS t)
 RETURN @has
END
go
```

IsReservationByCompany

Mówi, czy dana rezerwacja po ID jest zrobiona przez firmę

```
create function IsReservationByCompany (@ReservationID int)
returns bit
as
begin
 declare @CompanyID int
 select @CompanyID = Companies.CompanyID
 from ConferenceReservations
 inner join Customers on ConferenceReservations.CustomerID = Customers.CustomerID
 left join Companies on customers.CustomerID = CompanyID
 where ReservationID = @ReservationID
 if @CompanyID is not null begin return 1 end
```

```
 return 0
end
go
```

IsReservationPaid

Mówi, czy rezerwacja została opłacona

```
CREATE FUNCTION IsReservationPaid(@ReservationID INT)
RETURNS BIT
begin
RETURN (SELECT COUNT(*) FROM
  (SELECT DatePaid
  FROM ConferenceReservations
  WHERE DatePaid IS NOT NULL AND ReservationID = @ReservationID) t)
END
go
```

NewPriceAtTheDayAfterPrevious

Sprawdza, czy nowy próg cenowy jest zaraz po poprzednim.

```
CREATE FUNCTION [dbo].[NewPriceAtTheDayAfterPrevious](@ConferenceID INT)
RETURNS int
BEGIN

 IF (SELECT COUNT(*) FROM dbo.ConferencePricetables) = 1 RETURN 1

 DECLARE @Dates TABLE (
  DateID INT PRIMARY KEY IDENTITY(1,1),
  StartDate date,
  EndDate date
 )
 INSERT INTO @Dates (StartDate, EndDate)
 SELECT TOP 2 PriceStartsOn, PriceEndsOn
 FROM dbo.ConferencePricetables
 WHERE ConferenceID = @ConferenceID
 ORDER BY PriceStartsOn DESC

 DECLARE @PrevStepEnd DATE = (SELECT EndDate FROM @Dates WHERE DateID = 2)
 DECLARE @NewStepStart DATE = (SELECT StartDate FROM @Dates WHERE DateID = 1)
 RETURN DATEDIFF(DAY, @PrevStepEnd, @NewStepStart)
END
go
```

OrganisedLaterThanHired

Na potrzeby warunków integralnościowych sprawdza, czy pracownik zorganizował tylko konferencje po jego dacie zatrudnienia.

```
CREATE FUNCTION OrganisedLaterThanHired (@EmployeeID int, @ConferenceID int)
RETURNS int
BEGIN
 DECLARE @HireDate DATE = (SELECT HireDate FROM dbo.OurEmployees WHERE EmployeeID =
@EmployeeID)
 DECLARE @ConfDate DATE = (SELECT StartDate FROM Conferences WHERE ConferenceID =
@ConferenceID)
 RETURN DATEDIFF(DAY, @HireDate, @ConfDate)
END
go
```

ReservationEarlierThanConferenceDay

Sprawdza, czy dana rezerwacja została uczyniona przed danym dniem konferencji.

```
CREATE FUNCTION ReservationEarlierThanConferenceDay(@ConferenceDayID int,
@ReservationID INT)
RETURNS INT
BEGIN
 DECLARE @ConfDate DATE = (SELECT Date FROM dbo.ConferenceDays WHERE ConferenceDayID =
@ConferenceDayID)
 DECLARE @Orderdate DATE = (SELECT DateOrdered FROM dbo.ConferenceReservations WHERE
ReservationID = @ReservationID)
 RETURN DATEDIFF(DAY, @Orderdate, @ConfDate)
END
go
```

ReservationPrices

Zwraca tabelę ze zniżkami za datę rezerwacji dla wszystkich konferencji z danej rezerwacji.

```
create function ReservationPrices (@DateOrdered date, @ReservationID int)
returns table
as
return
(
    select base.ConferenceID, base.BasePriceForDay * dis.Discount as AdultPrice,
   base.BasePriceForDay * dis.Discount * stdis.StudentDiscount as StudentPrice
 from dbo.BaseDayPrices(@ReservationID) base
 join dbo.DiscountForReservations(@DateOrdered, @ReservationID) dis
 on
 base.ConferenceID = dis.ConferenceID
 join dbo.StudentDiscountForReservations(@ReservationID) stdis
 on dis.ConferenceID = stdis.ConferenceID
);
go
```

ReservedSeatsForWorkshop

Zwraca sumę miejsc zarezerwowanych na dany warsztat (po ID).

```
CREATE FUNCTION ReservedSeatsForWorkshop(@ConferenceDayWorkshopID int)
RETURNS int
BEGIN
 DECLARE @Sum INT = (SELECT SUM(ReservedSeats)
      FROM WorkshopReservation
      WHERE ConferenceDayWorkshopID = @ConferenceDayWorkshopID)
 RETURN @Sum
end
go
```

ReservedSeatsPerConferenceDay

Zwraca ilość zarezerwowanych miejsc na daną konferencję po ID

```
CREATE FUNCTION [dbo].[ReservedSeatsPerConferenceDay](@ConferenceDayID INT)
RETURNS INT
BEGIN
 DECLARE @number INT = (SELECT SUM(ReservedAdultSeats) + SUM(ReservedStudentSeats)
       FROM dbo.ConferenceDayReservation
       WHERE ConferenceDayID = @ConferenceDayID)
    IF @number IS NULL BEGIN SET @number = 0 end
 RETURN @number
END
go
```

StudentDiscountForReservations

Zwraca tabelę ze zniżkami studenckimi dla wszystkich konferencji z danej rezerwacji.

```
CREATE function [dbo].[StudentDiscountForReservations] (@ReservationID int)
returns table
as
return
(
    SELECT distinct c.ConferenceID, c.StudentDiscount
 from ConferenceDayReservation cdr
 join ConferenceDays cd
 on cd.ConferenceDayID = cdr.ConferenceDayID
 join Conferences c
 on cd.ConferenceID = c.ConferenceID
 where cdr.ReservationID = @ReservationID
);
go
```

ViewOrdersByEmailAsCustomer

Zwraca informacje o wszystkich zamówieniach danego klienta po adresie e-mail.

```
CREATE FUNCTION ViewOrdersByEmailAsCustomer (@CustomerEmail VARCHAR(100))
RETURNS @Data TABLE (
 DateOrdered DATE,
 DatePaid DATE,
 TotalAmount MONEY
)
BEGIN
 INSERT INTO @Data (DateOrdered, DatePaid, TotalAmount)
 SELECT DateOrdered, DatePaid, dbo.CalculatePriceForReservation(@CustomerEmail,
DateOrdered)
 FROM dbo.ConferenceReservations
 INNER JOIN dbo.CustomerContactData ON CustomerContactData.CustomerID =
ConferenceReservations.CustomerID
 WHERE Email = @CustomerEmail
 RETURN
END
go
```

WorkshopReservationOnDayReservationConference

Zwraca 0 jeśli rezerwacja jest na warsztat tego samego dnia co rezerwacja konferencji.

```
CREATE FUNCTION WorkshopReservationOnDayReservationConference (@DayReservationID INT,
@WorkshopInDayID INT)
RETURNS INT
BEGIN
 DECLARE @ConferenceDayAtReservation INT = (SELECT ConferenceDayID FROM
dbo.ConferenceDayReservation WHERE DayReservationID = @DayReservationID)
 DECLARE @ConferenceDayAtWorkshop INT = (SELECT ConferenceDayID FROM
dbo.ConferenceDayWorkshops WHERE ConferenceDayWorkshopID = @WorkshopInDayID)
 RETURN @ConferenceDayAtReservation - @ConferenceDayAtWorkshop
END
go
```

WorkshopSeatsLimit

Zwraca limit miejsc na dany warsztat po ID

```
CREATE FUNCTION [dbo].[WorkshopSeatsLimit](@WorkshopID INT)
RETURNS INT
AS
BEGIN
 DECLARE @Limit INT = (SELECT ParticipantsLimit
       FROM dbo.ConferenceDayWorkshops
       WHERE ConferenceDayWorkshopID = @WorkshopID)
 RETURN @Limit
```

```
end
go
```

Procedury

AddOurEmployee

Dodaj własnego pracownika.

```
create procedure AddOurEmployee
 @FirstName varchar(30),
 @LastName varchar(50),
 @BirthDate date,
 @HireDate date,
 @Phone varchar(15),
 @Street varchar(74),
 @HouseNumber varchar(5),
 @AppartmentNumber int,
 @CityName varchar(80),
 @RegionName varchar(80),
 @CountryName varchar(80),
 @PostalCode char(6),
 @Email varchar(100)
as
begin
 declare @CityID int
 exec FindCity @CityName, @RegionName, @CountryName, @CityID
 insert into OurEmployees (FirstName, LastName, BirthDate, HireDate, Phone, Street
,HouseNumber ,AppartmentNumber ,CityID ,PostalCode ,Email)
 values (
  @FirstName, @LastName, @BirthDate, @HireDate, @Phone, @Street, @HouseNumber,
@AppartmentNumber, @CityID, @PostalCode, @Email
 )
end
go
```

AddParticipantToWorkshop

Dodaj uczestnika do warsztatu po e-mailu

```
CREATE PROCEDURE AddParticipantToWorkshop
 @ParticipantEmail VARCHAR(100),
 @ConferenceName VARCHAR(200),
 @Date DATE,
 @CustomerEmail VARCHAR(100),
 @DateOrdered DATE,
 @WorkshopName VARCHAR(200),
 @StartTime TIME
AS
BEGIN
BEGIN TRY
 BEGIN TRAN tr
  DECLARE @ConferenceDayParticipantID INT, @ParticipantID INT,
@ConferenceDayReservationID INT
        EXEC @ParticipantID = dbo.FindParticipantByEmail @Email = @ParticipantEmail --
varchar(100)
        EXEC dbo.FindConferenceDayReservation @ConferenceName = @ConferenceName,
-- varchar(200)
                                              @ConfDayDate = @Date,
-- date
                                              @CustomerEmail = @CustomerEmail,
-- varchar(100)
                                              @DateOrdered = @DateOrdered,
-- date
                                              @ConferenceDayReservationID =
@ConferenceDayReservationID OUTPUT -- int
```

```
        -- Znaleźć ConferenceDayParticipantID do dodania
  SELECT @ConferenceDayParticipantID = ConferenceDayParticipantID
  FROM dbo.ConferenceDayParticipants
  WHERE ParticipantID = @ParticipantID AND ConferenceDayReservationID =
@ConferenceDayReservationID
  PRINT 'DayParticipantID ' + CAST (@ConferenceDayParticipantID AS VARCHAR)
  -- Znaleźć ConferenceDayWorkshopID
  DECLARE @ConferenceDayWorkshopID INT;
  EXEC dbo.FindWorkshopInDay @ConferenceName = @ConferenceName,
-- varchar(200)
                             @Date = @Date,                                    --
date
                             @WorkshopName = @WorkshopName,
-- varchar(200)
           @StartTime = @StartTime,
                             @ConferenceDayWorkshopID = @ConferenceDayWorkshopID OUTPUT
-- int
  PRINT 'ConferenceDayWorkshopID ' + CAST(@ConferenceDayWorkshopID AS VARCHAR)

  INSERT INTO dbo.WorkshopParticipants
  (
      ConferenceDayParticipantID,
      ConferenceDayWorkshopID
  )
  VALUES
  (   @ConferenceDayParticipantID, -- ConferenceDayParticipantID - int
      @ConferenceDayWorkshopID  -- ConferenceDayWorkshopID - int
      )
 COMMIT TRAN tr
END TRY
BEGIN CATCH
 PRINT ERROR_MESSAGE()
 ROLLBACK TRAN tr
END CATCH
END

go
```

AddPriceStep

Dodaj przedział cenowy dla danej konferencji

```
CREATE procedure [dbo].[AddPriceStep]
 @ConferenceName varchar(200),
 @ConferenceStartDate date,
 @PriceStartsOn date,
 @PriceEndsOn date,
 @DiscountRate real
as
declare @ConferenceID int, @c int
exec FindConference @ConferenceName, @ConferenceStartDate, @ConferenceID output, @c
begin
 if @ConferenceID is null
 begin
  print 'Nie znaleziono konferencji'
  return
 end
 insert into ConferencePricetables (ConferenceID, PriceStartsOn, PriceEndsOn,
DiscountRate)
 values (@ConferenceID, @PriceStartsOn, @PriceEndsOn, @DiscountRate)
end
go
```

AddPrivateCustomer

Dodaje nowego klienta prywatnego

```
CREATE procedure [dbo].[AddPrivateCustomer]
 @FirstName VARCHAR(30),
 @LastName VARCHAR(50),
 @ParticipantPhone nvarchar(15),
 @Email VARCHAR(100),
 @Street nvarchar(80),
 @HouseNumber nvarchar(5),
 @AppartmentNumber int,
 @PostalCode char(6),
 @CityName varchar(80),
 @RegionName varchar(80),
 @CountryName varchar(80)
as
begin
begin try
 begin tran tr
  DECLARE @NewParticipantID int
  EXEC dbo.NewParticipant @FirstName, -- varchar(30)
                          @LastName,  -- varchar(50)
                          @ParticipantPhone,     -- varchar(15)
                          @Email,      -- varchar(50)
        @NewParticipantID OUTPUT
  declare @CityID int
  exec FindCity @CityName, @RegionName, @CountryName, @CityID output
  insert into Customers (Street, HouseNumber, AppartmentNumber, PostalCode, CityID)
  values (
   @Street, @HouseNumber, @AppartmentNumber, @PostalCode, @CityID
  )
  insert into PrivateCustomers (ParticipantID, CustomerID)
  values (
   @NewParticipantID, (select max(CustomerID) from Customers)
  )
 commit tran find
end try
begin catch
 rollback tran tr
end catch
end
go
```

AddWorkshopAtDay

Dodaj warsztat do danego dnia konferencji

```
create procedure AddWorkshopAtDay
 @WorkshopName varchar(100),
 @ConferenceName varchar(200),
 @Day smallint,
 @StartTime time,
 @EndTime time,
 @Price money,
 @ParticipantsLimit int
as
declare @ConferenceDayID int,
  @WorkshopID int;
begin
 set @WorkshopID = (select WorkshopID from Workshops where Name = @WorkshopName)
 if @WorkshopID is null begin
  print 'Brak warsztatu o podanej nazwie w bazie'
  return
 end
 set @ConferenceDayID = (select ConferenceDayID
       from ConferenceDays
       where DayOrdinal = @Day and
          ConferenceID = (select ConferenceID
              from Conferences
```

```
              where Name = @ConferenceName))
 if @ConferenceDayID is null begin
  print 'Nie znaleziono konferencji'
  return
 end
 insert into ConferenceDayWorkshops (ConferenceDayID, WorkshopID, StartTime, EndTime,
Price, ParticipantsLimit)
 values (
   @ConferenceDayID,
   @WorkshopID,
   @StartTime,
   @EndTime,
   @Price,
   @ParticipantsLimit
     )
end
go
```

BindOurEmployeeWithConference

Dodaje pracownika jako odpowiedzialnego za daną konferencję.

```
CREATE procedure [dbo].[BindOurEmployeeWithConference]
 @EmpEmail VARCHAR(100),
 @ConferenceName varchar(200)
as
begin
 insert into ConferenceEmployees (EmployeeID, ConferenceID) values (
  (select EmployeeID from OurEmployees where Email = @EmpEmail),
  (select ConferenceID from Conferences where Name = @ConferenceName)
 )
end
go
```

BindParticipantWithCompany

Powiązuje danego uczestnika z firmą która zarezerwowała jego miejsce.

```
CREATE procedure [dbo].[BoundParticipantWithCompany]
 @Email varchar(100),
 @NIP char(10)
as
declare @ParticipantID int,
  @CompanyID int;
exec @ParticipantID = FindParticipantByEmail @Email;
EXEC @CompanyID = dbo.FindCompanyByNIP @NIP = @NIP -- char(10)

begin
insert into EmployeesOfCompanies (ParticipantID, CompanyID)
values (
 @ParticipantID,
 @CompanyID
)
end
go
```

DeleteUnpaidReservations

Usuwa wszystkie nieopłacone po tygodniu rezerwacje

```
create procedure DeleteUnpaidReservations as
begin
begin try
 begin tran tr
  delete from ConferenceReservations
  where DatePaid is null and DATEDIFF(day, DateOrdered, convert(date, getdate())) > 7
```

```
 commit tran tr
end try
begin catch rollback tran tr end catch
end -- dopisać trigger usuwający rezerwację dni konferencji, warsztatów itd.
go
```

FillReservation

Uzupełnia uczestnika w danej rezerwacji

```
CREATE PROCEDURE [dbo].[FillReservation]
 @CustomerEmail VARCHAR(100),
 @DateOrdered DATE,
 @ConferenceName VARCHAR(200),
 @ConfDayDate DATE,
 @FirstName VARCHAR(30),
 @LastName VARCHAR(50),
 @ParticipantPhone VARCHAR(15),
 @ParticipantEmail VARCHAR(100),
 @StudentCardNumber VARCHAR(10),
 @ParticipantID INT OUTPUT
AS
BEGIN
BEGIN TRY
 BEGIN TRANSACTION tr
  -- Wyszukaj czy w bazie nie ma już uczestnika o takim mailu
  DECLARE @FoundID int
  EXEC @FoundID = dbo.FindParticipantByEmail @Email = @ParticipantEmail -- varchar(100)

  -- Sprawdź czy dane są pełne
  IF (@FirstName IS NULL OR @LastName IS NULL OR @ParticipantEmail IS NULL) AND
@FoundID IS null BEGIN
   RAISERROR ('Dane niepełne', 11,1)
  END

  -- Znajdź rezerwację dnia
  DECLARE @ReservationID INT
  EXEC FindConferenceDayReservation @ConferenceName, @ConfDayDate, @CustomerEmail,
@DateOrdered, @ReservationID output
  IF @ReservationID IS NULL BEGIN
   RAISERROR ('Nie znaleziono rezerwacji', 11,1)
  END

  -- Znajdź wszystkie nieuzupełnione ParticipantID z tej rezerwacji
  DECLARE @EmptyParticipantIDs TABLE (ParticipantID INT NOT NULL)
  INSERT INTO @EmptyParticipantIDs (ParticipantID)
   SELECT cdp.ParticipantID
   FROM dbo.ConferenceDayParticipants cdp
   INNER JOIN dbo.Participants ON Participants.ParticipantID = cdp.ParticipantID
   WHERE ConferenceDayReservationID = @ReservationID AND LastName IS NULL
  DECLARE @size INT = (SELECT COUNT(*) FROM @EmptyParticipantIDs)

  -- Wybierz ID które trzeba uzupełnić
  IF @StudentCardNumber IS NULL
   SET @ParticipantID = (SELECT MIN(ParticipantID) FROM (SELECT * FROM
@EmptyParticipantIDs EXCEPT SELECT ParticipantID FROM dbo.Students) t);
  ELSE
   SET @ParticipantID = (SELECT MIN(ParticipantID) FROM (SELECT * FROM
@EmptyParticipantIDs INTERSECT SELECT ParticipantID FROM dbo.Students) t);

  IF @ParticipantID IS NULL AND @StudentCardNumber IS NULL RAISERROR ('Już nie ma
miejsc dla dorosłych',11,1)
  IF @ParticipantID IS NULL AND @StudentCardNumber IS NOT NULL RAISERROR ('Już nie ma
miejsc dla studentów',11,1)

  -- Jeśli jest jeszcze nieuzupełniona rezerwacja
  IF @ParticipantID IS NOT NULL BEGIN
```

```
   -- Jeśli już jest uczestnik o takim mailu
   IF @FoundID IS NOT NULL BEGIN
    UPDATE ConferenceDayParticipants
     SET ParticipantID = @FoundID
     WHERE ParticipantID = @ParticipantID AND ConferenceDayReservationID =
@ReservationID
   END ELSE BEGIN
    UPDATE dbo.Participants
     SET FirstName = @FirstName, LastName = @LastName, Email = @ParticipantEmail, Phone
= @ParticipantPhone
     WHERE ParticipantID = @ParticipantID
    UPDATE dbo.Students
     SET StudentCardNumber = @StudentCardNumber
     WHERE ParticipantID = @ParticipantID

   END
  END

 COMMIT TRANSACTION tr
END TRY
BEGIN CATCH
 PRINT ERROR_MESSAGE()
 ROLLBACK TRAN tr
END CATCH
END
go
```

FindCity

Znajduje miasto lub dodaje go do bazy, jeśli nie było wpisane

```
CREATE procedure [dbo].[FindCity]
 @CityName nvarchar(80),
 @RegionName nvarchar(80),
 @CountryName nvarchar(80),
 @CityID int output
as
begin
begin try
 begin tran find
  if @RegionName is null or @CountryName is null raiserror (15600, -1,-1, 'FindCity')
  set @CityID = (select Cityid
        from Cities
        inner join Regions on Cities.RegionID = Regions.RegionID
        inner join Countries on Countries.CountryID = Regions.CountryID
        where CityName = @CityName and RegionName = @RegionName and CountryName =
@CountryName)
  if @CityID is null begin
   declare @RegionID int
   exec FindRegion @RegionName, @CountryName, @RegionID output
   insert into Cities (CityName, RegionID) values (@CityName, @RegionID)
   set @CityID = (select max(CityID) from Cities)
  end
 commit tran find
end try
begin catch
 rollback tran find
end catch
end
go
```

FindConference

Znajduje konferencję i dzień konferencji po nazwie i dacie

```
CREATE procedure [dbo].[FindConference]
 @ConferenceName varchar(200),
 @Date date,
 @ConferenceID int output,
 @ConferenceDayID int output
as
begin
begin try
 begin tran tr
  select @ConferenceID = Conferences.ConferenceID
  from Conferences
  inner join ConferenceDays on Conferences.ConferenceID = ConferenceDays.ConferenceID
  where name = @ConferenceName AND (@date BETWEEN StartDate AND EndDate)
  if @ConferenceID is not null begin
   select @ConferenceDayID = ConferenceDayID
   from ConferenceDays
   where ConferenceID = @ConferenceID and Date = @Date
  end
 commit tran tr
end try
begin catch
 rollback tran tr
end catch
end
go
```

FindConferenceDayReservation

Znajduje ID rezerwacji dnia po nazwie konferencji, dniu, dniu zamówienia i e-mailu klienta

```
create PROCEDURE [dbo].[FindConferenceDayReservation]
 @ConferenceName VARCHAR(200), @ConfDayDate DATE, @CustomerEmail VARCHAR(100),
@DateOrdered DATE,
 @ConferenceDayReservationID INT OUTPUT
AS
BEGIN
 DECLARE @ConferenceDayID INT
 EXEC dbo.FindConference @ConferenceName = @ConferenceName,                    --
varchar(200)
                        @Date = @ConfDayDate,                       -- date
                        @ConferenceID = NULL,       -- int
                        @ConferenceDayID = @ConferenceDayID OUTPUT -- int
 DECLARE @ReservationID INT;
 EXEC dbo.FindReservation @CustomerEmail = @CustomerEmail,                    --
varchar(100)
                         @DateOrdered = @DateOrdered,           -- date
                         @ReservationID = @ReservationID OUTPUT -- int
 (SELECT @ConferenceDayReservationID = DayReservationID
 FROM dbo.ConferenceDayReservation
 WHERE ReservationID = @ReservationID AND ConferenceDayID = @ConferenceDayID)
END
go
```

FindCountry

Znajduje kraj lub dodaje go do bazy, jeśli nie ma go w bazie.

```
CREATE procedure [dbo].[FindCountry]
 @CountryName varchar(80),
 @CountryID int OUTPUT
as
begin
 set nocount on
 begin try
  begin TRAN FIND
   SET @CountryID = (select countryID
```

```
        from Countries
        where countryname = @CountryName)
   if(@CountryID is null) begin
    insert into Countries (CountryName)
    values (@CountryName);
    set @CountryID = @@IDENTITY;
   end
  COMMIT TRAN FIND
 end try
 begin catch
  rollback tran FIND
 end catch
end;
go
```

FindRegion

Znajduje region lub dodaje go do bazy, jeśli nie było go w bazie

```
CREATE procedure [dbo].[FindRegion]
 @RegionName nvarchar(80),
 @CountryName nvarchar(80),
 @RegionID int output
as
begin
 begin try
  begin tran find
   if @CountryName is null raiserror (15600, -1, -1, 'FindRegion')
   set @RegionID = (select RegionID
        from Regions
        inner join Countries on Countries.CountryID = Regions.CountryID
        where regionname = @RegionName and CountryName = @CountryName)
   if @RegionID is null begin
    declare @CountryID int
    exec FindCountry @CountryName, @CountryID output
    insert into Regions (RegionName, CountryID) values (@RegionName, @CountryID)
    set @RegionID = (select max(RegionID) from Regions)
   end
  commit tran find
 end try
 begin catch
  rollback tran find
 end catch
end
go
```

FindReservation

Znajduje rezerwację po mailu klienta i dacie zamówienia

```
CREATE procedure [dbo].[FindRegion]
 @RegionName nvarchar(80),
 @CountryName nvarchar(80),
 @RegionID int output
as
begin
 begin try
  begin tran find
   if @CountryName is null raiserror (15600, -1, -1, 'FindRegion')
   set @RegionID = (select RegionID
        from Regions
        inner join Countries on Countries.CountryID = Regions.CountryID
        where regionname = @RegionName and CountryName = @CountryName)
   if @RegionID is null begin
    declare @CountryID int
    exec FindCountry @CountryName, @CountryID output
    insert into Regions (RegionName, CountryID) values (@RegionName, @CountryID)
```

```
    set @RegionID = (select max(RegionID) from Regions)
   end
  commit tran find
 end try
 begin catch
  rollback tran find
 end catch
end
go
```

FindWorkshopInDay

Znajduje warsztat w dniu konferencji po nazwie, dacie i godzinie rozpoczęcia

```
CREATE PROCEDURE [dbo].[FindWorkshopInDay]
 @ConferenceName VARCHAR(200),
 @Date DATE,
 @WorkshopName VARCHAR(200),
 @StartTime TIME,
 @ConferenceDayWorkshopID INT OUTPUT
AS
BEGIN
 DECLARE @WorkshopID INT, @ConferenceDayID INT
 EXEC @WorkshopID = FindWorkshop @Name = @WorkshopName -- varchar(200)
 EXEC dbo.FindConference @ConferenceName = @ConferenceName,                    --
varchar(200)
                        @Date = @Date,                        -- date
                        @ConferenceID = null,        -- int
                        @ConferenceDayID = @ConferenceDayID OUTPUT -- int
 IF @WorkshopID IS NULL RAISERROR('Nie ma tekiego warsztatu', 11,1)
 SELECT @ConferenceDayWorkshopID = ConferenceDayWorkshopID
 FROM dbo.ConferenceDayWorkshops
 WHERE WorkshopID = @WorkshopID AND ConferenceDayID = @ConferenceDayID AND StartTime =
@StartTime
END
go
```

Invoice

Zwraca fakturę (wyszczególnione elementy zamówienia i dane klienta)

```
CREATE PROCEDURE [dbo].[Invoice]
 @CustomerEmail VARCHAR(100),
 @DateOrdered DATE,
 @InvoiceCustomerData VARCHAR(500) output
AS
BEGIN

 DECLARE @ReservationId INT
 EXEC dbo.FindReservation @CustomerEmail = @CustomerEmail,                    --
varchar(100)
                          @DateOrdered = @DateOrdered,            -- date
                          @ReservationID = @ReservationID OUTPUT -- int

 SELECT @InvoiceCustomerData = (Name + CHAR(10) + Address) FROM CustomerContactData
WHERE CustomerID =
 (SELECT CustomerID FROM dbo.ConferenceReservations WHERE ReservationID =
@ReservationId)

 DECLARE @DayReservations TABLE (
  ID INT PRIMARY KEY IDENTITY(1,1),
  ConfDayID INT,
  AdultSeats INT,
  StudentSeats INT,
  DayDate DATE,
  ConfName VARCHAR(200),
  ConfID int
```

```
 )
 INSERT INTO @DayReservations (ConfDayID, AdultSeats, StudentSeats, DayDate, ConfName,
ConfID)
 SELECT ConferenceDayReservation.ConferenceDayID, ReservedAdultSeats,
ReservedStudentSeats, dbo.ConferenceDays.Date, Conferences.Name,
Conferences.ConferenceID
 FROM dbo.ConferenceDayReservation
 INNER JOIN dbo.ConferenceDays ON ConferenceDays.ConferenceDayID =
ConferenceDayReservation.ConferenceDayID
 INNER JOIN dbo.Conferences ON Conferences.ConferenceID = ConferenceDays.ConferenceID
 WHERE ReservationID = @ReservationId

 DECLARE @WorkshopReserv TABLE (
  ID INT PRIMARY KEY IDENTITY(1,1),
  ConfDayID INT,
  Seats INT,
  DayDate date,
  ConfName VARCHAR(200),
  WorkshopName VARCHAR(200),
  Price MONEY,
  StartTime TIME
 )
 INSERT INTO @WorkshopReserv (ConfDayID, Seats, DayDate, ConfName, WorkshopName, Price,
StartTime)
 SELECT dbo.ConferenceDayReservation.ConferenceDayID,
dbo.WorkshopReservation.ReservedSeats, ConferenceDays.Date, Conferences.Name,
Workshops.Name, Price, StartTime
 FROM dbo.ConferenceDayReservation
 INNER JOIN dbo.ConferenceDays ON ConferenceDays.ConferenceDayID =
ConferenceDayReservation.ConferenceDayID
 INNER JOIN dbo.Conferences ON Conferences.ConferenceID = ConferenceDays.ConferenceID
 INNER JOIN dbo.ConferenceDayWorkshops ON ConferenceDayWorkshops.ConferenceDayID =
ConferenceDays.ConferenceDayID
 INNER JOIN dbo.Workshops ON Workshops.WorkshopID = ConferenceDayWorkshops.WorkshopID
 INNER JOIN dbo.WorkshopReservation ON WorkshopReservation.ConferenceDayWorkshopID =
ConferenceDayWorkshops.ConferenceDayWorkshopID AND
WorkshopReservation.ConferenceDayReservationID =
ConferenceDayReservation.DayReservationID
 WHERE ReservationID = @ReservationId

 DECLARE @Invoice TABLE (
  Description varchar(500),
  Quantity INT,
  BasePrice MONEY,
  OrderDiscount REAL,
  StudentDiscount REAL,
  FinalPrice REAL
 )

 DECLARE @ReservPointer INT = 1, @ReservSize INT = (SELECT COUNT(*) FROM (SELECT * FROM
@DayReservations) t), @Total REAL = 0.0
 WHILE @ReservPointer <= @ReservSize BEGIN
  DECLARE @AdultSeats INT = (SELECT AdultSeats FROM @DayReservations WHERE ID =
@ReservPointer)
  DECLARE @StudentSeats INT = (SELECT StudentSeats FROM @DayReservations WHERE ID =
@ReservPointer)
  DECLARE @ConfName VARCHAR(200) = (SELECT ConfName FROM @DayReservations WHERE Id =
@ReservPointer)
  DECLARE @ConfDate DATE = (SELECT DayDate FROM @DayReservations WHERE ID =
@ReservPointer)
  DECLARE @BaseDayPrice money, @DiscountForDay REAL, @DiscountForStudent REAL
        set @BaseDayPrice = (SELECT BasePriceForDay FROM
dbo.BaseDayPrices(@ReservationId) WHERE ConferenceID = (SELECT ConfID FROM
@DayReservations WHERE ID = @ReservPointer))
  set @DiscountForDay = (SELECT Discount FROM dbo.DiscountForReservations(@DateOrdered,
@ReservationId) WHERE ConferenceID = (SELECT ConfID FROM @DayReservations WHERE ID =
@ReservPointer))
```

```
  set @DiscountForStudent = (SELECT StudentDiscount FROM
dbo.StudentDiscountForReservations(@ReservationId) WHERE ConferenceID = (SELECT ConfID
FROM @DayReservations WHERE ID = @ReservPointer))
  INSERT INTO @Invoice
  (
      Description,
      Quantity,
      BasePrice,
      OrderDiscount,
      StudentDiscount,
      FinalPrice
  )
  VALUES
  (   '"' + CAST(@ConfName AS VARCHAR) + '" ' + CAST(@ConfDate AS VARCHAR) + ' -
miejsca normalne',    -- Description - varchar(400)
      @AdultSeats,     -- Quantity - int
      @BaseDayPrice, -- BasePrice - money
      @DiscountForDay,  -- OrderDiscount - real
      0,  -- StudentDiscount - real
      @AdultSeats * @BaseDayPrice * (1 - @DiscountForDay)    -- FinalPrice - real
      )
  INSERT INTO @Invoice
  (
      Description,
      Quantity,
      BasePrice,
      OrderDiscount,
      StudentDiscount,
      FinalPrice
  )
  VALUES
  (   '"' + CAST(@ConfName AS VARCHAR) + '" ' + CAST(@ConfDate AS VARCHAR) + ' -
miejsca studenckie',    -- Description - varchar(400)
      @StudentSeats,     -- Quantity - int
      @BaseDayPrice, -- BasePrice - money
      @DiscountForDay,  -- OrderDiscount - real
      @DiscountForStudent,  -- StudentDiscount - real
      @StudentSeats * (1 - @DiscountForDay) * (1 - @DiscountForStudent) * @BaseDayPrice
-- FinalPrice - real
      )

 SET @Total = @Total + @StudentSeats * (1 - @DiscountForDay) * (1 -
@DiscountForStudent) * @BaseDayPrice + @AdultSeats * @BaseDayPrice * (1 -
@DiscountForDay)
 SET @ReservPointer = @ReservPointer + 1
 END

 SET @ReservPointer = 1
 SET @ReservSize = (SELECT COUNT(*) FROM (SELECT * FROM @WorkshopReserv) t)
 WHILE @ReservPointer <= @ReservSize BEGIN
  DECLARE @Seats INT = (SELECT Seats FROM @WorkshopReserv WHERE ID = @ReservPointer)
  DECLARE @ConfName2 VARCHAR(200) = (SELECT ConfName FROM @WorkshopReserv WHERE Id =
@ReservPointer)
  DECLARE @ConfDate2 DATE = (SELECT DayDate FROM @WorkshopReserv WHERE ID =
@ReservPointer)
  DECLARE @Time TIME = (SELECT StartTime FROM @WorkshopReserv WHERE ID =
@ReservPointer)
  DECLARE @WorkName VARCHAR(200) = (SELECT WorkshopName FROM @WorkshopReserv WHERE ID =
@ReservPointer)
  DECLARE @Price MONEY = (SELECT Price FROM @WorkshopReserv WHERE ID = @ReservPointer)
  INSERT INTO @Invoice
  (
      Description,
      Quantity,
      BasePrice,
      OrderDiscount,
      StudentDiscount,
```

```
      FinalPrice
  )
  VALUES
  (   'Miejsca na warsztat "' + @WorkName + '" podczas konferencji "' + @ConfName2 + '"
' + CAST(@ConfDate2 AS VARCHAR) + ' godz. ' + CAST(@Time AS VARCHAR(5)),
      @Seats,     -- Quantity - int
      @Price, -- BasePrice - money
      0.0,  -- OrderDiscount - real
      0.0,  -- StudentDiscount - real
      @Seats * @Price   -- FinalPrice - real
      )
  SET @Total = @Total + @Seats * @Price
  SET @ReservPointer = @ReservPointer + 1
 END


 INSERT INTO @Invoice
 (
     Description,
     Quantity,
     BasePrice,
     OrderDiscount,
     StudentDiscount,
     FinalPrice
 )
 VALUES
 (   'Razem',    -- Description - varchar(400)
     null,     -- Quantity - int
     NULL, -- BasePrice - money
     null,   -- OrderDiscount - real
     null,   -- StudentDiscount - real
     @Total    -- FinalPrice - real
     )

 SELECT * FROM @Invoice WHERE Quantity > 0 OR Quantity IS NULL
END
go
```

MarkReservationAsPaid

Oznacza datę opłacenia rezerwacji jako datę dzisiejszą

```
CREATE procedure [dbo].[MarkReservationAsPaid]
 @Email varchar(100),
 @DateOrdered date
as
declare  @ReservationID int,
  @CustomerID int;
EXEC @CustomerID = dbo.FindCustomerByEmail @Email -- varchar(15)

begin
 EXEC dbo.FindReservation @CustomerEmail = @Email,                    -- varchar(15)
        @DateOrdered = @DateOrdered,
                        @ReservationID = @ReservationID OUTPUT -- int

 update ConferenceReservations
 set DatePaid = convert (date, getdate())
 where ReservationID = @ReservationID;
end
go
```

NewCompany

Wprowadza nową firmę do bazy

```
CREATE PROCEDURE NewCompany
 @CompanyName NVARCHAR(150),
 @NIP CHAR(10),
 @Phone VARCHAR(15),
 @Email VARCHAR(100),
 @Street NVARCHAR(74),
 @HouseNumber VARCHAR(5),
 @AppartmentNumber INT,
 @CityName VARCHAR(80),
 @PostalCode CHAR(6),
 @RegionName VARCHAR(80),
 @CountryName VARCHAR(80)
AS
BEGIN
BEGIN TRY
 BEGIN TRAN tr
  DECLARE @cityID INT
  EXEC dbo.FindCity @CityName = @CityName,           -- nvarchar(80)
                    @RegionName = @RegionName,         -- nvarchar(80)
                    @CountryName = @CountryName,        -- nvarchar(80)
                    @CityID = @CityID OUTPUT -- int
  INSERT INTO dbo.Customers
  (
      Street,
      HouseNumber,
      AppartmentNumber,
      CityID,
      PostalCode
  )
  VALUES
  (   @Street, -- Street - nvarchar(74)
      @HouseNumber, -- HouseNumber - nvarchar(5)
      @AppartmentNumber,    -- AppartmentNumber - int
      @cityID,    -- CityID - int
      @PostalCode   -- PostalCode - char(6)
      )
  DECLARE @CompanyID INT = (SELECT MAX(CustomerID) FROM dbo.Customers)
  INSERT INTO dbo.Companies
  (
      CompanyID,
      CompanyName,
      NIP,
      Phone,
      Email
  )
  VALUES
  (   @CompanyID,    -- CompanyID - int
      @CompanyName, -- CompanyName - nvarchar(150)
      @NIP,  -- NIP - char(10)
      @Phone,  -- Phone - varchar(12)
      @Email   -- Email - varchar(100)
      )
 COMMIT TRAN tr
END TRY
BEGIN CATCH
 ROLLBACK TRAN tr
END CATCH
END
go
```

NewConference

Dodaje nową konferencję

```
CREATE procedure [dbo].[NewConference]
 @Name varchar(200),
 @StartDate date,
```

```
 @EndDate date,
 @BasePrice money,
 @StudentDiscount real,
 @ParticipantLimit int,
 @Street varchar(74),
 @HouseNumber varchar(5),
 @AppartmentNumber int,
 @City varchar(80),
 @Region varchar(80),
 @Country varchar(80),
 @PostalCode char(6)
as
begin
declare @CityID int;
exec FindCity @City, @Region, @Country, @CityID output;
insert into Conferences (Name, StartDate, EndDate, BasePriceForDay, StudentDiscount,
ParticipantsLimit,
      Street, HouseNumber, AppartmentNumber, CityID, PostalCode)
values (
  @Name,
  @StartDate,
  @EndDate,
  @BasePrice,
  @StudentDiscount,
  @ParticipantLimit,
  @Street,
  @HouseNumber,
  @AppartmentNumber,
  @CityID,
  @PostalCode
    )
end
go
```

NewConferenceReservation

Dodaje do bazy nowe zamówienie

```
CREATE procedure [dbo].[NewConferenceReservation]
 @CustomerEmail varchar(100),
 @ReservationID int output
as
begin
begin try
 begin tran tr
  declare @CustomerID int
  EXEC @CustomerID = dbo.FindCustomerByEmail @CustomerEmail -- varchar(15)

  if @CustomerID is not null begin
   insert into ConferenceReservations (CustomerID)
   values (@CustomerID)
   set @ReservationID = @@IDENTITY
  end
 commit tran tr
end try
begin CATCH
 PRINT ERROR_MESSAGE()
 rollback tran tr
end catch
end
go
```

NewDayReservation

Dodaje nową rezerwację dnia

```
CREATE procedure [dbo].[NewDayReservation]
 @CustomerEmail varchar(100),
 @ConferenceName varchar(200),
 @ConferenceDayDate date,
 @OrderDate DATE,
 @AdultSeats int,
 @StudentSeats int
as
begin
begin try
 begin tran tr
  DECLARE @ConferenceID INT,
          @ConferenceDayID INT;
  EXEC dbo.FindConference @ConferenceName = @ConferenceName,                    --
varchar(200)
                          @Date = @ConferenceDayDate,                       -- date
                          @ConferenceID = @ConferenceID OUTPUT,       -- int
                          @ConferenceDayID = @ConferenceDayID OUTPUT -- int
  if @ConferenceDayID is not null begin
   declare @ReservationID int
   exec FindReservation @CustomerEmail, @OrderDate, @ReservationID OUTPUT
   insert into dbo.ConferenceDayReservation
   (ConferenceDayID, ReservedAdultSeats, ReservedStudentSeats, ReservationID)
   values (@ConferenceDayID, @AdultSeats, @StudentSeats, @ReservationID)
  end
 commit tran tr
end try
begin CATCH
 PRINT ERROR_MESSAGE()
 rollback tran tr end catch
end
go
```

NewParticipant

Dodaje uczestnika

```
CREATE procedure [dbo].[NewParticipant]
 @FirstName varchar(30),
 @LastName varchar(50),
 @Phone varchar(15),
 @Email varchar(50),
 @ParticipantID INT OUTPUT
as
begin
insert into Participants (FirstName, LastName, Phone, Email) values (@FirstName,
@LastName, @Phone, @Email)
SET @ParticipantID = (SELECT MAX(ParticipantID) FROM dbo.Participants)
end
go
```

NewWorkshop

Dodaje warsztat do relacji słownikowej

```
create procedure NewWorkshop
 @Name varchar(100),
 @Description varchar(1000)
as
begin
insert into Workshops (Name, Description) values (@Name, @Description)
end
go
```

NewWorkshopReservation

Dodaje nową rezerwację na warsztat

```
CREATE PROCEDURE [dbo].[NewWorkshopReservation]
 @ConferenceName VARCHAR(200),
 @ConfDayDate DATE,
 @WorkshopName VARCHAR(200),
 @StartTime TIME ,
 @CustomerEmail VARCHAR(100),
 @DateConferenceOrdered DATE,
 @SeatsReserved INT
AS
BEGIN
BEGIN TRY
 BEGIN TRAN TR
  DECLARE @DayReservationID INT, @WorkshopInDayID INT
  EXEC FindWorkshopInDay @ConferenceName, @ConfDayDate, @WorkshopName, @StartTime,
@WorkshopInDayID OUTPUT
  EXEC dbo.FindConferenceDayReservation @ConferenceName = @ConferenceName,
-- varchar(200)
                                        @ConfDayDate = @ConfDayDate,
-- date
                                        @CustomerEmail = @CustomerEmail,
-- varchar(100)
                                        @DateOrdered = @DateConferenceOrdered,
-- date
                                        @ConferenceDayReservationID = @DayReservationID
OUTPUT -- int
  IF @DayReservationID IS NULL RAISERROR('Nie znaleziono rezerwacji',11,1)
  IF @WorkshopInDayID IS NULL RAISERROR ('Nie znaleziono warsztatu',11,1)
  INSERT INTO dbo.WorkshopReservation
  (
      ConferenceDayWorkshopID,
      ConferenceDayReservationID,
      ReservedSeats
  )
  VALUES
  (   @WorkshopInDayID, -- ConferenceDayWorkshopID - int
      @DayReservationID, -- ConferenceDayReservationID - int
      @SeatsReserved  -- ReservedSeats - int
      )
 COMMIT TRAN TR
END TRY
BEGIN CATCH
 PRINT ERROR_MESSAGE()
 ROLLBACK TRAN TR
END CATCH
end
go
```

ShowParticipantsOfConference

Pokazuje uczestników konferencji o danym ID

```
create procedure ShowParticipantsOfConference
 @ConferenceID int
as
begin
 select FirstName, LastName, participants.Phone, CompanyName
 from ConferenceDayParticipants
 inner join Participants on Participants.ParticipantID =
ConferenceDayParticipants.ParticipantID
 inner join EmployeesOfCompanies on Participants.ParticipantID =
EmployeesOfCompanies.ParticipantID
 inner join Companies on EmployeesOfCompanies.CompanyID = Companies.CompanyID
 inner join ConferenceDayReservation on ConferenceDayReservationID = DayReservationID
```

```
 inner join ConferenceDays on ConferenceDayReservation.ConferenceDayID =
ConferenceDays.ConferenceDayID
 where ConferenceDays.ConferenceID = @ConferenceID
end
go
```

ShowParticipantsOfConferenceDay

Pokazuje uczestników dnia konferencji o danym ID i danym numerze dnia wewnątrz konferencji.

```
create procedure ShowParticipantsOfConferenceDay
 @ConferenceID int,
 @ConferenceDayOrdinal int
as
begin
 select FirstName, LastName, participants.Phone, CompanyName
 from ConferenceDayParticipants
 inner join Participants on Participants.ParticipantID =
ConferenceDayParticipants.ParticipantID
 inner join EmployeesOfCompanies on Participants.ParticipantID =
EmployeesOfCompanies.ParticipantID
 inner join Companies on EmployeesOfCompanies.CompanyID = Companies.CompanyID
 inner join ConferenceDayReservation on ConferenceDayReservationID = DayReservationID
 inner join ConferenceDays on ConferenceDayReservation.ConferenceDayID =
ConferenceDays.ConferenceDayID
 where ConferenceDays.DayOrdinal = @ConferenceDayOrdinal and
ConferenceDays.ConferenceID = @ConferenceID
end
go
```

Generator

Dane do bazy zostały w większości wygenerowane przez program SQL Data Generator firmy RedGate. Do danych, które wymagały większej precyzji z powodu narzuconych dokładnie warunków integralnościowych napisano programy w Javie generujące polecenia INSERT INTO.

Kod generujący uzupełnianie pustych krotek Students powstałych w wyniku działania triggera

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class Shuffler {

    static String randomID() {
        String toReturn = "";
        for(int i = 0; i < 6; i++) {
            toReturn = toReturn.concat(Integer.toString(new Random().nextInt(10)));
        }
        return toReturn;
    }

    public static void main(String ... args) throws IOException {

        Path c = Path.of("C:\\Program Files (x86)\\Red Gate\\SQL Data Generator
4\\Config\\NamesFirst.txt");
```

```
        Path s = Path.of("C:\\Program Files (x86)\\Red Gate\\SQL Data Generator
4\\Config\\NamesLast.txt");
        Path p = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\participantids.txt");
        Path out = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\res2.sql");

        List<String> firstNames = Files.lines(c).collect(Collectors.toList());
        List<String> lastNames = Files.lines(s).collect(Collectors.toList());
        List<String> participantIds = Files.lines(p).collect(Collectors.toList());
        Random r = new Random();

        try(FileWriter fw = new FileWriter(out.toFile(), true);
            BufferedWriter bw = new BufferedWriter(fw);
            PrintWriter outt = new PrintWriter(bw)) {
            for (String participantId : participantIds) {
                outt.println("update Students set StudentCardNumber = '" + randomID() +
"' where ParticipantID = " + participantId + "\ngo");
            }

        } catch(IOException e) {
            e.printStackTrace();
        }

    }

}
```

Kod generujący następujące po sobie progi cenowe

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.text.DecimalFormat;
import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class Shuffler {

    public static void main(String ... args) throws IOException {

        Path c = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\createdates.txt");
        Path s = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\startdates.txt");
        Path cid = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\confids.txt");
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");
        List<String> conferenceIds = Files.lines(cid).collect(Collectors.toList());
        List<LocalDate> createDates = Files.lines(c).map(d ->
LocalDate.from(dtf.parse(d))).collect(Collectors.toList());
        List<LocalDate> startDates = Files.lines(s).map(d ->
LocalDate.from(dtf.parse(d))).collect(Collectors.toList());

        Random r = new Random();

        for(int i = 0; i < conferenceIds.size(); i++) {
            LocalDate createdOn = createDates.get(i).minusDays(1);
            LocalDate startOn = startDates.get(i);
            String conferenceId = conferenceIds.get(i);
            System.out.println("insert into ConferencePricetables (ConferenceID,
PriceStartsOn, PriceEndsOn, DiscountRate) " +
                    "values (" + conferenceId + ", '" + dtf.format(createdOn) + "', '"
+ dtf.format(createdOn) + "', 1)\ngo");
```

```
            double currentDiscount = (r.nextDouble() + 1) / 2;
            LocalDate currentEndDiscountDate = createdOn;
            while(currentDiscount > 0) {
                LocalDate newDiscountStart = currentEndDiscountDate.plusDays(1);
                int discountLength = r.nextInt(10) + 7;
                LocalDate newDiscountEnd = newDiscountStart.plusDays(discountLength);
                if(newDiscountEnd.isAfter(startOn.minusDays(1))) break;
                currentDiscount -= r.nextDouble() / 2;
                if(currentDiscount < 0) break;
                System.out.println("insert into ConferencePricetables (ConferenceID,
PriceStartsOn, PriceEndsOn, DiscountRate) " +
                        "values (" + conferenceId + ", '" +
dtf.format(newDiscountStart) + "', '" + dtf.format(newDiscountEnd) + "', " +
currentDiscount + ")");
                System.out.println("go");
                currentEndDiscountDate = newDiscountEnd;
            }
        }

    }

}
```

Kod generujący uzupełnianie pustych krotek Participants powstałych w wyniku działania triggera

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class Shuffler {

    static String randomPhone() {
        String phone = "";
        Random r = new Random();
        for(int i = 0; i < 9; i++) {
            phone = phone.concat(Character.toString((char) (r.nextInt(9) + 49)));
        }
        return phone;
    }

    static String randomEmail() {
        String allowedChars = "abcdefghijklmnopqrstuvwxyz0123456789";
        String email = "";
        Random r = new Random();
        int length = r.nextInt(8) + 6;
        for(int i = 0; i < length; i++) {
            email =
email.concat(Character.toString(allowedChars.charAt(r.nextInt(allowedChars.length()))))
;
        }
        return email + "@example.com";
    }

    public static void main(String ... args) throws IOException {

        Path c = Path.of("C:\\Program Files (x86)\\Red Gate\\SQL Data Generator
4\\Config\\NamesFirst.txt");
        Path s = Path.of("C:\\Program Files (x86)\\Red Gate\\SQL Data Generator
4\\Config\\NamesLast.txt");
        Path p = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\participantids.txt");
```

```
        Path out = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\res2.sql");

        List<String> firstNames = Files.lines(c).collect(Collectors.toList());
        List<String> lastNames = Files.lines(s).collect(Collectors.toList());
        List<String> participantIds = Files.lines(p).collect(Collectors.toList());
        Random r = new Random();

        try(FileWriter fw = new FileWriter(out.toFile(), true);
            BufferedWriter bw = new BufferedWriter(fw);
            PrintWriter outt = new PrintWriter(bw)) {
            for (String participantId : participantIds) {
                outt.println("update Participants set FirstName = '" +
firstNames.get(r.nextInt(firstNames.size())) + "', " +
                        "LastName = '" + lastNames.get(r.nextInt(lastNames.size())) +
"', Phone = '" + randomPhone() +
                        "', Email = '" + randomEmail() + "' where LastName is null and
ParticipantID = " + participantId + "\ngo");
            }

        } catch(IOException e) {
            e.printStackTrace();
        }

    }

}
```

Kod generujący nazwy warsztatów

```
import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Random;

public class Shuffler {

    public static void main(String ... args) {

        List<String> words =
Arrays.asList("Zabawa","Czesto","Nigdy","Czlowiek","Zdrowie","Odpoczynek",
                "Zaufanie", "Prawie", "Zupelnie", "Pewnosc", "Czystosc",
"Jednoznacznie", "Praktycznie", "Rower", "Kanibalizm",
                "Z Pomyslem", "Polska", "Reedukacja", "Ciekawie");

        for(int i = 0; i < 1000; i++) {
            int l = new Random().nextInt(4) + 2;
            Collections.shuffle(words);
            StringBuilder sb = new StringBuilder();
            for(int j = 0; j < l; j++) sb.append(words.get(j)).append(" ");
            System.out.println(sb.toString().trim());
        }

    }

}
```

Kod generujący uczestników warsztatów (indeksy w plikach spełniają constrainty)

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.List;
```

```
import java.util.stream.Collectors;

public class Shuffler {

    public static void main(String ... args) throws IOException {

        Path c = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\workshopids.txt");
        Path s = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\participantids.txt");
        Path out = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\res.sql");

        List<String> workshopIds = Files.lines(c).collect(Collectors.toList());
        List<String> participantIds = Files.lines(s).collect(Collectors.toList());

        try(FileWriter fw = new FileWriter(out.toFile(), true);
            BufferedWriter bw = new BufferedWriter(fw);
            PrintWriter outt = new PrintWriter(bw)) {
            for (int i = 0; i < workshopIds.size(); i++) {
                outt.println("insert into WorkshopParticipants
(ConferenceDayParticipantID, ConferenceDayWorkshopID) values (" +
                        participantIds.get(i) + ", " + workshopIds.get(i) + ")\ngo");
            }

        } catch(IOException e) {
            e.printStackTrace();
        }

    }

}
```

Kod generujący rezerwacje na warsztaty

```
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.StandardOpenOption;
import java.util.List;
import java.util.Random;
import java.util.stream.Collectors;

public class Shuffler {

    public static void main(String ... args) throws IOException {

        Path p = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\dayreservationids.txt");
        List<Integer> reservationIds =
Files.lines(p).mapToInt(Integer::parseInt).boxed().collect(Collectors.toList());
        Path w = Path.of("C:\\Users\\Lenovo\\Desktop\\AGH\\nauka\\Sem 3\\Bazy
Danych\\projekt\\generator danych\\workshopids.txt");
        List<Integer> workshopIds =
Files.lines(w).mapToInt(Integer::parseInt).boxed().collect(Collectors.toList());
        int reservationIdsSize = reservationIds.size();
        int workshopIdsSize = workshopIds.size();

        Random r = new Random();

        for(int i = 0; i < 50000; i++) {
            int seats = r.nextInt(20) + 1;
            int index = r.nextInt(11930);
            int reservationId = reservationIds.get(index);
            int workshopId = workshopIds.get(index);
            StringBuilder sql = new StringBuilder("insert into WorkshopReservation " +
                    "(ConferenceDayWorkshopID, ConferenceDayReservationID,
```

```
ReservedSeats) values (");
            if (r.nextDouble() > 3.0 / 4.0) {
                seats = 1;
            }
            sql.append(workshopId + ", " + reservationId + ", " + seats + ")");
            sql.append("\ngo");
            System.out.println(sql);
        }

    }

}
```

Poprawki do kodu wygenerowanego przez SQL Data Generator

```
UPDATE dbo.Conferences SET PostalCode = NULL WHERE CityID IS NULL

UPDATE Conferences SET StudentDiscount = ROUND(StudentDiscount,2)

UPDATE Conferences SET BasePriceForDay = ROUND(BasePriceForDay,2)

DELETE FROM dbo.ConferenceDayWorkshops WHERE
[dbo].[ConferenceSize]([ConferenceDayID])<[ParticipantsLimit]

UPDATE dbo.ConferenceDayWorkshops SET price = 0 WHERE price IS NULL

UPDATE dbo.ConferenceDayWorkshops SET price = ROUND(Price, 1)

UPDATE dbo.ConferenceDayWorkshops SET StartTime = CONVERT(varchar(5), StartTime)

UPDATE dbo.ConferenceDayWorkshops SET EndTime = CONVERT(VARCHAR(5), EndTime)

DELETE FROM ConferenceReservations WHERE ReservationID NOT IN (SELECT ReservationID
FROM ConferenceDayReservation)

DELETE FROM ConferencePricetables WHERE DiscountRate < 0.01
```