AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Wydział Informatyki, Elektroniki i Telekomunikacji

Podstawy baz danych

System zarządzania konferencjami

Dokumentacja projektu

Autor:
Piotr Wróbel

Opieka nad projektem:
dr inż. Robert Marcjan

semestr zimowy 2018/19

Spis treści

1. Opis funkcji systemu

Przedstawiony projekt ma na celu stworzenie bazy danych wspomagającej działalność firmy organizującej konferencje. Rejestracja organizowana jest poprzez stronę internetową, klientami mogą być zarówno osoby prywatne jak i firmy. Możliwa jest zbiorcza rezerwacja wielu miejsc z podaniem danych uczestników w późniejszym terminie. Dla konferencji wielodniowych nie ma konieczności zapisywania się na cały ich przebieg i dla każdej osoby biorącej w nich udział istnieje możliwość indywidualnego wyboru poszczególnych dni oraz warsztatów (jeżeli takie są organizowane).

1.1. Funkcjonalności systemu

Do aktorów korzystających z systemów należą administrator systemu, organizator konferencji oraz klient (indywidualny bądź firma) o następujących funkcjonalnościach:

1.1.1. Administrator systemu

— Przydzielanie uprawnień pozostałym użytkownikom
— Pełen dostęp do bazy danych

1.1.2. Organizator konferencji

— Dodanie nowej konferencji wraz z jej nazwą, terminem, liczbą dni, terminami ewentualnych warsztatów, obowiązującymi progami cenowymi i zniżkami studenckimi, limitem miejsc na poszczególne dni/warsztaty oraz miejscem odbywania się konferencji
— Wyświetlenie listy wszystkich klientów
— Wyświetlenie listy wszystkich uczestników danej konferencji z możliwością sporządzenia jej tylko dla poszczególnych dni lub warsztatów
— Dostęp do danych o płatnościach, możliwość anulowania nieopłaconej rezerwacji
— Możliwość modyfikacji danych konferencji wraz z wymuszeniem konsekwencji zmian (np. w przypadku zmiany liczby miejsc na konkretny warsztat)
— Uniemożliwienie klientowi rejestracji na kolidujące ze sobą warsztaty
— Generowanie raportu statystycznego na temat klientów, którzy najczęściej korzystają z usług danego organizatora

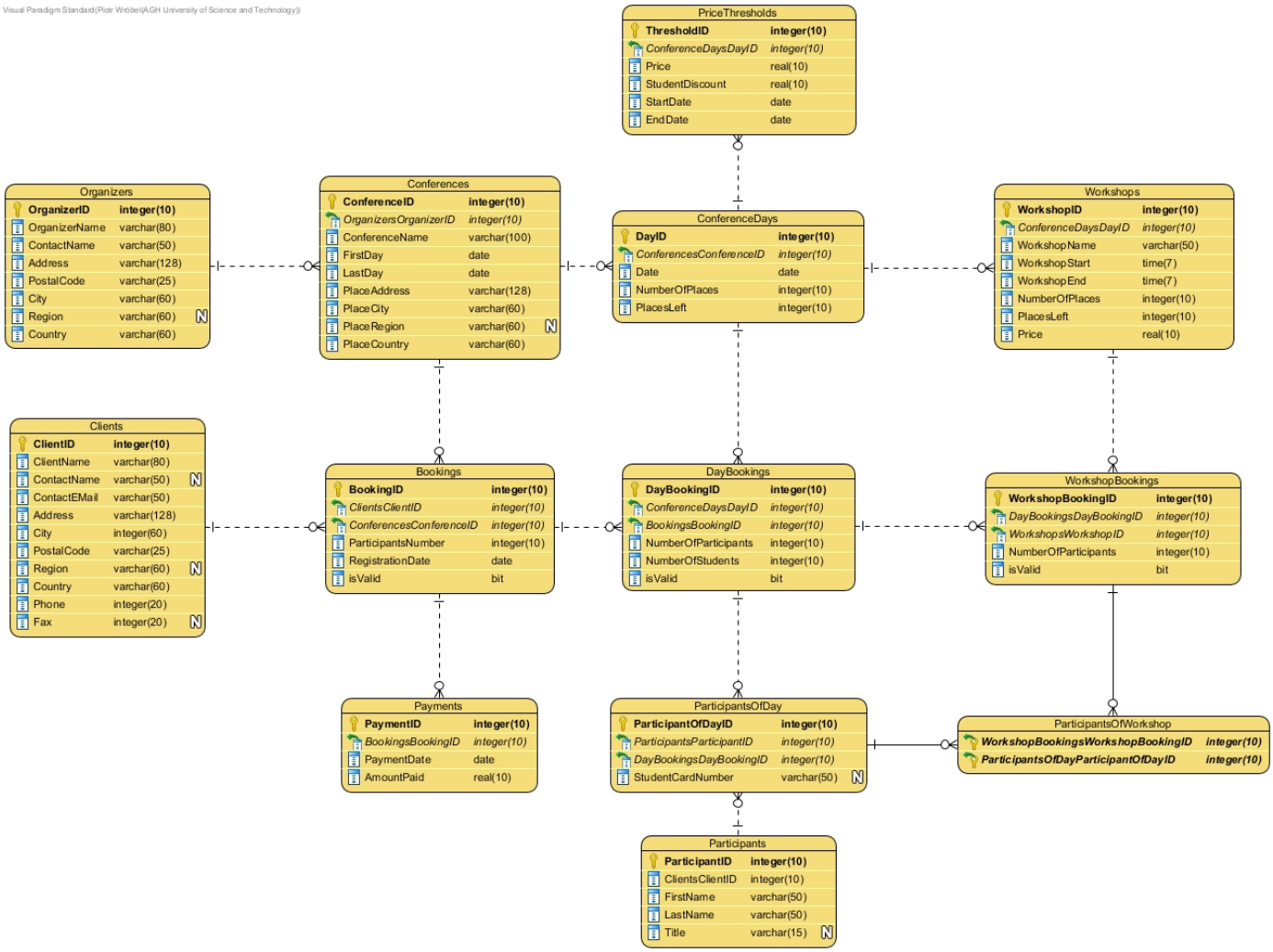
1.1.3. Klient

— Prezentowanie informacji o konferencji: jej nazwy, dni, odbywających się warsztatów oraz poszczególnych cen
— Rezerwacja określonej ilości miejsc na wybrane dni, bez konieczności podawania danych uczestników

— Rejestracja uczestników na warsztaty
— Modyfikacja danych rezerwacji w zakresie ilości miejsc, przydziału dni lub warsztatów
— Uzupełnienie danych osobowych uczestników w odpowiednim terminie
— Udostępnienie informacji o historii płatności związanych z daną konferencją i pozostałą do zapłaty kwotą

2. Schemat bazy danych

2.1. Diagram bazy danych



Rysunek 2.1. Schemat projektowanej bazy danych wykonany w programie Visual Paradigm

2.2. Opis tabel

2.2.1. Tabela Organizers

Zawiera dane organizatorów konferencji – nazwę organizatora, dane osoby kontaktowej i adres.

Pola tabeli:

— **OrganizerID** – identyfikator organizatora, **klucz główny**
— **OrganizerName** – nazwa organizatora
— **ContactName** – imię i nazwisko osoby kontaktowej
— **Address** – ulica i numer lokalu organizatora
— **Postal Code** – kod pocztowy
— **City** – miasto
— **Region** – województwo/stan (jeśli dotyczy, inaczej pole ma wartoć null)
— **Country** – kraj

Kod SQL generujący tabelę:

```
1  create table Organizers (
2    OrganizerID            int not null identity,
3    OrganizerName          nvarchar(80) not null,
4    ContactName            nvarchar(50) not null,
5    Address                nvarchar(128) not null,
6    PostalCode             nvarchar(25) not null,
7    City                   nvarchar(60) not null,
8    Region                 nvarchar(60) default null,
9    Country                nvarchar(60) not null,
10   constraint PK_Organizers
11     primary key (OrganizerID)
12 )
```

Warunki integralnościowe

— pole OrganizerID jest kluczem głównym
— wszystkie pola poza Region nie mogą mieć wartości pustej

2.2.2. Tabela Conferences

Zawiera dane konferencji – jej nazwę, dni w których się odbywa oraz dane adresowe miejsca

Pola tabeli:

— **ConferenceID** – identyfikator konferencji, **klucz główny**
— **OrganizerID** – identyfikator organizatora konferencji, **klucz obcy**
— **ConferenceName** – nazwa konferencji
— **FirstDay** – pierwszy dzień konferencji
— **LastDay** – ostatni dzień konferencji
— **PlaceAddress** – ulica i numer lokalu miejsca, w którym odbywa się konferencja
— **PlaceCity** – miasto, w którym odbywa się konferencja
— **PlaceRegion** – województwo/stan, w którym odbywa się konferencja
— **PlaceCountry** – kraj, w którym odbywa się konferencja

Kod SQL generujący tabelę:

```
create table Conferences (
  ConferenceID          int not null identity,
  OrganizerID           int not null,
  ConferenceName        nvarchar(100) not null,
  FirstDay              date not null,
  LastDay               date not null,
  PlaceAddress          nvarchar(128) not null,
  PlaceCity             nvarchar(60) not null,
  PlaceRegion           nvarchar(60) default null,
  PlaceCountry          nvarchar(60) not null,
  constraint PK_Conferences
    primary key (ConferenceID),
  constraint Conferences_Organizers
    foreign key (OrganizerID)
    references Organizers (OrganizerID),
  constraint CK_LastDay_FirstDay_Conferences
    check (LastDay >= FirstDay)
)
```

Warunki integralnościowe

— pole ConferenceID jest kluczem głównym
— pole OrganizerID jest kluczem obcym
— wszystkie pola poza PlaceRegion muszą być niepuste
— data zapisana w polu LastDay nie może być wcześniejsza niż ta z FirstDay

2.2.3. Tabela Clients

Zawiera dane klientów firmowych jak i indywidualnych

Pola tabeli:

— **ClientID** – identyfikator klienta, **klucz główny**
— **ClientName** – nazwa klienta firmowego/imię i nazwisko klienta indywidualnego
— **ContactName** – imię i nazwisko osoby kontaktowej dla klientów firmowych
— **ContactEMail** – adres e-mailowy klienta
— **Address** – ulica i numer lokalu klienta
— **City** – miasto zamieszkania klienta
— **PostalCode** – kod pocztowy klienta
— **Region** – stan/województwo zamieszkania klienta
— **Country** – kraj zamieszkania klienta
— **Phone** – telefon do klienta
— **Fax** – fax klienta

Kod SQL generujący tabelę:

```
1  create table Clients (
2    ClientID              int not null identity,
3    ClientName            nvarchar(80) not null,
4    ContactName           nvarchar(50) default null,
5    ContactEMail          nvarchar(50) not null check
6      (ContactEMail like '%_@_%._%'),
7    Address               nvarchar(128) not null,
8    City                  nvarchar(60) not null,
9    PostalCode            nvarchar(25) not null,
10   Region                nvarchar(60) default null,
11   Country               nvarchar(60) not null,
12   Phone                 varchar(20) not null,
13   Fax                   varchar(20) default null,
14   constraint PK_Clients
15     primary key (ClientID)
16 )
```

Warunki integralnościowe

— pole ClientID jest kluczem głównym
— pola ClientName, ClientEMail, Address, City, PostalCode, Country, Phone nie mogą być puste, pozostałe są uzupełniane w razie potrzeby (np. dla klienta firmowego pole ContactName)
— pole ContactEMail musi mieć określony format

2.2.4. Tabela Bookings

Zawiera dane rezerwacji wykonanych przez klientów

Pola tabeli:

— **BookingID** – identyfikator rezerwacji, **klucz główny**
— **ClientID** – identyfikator klienta, który wykonał rezerwację, **klucz obcy**
— **ConferenceID** – identyfikator konferencji, na którą wykonano rezerwację, **klucz obcy**
— **ParticipantsNumber** – liczba osób, dla których wykonywana jest rezerwacja
— **RegistrationDate** – data wykonania rezerwacji
— **isValid** – czy rezerwacja jest ważna

Kod SQL generujący tabelę:

```
create table Bookings (
  BookingID             int not null identity,
  ClientID              int not null,
  ConferenceID          int not null,
  ParticipantsNumber    int not null check
    (ParticipantsNumber > 0),
  RegistrationDate      datetime not null default getdate(),
  isValid               bit not null default 1,
  constraint PK_Bookings
    primary key (BookingID),
  constraint Bookings_Clients
    foreign key (ClientID)
    references Clients (ClientID),
  constraint Bookings_Conferences
    foreign key (ConferenceID)
    references Conferences (ConferenceID)
)
```

Warunki integralnościowe

— pole BookingID jest kluczem głównym
— pola ClientID i ConferenceID są kluczami obcymi
— pozostałe pola muszą być niepuste
— pole ParticipantsNumber musi zawierać liczbę większą od 0

2.2.5. Tabela Payments

Zawiera dane dotyczące płatności wykonanych dla danej rezerwacji

Pola tabeli:

— **PaymentID** – identyfikator płatności, **klucz główny**
— **BookingID** – identyfikator rezerwacji, **klucz obcy**
— **PaymentDate** – data wykonania płatności
— **AmountPaid** – zapłacona kwota

Kod SQL generujący tabelę:

```
1  create table Payments (
2    PaymentID              int not null identity ,
3    BookingID              int not null,
4    PaymentDate            datetime not null default getdate(),
5    AmountPaid             money not null check
6      (AmountPaid > 0),
7    constraint PK_Payments
8      primary key (PaymentID),
9    constraint Payments_Bookings
10     foreign key (BookingID)
11     references Bookings (BookingID)
12 )
```

Warunki integralnościowe

— pole PaymentID jest kluczem głównym
— pole Booking ID jest kluczem obcym
— pozostałe pola muszą być niepuste
— pole AmountPaid musi zawierać liczbę większą od 0

2.2.6. Tabela ConferenceDays

Zawiera informacje dotyczące ilości miejsc na poszczególne dni konferencji

Pola tabeli:

— **DayID** – identyfikator dnia, **klucz główny**
— **ConferenceID** – identyfikator konferencji, **klucz obcy**
— **Date** – data
— **NumberOfPlaces** – liczba miejsc na dany dzień
— **PlacesLeft** – pozostała liczba miejsc

Kod SQL generujący tabelę:

```
create table ConferenceDays (
  DayID                 int not null identity,
  ConferenceID          int not null,
  Date                  date not null,
  NumberOfPlaces        int not null check
    (NumberOfPlaces > 0),
  PlacesLeft            int not null check
    (PlacesLeft >= 0),
  constraint PK_ConferenceDays
    primary key (DayID),
  constraint ConferenceDays_Conferences
    foreign key (ConferenceID)
    references Conferences (ConferenceID),
  constraint CK_Places_ConferenceDays
    check (PlacesLeft <= NumberOfPlaces),
  constraint Unique_Days_Conf
    unique (ConferenceID, Date)
)
```

Warunki integralnościowe

— pole DayID jest kluczem głównym
— pole ConferenceID jest kluczem obcym
— pole Date musi być niepuste, pozostałe organizator może uzupełnić w późniejszym terminie
— pole NumberOfPlaces musi zawierać liczbę większą od 0
— pole PlacesLeft musi zawierać liczbę nie mniejszą od 0 i nie większą od NumberOfPlaces, automatyczne ustawienie tej wartości zostanie rozwiązane za pomocą triggera
— każda para (ConferenceID, Date) musi być jedyna

2.2.7. Tabela PriceThresholds

Zawiera informacje dotyczące progów cenowych na poszczególne dni konferencji

Pola tabeli:

— **ThresholdID** – identyfikator progu, **klucz główny**
— **DayID** – identyfikator dnia konferencji, **klucz obcy**
— **Price** – cena
— **StudentDiscount** – zniżka studencka (o ile jest określana)
— **StartDate** – początek obowiązywania danego progu
— **EndDate** – koniec obowiązywania danego progu

Kod SQL generujący tabelę:

```
create table PriceThresholds (
  ThresholdID            int not null identity,
  DayID                  int not null,
  Price                  money not null check
    (Price >= 0.0),
  StudentDiscount        real not null default 0 check
    (StudentDiscount between 0.0 and 100.0),
  StartDate              date not null,
  EndDate                date not null,
  constraint PK_PriceThresholds
    primary key (ThresholdID),
  constraint PriceThresholds_ConferenceDays
    foreign key (DayID)
    references ConferenceDays (DayID),
  constraint CK_Dates_PriceThresholds
    check (EndDate >= StartDate)
)
```

Warunki integralnościowe

— pole ThresholdID jest kluczem głównym
— pole DayID jest kluczem obcym
— wszystkie pola muszą być niepuste
— zawartość pola Price musi być większa od 0
— zawartość pola StudentDiscount musi być między 0 a 100
— data zapisana w EndDate musi być niewcześniejsza niż ta w StartDate

2.2.8. Tabela Workshops

Zawiera informacje dotyczące warsztatów odbywających się w poszczególnych dniach konferencji

Pola tabeli:

— **WorkshopID** – identyfikator warsztatu, **klucz główny**
— **DayID** – identyfikator dnia konferencji, **klucz obcy**
— **WorkshopName** – nazwa warsztatu
— **WorkshopStart** – godzina rozpoczęcia
— **WorkshopEnd** – godzina zakończenia
— **NumberOfPlaces** – liczba miejsc
— **PlacesLeft** – liczba pozostałych miejsc
— **Price** – cena

Kod SQL generujący tabelę:

```
create table Workshops (
  WorkshopID              int not null identity,
  DayID                   int not null,
  WorkshopName            nvarchar(50) not null,
  WorkshopStart           time not null,
  WorkshopEnd             time not null,
  NumberOfPlaces          int not null check
    (NumberOfPlaces > 0),
  PlacesLeft              int not null check
    (PlacesLeft >= 0),
  Price                   money not null default 0 check
    (Price >= 0.0),
  constraint PK_Workshops
    primary key (WorkshopID),
  constraint Workshops_ConferenceDays
    foreign key (DayID)
    references ConferenceDays (DayID),
  constraint CK_Time_Workshops
    check (WorkshopEnd > WorkshopStart),
  constraint CK_Places_Workshops
    check (PlacesLeft <= NumberOfPlaces)
)
```

Warunki integralnościowe

— pole WorkshopID jest kluczem głównym
— pole DayID jest kluczem obcym
— wszystkie pola muszą być niepuste
— zawartość pola NumberOfPlaces musi być większa od 0
— zawartość pól PlacesLeft, Price musi być nie mniejsza od 0
— godzina zapisana w End musi być późniejsza niż ta w Start

2.2.9. Tabela DayBookings

Zawiera szczegółowe dane rezerwacji dotyczące ilości miejsc rezerwowanych na poszczególne dni konferencji

Pola tabeli:

— **DayBookingID** – identyfikator rezerwacji na dany dzień, **klucz główny**
— **DayID** – identyfikator dnia konferencji, **klucz obcy**

— **BookingID** – identyfikator rezerwacji, **klucz obcy**
— **NumberOfParticipants** – liczba uczestników (włącznie ze studentami)
— **NumberOfStudents** – liczba uczestniczących studentów
— **isValid** – czy rezerwacja jest ważna

Kod SQL generujący tabelę:

```
create table DayBookings (
  DayBookingID          int not null identity,
  DayID                 int not null,
  BookingID             int not null,
  NumberOfParticipants  int not null check
    (NumberOfParticipants > 0),
  NumberOfStudents      int not null default 0 check
    (NumberOfStudents >= 0),
  isValid               bit not null default 1,
  constraint PK_DayBookings
    primary key (DayBookingID),
  constraint DayBookings_ConferenceDays
    foreign key (DayID)
    references ConferenceDays (DayID),
  constraint DayBookings_Bookings
    foreign key (BookingID)
    references Bookings (BookingID),
  constraint CK_Students_DayBookings
    check (NumberOfStudents <= NumberOfParticipants),
  constraint unique_booking
    unique (DayID, BookingID)
)
```

Warunki integralnościowe

— pole DayBookingID jest kluczem głównym
— pola DayID i BookingID są kluczami obcymi
— wszystkie pola muszą być niepuste
— zawartość pola NumberOfParticipants musi być większa od 0
— zawartość pola NumberOfStudents musi być nie większa niż pola NumberOfParticipants i nie mniejsza niż 0
— każda para (DayID, BookingID) musi być jedyna

2.2.10. Tabela WorkshopBookings

Zawiera szczegółowe dane rezerwacji dotyczące ilości rezerwowanych miejsc na warsztaty odbywające się w danym dniu konferencji

Pola tabeli:

— **WorkshopBookingID** – identyfikator rezerwacji na określony warsztat, **klucz główny**
— **DayBookingID** – identyfikator rezerwacji na określony dzień konferencji, **klucz obcy**
— **WorkshopID** – identyfikator warsztatu, **klucz obcy**
— **NumberOfParticipants** – liczba uczestników
— **isValid** – czy rezerwacja jest ważna

Kod SQL generujący tabelę:

```
create table WorkshopBookings (
  WorkshopBookingID      int not null identity,
  DayBookingID           int not null,
  WorkshopID             int not null,
  NumberOfParticipants   int not null check
    (NumberOfParticipants > 0),
  isValid                bit not null default 1,
  constraint PK_WorkshopBookings
    primary key (WorkshopBookingID),
  constraint WorkshopBookings_DayBookings
    foreign key (DayBookingID)
    references DayBookings (DayBookingID),
  constraint Workshop_WorkshopBookings
    foreign key (WorkshopID)
    references Workshops (WorkshopID)
  constraint Unique_Workshop_on_day
    unique (WorkshopID, DayBookingID)
)
```

Warunki integralnościowe

— pole WorkshopBookingID jest kluczem głównym
— pola DayBookingID i WorkshopID są kluczami obcymi
— pola NumberOfParticipants i isValid muszą być niepuste
— zawartość pola NumberOfParticipants musi być większa od 0
— każda para (WorkshopID, DayBookingID) musi być jedyna

2.2.11. Tabela Participants

Zawiera dane uczestników konferencji, uzupełniane przez klienta po wykonaniu rezerwacji do 2 tygodni przed rozpoczęciem konferencji

Pola tabeli:

— **ParticipantID** – identyfikator uczestnika, **klucz główny**
— **FirstName** – Imię/imiona uczestnika
— **LastName** – Nazwisko uczestnika
— **Title** – tytuł uczestnika

Kod SQL generujący tabelę:

```
create table Participants (
  ParticipantID          int not null identity,
  FirstName              nvarchar(50) not null,
  LastName               nvarchar(50) not null,
  Title                  nvarchar(15) default null,
  constraint PK_Participants
    primary key (ParticipantID)
)
```

Warunki integralnościowe

— pole ParticipantID jest kluczem głównym
— pola FirstName, LastName i isIDCardTaken muszą być niepuste

2.2.12. Tabela ParticipantsOfDay

Przypisanie uczestników do zarezerwowanych miejsc na poszczególne dni

Pola tabeli:

— **ParticipantOfDayID** – identyfikator przypisania do dnia, **klucz główny**
— **ParticipantID** – identyfikator uczestnika, **klucz obcy**
— **DayBookingID** – identyfikator rezerwacji na określony dzień, **klucz obcy**
— **StudentCardNumber** – numer legitymacji studenckiej uczestnika (w tej tabeli, aby dla danej rezerwacji pojawial się tylko aktualny stan posiadania przez uczestnika statusu studenta)

Kod SQL generujący tabelę:

```
create table ParticipantsOfDay (
  ParticipantOfDayID     int not null identity,
  ParticipantID          int not null,
```

```
  DayBookingID          int not null,
  StudentCardNumber     varchar(50) default null,
  constraint PK_ParticipantsOfDay
    primary key (ParticipantOfDayID),
  constraint ParticipantsOfDay_Participants
    foreign key (ParticipantID)
    references Participants (ParticipantID),
  constraint ParticipantsOfDay_DayBookings
    foreign key (DayBookingID)
    references DayBookings (DayBookingID),
  constraint Unique_Participant_for_Day
    unique (ParticipantID, DayBookingID)
)
```

Warunki integralnościowe

— pole ParticipantOfDayID jest kluczem głównym
— pola ParticipantID, DayBookingID są kluczami obcymi
— każda para (ParticipantID, DayBookingID) musi być jedyna

2.2.13. Tabela ParticipantsOfWorkshop

Przypisanie uczestników do zarezerwowanych miejsc na warsztaty

Pola tabeli:

— **WorkshopBookingID** – identyfikator rezerwacji dla warsztatu, **klucz główny**
— **ParticipantOfDayID** – identyfikator przypisania do dnia, **klucz główny**

Kod SQL generujący tabelę:

```
create table ParticipantsOfWorkshop (
  WorkshopBookingID     int not null,
  ParticipantOfDayID    int not null,
  constraint PK_ParticipantsOfWorkshop
    primary key  (WorkshopBookingID, ParticipantOfDayID),
  constraint ParticipantsOfWorkshop_WorkshopBookings
    foreign key (WorkshopBookingID)
    references WorkshopBookings (WorkshopBookingID),
  constraint ParticipantsOfWorkshop_ParticipantsOfDay
    foreign key (ParticipantOfDayID)
    references ParticipantsOfDay (ParticipantOfDayID)
)
```

Warunki integralnościowe

— pola WorkshopBookingID, ParticipantOfDayID tworzą klucz główny, co gwarantuje ich unikalność, jednocześnie są kluczami obcymi, tabela definiuje relację wiele-do-wiele

3. Widoki

3.1. Widoki związane z konferencją

3.1.1. FutureConferences

Pokazuje nadchodzące i trwające obecnie konferencje.

```
1  create view FutureConferences
2  as
3    select ConferenceID, ConferenceName, FirstDay, LastDay,
4      PlaceCity, PlaceCountry
5    from Conferences
6    where FirstDay > getdate() or getdate() between FirstDay and LastDay
```

3.1.2. DaysWithFreePlaces

Pokazuje przyszłe dni konferencji z wolnymi miejscami

```
1  create view DaysWithFreePlaces
2  as
3    select Conferences.ConferenceID, Conferences.ConferenceName,
4      CD.DayID, CD.Date, CD.PlacesLeft,
5      dbo.PriceOfDayOn(getdate(), CD.DayID) as Price
6    from Conferences
7    inner join ConferenceDays CD
8    on Conferences.ConferenceID = CD.ConferenceID
9    where CD.PlacesLeft > 0
10   and CD.Date >= getdate()
```

3.1.3. WorkshopsWithFreePlaces

Pokazuje przyszłe warsztaty z wolnymi miejscami

```
1  create view WorkshopsWithFreePlaces
2  as
3    select Conferences.ConferenceID, Conferences.ConferenceName,
4      CD.DayID, CD.Date,
5      W.WorkshopID, W.WorkshopName,
6      W.WorkshopStart, W.WorkshopEnd,
```

```
7       W.PlacesLeft, W.Price
8     from Conferences
9     inner join ConferenceDays CD
10    on Conferences.ConferenceID = CD.ConferenceID
11    inner join Workshops W
12    on CD.DayID = W.DayID
13    where W.PlacesLeft > 0
14    and CD.Date >= getdate()
```

3.1.4. ParticipantsOfConference

Pokazuje wszystkich zapisanych uczestników konferencji

```
1  create view ParticipantsOfConference
2  as
3    select distinct Conferences.ConferenceID,
4      Conferences.ConferenceName,
5      P.ParticipantID, P.Title,
6      P.LastName, P.FirstName
7    from Conferences
8    inner join Bookings B
9    on Conferences.ConferenceID = B.ConferenceID
10   inner join DayBookings DB
11   on B.BookingID = DB.BookingID
12   inner join ParticipantsOfDay POD
13   on DB.DayBookingID = POD.DayBookingID
14   inner join Participants P
15   on POD.ParticipantID = P.ParticipantID
16   where B.isValid = 1
17   and DB.isValid = 1
```

3.1.5. ClientsOfConference

Pokazuje wszystkich klientów, którzy wykonali ważną rezerwację.

```
1  create view ClientsOfConference
2  as
3    select Conferences.ConferenceID,
4      Conferences.ConferenceName,
5      C.ClientID, C.ClientName,
6      isnull(C.ContactName, C.ClientName) as ContactName,
7      C.ContactEMail, C.Address, C.PostalCode,
8      isnull(C.Region, '-') as Region, C.Country,
```

```
    B.ParticipantsNumber
  from Conferences
  inner join Bookings B
  on Conferences.ConferenceID = B.ConferenceID
  inner join Clients C
  on B.ClientID = C.ClientID
  where B.isValid = 1
```

3.1.6. CancelledBookings

Pokazuje anulowane rezerwacje.

```
create view CancelledBookings
as
  select BookingID, ClientID, ConferenceID, ParticipantsNumber
  from Bookings
  where isValid = 0
```

3.1.7. ClientStatistics

Pokazuje statystyki klientów – liczbę wykonanych ważnych rezerwacji i całkowitą sumę płatności

```
create view ClientStatistics
as
  select ClientID, ClientName,
    dbo.NumberOfBookedConferences(ClientID)
    as BookedConferencesNumber,
    dbo.TotalPayments(ClientID)
    as TotalPayments
  from Clients
```

3.2. Widoki związane z płatnościami

3.2.1. NotFullyPaidBookings

Pokazuje nieopłacone w całości ważne rezerwacje.

```
create view NotFullyPaidBookings
as
  select BookingID, ClientID, ConferenceID,
    ParticipantsNumber, RegistrationDate,
    dbo.AmountLeftToPayForBooking(BookingID)
    as LeftToPay,
```

```
7       (7 - dbo.DaysFromRegistration(BookingID, getdate()))
8       as LeftTimeToPay
9     from Bookings
10    where isValid = 1
```

3.2.2. FullyPaidBookings

Pokazuje opłacone w całości ważne rezerwacje.

```
1  create view FullyPaidBookings
2  as
3    select BookingID, ClientID, ConferenceID,
4      ParticipantsNumber, RegistrationDate,
5      dbo.AmountPaidForBooking(BookingID) as MoneyPaid
6    from Bookings
7    where dbo.AmountLeftToPayForBooking(BookingID) = 0
8    and isValid = 1
```

3.3. Widoki związane z zapisami

3.3.1. DayBookingsWithoutCompleteParticipants

Pokazuje dni konferencji, na które istnieją rezerwacje z niepełną listą uczestników.

```
1  create view DayBookingsWithoutCompleteParticipants
2  as
3    select DayBookings.DayID, B.BookingID, B.ClientID,
4      B.ConferenceID,
5      dbo.PlacesOnDayBookingLeft(DayBookings.DayBookingID)
6      as UntakenPlaces
7    from DayBookings
8    inner join Bookings B
9    on DayBookings.BookingID = B.BookingID
10   where dbo.PlacesOnDayBookingLeft(DayBookings.DayBookingID) > 0
11     and DayBookings.isValid = 1
```

3.3.2. WorkshopBookingsWithoutCompleteParticipants

Pokazuje warsztaty, na które istnieją rezerwacje z niepełną listą uczestników.

```
1  create view WorkshopBookingsWithoutCompleteParticipants
2  as
3    select WorkshopBookings.WorkshopID, B.BookingID, B.ClientID,
```

```
    B.ConferenceID,
    dbo.PlacesOnWorkshopBookingLeft(
      WorkshopBookings.WorkshopBookingID)
    as UntakenPlaces
  from WorkshopBookings
  inner join DayBookings D
  on WorkshopBookings.DayBookingID = D.DayBookingID
  inner join Bookings B
  on D.BookingID = B.BookingID
  where dbo.PlacesOnWorkshopBookingLeft(
    WorkshopBookings.WorkshopBookingID) > 0
    and WorkshopBookings.isValid = 11
```

4. Procedury

4.1. Procedury związane z organizacją konferencji i warsztatów

4.1.1. AddOrganizer

Dodaje nowego organizatora do tabeli Organizers.

```
create procedure AddOrganizer
  @orgName nvarchar(50),
  @contactName nvarchar(50),
  @address nvarchar(128),
  @postalCode nvarchar(10),
  @city nvarchar(25),
  @region nvarchar(25),
  @country nvarchar(25)
as
  begin try
    insert into Organizers
    (OrganizerName, ContactName, Address, PostalCode, City,
      Region, Country)
    values
    (@orgName, @contactName, @address, @postalCode, @city,
      @region, @country)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding Organizer. Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.1.2. AddConference

Dodaje nową konferencję do tabeli Conferences.

```
create procedure AddConference
  @organizerID int,
  @conferenceName nvarchar(100),
  @firstDay date,
```

```
  @lastDay date,
  @placeAddress nvarchar(128),
  @placeCity nvarchar(25),
  @placeRegion nvarchar(25),
  @placeCountry nvarchar(25)
as
  begin try
    if not exists(select * from Organizers
      where OrganizerID = @organizerID)
      throw 50001, 'Organizer does not exist!', 1
    insert into Conferences
    (OrganizerID, ConferenceName, FirstDay, LastDay,
      PlaceAddress, PlaceCity, PlaceRegion, PlaceCountry)
    values
    (@organizerID, @conferenceName, @firstDay, @lastDay,
      @placeAddress, @placeCity, @placeRegion, @placeCountry)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding Conference. Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.1.3. AddConferenceDay

Dodaje nowy dzień konferencji do tabeli ConferenceDays.

```
create procedure AddConferenceDay
  @confID int,
  @date date,
  @numOfPlaces int
as
  begin try
    if not exists(select * from Conferences
      where ConferenceID = @confID)
      throw 50001, 'Conference does not exist!', 1
    if exists(select * from ConferenceDays
      where ConferenceID = @confID and Date = @date)
      throw 50001, 'This day is already declared!', 1
    insert into ConferenceDays
    (ConferenceID, Date, NumberOfPlaces, PlacesLeft)
    values
```

```
    (@confID, @date, @numOfPlaces, @numOfPlaces)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding ConferenceDay. Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.1.4. AddWorkshop

Dodaje nowy warsztat do tabeli Workshops.

```
create procedure AddWorkshop
  @dayID int,
  @workshopName nvarchar(50),
  @workshopStart time,
  @workshopEnd time,
  @numOfPlaces int,
  @price money
as
  begin try
    if not exists(select * from ConferenceDays
      where DayID = @dayID)
      throw 50001, 'Day of Conference does not exist!', 1
    insert into Workshops
    (DayID, WorkshopName, WorkshopStart, WorkshopEnd,
      NumberOfPlaces, PlacesLeft, Price)
    values
    (@dayID, @workshopName, @workshopStart, @workshopEnd,
      @numOfPlaces, @numOfPlaces, @price)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding Workshop. Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.1.5. AddPriceThreshold

Dodaje nowy próg cenowy do tabeli PriceThresholds, sprawdzając czy nie wprowadza on kolizji z istniejącymi.

```
create procedure AddPriceThreshold
```

```
  @dayID int,
  @price money,
  @studentDiscount real,
  @startDate date,
  @endDate date
as
  begin try
    if not exists(select * from ConferenceDays
      where DayID = @dayID)
      throw 50001, 'Day of Conference does not exist!', 1
    if exists(select * from PriceThresholds
      where (StartDate between @startDate and @endDate)
      or (EndDate between @startDate and @endDate))
      throw 50001, 'Collision with existing threshold!', 1
    insert into PriceThresholds
    (DayID, Price, StudentDiscount, StartDate, EndDate)
    values
    (@dayID, @price, @studentDiscount, @startDate, @endDate)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding PriceThreshold. Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.1.6. ChangeConferenceDetails

Umożliwia edycję danych dotyczących konferencji

```
create procedure ChangeConferenceDetails
  @confID int,
  @newName nvarchar(100),
  @newStart date,
  @newEnd date,
  @newAddress nvarchar(128),
  @newCity nvarchar(25),
  @newRegion nvarchar(25),
  @newCountry nvarchar(25)
as
  begin try
    if not exists(select * from Conferences
      where ConferenceID = @confID)
```

```
14        throw 50001, 'Conference does not exist!', 1
15      if @newName is not null
16      begin
17        update Conferences
18        set ConferenceName = @newName
19        where ConferenceID = @confID
20      end
21      if @newStart is not null
22      begin
23        update Conferences
24        set FirstDay = @newStart
25        where ConferenceID = @confID
26      end
27      if @newEnd is not null
28      begin
29        update Conferences
30        set LastDay = @newEnd
31        where ConferenceID = @confID
32      end
33      if @newAddress is not null
34      begin
35        update Conferences
36        set PlaceAddress = @newAddress
37        where ConferenceID = @confID
38      end
39      if @newCity is not null
40      begin
41        update Conferences
42        set PlaceCity = @newCity
43        where ConferenceID = @confID
44      end
45      if @newRegion is not null
46      begin
47        update Conferences
48        set PlaceRegion = @newRegion
49        where ConferenceID = @confID
50      end
51      if @newCountry is not null
52      begin
53        update Conferences
54        set PlaceCountry = @newCountry
```

```
55        where ConferenceID = @confID
56      end
57    end try
58    begin catch
59      declare @errorMessage nvarchar(2048)
60      = 'Error with editing Conference details.
61        Message: ' + error_message();
62      throw 50001, @errorMessage, 1
63    end catch
```

4.1.7. ChangeDayPlaces

Zmienia liczbę miejsc dla danego dnia konferencji, pilnując czy po zmianie miejsc będzie co najmniej tyle, ile zostało zarezerwowanych.

```
1  create procedure ChangeDayPlaces
2    @dayID int,
3    @newNumberOfPlaces int
4  as
5    begin try
6      if not exists(select * from ConferenceDays
7        where DayID = @dayID)
8        throw 50001, 'Conference Day does not exist!', 1
9
10     declare @diff int = (select NumberOfPlaces
11       from ConferenceDays
12        where DayID = @dayID) - @newNumberOfPlaces
13     declare @placesLeft int = (select PlacesLeft
14       from ConferenceDays
15       where DayID = @dayID)
16     if @diff > @placesLeft
17       throw 50001, 'There are too many participants
18         of that day to set this number of places!', 1
19
20     update ConferenceDays
21     set NumberOfPlaces = @newNumberOfPlaces
22     where DayID = @dayID
23   end try
24   begin catch
25     declare @errorMessage nvarchar(2048)
26     = 'Error with editing Conference Day places.
27       Message: ' + error_message();
```

```
    throw 50001, @errorMessage, 1
  end catch
```

4.1.8. ChangeWorkshopPlaces

Zmienia liczbę miejsc dla danego warsztatu, pilnując czy po zmianie miejsc będzie co najmniej tyle, ile zostało zarezerwowanych.

```
create procedure ChangeWorkshopPlaces
  @workshopID int,
  @newNumberOfPlaces int
as
  begin try
    if not exists(select * from Workshops
      where WorkshopID = @workshopID)
      throw 50001, 'Workshop does not exist!', 1

    declare @diff int = (select NumberOfPlaces
      from Workshops
      where WorkshopID = @workshopID) - @newNumberOfPlaces
    declare @placesLeft int = (select PlacesLeft
      from Workshops
      where WorkshopID = @workshopID)
    if @diff > @placesLeft
      throw 50001, 'There are too many participants
        of that workshop to set this number of places!', 1

    update Workshops
    set NumberOfPlaces = @newNumberOfPlaces
    where WorkshopID = @workshopID
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with editing Workshop places.
      Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.1.9. ChangeWorkshopPrice

Zmienia cenę dla danego warsztatu.

```
create procedure ChangeWorkshopPrice
```

```
  @workshopID int,
  @newPrice money
as
  begin try
    if not exists(select * from Workshops
      where WorkshopID = @workshopID)
      throw 50001, 'Workshop does not exist!', 1

    update Workshops
    set Price = @newPrice
    where WorkshopID = @workshopID
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with editing Workshop price.
      Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.1.10. ChangePriceThreshold

Zmienia dany próg cenowy, sprawdzając czy zmiana nie wprowadzi kolizji z pozostałymi.

```
create procedure ChangePriceThreshold
  @thresholdID int,
  @newPrice money,
  @newStart date,
  @newEnd date,
  @newStudentDiscount real
as
  begin try
    if not exists(select * from PriceThresholds
      where ThresholdID = @thresholdID)
      throw 50001, 'Price Threshold does not exist!', 1

    if @newPrice is not null
    begin
      update PriceThresholds
      set Price = @newPrice
      where ThresholdID = @thresholdID
    end

```

```
20      if @newStudentDiscount is not null
21      begin
22        update PriceThresholds
23        set StudentDiscount = @newStudentDiscount
24        where ThresholdID = @thresholdID
25      end
26
27      declare @oldEnd date = (select EndDate
28        from PriceThresholds
29        where ThresholdID = @thresholdID)
30      declare @oldStart date = (select StartDate
31        from PriceThresholds
32        where ThresholdID = @thresholdID)
33
34      if @newStart is not null
35      begin
36        if exists(select * from PriceThresholds
37          where (StartDate between @newStart and @oldEnd)
38          or (EndDate between @newStart and @oldEnd))
39          throw 50001, 'Collision with existing threshold!', 1
40        update PriceThresholds
41        set StartDate = @newStart
42        where ThresholdID = @thresholdID
43
44        set @oldStart = @newStart
45      end
46
47      if @newEnd is not null
48      begin
49        if exists(select * from PriceThresholds
50          where (StartDate between @oldStart and @newEnd)
51          or (EndDate between @oldStart and @newEnd))
52          throw 50001, 'Collision with existing threshold!', 1
53        update PriceThresholds
54        set EndDate = @newEnd
55        where ThresholdID = @thresholdID
56      end
57    end try
58    begin catch
59      declare @errorMessage nvarchar(2048)
60      = 'Error with editing Price Threshold.
```

```
61        Message: ' + error_message();
62      throw 50001, @errorMessage, 1
63    end catch
```

4.2. Procedury związane z obsługą rezerwacji

4.2.1. AddClient

Dodaje nowego klienta do tabeli Clients.

```
1  create procedure AddClient
2    @clientName nvarchar(50),
3    @contactName nvarchar(50),
4    @contactEmail nvarchar(50),
5    @address nvarchar(128),
6    @city nvarchar(25),
7    @postalcode nvarchar(10),
8    @region nvarchar(25),
9    @country nvarchar(25),
10   @phone varchar(20),
11   @fax varchar(20)
12 as
13   begin try
14     insert into Clients
15     (ClientName, ContactName, ContactEMail, Address,
16       City, PostalCode, Region, Country, Phone, Fax)
17     values
18       (@clientName, @contactName, @contactEmail, @address,
19     @city, @postalcode, @region, @country, @phone, @fax)
20   end try
21   begin catch
22     declare @errorMessage nvarchar(2048)
23     = 'Error with adding Client. Message: ' + error_message();
24     throw 50001, @errorMessage, 1
25   end catch
```

4.2.2. AddBooking

Dodaje nową rezerwację do tabeli Bookings.

```
1  create procedure AddBooking
2    @clientID int,
```

```
  @conferenceID int,
  @participantsNumber int
as
  begin try
    if not exists(select * from Clients
      where ClientID = @clientID)
      throw 50001, 'Client does not exist!', 1
    if not exists(select * from Conferences
      where ConferenceID = @conferenceID)
      throw 50001, 'Conference does not exist!', 1

    insert into Bookings
    (ClientID, ConferenceID, ParticipantsNumber,
      RegistrationDate, isValid)
    values
    (@clientID, @conferenceID, @participantsNumber,
      getdate(), 1)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding Booking. Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.2.3. AddDayBooking

Dodaje nową rezerwację dla danego dnia do tabeli DayBookings, sprawdzając czy nie zarezerwowano zbyt dużej liczby miejsc i aktualizując następnie liczbę wolnych miejsc na dany dzień.

```
create procedure AddDayBooking
  @dayID int,
  @bookingID int,
  @numberOfParticipants int,
  @numberOfStudents int
as
  begin try
    if not exists(select * from ConferenceDays
      where DayID = @dayID)
      throw 50001, 'Conference Day does not exist!', 1
    if not exists(select * from Bookings
      where BookingID = @bookingID)
      throw 50001, 'Booking does not exist!', 1
```

```
14
15      declare @placesLeft int = (select PlacesLeft
16        from ConferenceDays
17        where DayID = @dayID)
18      if @numberOfParticipants > @placesLeft
19        throw 50001, 'Number of participants bigger
20          than free places!', 1
21      if @numberOfParticipants < @numberOfStudents
22        throw 50001, 'Incorrect number of students!', 1
23
24      declare @bookedPlaces int = (select ParticipantsNumber
25        from Bookings
26        where BookingID = @bookingID)
27      if @bookedPlaces < @numberOfParticipants
28        throw 50001, 'Declared number of participants
29          is bigger than number of booked places!', 1
30
31      insert into DayBookings
32      (DayID, BookingID, NumberOfParticipants,
33        NumberOfStudents, isValid)
34      values
35      (@dayID, @bookingID, @numberOfParticipants,
36        @numberOfStudents, 1)
37
38      update ConferenceDays
39      set PlacesLeft = @placesLeft - @numberOfParticipants
40      where DayID = @dayID
41    end try
42    begin catch
43      declare @errorMessage nvarchar(2048)
44      = 'Error with adding Day Booking. Message: ' + error_message();
45      throw 50001, @errorMessage, 1
46    end catch
```

4.2.4. AddWorkshopBooking

Dodaje nową rezerwację na warsztat do tabeli WorkshopBookings, sprawdzajac czy zgadzają się dzień konferencji z dniem warsztatu, zadeklarowana liczba uczestników nie przekracza zerezerwowanej na dany dzień oraz liczby wolnych miejsc, a następnie aktualizuje liczbę wolnych miejsc.

```
1  create procedure AddWorkshopBooking
2    @dayBookingID int,
```

```
3   @workshopID int,
4   @numberOfParticipants int
5 as
6   begin try
7     if not exists(select * from DayBookings
8       where DayBookingID = @dayBookingID)
9       throw 50001, 'Day Booking does not exist!', 1
10    if not exists(select * from Workshops
11      where WorkshopID = @workshopID)
12      throw 50001, 'Workshop does not exist!', 1
13
14    declare @bookedPlacesForDay int = (select NumberOfParticipants
15      from DayBookings
16      where DayBookingID = @dayBookingID)
17    if @numberOfParticipants > @bookedPlacesForDay
18      throw 50001, 'Number of participants bigger than
19        number of booked places for that dat!', 1
20
21    declare @bookedDayID int = (select DayID
22      from DayBookings
23      where DayBookingID = @dayBookingID)
24    declare @workshopDayID int = (select WorkshopID
25      from Workshops
26      where WorkshopID = @workshopID)
27    if @bookedDayID <> @workshopDayID
28      throw 50001, 'Day connected with Day Booking
29        and day of Workshop do not match!', 1
30
31    declare @placesLeft int = (select PlacesLeft
32      from Workshops
33      where WorkshopID = @workshopID)
34    if @numberOfParticipants > @placesLeft
35      throw 50001, 'Number of participants bigger
36        than free places!', 1
37
38    insert into WorkshopBookings
39    (DayBookingID, WorkshopID, NumberOfParticipants, isValid)
40    values
41    (@dayBookingID, @workshopID, @numberOfParticipants, 1)
42
43    update Workshops
```

```
    set PlacesLeft = @placesLeft - @numberOfParticipants
    where WorkshopID = @workshopID
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding Workshop Booking.
      Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.2.5. AddPayment

Dodaje nową płatność do tabeli Payments.

```
create procedure AddPayment
  @bookingID int,
  @amountPaid money
as
  begin try
    if not exists(select * from Bookings
      where BookingID = @bookingID)
      throw 50001, 'Booking does not exist!', 1

    if dbo.AmountLeftToPayForBooking(@bookingID) <= 0
      throw 50001, 'Booking is already fully paid', 1

    insert into Payments
    (BookingID, PaymentDate, AmountPaid)
    values
    (@bookingID, getdate(), @amountPaid)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding Payment.
      Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.2.6. CancelUnpaidBookings

Anuluje nieopłacone w terminie rezerwacje (podanym jako argument funkcji)

```
create procedure CancelUnpaidBookings
```

```
  @dayThreshold int
as
  begin
    update Bookings
    set isValid = 0
    where dbo.DaysFromRegistration(BookingID, getdate())
      > @dayThreshold
    and dbo.AmountLeftToPayForBooking(BookingID) <> 0
  end
```

4.2.7. ChangeNumberOfBookingParticipants

Zmienia liczbę osób dla rezerwacji.

```
create procedure ChangeNumberOfBookingParticipants
  @bookingID int,
  @newNumberOfParticipants int
as
  begin try
    if not exists (select * from Bookings
      where BookingID = @bookingID)
      throw 50001, 'Booking does not exist!', 1

    update Bookings
    set ParticipantsNumber = @newNumberOfParticipants
    where BookingID = @bookingID
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with editing Booking number of participants.
      Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.2.8. ChangeNumberOfDayParticipants

Zmienia całkowitę liczbę osób lub liczbę studentów dla rezerwacji, sprawdzając czy nowe wartości nie przekraczają liczby wolnych miejsc lub liczby zarezerwowanych miejsc na konferencję oraz zmieniając liczby zarezerwowanych miejsc na warsztaty powiązane z rezerwacją dla danego dnia, jeśli liczba wcześniejszych miejsc jest większa niż nowa podana.

```
create procedure ChangeNumberOfDayParticipants
  @dayBookingID int,
```

```
3   @newNumberOfParticipants int,
4   @newNumberOfStudents int
5 as
6   begin try
7     if not exists (select * from DayBookings
8       where DayBookingID = @dayBookingID)
9       throw 50001, 'This Day Booking does not exist!', 1
10
11    declare @bookingID int = (select BookingID
12      from DayBookings
13      where DayBookingID = @dayBookingID)
14    declare @bookedPlaces int = (select ParticipantsNumber
15      from Bookings
16      where BookingID = @bookingID)
17    if @newNumberOfParticipants > @bookedPlaces
18      throw 50001, 'Given number of participants bigger
19        than value in booking', 1
20
21    declare @dayid int = (select DayID from DayBookings
22      where DayBookingID = @dayBookingID)
23    declare @placesLeft int = (select PlacesLeft
24      from ConferenceDays
25      where DayID = @dayid)
26    declare @currentPlaces int = (select NumberOfParticipants
27      from DayBookings
28      where DayBookingID = @dayBookingID)
29
30    if @newNumberOfParticipants is not null
31    begin
32      if (@newNumberOfParticipants - @currentPlaces) > @placesLeft
33        throw 50001, 'Too few free places left to set this number!', 1
34
35      update DayBookings
36      set NumberOfParticipants = @newNumberOfParticipants
37      where DayBookingID = @dayBookingID
38
39      update ConferenceDays
40      set PlacesLeft = @placesLeft -
41        (@newNumberOfParticipants - @currentPlaces)
42      where DayID = @dayid
43
```

```
44        update WorkshopBookings
45        set NumberOfParticipants = @newNumberOfParticipants
46        where DayBookingID = @dayBookingID
47        and NumberOfParticipants > @newNumberOfParticipants
48
49        set @currentPlaces = @newNumberOfParticipants
50      end
51      if @newNumberOfStudents is not null
52      begin
53        if (@newNumberOfStudents > @currentPlaces)
54          throw 50001, 'New number of students bigger
55            than declared number of participants!', 1
56
57        update DayBookings
58        set NumberOfStudents = @newNumberOfStudents
59        where DayBookingID = @dayBookingID
60      end
61
62    end try
63    begin catch
64      declare @errorMessage nvarchar(2048)
65      = 'Error with editing Booking number
66        of participants for this day. Message: ' + error_message();
67      throw 50001, @errorMessage, 1
68    end catch
```

4.2.9. ChangeNumberOfWorkshopParticipants

Zmienia liczbę osób dla rezerwacji warsztatu, sprawdzając poprawność nowych danych – czy nie przekraczają liczby wolnych miejsc oraz liczby miejsc zarezerwowanych.

```
1  create procedure ChangeNumberOfWorkshopParticipants
2    @workshopBookingID int,
3    @newNumberOfParticipants int
4  as
5    begin try
6      if not exists(select * from WorkshopBookings
7        where WorkshopBookingID = @workshopBookingID)
8        throw 50001, 'Booking for workshop does not exist!', 1
9
10     declare @dayBookingID int = (select DayBookingID
11       from WorkshopBookings
```

```
12        where WorkshopBookingID = @workshopBookingID)
13      declare @placesBookedForDay int = (select NumberOfParticipants
14        from DayBookings
15        where DayBookingID = @dayBookingID)
16      if @newNumberOfParticipants > @placesBookedForDay
17        throw 50001, 'Given number of participants bigger
18          than value booked for this day!', 1
19
20      declare @workshopID int = (select WorkshopID
21        from WorkshopBookings
22        where WorkshopBookingID = @workshopBookingID)
23      declare @placesLeft int = (select PlacesLeft
24        from Workshops
25        where WorkshopID = @workshopID)
26      declare @currentPlaces int = (select NumberOfParticipants
27        from WorkshopBookings
28        where WorkshopBookingID = @workshopBookingID)
29      if (@newNumberOfParticipants - @currentPlaces) > @placesLeft
30        throw 50001, 'Too few free places to set this number!', 1
31
32      update WorkshopBookings
33      set NumberOfParticipants = @newNumberOfParticipants
34      where WorkshopBookingID = @workshopBookingID
35
36      update Workshops
37      set PlacesLeft = @placesLeft -
38        (@newNumberOfParticipants - @currentPlaces)
39      where WorkshopID = @workshopID
40    end try
41    begin catch
42      declare @errorMessage nvarchar(2048)
43      = 'Error with editing Booking number
44        of participants for this workshop. Message: ' + error_message();
45      throw 50001, @errorMessage, 1
46    end catch
```

4.2.10. CancelBooking

Anuluje całą rezerwację.

```
1  create procedure CancelBooking
2    @bookingID int
```

```
as
  begin try
    if not exists(select * from Bookings
      where BookingID = @bookingID)
      throw 50001, 'Booking does not exist!', 1

    if (select isValid from Bookings
      where BookingID = @bookingID) = 0
      throw 50001, 'Booking is already cancelled!', 1

    if dbo.AmountPaidForBooking(@bookingID) <> 0
      throw 50001, 'Booking has got non-zero payments!', 1

    update Bookings
    set isValid = 0
    where BookingID = @bookingID
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with cancelling Booking.
      Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.2.11. CancelDayBooking

Anuluje rezerwację dla dnia, aktualizując liczbę wolnych miejsc.

```
create procedure CancelDayBooking
  @dayBookingID int
as
  begin try
    if not exists(select * from DayBookings
      where DayBookingID = @dayBookingID)
      throw 50001, 'Booking for day does not exist!', 1

    if (select isValid from DayBookings
      where DayBookingID = @dayBookingID) = 0
      throw 50001, 'Booking for day is already cancelled!', 1

    declare @bookingID int = (select BookingID
      from DayBookings
```

```
15        where DayBookingID = @dayBookingID)
16      if dbo.AmountPaidForBooking(@bookingID) <> 0
17        throw 50001, 'Booking has got non-zero payments!', 1
18
19      update DayBookings
20      set isValid = 0
21      where DayBookingID = @dayBookingID
22
23      declare @dayID int = (select DayID
24        from DayBookings
25        where DayBookingID = @dayBookingID)
26      declare @numberOfPlacesMadeFree int =
27        (select NumberOfParticipants
28        from DayBookings
29        where DayBookingID = @dayBookingID)
30      declare @placesLeft int = (select PlacesLeft
31        from ConferenceDays
32        where DayID = @dayID)
33      update ConferenceDays
34      set PlacesLeft = @placesLeft + @numberOfPlacesMadeFree
35      where DayID = @dayID
36    end try
37    begin catch
38      declare @errorMessage nvarchar(2048)
39      = 'Error with cancelling Day Booking.
40        Message: ' + error_message();
41      throw 50001, @errorMessage, 1
42    end catch
```

4.2.12. CancelWorkshopBooking

Anuluje rezerwację na warsztat, aktualizując liczbę wolnych miejsc.

```
1  create procedure CancelWorkshopBooking
2    @workshopBookingID int
3  as
4    begin try
5      if not exists(select * from WorkshopBookings
6        where WorkshopBookingID = @workshopBookingID)
7        throw 50001, 'Booking for workshop does not exist!', 1
8
9      if (select isValid from WorkshopBookings
```

```
10        where WorkshopBookingID = @workshopBookingID) = 0
11        throw 50001, 'Booking for workshop is already cancelled!', 1
12
13      declare @bookingID int = (select DayBookings.BookingID
14              from DayBookings
15                inner join WorkshopBookings WB
16                  on DayBookings.DayBookingID = WB.DayBookingID
17              where WB.WorkshopBookingID = @workshopBookingID)
18      if dbo.AmountPaidForBooking(@bookingID) <> 0
19        throw 50001, 'Booking has got non-zero payments!', 1
20
21      update WorkshopBookings
22      set isValid = 0
23      where WorkshopBookingID = @workshopBookingID
24
25      declare @workshopID int = (select WorkshopID
26        from WorkshopBookings
27        where WorkshopBookingID = @workshopBookingID)
28      declare @numberOfPlacesMadeFree int =
29        (select NumberOfParticipants
30        from WorkshopBookings
31        where WorkshopBookingID = @workshopBookingID)
32      declare @placesLeft int = (select PlacesLeft
33        from Workshops
34        where WorkshopID = @workshopID)
35
36      update Workshops
37      set PlacesLeft = @placesLeft + @numberOfPlacesMadeFree
38      where WorkshopID = @workshopID
39    end try
40    begin catch
41      declare @errorMessage nvarchar(2048)
42      = 'Error with cancelling Workshop Booking.
43        Message: ' + error_message();
44      throw 50001, @errorMessage, 1
45    end catch
```

4.3. Procedury związane z zapisami uczestników na zarezerwowane miejsca

4.3.1. AddParticipant

Dodaje dane uczestnika do tabeli Participants.

```
create procedure AddParticipant
  @firstName nvarchar(50),
  @lastName nvarchar(50),
  @title nvarchar(15)
as
  begin try
    insert into Participants
    (FirstName, LastName, Title)
    values
    (@firstName, @lastName, @title)
  end try
  begin catch
    declare @errorMessage nvarchar(2048)
    = 'Error with adding Participant.
      Message: ' + error_message();
    throw 50001, @errorMessage, 1
  end catch
```

4.3.2. AddParticipantToDayBooking

Zapisuje uczestnika na dany dzień konferencji, sprawdzając czy są jeszcze niezajęte miejsca.

```
create procedure AddParticipantToDayBooking
  @participantID int,
  @dayBookingID int,
  @studentCardNumber varchar(15)
as
  begin try
    if not exists(select * from DayBookings
      where DayBookingID = @dayBookingID)
      throw 50001, 'Booking for day does not exist!', 1

    if not exists(select * from Participants
      where ParticipantID = @participantID)
      throw 50001, 'Participant does not exist!', 1

    if dbo.PlacesOnDayBookingLeft(@dayBookingID) = 0
```

```
16        throw 50001, 'All places from booking are occupied!', 1
17
18      insert into ParticipantsOfDay
19      (ParticipantID, DayBookingID, StudentCardNumber)
20      values
21      (@participantID, @dayBookingID, @studentCardNumber)
22    end try
23    begin catch
24      declare @errorMessage nvarchar(2048)
25      = 'Error with adding Participant to a Day.
26        Message: ' + error_message();
27      throw 50001, @errorMessage, 1
28    end catch
```

4.3.3. AddParticipantToWorkshopBooking

Zapisuje uczestnika na dany warsztat, sprawdzając czy zostały jeszcze niewypełnione miejsca.

```
1  create procedure AddParticipantToWorkshopBooking
2    @workshopBookingID int,
3    @participantOfDayID int
4  as
5    begin try
6      if not exists(select * from ParticipantsOfDay
7        where ParticipantOfDayID = @participantOfDayID)
8        throw 50001, 'This day booking does not exist!', 1
9
10     if not exists(select * from WorkshopBookings
11       where WorkshopBookingID = @workshopBookingID)
12       throw 50001, 'This booking for workshop does not exist', 1
13
14     if dbo.PlacesOnWorkshopBookingLeft(@workshopBookingID) = 0
15       throw 50001, 'All places from workshop booking are occupied!', 1
16
17     insert into ParticipantsOfWorkshop
18     (WorkshopBookingID, ParticipantOfDayID)
19     values
20     (@workshopBookingID, @participantOfDayID)
21   end try
22   begin catch
23     declare @errorMessage nvarchar(2048)
24     = 'Error with adding Participant to a Workshop.
```

```
25        Message: ' + error_message();
26      throw 50001, @errorMessage, 1
27    end catch
```

4.3.4. DeleteParticipantFromDayBooking

Wypisuje uczestnika z danego dnia konferencji.

```
1  create procedure DeleteParticipantFromDayBooking
2    @participantOfDayID int
3  as
4    begin try
5      if not exists(select * from ParticipantsOfDay
6        where ParticipantOfDayID = @participantOfDayID)
7        throw 50001, 'This participant to day assignment
8          does not exist!', 1
9
10     delete from ParticipantsOfDay
11     where ParticipantOfDayID = @participantOfDayID
12   end try
13   begin catch
14     declare @errorMessage nvarchar(2048)
15     = 'Error with deleting Participant from Day.
16       Message: ' + error_message();
17     throw 50001, @errorMessage, 1
18   end catch
```

4.3.5. DeleteParticipantFromWorkshop

Wypisuje uczestnika z danego warsztatu.

```
1  create procedure DeleteParticipantFromWorkshop
2    @participantOfDayID int,
3    @workshopBookingID int
4  as
5    begin try
6      if not exists(select * from ParticipantsOfWorkshop
7      where ParticipantOfDayID = @participantOfDayID
8      and WorkshopBookingID = @workshopBookingID)
9        throw 50001, 'This participant to workshop
10         assignment does not exist!', 1
11
12     delete from ParticipantsOfWorkshop
```

```
13      where ParticipantOfDayID = @participantOfDayID
14      and WorkshopBookingID = @workshopBookingID
15    end try
16    begin catch
17      declare @errorMessage nvarchar(2048)
18      = 'Error with deleting Participant from Workshop.
19        Message: ' + error_message();
20      throw 50001, @errorMessage, 1
21    end catch
```

5. Funkcje

5.1. Funkcje dotyczące konferencji i warsztatów

5.1.1. ConferenceDaysInfo

Zwraca tabelę zawierającą informacje o poszczególnych dniach konferencji.

```
create function ConferenceDaysInfo
(
@confID int
)
returns table
as
  return (select Date, NumberOfPlaces, PlacesLeft
          from ConferenceDays
          where ConferenceID = @confID)
```

5.1.2. WorkshopsInfo

Zwraca tabelę z informacjami o warsztatach odbywających się w danym dniu konferencji.

```
create function WorkshopsInfo
(
@confDayID int
)
returns table
as
  return (select WorkshopName, WorkshopStart, WorkshopEnd,
                 NumberOfPlaces, PlacesLeft, Price
          from Workshops
          where DayID = @confDayID)
```

5.1.3. DayParticipantsList

Zwraca listę uczestników danego dnia konferencji.

```
create function DayParticipantsList
(
@confDayID int
)
returns table
as
  return (select Participants.Title, Participants.FirstName,
                 Participants.LastName
  from ConferenceDays
          inner join DayBookings DB
            on ConferenceDays.DayID = DB.DayID
          inner join ParticipantsOfDay POD
            on DB.DayBookingID = POD.DayBookingID
          inner join Participants
            on POD.ParticipantID = Participants.ParticipantID
  where DB.DayID = @confDayID and DB.isValid = 1)
```

5.1.4. WorkshopParticipantsList

Zwraca listę uczestników danego warsztatu.

```
create function WorkshopParticipantsList
(
@workshopID int
)
returns table
as
  return (select P.Title, P.FirstName, P.LastName
  from Workshops
          inner join WorkshopBookings WB
            on Workshops.WorkshopID = WB.WorkshopID
          inner join ParticipantsOfWorkshop POW
            on WB.WorkshopBookingID = POW.WorkshopBookingID
          inner join ParticipantsOfDay POD
            on POW.ParticipantOfDayID = POD.ParticipantOfDayID
          inner join Participants P
            on POD.ParticipantID = P.ParticipantID
  where Workshops.WorkshopID = @workshopID and WB.isValid = 1)
```

5.1.5. PlacesOnDayBookingLeft

Oblicza ilość miejsc, jakie zostały do wypełnienia uczesnikami w rezerwacji dla danego dnia.

```
create function PlacesOnDayBookingLeft
(
@dayBookingID int
)
returns int
as
  begin
    declare @BookedPlaces int =
      (select NumberOfParticipants
      from DayBookings
      where DayBookingID = @dayBookingID)
    declare @TakenPlaces int =
      (select count(ParticipantOfDayID)
      from ParticipantsOfDay
      where DayBookingID = @dayBookingID)
    return @BookedPlaces - @TakenPlaces
  end
```

5.1.6. PlacesOnWorkshopBookingLeft

Oblicza ilość miejsc, jakie zostały do wypełnienia uczesnikami w rezerwacji dla danego warsztatu.

```
create function PlacesOnWorkshopBookingLeft
(
@workshopBookingID int
)
returns int
as
  begin
    declare @BookedPlaces int =
      (select NumberOfParticipants
      from WorkshopBookings
      where WorkshopBookingID = @workshopBookingID)
    declare @TakenPlaces int =
      (select count(ParticipantOfDayID)
      from ParticipantsOfWorkshop
      where WorkshopBookingID = @workshopBookingID)
    return @BookedPlaces - @TakenPlaces
  end
```

5.1.7. RegisteredStudents

Oblicza ilość studentów przypisanych do danej rezerwacji dnia.

```
create function RegisteredStudents
(
@dayBookingID int
)
returns int
as
  begin
    return (select count(ParticipantOfDayID) from ParticipantsOfDay
      where DayBookingID = @dayBookingID
      and StudentCardNumber is not null)
  end
```

5.2. Funkcje związane z opłatami

5.2.1. PriceOfDayOn

Zwraca koszt danego dnia konferencji dla podanej daty.

```
create function PriceOfDayOn
(
@day date,
@dayID int
)
returns money
as
  begin
    return (select PriceThresholds.Price
    from PriceThresholds
    where PriceThresholds.DayID = @dayID and
      @day between StartDate and EndDate)
  end
```

5.2.2. StudentDiscountOfDay

Zwraca wysokość zniżki studenckiej obowiązującej na dany dzień konferencji dla podanej daty.

```
create function StudentDiscountOfDay
(
@day date,
@dayID int
```

```
)
returns real
as
  begin
    return (select PriceThresholds.StudentDiscount
    from PriceThresholds
    where PriceThresholds.DayID = @dayID and
      @day between StartDate and EndDate)
  end
```

5.2.3. PricesOfConferenceDays

Zwraca tabelę z podanymi cenami poszczególnych dni konferencji przy rezerwacji dla podanej daty.

```
create function PricesOfConferenceDays
(
@confID int,
@dateOfCheck date
)
returns table
as
  return (select ConferenceDays.Date,
                 ConferenceDays.PlacesLeft,
                 dbo.PriceOfDayOn(@dateOfCheck, ConferenceDays.DayID)
                 as Price
  from ConferenceDays
  where ConferenceID = @confID)
```

5.2.4. PriceThresholdsForConferenceDays

Zwraca tabelę z podanymi wszystkimi progami cenowymi dla danej konferencji.

```
create function PriceThresholdsForConferenceDays
(
@confID int
)
returns table
as
  return (select ConferenceDays.Date, PT.StartDate,
                 PT.EndDate, PT.Price,
                 PT.StudentDiscount
  from ConferenceDays
         inner join PriceThresholds PT
```

```
12              on ConferenceDays.DayID = PT.DayID
13    where ConferenceDays.ConferenceID = @confID)
```

5.2.5. TotalPriceOfDayBooking

Oblicza całkowity koszt rezerwacji dla wszystkich dni konferencji z uwzględnieniem zniżki studenckiej (ale bez warsztatów).

```
1  create function TotalPriceOfDayBooking
2  (
3  @bookingID int,
4  @day date
5  )
6  returns money
7  as
8    begin
9      return (select isnull(sum((DayBookings.NumberOfParticipants -
10              0.01 * dbo.StudentDiscountOfDay(@day, DayBookings.DayID) *
11              DayBookings.NumberOfStudents) *
12              dbo.PriceOfDayOn(@day, DayBookings.DayID)), 0)
13     from DayBookings
14     where BookingID = @bookingID and isValid = 1)
15   end
```

5.2.6. TotalPriceOfWorkshopBooking

Oblicza całkowity koszt rezerwacji dla warsztatów.

```
1  create function TotalPriceOfWorkshopBooking
2  (
3  @bookingID int
4  )
5  returns money
6  as
7    begin
8      return (select isnull(sum(WorkshopBookings.NumberOfParticipants *
9               W.Price), 0)
10     from WorkshopBookings
11             inner join Workshops W
12               on WorkshopBookings.WorkshopID = W.WorkshopID
13             inner join DayBookings DB
14               on WorkshopBookings.DayBookingID = DB.DayBookingID
15             inner join Bookings B
```

```
16              on DB.BookingID = B.BookingID
17      where B.BookingID = @bookingID
18        and WorkshopBookings.isValid = 1)
19    end
```

5.2.7. TotalPriceOfBooking

Oblicza całkowity koszt rezerwacji.

```
1  create function TotalPriceOfBooking
2  (
3  @bookingID int
4  )
5  returns money
6  as
7    begin
8      return (select dbo.TotalPriceOfDayBooking(@bookingID,
9                    Bookings.RegistrationDate) +
10                   dbo.TotalPriceOfWorkshopBooking(@bookingID)
11     from Bookings
12     where BookingID = @bookingID)
13   end
```

5.2.8. AmountPaidForBooking

Oblicza sumę wykonanych dotychczas opłat na poczet danej rezerwacji.

```
1  create function AmountPaidForBooking
2  (
3  @bookingID int
4  )
5  returns money
6  as
7    begin
8      return (select isnull(sum(P.AmountPaid), 0)
9      from Bookings
10     left outer join Payments P
11     on Bookings.BookingID = P.BookingID
12     where Bookings.BookingID = @bookingID)
13   end
```

5.2.9. AmountLeftToPayForBooking

Oblicza kwotę pozostałą do zapłaty za rezerwację.

```
create function AmountLeftToPayForBooking
(
@bookingID int
)
returns money
as
  begin
    return dbo.TotalPriceOfBooking(@bookingID) -
           dbo.AmountPaidForBooking(@bookingID)
  end
```

5.3. Funkcje związane ze statystyką

5.3.1. NumberOfBookedConferences

Oblicza ilość ważnych rezerwacji dokonanych przez danego klienta.

```
create function NumberOfBookedConferences
(
@clientID int
)
returns int
as
  begin
    return (select count(*)
    from Bookings
    where ClientID = @clientID and isValid = 1)
  end
```

5.3.2. TotalPayments

Oblicza całkowitą kwotę wszystkich płatności wykonanych przez danego klienta.

```
create function TotalPayments
(
@clientID int
)
returns money
as
  begin
```

```
8       return (select sum(dbo.AmountPaidForBooking(BookingID))
9         from Bookings
10        where ClientID = @clientID
11        and isValid = 1)
12    end
```

5.4. Dodatkowe funkcje

5.4.1. IsThereACollisionBetweenWorkshops

Sprawdza czy dwa podane warsztaty ze sobą kolidują.

```
create function IsThereACollisionBetweenWorkshops
(
@Workshop1ID int,
@Workshop2ID int
)
returns bit
as
  begin
    declare @dayW1 int = (select DayID
      from Workshops
      where WorkshopID = @Workshop1ID)
    declare @dayW2 int = (select DayID
      from Workshops
      where WorkshopID = @Workshop2ID)

    if @dayW1 <> @dayW2
      return 0

    declare @startW1 time = (select WorkshopStart
      from Workshops
      where WorkshopID = @Workshop1ID)
    declare @endW1 time = (select WorkshopStart
      from Workshops
      where WorkshopID = @Workshop1ID)
    declare @startW2 time = (select WorkshopStart
      from Workshops
      where WorkshopID = @Workshop2ID)
    declare @endW2 time = (select WorkshopStart
      from Workshops
      where WorkshopID = @Workshop2ID)
    if @startW1 > @startW2 and @endW2 > @startW1
      return 1
    else if @startW2 > @startW1 and @endW1 > @startW2
      return 1
    return 0
  end
```

5.4.2. DaysLeftToConference

Oblicza ilość dni pozostałych do konferencji

```
create function DaysLeftToConference
(
@conferenceID int,
@currentDay date
)
returns int
as
  begin
    return datediff(day, @currentDay,
      (select FirstDay
       from Conferences
       where ConferenceID = @conferenceID))
  end
```

5.4.3. DaysFromRegistration

Oblicza ilość dni, które upłynęły od wykonania rezerwacji.

```
create function DaysFromRegistration
(
@bookingID int,
@currentDay date
)
returns int
as
  begin
    return datediff(day, @currentDay,
      (select RegistrationDate
      from Bookings
      where BookingID = @bookingID))
  end
```

6. Triggery

6.1. Triggery związane z obsługą konferencji

6.1.1. CheckWorkshopPlaceLimitsAfterChangingLimitForDay

Sprawdza, czy po modyfikacji liczby dostępnych miejsc na dzień nie istnieje warsztat na który jest wyższy limi miejsc.

```
create trigger CheckWorkshopPlaceLimitsAfterChangingLimitForDay
  on ConferenceDays
  after update
as
begin
  if exists(select * from Workshops
  inner join ConferenceDays CD on Workshops.DayID = CD.DayID
  where Workshops.NumberOfPlaces > CD.NumberOfPlaces)
    throw 50001, 'There exists at least one workshop
      with bigger number of places than changed value!', 1
end
```

6.1.2. CheckDayPlaceLimitsFitWorkshopLimits

Sprawdza, czy po modyfikacji liczby dostępnych miejsc na warsztat nie została przekroczona maksymalna dopuszczalna liczba miejsc dla danego dnia.

```
create trigger CheckDayPlaceLimitsFitWorkshopLimits
  on Workshops
  after insert, update
as
begin
  if exists(select * from Workshops
  inner join ConferenceDays CD on Workshops.DayID = CD.DayID
  where Workshops.NumberOfPlaces > CD.NumberOfPlaces)
    throw 50001, 'The set number of places is bigger
      than limit for conference day!', 1
end
```

6.1.3. CheckIfEndOfThresholdIsNotLaterThanConfDay

Sprawdza, czy dodany próg cenowy nie kończy się później niż konferencja.

```
create trigger CheckIfEndOfThresholdIsNotLaterThanConfDay
  on PriceThresholds
  after insert, update
as
  begin
    declare @day date = (select ConferenceDays.Date
    from ConferenceDays
    inner join inserted
    on ConferenceDays.DayID = inserted.DayID)
    declare @endDate date = (select inserted.EndDate
      from inserted)

    if @endDate > @day
      throw 50001, 'End of threshold is later
        than the day of conference connected with it!', 1
  end
```

6.1.4. CheckIfAddedDayIsCorrect

Sprawdza, czy dodany dzień konferencji ma poprawną datę.

```
create trigger CheckIfAddedDayIsCorrect
  on ConferenceDays
  after insert, update
as
  begin
    if exists(select * from inserted
      inner join Conferences
      on Conferences.ConferenceID = inserted.ConferenceID
      where inserted.Date not
      between Conferences.FirstDay and Conferences.LastDay)
    throw 50001, 'Inserted day is not
    in declared time period for conference!', 1
  end
```

6.2. Triggery związane z rezerwacjami

6.2.1. MoreDayParticipantsThanReservations

Sprawdza, czy po modyfikacji liczby zarezerwowanych miejsc na dzień, liczba zapisanych uczestników nie jest większa od nowej liczby rezerwowanych miejsc.

```
create trigger MoreDayParticipantsThanReservations
  on DayBookings
  after update
as
begin
  if exists(select * from DayBookings
    where dbo.PlacesOnDayBookingLeft(DayBookingID) < 0)
    throw 50001, 'New number of booked places
      for day is too small for all participants!', 1
end
```

6.2.2. MoreWorkshopParticipantsThanReservations

Sprawdza, czy po modyfikacji liczby zarezerwowanych miejsc na warsztat, liczba zapisanych uczestników nie jest większa od nowej liczby rezerwowanych miejsc.

```
create trigger MoreWorkshopParticipantsThanReservations
  on WorkshopBookings
  after update
as
begin
  if exists(select  * from WorkshopBookings
    where dbo.PlacesOnWorkshopBookingLeft(WorkshopBookingID) < 0)
    throw 50001, 'New number of booked places
      for workshop is too small for all participants!', 1
end
```

6.2.3. CheckValidDateOfConferenceForBooking

Sprawdza, czy rezerwacja nie jest wykonywana na zakończoną już konferencję.

```
create trigger CheckValidDateOfConferenceForBooking
  on Bookings
  after insert
as
  begin
    if exists(select * from Conferences
```

```
7         inner join Bookings B
8         on Conferences.ConferenceID = B.ConferenceID
9         where B.RegistrationDate > Conferences.LastDay)
10        throw 50001, 'Booking made for past Conference!', 1
11    end
```

6.2.4. CheckValidConferenceDaysToBook

Sprawdza, czy rezerwacja dla danego dnia odnosi się do poprawnej konferencji.

```
1  create trigger CheckValidConferenceDaysToBook
2    on DayBookings
3    after insert, update
4  as
5    begin
6      if exists(select * from DayBookings
7        inner join Bookings B
8        on DayBookings.BookingID = B.BookingID
9        inner join ConferenceDays CD
10       on DayBookings.DayID = CD.DayID
11       where CD.ConferenceID <> B.ConferenceID)
12     throw 50001, 'Day booking made for incorrect conference!', 1
13   end
```

6.2.5. CheckValidDateOfConferenceDayForBooking

Sprawdza, czy rezerwacja nie jest wykonywana dla przeszłego dnia konferencji.

```
1  create trigger CheckValidDateOfConferenceDayForBooking
2    on DayBookings
3    after insert
4  as
5    begin
6      if exists(select * from DayBookings
7        inner join Bookings B
8        on DayBookings.BookingID = B.BookingID
9        inner join ConferenceDays CD
10       on DayBookings.DayID = CD.DayID
11       where B.RegistrationDate > CD.Date)
12       throw 50001, 'Booking made for past day of conference!', 1
13   end
```

6.2.6. CancelDayBookingsAfterCancelledBooking

Anuluje rezerwacje na konkretne dni po anulowaniu rezerwacji.

```
create trigger CancelDayBookingsAfterCancelledBooking
  on Bookings
  after update
as
  begin
    update DayBookings
    set isValid = 0
    where BookingID in
          (select inserted.BookingID from inserted
           inner join deleted
           on deleted.BookingID = inserted.BookingID
           where deleted.isValid = 1 and inserted.isValid = 0)
  end
```

6.2.7. CancelWorkshopBookingsAfterCancelledDayBooking

Anuluje rezerwacje na warsztaty po anulowaniu rezerwacji dnia.

```
create trigger CancelWorkshopBookingsAfterCancelledDayBooking
  on DayBookings
  after update
as
  begin
    update WorkshopBookings
    set isValid = 0
    where DayBookingID in
          (select inserted.DayBookingID from inserted
           inner join deleted
           on deleted.DayBookingID = inserted.DayBookingID
           where deleted.isValid = 1 and inserted.isValid = 0)
  end
```

6.2.8. DeleteParticipantsOfDayAfterCancelledDayBooking

Usuwa z listy uczestników po anulowaniu rezerwacji dnia.

```
create trigger DeleteParticipantsOfDayAfterCancelledDayBooking
  on DayBookings
  after update
as
```

```
  begin
    delete from ParticipantsOfDay
    where DayBookingID in
          (select inserted.DayBookingID from inserted
            inner join deleted
            on deleted.DayBookingID = inserted.DayBookingID
            where deleted.isValid = 1 and inserted.isValid = 0)
  end
```

6.2.9. DeleteParticipantsOfWSAfterCancelledWorkshopBooking

Usuwa z listy uczestników warsztatu po anulowaniu rezerwacji warsztatu.

```
create trigger DeleteParticipantsOfWSAfterCancelledWorkshopBooking
  on WorkshopBookings
  after update
as
  begin
    delete from ParticipantsOfWorkshop
    where WorkshopBookingID in
          (select inserted.WorkshopBookingID from inserted
            inner join deleted
            on deleted.WorkshopBookingID = inserted.WorkshopBookingID
            where deleted.isValid = 1 and inserted.isValid = 0)
  end
```

6.2.10. DeleteWSParticipantsAfterDeletingDayParticipants

Usuwa z listy uczestników warsztatu po usunięciu zapisu uczestnika na dzień konferencji.

```
create trigger DeleteWSParticipantsAfterDeletingDayParticipants
  on ParticipantsOfDay
  after delete
as
  begin
    delete from ParticipantsOfWorkshop
    where ParticipantOfDayID in
          (select ParticipantOfDayID from deleted)
  end
```

6.2.11. CheckNumberOfStudents

Sprawdza czy dla rezerwacji z zapisanymi wszystkimi uczestnikami liczba zapisanych studentów jest zgodna z podaną.

```
create trigger CheckNumberOfStudents
  on ParticipantsOfDay
  after insert, update
as
  begin
    if exists(select * from DayBookings
      where dbo.PlacesOnDayBookingLeft(DayBookings.DayBookingID) = 0
      and NumberOfStudents <>
      dbo.RegisteredStudents(DayBookings.DayBookingID))
      throw 50001, 'Declared number of students
      does not match number of students on the list!', 1
  end
```

6.2.12. CheckWorkshopOverlap

Sprawdza przy zapisie na warsztat czy nie występuje kolizja z innym warsztatem, na który uczestnik jest zapisany.

```
create trigger CheckWorkshopOverlap
  on ParticipantsOfWorkshop
  after insert
as
  begin
    if exists(select * from inserted
      inner join ParticipantsOfWorkshop as POW
      on POW.ParticipantOfDayID = inserted.ParticipantOfDayID
      inner join WorkshopBookings WB
      on POW.WorkshopBookingID = WB.WorkshopBookingID
      inner join WorkshopBookings WB2
      on inserted.WorkshopBookingID = WB2.WorkshopBookingID
      where
      dbo.IsThereACollisionBetweenWorkshops
      (WB.WorkshopID, WB2.WorkshopID) = 1
      and WB.WorkshopID <> WB2.WorkshopID)
      throw 50001, 'There is a collision between workshops!', 1
  end
```

7. Indeksy

Poza indeksami utworzonymi automatycznie dla kluczy głównych i warunków integralnościowych Unique, zdecydowano się na założenie indeksów na klucze główne najczęściej używanych tabel, ze względu na dużą ilość złączeń wykonywanych w poszczególnych funkcjach, procedurach i triggerach.

```
1  create index Ind_POD_DayBk
2    on ParticipantsOfDay (DayBookingID)
3  create index Ind_DayBookings_ConfDays
4    on DayBookings (DayID)
5  create index Ind_DayBookings_Bookings
6    on DayBookings (BookingID)
7  create index Ind_WorkshopBookings_DB
8    on WorkshopBookings (DayBookingID)
9  create index Ind_WorkshopBookings_Workshops
10   on WorkshopBookings (WorkshopID)
11 create index Ind_Payments_Bookings
12   on Payments (BookingID)
13 create index Ind_Bookings_Conferences
14   on Bookings (ConferenceID)
15 create index Ind_Bookings_Clients
16   on Bookings (ClientID)
17 create index Ind_ConfDays_Conferences
18   on ConferenceDays (ConferenceID)
19 create index Ind_Thresholds_ConfDays
20   on PriceThresholds (DayID)
21 create index Ind_Workshops_ConfDays
22   on Workshops (DayID)
23 create index Ind_Conferences_Organizers
24   on Conferences (OrganizerID)
```

8. Role

8.1. Administrator systemu

Osoba posiadająca wszystkie możliwe uprawnienia w bazie: wgląd do wszystkich tabel, możliwość modyfikacji elementów bazy oraz dodawania nowych lub usuwania istniejących.

8.2. Organizator konferencji

Osoba uzupełniająca dane konferencji oraz monitorująca rezerwacje klientów i ich płatności.

8.2.1. Tabele

— Organizers
— Conferences
— ConferenceDays
— Workshops
— PriceThresholds

8.2.2. Widoki

— FutureConferences
— DaysWithFreePlaces
— WorkshopsWithFreePlaces
— ParticipantsOfConference
— ClientsOfConference
— CancelledBookings
— ClientStatistics
— NotFullyPaidBookings
— FullyPaidBookings
— DayBookingsWithoutCompleteParticipants
— WorkshopBookingWithoutCompleteParticipants

8.2.3. Procedury

— AddConference
— AddConferenceDay
— AddWorkshop
— AddPriceThreshold
— ChangeConferenceDetails

— ChangeDayPlaces
— ChangeWorkshopPlaces
— ChangeWorkshopPrice
— ChangePriceThreshold
— CancelUnpaidBookings

8.2.4. Funkcje

— DayParticipantsList
— WorkshopParticipantsList
— NumberOfBookedConferences
— TotalPayments
— DaysLeftToConference
— DaysFromRegistration

8.3. Klient

Osoba dokonująca rezerwacji indywidualnej lub grupowej, odpowiedzialna za wykonanie płatności oraz zadeklarowanie odpowiedniej liczby miejsc na dni konferencji oraz warsztaty, a następnie uzupełnienie danych uczestników i ich przyporządkowanie do poszczególnych dni i warsztatów.

8.3.1. Tabele

— Clients
— Bookings
— DayBookings
— WorkshopBookings
— ParticipantsOfDay
— ParticipantsOfWorkshop
— Participants

8.3.2. Widoki

— FutureConferences
— DaysWithFreePlaces
— WorkshopsWithFreePlaces
— DayBookingsWithoutCompleteParticipants
— WorkshopBookingsWithoutCompleteParticipants

8.3.3. Procedury

— AddClient
— AddBooking
— AddDayBooking

— AddWorkshopBooking
— AddPayment
— ChangeNumberOfBookingParticipants
— ChangeNumberOfDayParticipants
— ChangeNumberOfWorkshopParticipants
— CancelBooking
— CancelDayBooking
— CancelWorkshopBooking
— AddParticipant
— AddParticipantToDayBooking
— AddParticipantToWorkshopBooking
— DeleteParticipantFromDayBooking
— DeleteParticipantFromWorkshop

8.3.4. Funkcje

— ConferenceDaysInfo
— WorkshopsInfo
— PlacesOnDayBookingLeft
— PlacesOnWorkshopBookingLeft
— RegisteredStudents
— PriceOfDayOn
— StudentDiscountOfDay
— PricesOfConferenceDays
— PriceThresholdsForConferenceDays
— TotalPriceOfDayBooking
— TotalPriceOfWorkshopBooking
— TotalPriceOfBooking
— AmountPaidForBooking
— AmountLeftToPayForBooking
— IsThereACollisionBetweenWorkshops
— DaysLeftToConference
— DaysFromRegistration

9. Generator danych

9.1. Opis generatora

Generator został napisany w języku Python z wykorzystaniem losowych danych pobranych z serwisu Mockaroo.com. Pobrano zestawy adresów, nazw firm, nazw konferencji (jako nazwa przemysłu + „conference of" + nazwa sektora biznesowego), cen dla konferencji, adresów e-mailowych, pełnych imion, liczb dla liczb miejsc, numerów telefonów, numerów kart kredytowych (użytych tutaj jako numery legitymacji studenckich) oraz nazwy warsztatów (jako nazwa sektora biznesowego + „workshop").

Tworzonych jest 10 organizatorów konferencji, dla każdego z nich po 3 konferencje dla każdego z lat: 2017, 2018 i 2019. Długość trwania konferencji jest losowa i wynosi od 2 do 4 dni, tak samo losowo wybierana jest liczba dostępnych miejsc pomiędzy 75 a 250. Dla każdego dnia tworzone są 4 progi cenowe: pierwszy z datą końcową na 3 miesiące przed początkiem konferencji (i datą początkową w latach 70. XX wieku), a każdy następny na okres kolejnego miesiąca. Cena dla pierwszego progu jest losowana i podnoszona o 15% dla każdego następnego. Każdy dzień ma wybierane losowo od 2 do 5 warsztatów, z liczbą miejsc wybraną losowo między 10 a 100% miejsc dla danego dnia oraz ceną na poziomie nie większym niż 10 % najwyższego progu cenowego podniesionego o 15%.

Tworzonych jest 750 klientów, dla każdego z nich losowo pomiędzy 2 a 4 rezerwacjami. Liczba miejsc dla rezerwacji jest nie większa niż 15% liczby dostępnych miejsc na dzień z najwyższą ich liczbą. Data wykonania rezerwacji wybrana jest pomiędzy 1 a 6 miesiącami przed konferencją. Dla każdej rezerwacji tworzone jest między 0 a 2 płatnościami o losowych kwotach. Dla każdego dnia z wybranej konferencji z 50% prawdopodobieństwem wykonywana jest rezerwacja dla nie więcej niż pozostałej ilości miejsc dla danego dnia (lub zarezerwowanej liczby, jeśli ta jest mniejsza). Dla każdego z warsztatów odbywających się danego dnia wykonywana jest rezerwacja z prawdopodobieństwem wynoszącym 50% na nie więcej niż połowę maksymalnej wartości miejsc możliwych do zajecia. Dla dni i warsztatów dodatkowo generowane są polecenia odpowiednio modyfikujące wartości pól PlacesLeft (aby nie tracić informacji o identyfikatorach odpowiednich rezerwacji nie używano tutaj procedury AddDay/WorkshopBooking).

Generowanych jest 1000 uczestników (maksymalna wielkość wygenerowanych danych z serwisu dla konta bez płatnej subskrypcji), o prawdopodobieństwie posiadania tytułu wynoszącym 50%. Następnie dla każdej rezerwacji z prawdopodobieństwem 50% podejmowana jest decyzja o tym czy wypełniać wszystkie miejsca uczestnikami, a następnie wykonywany jest zapis wszystkich studentów oraz pozostałych uczestników (tak, aby w ramach jednego dnia/warsztatu nie następowały powtórki).

9.2. Kod generatora

```
comp_names = open('company-names.csv', 'r')
comp_names_list = comp_names.readlines()
comp_names.close()
for i in range(0, len(comp_names_list)):
    if '"' in comp_names_list[i]:
        comp_names_list[i] = comp_names_list[i].replace('"', '')

full_names = open('full-names.csv', 'r')
full_names_list = full_names.readlines()
full_names.close()
for i in range(0, len(full_names_list)):
    if "'" in full_names_list[i]:
        full_names_list[i] = full_names_list[i].replace("'", "")

addresses = open('addresses.csv', 'r')
addresses_list = addresses.readlines()
addresses.close()
for i in range(0, len(addresses_list)):
    if "'" in addresses_list[i]:
        addresses_list[i] = addresses_list[i].replace("'", "")

conference_names = open('conf-names.csv', 'r')
conference_names_list = conference_names.readlines()
conference_names.close()

num_of_places = open('num-of-places.csv', 'r')
num_of_places_list = num_of_places.readlines()
num_of_places.close()

conf_prices = open('conf-prices.csv', 'r')
conf_prices_list = conf_prices.readlines()
conf_prices.close()
for i in range(0, len(conf_prices_list)):
    if "," in conf_prices_list[i]:
        conf_prices_list[i] = conf_prices_list[i].replace(",", ".")

workshop_names = open('workshop-names.csv', 'r')
workshop_names_list = workshop_names.readlines()
workshop_names.close()

e_mails = open('e-mails.csv', 'r')
e_mails_list = e_mails.readlines()
```

```
e_mails.close()

phones = open('phones.csv', 'r')
phones_list = phones.readlines()
phones.close()

titles = open('titles.csv', 'r')
titles_list = titles.readlines()
titles.close()

student_cards = open('student-cards.csv', 'r')
student_cards_list = student_cards.readlines()
student_cards.close()

def make_insert_query(table, columns, values):
    result = "insert into " + table + " ("
    for i in columns[0:-1]:
        result += i
        result += ", "
    result += columns[-1]
    result += ") values ("
    for i in values[0:-1]:
        if i == '':
            result += "null"
        else:
            result += "'" +  i.replace('\n', '') + "'"
        result += ", "
    if values[-1] == '':
        result += "null"
    else:
        result += "'" + values[-1].replace('\n', '') + "'"
    result += ") \n"
    return result

def make_update_query(table, column_to_set, value, where_column, \
where_value):
    result = "update " + table + " set " + column_to_set + " = " + \
    str(value) + " where " + where_column + " = " + str(where_value) + '\n'
    return result

def make_organizers(attributes):
    organizers_cols = ["OrganizerID", "OrganizerName", "ContactName",
    "Address", "PostalCode", "City", "Region", "Country"]
    return make_insert_query("Organizers", organizers_cols, attributes)
```

```
87
88  def make_conferences(attributes):
89      conferences_cols = ["ConferenceID", "OrganizerID", "ConferenceName",
90      "FirstDay", "LastDay", "PlaceAddress", "PlaceCity", "PlaceRegion",
91      "PlaceCountry"]
92      return make_insert_query("Conferences", conferences_cols, attributes)
93
94  def make_confdays(attributes):
95      confdays_cols = ["DayID", "ConferenceID", "Date", "NumberOfPlaces",
96      "PlacesLeft"]
97      return make_insert_query("ConferenceDays", confdays_cols, attributes)
98
99  def make_pricethr(attributes):
100     pricethr_cols = ["ThresholdID", "DayID", "Price", "StudentDiscount",
101     "StartDate", "EndDate"]
102     return make_insert_query("PriceThresholds", pricethr_cols, attributes)
103
104 def make_workshops(attributes):
105     workshops_cols = ["WorkshopID", "DayID", "WorkshopName", "WorkshopStart",
106     "WorkshopEnd", "NumberOfPlaces", "PlacesLeft", "Price"]
107     return make_insert_query("Workshops", workshops_cols, attributes)
108
109 def make_clients(attributes):
110     clients_cols = ["ClientID", "ClientName", "ContactName", "ContactEMail",
111     "Address", "City", "PostalCode", "Region", "Country", "Phone", "Fax"]
112     return make_insert_query("Clients", clients_cols, attributes)
113
114 def make_booking(attributes):
115     booking_cols = ["BookingID", "ClientID", "ConferenceID",
116     "ParticipantsNumber", "RegistrationDate", "isValid"]
117     return make_insert_query("Bookings", booking_cols, attributes)
118
119 def make_payment(attributes):
120     payment_cols = ["PaymentID", "BookingID", "PaymentDate", "AmountPaid"]
121     return make_insert_query("Payments", payment_cols, attributes)
122
123 def make_daybooking(attributes):
124     daybooking_cols = ["DayBookingID", "DayID", "BookingID",
125     "NumberOfParticipants", "NumberOfStudents", "isValid"]
126     return make_insert_query("DayBookings", daybooking_cols, attributes)
127
128 def make_workshopbooking(attributes):
129     workshopbooking_cols = ["WorkshopBookingID", "DayBookingID",
130     "WorkshopID", "NumberOfParticipants", "isValid"]
```

```
131     return make_insert_query("WorkshopBookings", workshopbooking_cols, \
132     attributes)
133 
134 def make_participant(attributes):
135     participant_cols = ["ParticipantID", "FirstName", "LastName", "Title"]
136     return make_insert_query("Participants", participant_cols, attributes)
137 
138 def make_participant_of_day(attributes):
139     pod_cols = ["ParticipantOfDayID", "ParticipantID", "DayBookingID",
140     "StudentCardNumber"]
141     return make_insert_query("ParticipantsOfDay", pod_cols, attributes)
142 
143 def make_participant_of_workshop(attributes):
144     pow_cols = ["WorkshopBookingID", "ParticipantOfDayID"]
145     return make_insert_query("ParticipantsOfWorkshop", pow_cols, attributes)
146 
147 import random
148 
149 set_command = "set identity_insert "
150 
151 organizers_out = open('gen-organizers.sql', 'w')
152 organizers_out.write(set_command + "Organizers on\n\n")
153 
154 conferences_out = open('gen-conferences.sql', 'w')
155 conferences_out.write(set_command + "Conferences on\n\n")
156 
157 confdays_out = open('gen-confdays.sql', 'w')
158 confdays_out.write(set_command + "ConferenceDays on\n\n")
159 
160 pricethr_out = open('gen-pricethr.sql', 'w')
161 pricethr_out.write(set_command + "PriceThresholds on\n\n")
162 
163 workshops_out = open('gen-workshops.sql', 'w')
164 workshops_out.write(set_command + "Workshops on\n\n")
165 
166 clients_out = open('gen-clients.sql', 'w')
167 clients_out.write(set_command + "Clients on\n\n")
168 
169 bookings_out = open('gen-bookings.sql', 'w')
170 bookings_out.write(set_command + "Bookings on\n\n")
171 
172 payments_out = open('gen-payments.sql', 'w')
173 payments_out.write(set_command + "Payments on\n\n")
174 
```

```
daybookings_out = open('gen-daybookings.sql', 'w')
daybookings_out.write(set_command + "DayBookings on\n\n")

workshopbookings_out = open('gen-workshopbookings.sql', 'w')
workshopbookings_out.write(set_command + "WorkshopBookings on\n\n")

participants_out = open('gen-participants.sql', 'w')
participants_out.write(set_command + "Participants on\n\n")

pod_out = open('gen-pods.sql', 'w')
pod_out.write(set_command + "ParticipantsOfDay on\n\n")

pow_out = open('gen-pows.sql', 'w')

day_id = 1
workshop_id = 1
price_thr_id = 1

conf_max_places = []
day_places = []
workshop_places_list = []
conf_start_days = []
conferences_days_ids = {}
conferences_days_workshops_ids = {}

first_years = ('2017', '2018', '2019')
first_months = ('01', '02', '03', '04', '05', '06', '07', '08', '09', '10',
'11', '12')
first_days = ('01', '02', '03', '04', '05', '06', '07', '08', '09', '10',
'11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22',
'23')
student_discounts = ['0', '10', '25', '50']
workshop_starts = ['09:00:00', '11:00:00', '14:00:00']

#making organizers and conferences (and its details)
for organizer_id in range(1,11):
    org_attributes = []
    org_attributes.append(str(organizer_id))
    org_attributes.append(comp_names_list[organizer_id])
    org_attributes.append(full_names_list[organizer_id])
    full_address = addresses_list[organizer_id].split(';')
    for i in full_address:
        org_attributes.append(i)
    if org_attributes[4] == '':
```

```
219         org_attributes[4] = str(random.randrange(30800,45000))
220     organizers_out.write(make_organizers(org_attributes))
221 
222     #conferences
223     first_dates = []
224     end_dates = []
225     for y in first_years:
226         for conf_num in range(0,3):
227             month = random.choice(first_months)
228             day = int(random.choice(first_days))
229             how_long = random.randrange(2,5)
230             first_dates.append(y + "-" + month + "-" + str(day))
231             end_dates.append(y + "-" + month + "-" + str(day + how_long))
232 
233     for conference_id in range(1 + (organizer_id - 1) * 9, \
234     (organizer_id) * 9 + 1):
235         conf_attributes = [str(conference_id), str(organizer_id)]
236         conf_attributes.append(conference_names_list[conference_id])
237         conf_attributes.append(first_dates[conference_id % 9])
238         conf_attributes.append(end_dates[conference_id % 9])
239         conf_full_address = addresses_list[10 + conference_id].split(';')
240         #10 to avoid the same address as organizers
241         conf_year = conf_attributes[3][0:4]
242         conf_month = conf_attributes[3][5:7]
243         conf_fday = int(conf_attributes[3][8:10])
244         conf_lday = int(conf_attributes[4][8:10])
245         for i in range(0, len(conf_full_address)):
246             if i != 1:
247             #on 1st position there is a postal code which is not needed
248             #in Conference table
249                 conf_attributes.append(conf_full_address[i])
250         conferences_out.write(make_conferences(conf_attributes))
251         conf_start_days.append(first_dates[conference_id % 9])
252 
253         #conference days
254         day_counter = 0
255         day_diff = conf_lday - conf_fday
256         temp_places = []
257         days_id_list = []
258 
259         while day_counter <= day_diff:
260             date = conf_year + "-" + conf_month + "-" + str(day_counter + \
261             conf_fday)
262             places = random.choice(num_of_places_list)
```

```
263             day_attributes = [str(day_id), str(conference_id), date,
264             places, places]
265             confdays_out.write(make_confdays(day_attributes))
266             day_places.append(int(places))
267             temp_places.append(int(places))
268             days_id_list.append(day_id)
269
270             #price thresholds
271             day_price = random.choice(conf_prices_list)[1:]
272             stud_disc = random.choice(student_discounts)
273             thr_starts = ['1970-01-01']
274             thr_year = int(conf_year)
275             thr_month = int(conf_month) - 3
276             if thr_month < 0:
277                 thr_month %= 12
278                 thr_year -= 1
279             elif thr_month == 0:
280                 thr_month = 12
281                 thr_year -= 1
282             thr_day = conf_fday + day_counter
283             thr_ends = [str(thr_year) + "-" + str(thr_month) +
284             "-" + str(thr_day)]
285             for thr_counter in range(0,3):
286                 thr_starts.append(str(thr_year) + "-"
287                 + str(thr_month) + "-" + str(thr_day + 1))
288                 thr_month += 1
289                 if thr_month > 12:
290                     thr_month %= 12
291                     thr_year += 1
292                 thr_ends.append(str(thr_year) + "-"
293                 + str(thr_month) + "-" + str(thr_day))
294
295             for price_num in range(0,4):
296                 pricethr_attributes = [str(price_thr_id), str(day_id),
297                 day_price, stud_disc, thr_starts[price_num],
298                 thr_ends[price_num]]
299                 pricethr_out.write(make_pricethr(pricethr_attributes))
300                 price_thr_id += 1
301                 price = float(day_price)
302                 price *= 1.15
303                 day_price = str(price)
304
305             #workshops
306             workshop_number = random.randrange(2,6)
```

```
307             workshop_list = []
308             for workshop_num in range(0, workshop_number):
309                 workshop_attributes = [str(workshop_id), str(day_id),
310                 random.choice(workshop_names_list)]
311                 workshop_start_time = random.choice(workshop_starts)
312                 workshop_attributes.append(workshop_start_time)
313                 workshop_end_time = str(int(workshop_start_time[0:2]) +
314                 random.randrange(1,4))
315                 workshop_end_time += ":00:00"
316                 workshop_attributes.append(workshop_end_time)
317                 workshop_places = int(random.uniform(0.1, 1.0) * int(places))
318                 if workshop_places == 0:
319                     workshop_places += 1
320                 workshop_attributes.append(str(workshop_places))
321                 workshop_attributes.append(str(workshop_places))
322                 workshop_attributes.append(str(random.uniform(0.0, 0.1) \
323                 * price))
324                 workshops_out.write(make_workshops(workshop_attributes))
325                 workshop_list.append(workshop_id)
326                 workshop_places_list.append(workshop_places)
327 
328                 workshop_id += 1
329             conferences_days_workshops_ids[day_id] = workshop_list
330 
331             day_counter += 1
332             day_id += 1
333         max_places = 0
334         for el in temp_places:
335             if el > max_places:
336                 max_places = el
337         conf_max_places.append(max_places)
338         conferences_days_ids[conference_id] = days_id_list
339 
340 booking_id = 1
341 payments_id = 1
342 day_booking_id = 1
343 workshop_booking_id = 1
344 bookings_days_ids_nums = {}
345 number_of_bookings = 0
346 booking_workshops_ids_nums = {}
347 
348 #Clients
349 for client_id in range(1, 751):
350     client_attributes = [str(client_id)]
```

```
    is_company = random.randint(0,2)
    if is_company == 1:
        client_attributes.append(comp_names_list[client_id + 20])
        #to avoid the same values as in organizers
        client_attributes.append(full_names_list[client_id + 20])
    else:
        client_attributes.append(full_names_list[client_id + 20])
        client_attributes.append('')
    client_attributes.append(e_mails_list[client_id])
    client_full_address = addresses_list[len(addresses_list) \
    - client_id].split(';')
    for i in client_full_address:
        client_attributes.append(i)
    buff = client_attributes[5]
    client_attributes[5] = client_attributes[6]
    client_attributes[6] = buff
    if client_attributes[6] == '':
        client_attributes[6] = str(random.randrange(30800,45000))
    client_attributes.append(phones_list[client_id])
    if is_company == 1:
        client_attributes.append(str(random.randrange(10005, 67889)))
    else:
        client_attributes.append('')
    clients_out.write(make_clients(client_attributes))

    #bookings
    number_of_bookings = random.randrange(2,5)
    for booking_num in range(0, number_of_bookings):
        booking_attributes = [str(booking_id), str(client_id)]
        booking_number_of_places = 0
        while booking_number_of_places == 0:
            conf_id = random.randrange(0, len(conf_max_places))
            if conf_max_places[conf_id] > 15:
                booking_number_of_places = \
                random.randrange(1, int(conf_max_places[conf_id] * 0.15))
            elif conf_max_places[conf_id] > 0:
                booking_number_of_places = 1
            else:
                booking_number_of_places = 0
        booking_attributes.append(str(conf_id + 1))
        booking_attributes.append(str(booking_number_of_places))

        conf_max_places[conf_id] -= booking_number_of_places

```

```
        conf_start_date = conf_start_days[conf_id]
        conf_year = int(conf_start_date[0:4])
        conf_month = int(conf_start_date[5:7])
        conf_day = int(conf_start_date[8:10])

        res_month_diff = random.randrange(1,6)
        res_month = conf_month - res_month_diff
        if res_month < 0:
            res_month %= 12
            res_year = conf_year - 1
        elif res_month == 0:
            res_month = 12
            res_year = conf_year - 1
        else:
            res_year = conf_year
        res_day = conf_day

        res_date = str(res_year) + "-"
        if res_month < 10:
            res_date += "0"
        res_date += str(res_month) + "-"
        if res_day < 10:
            res_date += "0"
        res_date += str(res_day)

        booking_attributes.append(res_date)
        booking_attributes.append('1')
        bookings_out.write(make_booking(booking_attributes))

        #Payments
        payments_number = random.randrange(0,3)
        pay_day = res_day
        for payment_made in range(0, payments_number):
            payments_attributes = [str(payments_id), str(booking_id)]
            pay_day += 1
            pay_date = res_date[0:8]
            if pay_day < 10:
                pay_date += "0"
            pay_date += str(pay_day)
            payments_attributes.append(pay_date)
            amount_paid = float(random.choice(conf_prices_list)[1:]) * 0.15
            payments_attributes.append(str(amount_paid))
            payments_out.write(make_payment(payments_attributes))
            payments_id += 1
```

```
439
440         #DaysBooking
441         book_days_info = []
442
443         for conf_day_id in conferences_days_ids[conf_id + 1]:
444             make_booking_for_this_day = random.randrange(0,2)
445             if make_booking_for_this_day == 1:
446                 days_booking_attributes = [str(day_booking_id),
447                 str(conf_day_id), str(booking_id)]
448                 max_allowed_places_taken = min(booking_number_of_places, \
449                 day_places[conf_day_id - 1])
450                 if max_allowed_places_taken == 0:
451                     continue
452                 elif max_allowed_places_taken == 1:
453                     booked_places = 1
454                 else:
455                     booked_places = random.randrange(1, \
456                     max_allowed_places_taken)
457
458                 days_booking_attributes.append(str(booked_places))
459                 number_of_students = random.randrange(0, booked_places + 1)
460                 days_booking_attributes.append(str(number_of_students))
461                 days_booking_attributes.append('1')
462                 day_places[conf_day_id - 1] -= booked_places
463
464                 daybookings_out.write( \
465                 make_daybooking(days_booking_attributes))
466                 daybookings_out.write(make_update_query("ConferenceDays", \
467                 "PlacesLeft", day_places[conf_day_id - 1], "DayID", \
468                 conf_day_id))
469
470                 book_days_info.append((day_booking_id, booked_places, \
471                 number_of_students))
472
473                 #WorkshopBooking
474                 workshop_booking_info = []
475
476                 for workshop_to_book_id in \
477                 conferences_days_workshops_ids[conf_day_id]:
478                     make_booking_for_this_workshop = random.randrange(0,2)
479                     if make_booking_for_this_workshop == 1:
480                         workshop_booking_attributes = [
481                         str(workshop_booking_id), str(day_booking_id),
482                         str(workshop_to_book_id)]
```

```
                        max_allowed_ws_places_taken = min(booked_places, \
                        workshop_places_list[workshop_to_book_id - 1] // 3)
                        if max_allowed_ws_places_taken == 0:
                            continue
                        elif max_allowed_ws_places_taken == 1:
                            booked_workshop_places = 1
                        else:
                            booked_workshop_places = \
                            max_allowed_ws_places_taken // 2
                        workshop_booking_attributes.append(\
                        str(booked_workshop_places))
                        workshop_booking_attributes.append('1')
                        workshop_places_list[workshop_to_book_id - 1] -= \
                        booked_workshop_places

                        workshopbookings_out.write( \
                         make_workshopbooking(workshop_booking_attributes))
                        workshopbookings_out.write( \
                        make_update_query("Workshops", "PlacesLeft", \
                        workshop_places_list[workshop_to_book_id - 1], \
                        "WorkshopID", workshop_to_book_id))

                        workshop_booking_info.append((workshop_booking_id, \
                        booked_workshop_places))

                        workshop_booking_id += 1

                booking_workshops_ids_nums[day_booking_id] = \
                workshop_booking_info
                day_booking_id += 1
        bookings_days_ids_nums[booking_id] = book_days_info
        booking_id += 1

#Participants
for participant_id in range(1, 1001):
    participant_attributes = [str(participant_id)]
    participant_full_name = full_names_list[participant_id - 1]
    participant_full_name = participant_full_name.split()
    participant_attributes.append(participant_full_name[0])
    participant_attributes.append(participant_full_name[1])

    participant_with_title = random.randrange(0,2)
    if participant_with_title == 1:
        participant_title = random.choice(titles_list)
```

```
527             participant_attributes.append(participant_title)
528         else:
529             participant_attributes.append('')
530 
531         participants_out.write(make_participant(participant_attributes))
532 
533 participant_of_day_id = 1
534 
535 #ParticipantsOfDay
536 for booking_id in bookings_days_ids_nums.keys():
537     for booking_day_info in bookings_days_ids_nums[booking_id]:
538         booking_day_id = booking_day_info[0]
539         number_of_participants = booking_day_info[1]
540         number_of_students = booking_day_info[2]
541 
542         complete_all_places = random.randrange(0,2)
543         if complete_all_places != 0:
544             places_to_fill = number_of_participants
545         else:
546             places_to_fill = max(number_of_students, \
547             random.randrange(0, number_of_participants))
548 
549         day_participants_ids = []
550         first_pod_id = participant_of_day_id
551 
552         for participants_of_day in range(0, places_to_fill):
553             participant_id = random.randrange(1, 1001)
554             while participant_id in day_participants_ids:
555                 participant_id = random.randrange(1, 1001)
556             day_participants_ids.append(participant_id)
557             participants_of_day_attributes = [str(participant_of_day_id),
558             str(participant_id), str(booking_day_id)]
559 
560             if number_of_students > 0:
561                 student_card_number = random.choice(student_cards_list)
562                 number_of_students -= 1
563             else:
564                 student_card_number = ''
565             participants_of_day_attributes.append(student_card_number)
566 
567             pod_out.write( \
568             make_participant_of_day(participants_of_day_attributes))
569             participant_of_day_id += 1
570 
```

```
571
572         #ParticipantsOfWorkshop
573         for workshop_booking_info in \
574         booking_workshops_ids_nums[booking_day_id]:
575             workshop_booking_id = workshop_booking_info[0]
576             workshop_booked_places = workshop_booking_info[1]
577             workshop_filled_places = min(workshop_booked_places, \
578             places_to_fill)
579             taken_pod_ids = [i for i in range(first_pod_id, \
580             participant_of_day_id)]
581
582             for workshop_place in range(0, workshop_filled_places):
583                 pow_attributes = [str(workshop_booking_id)]
584                 pod_id_to_become_pow = random.choice(taken_pod_ids)
585                 taken_pod_ids.remove(pod_id_to_become_pow)
586                 pow_attributes.append(str(pod_id_to_become_pow))
587                 pow_out.write(make_participant_of_workshop(pow_attributes))
588
589
590
591 organizers_out.write('\n' + set_command + "Organizers off\n\n")
592 conferences_out.write('\n' + set_command + "Conferences off\n\n")
593 confdays_out.write('\n' + set_command + "ConferenceDays off\n\n")
594 pricethr_out.write('\n' + set_command + "PriceThresholds off\n\n")
595 workshops_out.write('\n' + set_command + "Workshops off\n\n")
596 clients_out.write('\n' + set_command + "Clients off\n\n")
597 bookings_out.write('\n' + set_command + "Bookings off\n\n")
598 payments_out.write('\n' + set_command + "Payments off\n\n")
599 daybookings_out.write('\n' + set_command + "DayBookings off\n\n")
600 workshopbookings_out.write('\n' + set_command + "WorkshopBookings off\n\n")
601 participants_out.write('\n' + set_command + "Participants off\n\n")
602 pod_out.write('\n' + set_command + "ParticipantsOfDay off\n\n")
603 pow_out.write('\n' + set_command + "ParticipantsOfWorkshop off\n\n")
604
605 organizers_out.close()
606 conferences_out.close()
607 confdays_out.close()
608 pricethr_out.close()
609 workshops_out.close()
610 clients_out.close()
611 bookings_out.close()
612 payments_out.close()
613 daybookings_out.close()
614 workshopbookings_out.close()
```

```
615 participants_out.close()
616 pod_out.close()
617 pow_out.close()
618
619 import os
620 os.system('cat gen-organizers.sql gen-conferences.sql gen-confdays.sql \
621 gen-pricethr.sql gen-workshops.sql gen-clients.sql gen-bookings.sql \
622 gen-payments.sql gen-daybookings.sql gen-workshopbookings.sql \
623 gen-participants.sql gen-pods.sql gen-pows.sql > generator.sql')
624 os.system('rm gen-*')
```