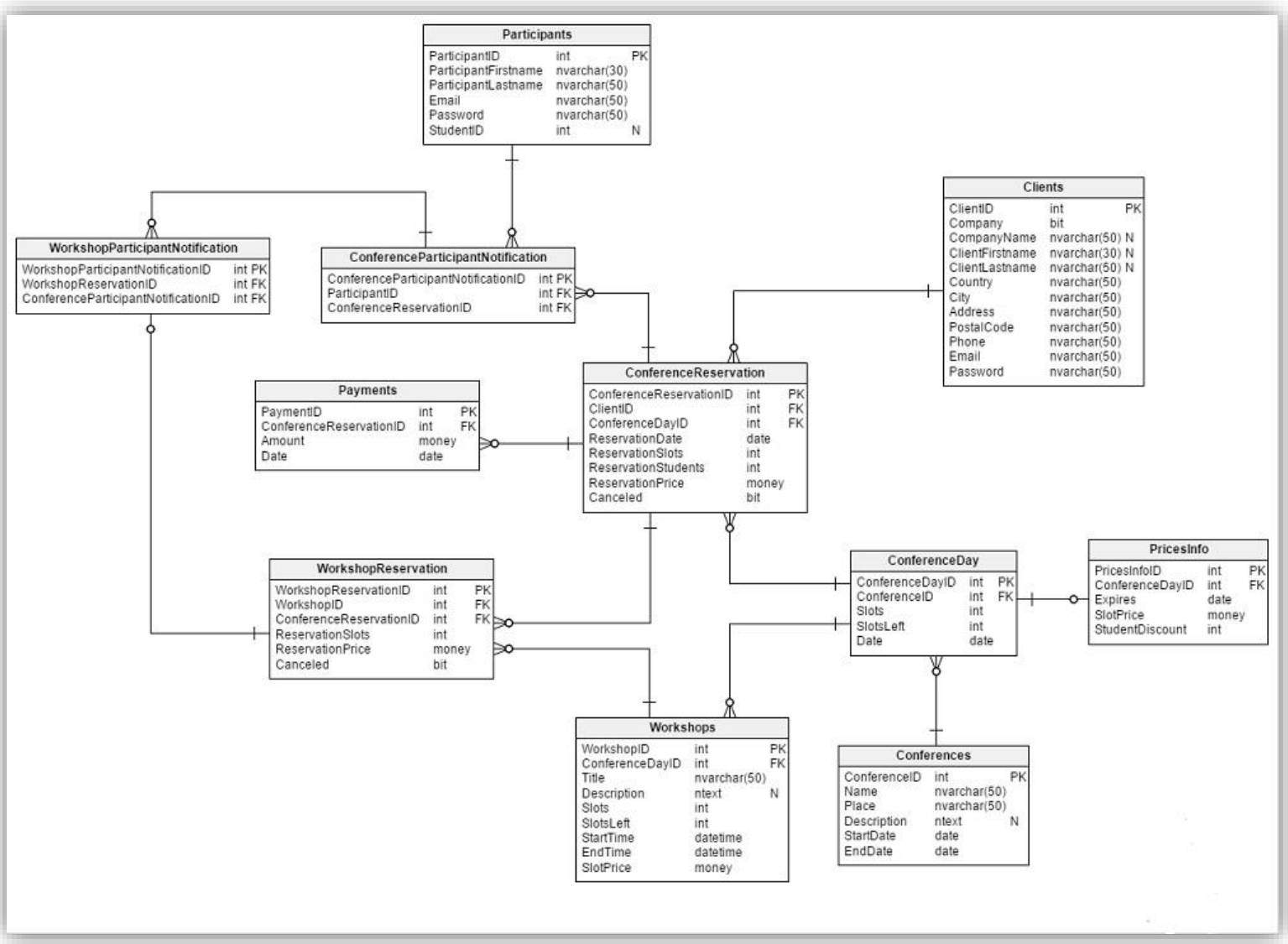
Podstawy baz danych I
Projekt systemu zarządzania konferencjami
Dokumentacja
Konrad Onieszczuk

1) Opis bazy danych

Celem projektu było stworzenie systemu wspomagającego firmę organizującą konferencje. Organizowane konferencje mogą być jedno lub kilkudniowe. Klientami mogą być zarówno indywidualne osoby jak i firmy, natomiast uczestnikami konferencji są osoby (firma nie musi podawać od razu przy rejestracji listy uczestników może zarezerwować odpowiednią ilość miejsc na określone dni oraz na warsztaty, natomiast na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić). Dla konferencji kilkudniowych, uczestnicy mogą rejestrować na dowolne z tych dni, dowolną liczbę osób. Klient może zmienić liczbę osób na rezerwacji, lub całkiem ją anulować (do 2 tygodni przed konferencją). System obsługiwany jest przez serwis www, który powinien korzystać z procedur i funkcji udostępnianych przez bazę. Baza dostarcza danych do wyświetlenia na stronie, natomiast poprzez serwis użytkownik może obsługiwać bazę bez bezpośredniego dostępu do niej.

2) Schemat bazy danych



3) Opis tabel * constrainty z serii *badgen* nie współpracują z generatorem

Tabela Clients

Przechowuje podstawowe dane o klientach firmy

- ClientID – klucz główny definiujący każdego klienta
- Company – bit informujący o tym czy klient jest firmą
- CompanyName – nazwa firmy, która może być nullem w przypadku gdy klient jest osobą prywatną
- ClientFirstname i ClientLastname – imiona i nazwiska klientów indywidualnych, które mogą być nullem w przypadku gdy klient jest firmą
- Dane adresowe typu kraj, miasto, kod pocztowy czy telefon niemogące być nullami, a dodatkowo z nałożonym checkiem na kod pocztowy i unikatowością numeru telefonu
- Unikatowy email i hasło są danymi dostępowymi na stronę www powiązaną z firmą

```
CREATE TABLE Clients (
    ClientID int  NOT NULL IDENTITY(1, 1),
    Company bit  NOT NULL,
    CompanyName nvarchar(50)  NULL,
    ClientFirstname nvarchar(30)  NULL,
    ClientLastname nvarchar(50)  NULL,
    Country nvarchar(50)  NOT NULL,
    City nvarchar(50)  NOT NULL,
    Address nvarchar(50)  NOT NULL,
    PostalCode nvarchar(50)  NOT NULL,
    Phone nvarchar(50)  NOT NULL,
    Email nvarchar(50)  NOT NULL,
    Password nvarchar(50)  NOT NULL,
    CONSTRAINT ClientsUnique UNIQUE (Email, Phone),
    CONSTRAINT check1 CHECK (PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'),
    CONSTRAINT check2 CHECK (Email LIKE '%[@]_%[.]__' OR
    Email LIKE '%[@]%[.]___' OR Email LIKE '%[@]%[.]____'  ),
    CONSTRAINT ClientsPK PRIMARY KEY  (ClientID)
)
```

Tabela ConferenceReservation

Przechowuje dane o rezerwacjah na konkretne dni konferencji

- ConferenceReservationID – klucz główny identyfikujący daną rezerwację
- ClientID – klucz obcy z tabeli ClientID pokazuje identyfikator rezerwującego
- ConferenceDayID – klucz obcy z tabeli ConferenceDay, wskazuje na identyfikator dnia konferencji, na który została zrobiona rezerwacja
- ReservationDate – data dokonania rezerwacji
- ReservationSlots i ReservationStudents – liczba zarezerwowanych miejsc odpowiednio dla wszystkich, tylko dla studentów oboszczona checkiem, który sprawdza czy jest liczbą dodatnią
- ReservationPrice – cena rezerwacji obliczana na podstawie daty rezerwacji i cennika odpowiadającego tej dacie w tabeli PricesInfo oboszczona checkiem sprawdzającym, czy jest liczbą dodatnią
- Cacneled – bit informujący, czy rezerwacja została anulowana

Żadne z pól nie może być nullem, a dodatkowo tabela posiada indeksy nieklastrowane na klucze obce w celu zwiększenia szybkości operacji, gdyż są one używane w procedurach, widokach, triggerach itp.

```
CREATE TABLE ConferenceReservation (
    ConferenceReservationID int  NOT NULL IDENTITY(1, 1),
    ClientID int  NOT NULL,
    ConferenceDayID int  NOT NULL,
    ReservationDate date  NOT NULL,
    ReservationSlots int  NOT NULL,
    ReservationStudents int  NOT NULL,
    ReservationPrice money  NOT NULL,
    Canceled bit  NOT NULL DEFAULT 0,
    CONSTRAINT check4 CHECK (ReservationSlots > 0),
    CONSTRAINT check5 CHECK (ReservationPrice >= 0),
    CONSTRAINT check7 CHECK (ReservationStudents >= 0),
    CONSTRAINT badgen1 CHECK (ReservationDate >= GETDATE()),
    CONSTRAINT ConferenceReservationPK PRIMARY KEY  (ConferenceReservationID)
)
CREATE NONCLUSTERED INDEX ConferenceReservationIndex1 on ConferenceReservation (ClientID ASC)
CREATE    NONCLUSTERED    INDEX    ConferenceReservationIndex2    on    ConferenceReservation
(ConferenceDayID ASC)
CREATE  NONCLUSTERED  INDEX  ConferenceReservationIndex3  on  ConferenceReservation  (ClientID
ASC,ConferenceDayID ASC)
ALTER TABLE ConferenceReservation ADD CONSTRAINT ConferenceReservation_Clients
    FOREIGN KEY (ClientID)
    REFERENCES Clients (ClientID)
ALTER TABLE ConferenceReservation ADD CONSTRAINT ConfReservation_ConfDay
    FOREIGN KEY (ConferenceDayID)
    REFERENCES ConferenceDay (ConferenceDayID)
```

Tabela ConferenceDay

Przechowuje informacje o dniach konferencji

- ConferenceDayID – klucz główny identyfikujacy dany dzień konferencji
- ConferenceID – klucz obcy do tabeli Conference, czyli identyfikatora konferencji, w której ten dzień jest
- Slots i SlotsLeft - to odpowiednio miejsca całkowita ilość miejsc na konferencję i ilość miejsc, która została, oba muszą być liczbami dodatnimi, przy czym Slots > 0, a SlotsLeft z oczywistych względów >= 0
- Date – to unikatowe pole z datą tego dnia konferencji

Żadne z pól nie może być nullem, a dodatkowo tabela posiada indeksy nieklastrowane na klucz obcy ConferenceID w celu zwiększenia szybkości operacji, gdyż jest używany w procedurach, widokach, triggerach itp.

```
CREATE TABLE ConferenceDay (
    ConferenceDayID int  NOT NULL IDENTITY(1, 1),
    ConferenceID int  NOT NULL,
    Slots int  NOT NULL,
    SlotsLeft int  NOT NULL,
    Date date  NOT NULL,
    CONSTRAINT ConferenceDayUnique UNIQUE (Date),
    CONSTRAINT check8 CHECK (Slots > 0),
```

```
    CONSTRAINT check8a CHECK (SlotsLeft >= 0),
    CONSTRAINT ConferenceDayPK PRIMARY KEY  (ConferenceDayID),
    CONSTRAINT badgen2 CHECK (Date > GETDATE())

)
CREATE NONCLUSTERED INDEX ConferenceDayIndex1 on ConferenceDay (ConferenceID ASC)
ALTER TABLE ConferenceDay ADD CONSTRAINT ConferenceDay_Conferences
    FOREIGN KEY (ConferenceID)
    REFERENCES Conferences (ConferenceID)
```

Tabela PricesInfo

Przechowuje informacje o cenach, zniżkach wraz z terminami wygaśnięcia dla każdego dnia konferencji

- PricesInfoID – klucz główny identyfikujący tabelę
- ConferenceDayID – klucz obcy do danego dnia konferencji na którą obowiązuje cena
- Expires – data wygaśnięcia danego progu
- SlotPrice – cena za jedno miejsce
- StudentDiscount – zniżka studencka, obliczana poprzez wartość tego pola * 0.01

Żadne z pól nie może być nullem, a dodatkowo tabela posiada indeks nieklastrowany na klucz obcy ConferenceDayID w celu zwiększenia szybkości operacji, gdyż jest używany w procedurach, widokach, triggerach itp

```
CREATE TABLE PricesInfo (
    PricesInfoID int  NOT NULL IDENTITY(1, 1),
    ConferenceDayID int  NOT NULL,
    Expires date  NOT NULL,
    SlotPrice money  NOT NULL,
    StudentDiscount int  NOT NULL,
    CONSTRAINT check9 CHECK (StudentDiscount between 0 and 100),
    CONSTRAINT check10 CHECK (SlotPrice >= 0),
    CONSTRAINT PricesInfoPK PRIMARY KEY  (PricesInfoID),
    CONSTRAINT badgen3 CHECK (Expires >= GETDATE())

)
CREATE NONCLUSTERED INDEX PricesInfoIndex1 on PricesInfo (ConferenceDayID ASC)
ALTER TABLE PricesInfo ADD CONSTRAINT PricesInfo_ConferenceDay
    FOREIGN KEY (ConferenceDayID)
    REFERENCES ConferenceDay (ConferenceDayID)
```

Tabela Conferences

Przechowuje informacje o konferencjach

- ConferenceID – klucz główny definiujący daną konferencję
- Name – nazwa konferencji nie mogąca być nullem
- Place – miejsce odbycia się konferencji również bez nulla
- Description – opis danej konferencji, może być nullem
- StartDate i EndDate – to odpowiednio unikatowe dni startu i końca, przy czym koniec nie może być wcześniej niż początek co jest sprawdzane checkiem

```
CREATE TABLE Conferences (
    ConferenceID int  NOT NULL IDENTITY(1, 1),
    Name nvarchar(50)  NOT NULL,
    Place nvarchar(50)  NOT NULL,
    Description ntext  NULL,
    StartDate date  NOT NULL,
    EndDate date  NOT NULL,
    CONSTRAINT ConferencesUnique UNIQUE (StartDate, EndDate),
    CONSTRAINT check11 CHECK (StartDate <= EndDate),
    CONSTRAINT ConferencesPK PRIMARY KEY  (ConferenceID),
    CONSTRAINT badgen4 CHECK (StartDate > GETDATE()),
    CONSTRAINT badgen5 CHECK (EndDate > GETDATE())
)
```

Tabela Workshops

Przechowuje informacje o warsztatach

- WorkshopID – klucz główny definiujący dany warsztat
- ConferenceDayID – klucz obcy do tabeli ConferenceDay, czyli do danego dnia konferencji, w którym się ten warsztat odbywa
- Title, Place, Description – pola opisowe danego warsztatu
- Slots, SlotsLeft - to odpowiednio miejsca całkowita ilość miejsc na warsztat i ilość miejsc, która została, oba muszą być liczbami dodatnimi, przy czym Slots > 0, a SlotsLeft z oczywistych względów >= 0
- SlotPrice – to cena miejsca na warsztat, > 0

Żadne z pól za wyjątkiem description nie może być nullem, a dodatkowo tabela posiada indeks nieklastrowany na klucz obcy ConferenceDayID w celu zwiększenia szybkości operacji, gdyż jest używany w procedurach, widokach, triggerach itp

```
CREATE TABLE Workshops (
    WorkshopID int  NOT NULL IDENTITY(1, 1),
    ConferenceDayID int  NOT NULL,
    Title nvarchar(50)  NOT NULL,
    Description ntext  NULL,
    Slots int  NOT NULL,
    SlotsLeft int  NOT NULL,
    StartTime datetime  NOT NULL,
    EndTime datetime  NOT NULL,
    SlotPrice money  NOT NULL,
    CONSTRAINT check12 CHECK (Slots > 0),
    CONSTRAINT check13 CHECK (SlotPrice >= 0),
    CONSTRAINT check14 CHECK (StartTime < EndTime),
    CONSTRAINT check15 CHECK (SlotsLeft >= 0),
    CONSTRAINT WorkshopsPK PRIMARY KEY  (WorkshopID),
    CONSTRAINT badgen6 CHECK (StartTime > GETDATE()),
    CONSTRAINT badgen7 CHECK (EndTime > GETDATE())
)
CREATE NONCLUSTERED INDEX WorkshopsIndex1 on Workshops (ConferenceDayID ASC)
ALTER TABLE Workshops ADD CONSTRAINT Workshops_ConferenceDay
    FOREIGN KEY (ConferenceDayID)
    REFERENCES ConferenceDay (ConferenceDayID)
```

Tabela WorkshopReservation

Przechowuje dane o rezerwacji na warsztaty

- WorkshopReservationID – klucz główny identyfikujący daną rezerwację
- WorkshopID – klucz obcy do tabeli Workshops, czyli na dany warsztat, na który zrobiona jest rezerwacja
- ConferenceReservationID – klucz obcy do tabeli ConferenceReservation , czyli do rezerwacji na konferencje, gdyż rezerwacja na warsztaty jest związana z wcześniejszą rezerwacją na konferencję
- ReservationSlots – zarezerwowana ilość miejsc na warsztat, > 0
- ReservationPrice – cena rezerwacji >=0 , gdyż warsztaty mogą być darmowe
- Canceled – bit informujący o tym, czy dana rezerwacja jest anulowana

Żadne z pól nie może być nullem, a dodatkowo tabela posiada indeksy nieklastrowane na klucz obcy ConferenceReservationID oraz WorkshopID w celu zwiększenia szybkości operacji, gdyż jest używany w procedurach, widokach, triggerach itp

```
CREATE TABLE WorkshopReservation (
    WorkshopReservationID int  NOT NULL IDENTITY(1, 1),
    WorkshopID int  NOT NULL,
    ConferenceReservationID int  NOT NULL,
    ReservationSlots int  NOT NULL,
    ReservationPrice money  NOT NULL,
    Canceled bit  NOT NULL DEFAULT 0,
    CONSTRAINT check16 CHECK (ReservationSlots > 0),
    CONSTRAINT check17 CHECK (ReservationPrice >= 0),
    CONSTRAINT WorkshopReservationPK PRIMARY KEY  (WorkshopReservationID)

)
CREATE NONCLUSTERED INDEX WorkshopReservationIndex1 on WorkshopReservation (WorkshopID ASC)
CREATE NONCLUSTERED INDEX WorkshopReservationIndex2 on WorkshopReservation
(ConferenceReservationID ASC)
CREATE NONCLUSTERED INDEX WorkshopReservationIndex3 on WorkshopReservation (WorkshopID
ASC,ConferenceReservationID ASC)
ALTER TABLE WorkshopReservation ADD CONSTRAINT WorkshopReservation_Workshops
    FOREIGN KEY (WorkshopID)
    REFERENCES Workshops (WorkshopID)
ALTER TABLE WorkshopReservation ADD CONSTRAINT WorkshopReservation_ConferenceReservation
    FOREIGN KEY (ConferenceReservationID)
    REFERENCES ConferenceReservation (ConferenceReservationID)
```

Tabela WorkshopParticipantNotification

Przechowuje uczestników zarezerwowanych na warsztat z warunkiem wcześniejszego bycia zarejestrowanym na dany dzień konferencji, w którym odbywa się warsztat

- WorkshopParticipantNotificationID – klucz główny identyfikujący zarezerwowane miejsce
- WorkshopReservationID – klucz obcy do tabeli WorkshopReservation, czyli na dany warsztat, na który zarezerwowane są miejsca
- ConferenceParticipantNotification – klucz obcy do tabeli ConferenceParticipantNotification, czyli spełnienie warunku wcześniejszej rezerwacji na konferencję

Żadne z pól nie może być nullem, a dodatkowo tabela posiada indeksy nieklastrowane na klucz obcy WorkshopReservationID oraz ConferenceParticipantNotificationID w celu zwiększenia szybkości operacji, gdyż jest używany w procedurach, widokach, triggerach itp

```
CREATE TABLE WorkshopParticipantNotification (
    WorkshopParticipantNotificationID int  NOT NULL IDENTITY(1, 1),
    WorkshopReservationID int  NOT NULL,
    ConferenceParticipantNotificationID int  NOT NULL,
    CONSTRAINT WorkshopParticipantsPK PRIMARY KEY  (WorkshopParticipantNotificationID)
)
CREATE NONCLUSTERED INDEX WorkshopParticipantNotificationIndex1 on
WorkshopParticipantNotification (WorkshopReservationID ASC)
CREATE NONCLUSTERED INDEX WorkshopParticipantNotificationIndex2 on
WorkshopParticipantNotification (ConferenceParticipantNotificationID ASC)
ALTER TABLE WorkshopParticipantNotification ADD CONSTRAINT
WorkshopParticipants_WorkshopReservation
    FOREIGN KEY (WorkshopReservationID)
    REFERENCES WorkshopReservation (WorkshopReservationID)
ALTER TABLE WorkshopParticipantNotification ADD CONSTRAINT WPN_CPN
    FOREIGN KEY (ConferenceParticipantNotificationID)
    REFERENCES ConferenceParticipantNotification (ConferenceParticipantNotificationID)
```

Tabela ConferenceParticipantNotification

Przechowuje uczestników związanych z rezerwacją dany dzień konferencji

- ConferenceParticipantNotificationID – klucz główny definiujący zarezerwowane miejsce
- ParticipantID – klucz obcy do tabeli Participants, czyli do tabeli z informacją o danym uczestniku
- ConferenceReservationID – klucz obcy do tabeli ConferenceReservation, czyli do rezerwacji na dany dzień konferencji, na który uczestnik jest zarejestrowany

Żadne z pól nie może być nullem, a dodatkowo tabela posiada indeksy nieklastrowane na klucz obcy ConferenceReservationID oraz ParticipantID w celu zwiększenia szybkości operacji, gdyż jest używany w procedurach, widokach, triggerach itp

```
CREATE TABLE ConferenceParticipantNotification (
    ConferenceParticipantNotificationID int  NOT NULL IDENTITY(1, 1),
    ParticipantID int  NOT NULL,
    ConferenceReservationID int  NOT NULL,
    CONSTRAINT ConferenceParticipantsPK PRIMARY KEY  (ConferenceParticipantNotificationID)
)
CREATE NONCLUSTERED INDEX ConferenceParticipantNotificationIndex1 on
ConferenceParticipantNotification (ConferenceReservationID ASC)
CREATE NONCLUSTERED INDEX ConferenceParticipantNotificationIndex2 on
ConferenceParticipantNotification (ParticipantID ASC)
CREATE NONCLUSTERED INDEX ConferenceParticipantNotificationIndex3 on
ConferenceParticipantNotification (ParticipantID ASC,ConferenceReservationID ASC)
ALTER TABLE ConferenceParticipantNotification ADD CONSTRAINT
ConferenceParticipants_Participants
    FOREIGN KEY (ParticipantID)
    REFERENCES Participants (ParticipantID)
```

```
ALTER TABLE ConferenceParticipantNotification ADD CONSTRAINT
ConferenceParticipants_ConferenceReservation
    FOREIGN KEY (ConferenceReservationID)
    REFERENCES ConferenceReservation (ConferenceReservationID)
```

Tabela Participants

Przechowuje podstawowe informacje o uczestnikach konferencji i warsztatów

- ParticipantID – klucz główny identyfikujący danego uczestnika
- Firstname i Lastname – imię i nazwisko danego uczestnika, bez nulli
- Unikatowy email i hasło są danymi dostępowymi na stronę www powiązaną z firmą
- StudentID – unikalny numer legitymacji studenckiej uprawniający do zniżki, może być nullem w przypadku gdy uczestnik nie jest studentem

```
CREATE TABLE Participants (
    ParticipantID int  NOT NULL IDENTITY(1, 1),
    ParticipantFirstname nvarchar(30)  NOT NULL,
    ParticipantLastname nvarchar(50)  NOT NULL,
    Email nvarchar(50)  NOT NULL,
    Password nvarchar(50)  NOT NULL,
    StudentID int  NULL,
    CONSTRAINT ParticipantsUnique UNIQUE (Email, StudentID),
    CONSTRAINT check18 CHECK (Email LIKE '%[@]_%[.]__' OR Email LIKE '%[@]%[.]___'
    OR Email LIKE '%[@]%[.]____'  ),
    CONSTRAINT ParticipantsPK PRIMARY KEY  (ParticipantID)
)
```

Tabela Payments

Przechowuje płatności klientów

- PaymentID – klucz główny identyfikujący daną płatność
- ConferenceReservationID – klucz obcy do tabeli ConferenceReservation, czyli rezerwacji związanej z daną płatnością
- Amount – wysokość wpłaty
- Date – data wpłaty

```
CREATE TABLE Payments (
    PaymentID int  NOT NULL IDENTITY(1, 1),
    ConferenceReservationID int  NOT NULL,
    Amount money  NOT NULL,
    Date date  NOT NULL,
    CONSTRAINT check6 CHECK (Amount >= 0),
    CONSTRAINT PaymentsPK PRIMARY KEY  (PaymentID),
    CONSTRAINT badgen8 CHECK (Date >= GETDATE())
)
ALTER TABLE Payments ADD CONSTRAINT Payments_ConferenceReservation
    FOREIGN KEY (ConferenceReservationID)
    REFERENCES ConferenceReservation (ConferenceReservationID)
```

4) Procedury

AddClientCompany

Procedura dodająca klienta biznesowego czyt. firmę do bazy danych, przyjmująca jako argumenty wszystkie pola z tabeli Clients za wyjątkiem pól odpowiadających klientowi prywatnemu, które wpisuje jako null i ustawia flagę

```
create procedure AddClientCompany
	@CompanyName nvarchar(50),
	@Country nvarchar(50),
	@City nvarchar(50),
	@Address nvarchar(50),
	@PostalCode nvarchar(50),
	@Phone nvarchar(50),
	@Email nvarchar(50),
	@Password nvarchar(50)
	as
	begin
		insert into Clients (Company, CompanyName, ClientFirstname, ClientLastname,
		 Country, City, Address, PostalCode, Phone, Email, Password)
		values ('true', @CompanyName, NULL, NULL, @Country, @City, @Address, @PostalCode,
		 @Phone, @Email, @Password)
	end
```

AddClientPrivate

Procedura dodająca klienta prywatnego do bazydanych, przyjmująca jako argumenty wszystkie pola z tabeli Clients za wyjątkiem pól odpowiadających klientowi biznesowemu, które wpisuje jako null i ustawia flagę

```
create procedure AddClientPrivate
	@ClientFirstname nvarchar(50),
	@ClientLastname nvarchar(50),
	@Country nvarchar(50),
	@City nvarchar(50),
	@Address nvarchar(50),
	@PostalCode nvarchar(50),
	@Phone nvarchar(50),
	@Email nvarchar(50),
	@Password nvarchar(50)
	as
	begin
		insert into Clients (Company, CompanyName, ClientFirstname, ClientLastname, Country,
		City, Address, PostalCode, Phone, Email, Password)
		values ('false', NULL, @ClientFirstname, @ClientLastname, @Country, @City, @Address,
		@PostalCode, @Phone, @Email, @Password)
	end
```

AddParticipant

Procedura dodająca uczestnika konferencji i/lub warsztatów jako argumenty przyjmująca wszystkie pola z tabeli Participants i decydująca o wartości null w polu StudentID ze względu na wartość argumentu @StudentID

```
create procedure AddParticipant
	@Firstname nvarchar(50),
	@Lastname nvarchar(50),
	@Email nvarchar(50),
	@Password nvarchar(50),
	@StudentID int
	as
	begin
		if (@StudentID <> 0)
			insert into Participants (ParticipantFirstname, ParticipantLastname, Email,
			Password, StudentID)
			values (@Firstname, @Lastname, @Email, @Password, @StudentID)
		else
			insert into Participants (ParticipantFirstname, ParticipantLastname, Email,
			Password, StudentID)
			values (@Firstname, @Lastname, @Email, @Password, NULL)
	end
```

AddConference

Procedura dodająca konferencję jako argumenty przyjmująca wszystkie pola z tabeli Conferences i decudująca o wartości null w polu Description w zależności od wartości argumentu @Description

```
create procedure AddConference
	@Name nvarchar(50),
	@Place nvarchar(50),
	@Description ntext,
	@StartDate date,
	@EndDate date
	as
	begin
		if (@Description not like '0')
		begin
			insert into Conferences (Name, Place, Description, StartDate, EndDate)
			values (@Name, @Place, @Description, @StartDate, @EndDate)
		end
		else
		begin
			insert into Conferences (Name, Place, Description, StartDate, EndDate)
			values (@Name, @Place, NULL, @StartDate, @EndDate)
		end
	end
```

AddConferenceDay

Procedura dodająca dzień konferencji na daną konferencję, przyjmująca jako argumenty jej ID, liczbę miejsc oraz datę. Procedura sprawdza, czy podana data rzeczywiście jest jednym z dni konferencji.

```
create procedure AddConferenceDay
	@ConferenceID int,
	@Slots int,
	@Date date
	as
	begin
		if (@Date between (select StartDate from Conferences where ConferenceID =
		@ConferenceID) and (select EndDate from Conferences where ConferenceID =
		@ConferenceID))
		begin
			insert into ConferenceDay (ConferenceID, Slots, SlotsLeft, Date)
```

```
                values (@ConferenceID, @Slots, @Slots, @Date)
        end
        else
        begin
                THROW 51000, 'Wrong data', 1
        end
end
```

AddPricesInfo

Procedura dodająca informacje o cenach i zniżkach na dany dzień konferencji wraz z datą wygaśnięcia.

```
create procedure AddPricesInfo
        @ConferenceDayID int,
        @Expires date,
        @SlotPrice money,
        @StudentDiscount int
        as
        begin
                insert into PricesInfo (ConferenceDayID, Expires, SlotPrice, StudentDiscount)
                values (@ConferenceDayID, @Expires, @SlotPrice, @StudentDiscount)
        end
```

AddWorkshop

Procedura dodająca warsztat, przyjmuje jako argumenty wszystkie pola z tabeli Workshop za wyjątkiem SlotsLeft, a także sprawdza, czy daty rozpoczęcia i zakończenia mieszczą się w jednym dniu oraz jak wcześniej decyduje o wartości null w polu Description w zależności od wartości argumentu @Description

```
create procedure AddWorkshop
        @ConferenceDayID nvarchar(50),
        @Title nvarchar(50),
        @Description ntext,
        @Slots int,
        @StartTime datetime,
        @EndTime datetime,
        @SlotPrice money
        as
        begin
                if (CONVERT(date, @StartTime) = CONVERT(date, @EndTime))
                begin
                        if (@Description not like '0')
                        begin
                                insert into Workshops (ConferenceDayID, Title, Description, Slots,
                                SlotsLeft, StartTime, EndTime, SlotPrice)
                                values (@ConferenceDayID, @Title, @Description, @Slots, @Slots,
                                @StartTime, @EndTime, @SlotPrice)
                        end
                        else
                                insert into Workshops (ConferenceDayID, Title, Description, Slots,
                                SlotsLeft, StartTime, EndTime, SlotPrice)
                                values (@ConferenceDayID, @Title, NULL, @Slots, @Slots, @StartTime,
                                @EndTime, @SlotPrice)
                end
                else
                begin
                        THROW 51000, 'Workshop cannot last more than 1 day', 1
                end
        end
```

AddConferenceReservation

Procedura dodająca rezerwację na konferencję, przyjmująca wszystkie argumenty z tabeli ConferenceReservation za wyjątkiem ReservationPrice, gdyż to liczymy w procedurze. Na starcie sprawdza, czy już nie jest wcześniej nie istnieje taka rezerwacja, oraz czy jest jeszcze tyle wolnych miejsc na ile chcemy zarezerwować, a na koncu czy w bazie są jakieś informacje odnośnie ceny przy rezerwacji.

```
create procedure AddConferenceReservation
        @ClientID int,
        @ConferenceDayID int,
        @ReservationDate date,
        @ReservationSlots int,
        @ReservationStudents int,
        @Canceled bit
        as
        begin
                if (((select ClientID from ConferenceReservation where ClientID=@ClientID and
                ConferenceDayID=@ConferenceDayID) is null) and @ReservationSlots <= (select SlotsLeft
                from ConferenceDay where ConferenceDayID = @ConferenceDayID))
                begin
                        if (@Canceled <> 1) -- na potrzeby generatora :)
                        begin
                                declare @SlotPrice int = (select top 1 SlotPrice from PricesInfo where
                                ConferenceDayID=@ConferenceDayID and Expires >= @ReservationDate order
                                by Expires)
                                declare @Discount int = (select top 1 StudentDiscount from PricesInfo
                                where ConferenceDayID=@ConferenceDayID and Expires >=
                                @ReservationDate)
                                if (@SlotPrice is null and @Discount is null)
                                begin
                                        THROW 51000, 'Theres is no info about prices connected', 1
                                end
                                else
                                begin
                                        update ConferenceDay
                                        set SlotsLeft = SlotsLeft-@ReservationSlots
                                        where ConferenceDayID = @ConferenceDayID
                                        insert into ConferenceReservation (ClientID, ConferenceDayID,
                                        ReservationDate, ReservationSlots, ReservationStudents,
                                        ReservationPrice, Canceled)
                                        values (@ClientID, @ConferenceDayID, @ReservationDate,
                                        @ReservationSlots, @ReservationStudents,
                                        @SlotPrice*(@ReservationSlots-
                                        @ReservationStudents)+@ReservationStudents*(1-@Discount*0.01),
                                        @Canceled)
                                end
                        end
                        else
                        begin
                                insert into ConferenceReservation (ClientID, ConferenceDayID,
                                ReservationDate, ReservationSlots, ReservationStudents,
                                ReservationPrice, Canceled)
                                values (@ClientID, @ConferenceDayID, @ReservationDate,
                                @ReservationSlots, @ReservationStudents, 0, @Canceled)
                        end
                end
                else
                begin
                        THROW 51000, 'Conference has not enough slots or is already registered', 1
                end
        end
```

AddWorkshopReservation

Procedura dodająca rezerwację na warsztat, przyjmująca wszystkie argumenty z tabeli ConferenceReservation za wyjątkiem ReservationPrice, gdyż to liczymy w procedurze. Na starcie sprawdza, czy już nie jest wcześniej nie istnieje taka rezerwacja oraz czy jest zarezerwowany wcześniej na konferencję (konferencja wiąże się z warsztatami – taki jest wymóg) a także czy jest jeszcze tyle wolnych miejsc na ile chcemy zarezerwować, a na koncu czy w bazie są jakieś informacje odnośnie ceny przy rezerwacji.

```
create procedure AddWorkshopReservation
        @WorkshopID int,
        @ConferenceReservationID int,
        @ReservationSlots int,
        @Canceled bit
        as
        begin
                declare @ClientID int = (select ClientID from ConferenceReservation where
                ConferenceReservationID = @ConferenceReservationID)
                declare @ConferenceDayID int = (select ConferenceDayID from ConferenceReservation where
                ConferenceReservationID = @ConferenceReservationID)
                if ((select ConferenceReservationID from ConferenceReservation where ClientID =
                @ClientID and ConferenceDayID = @ConferenceDayID) is not null and
                (select WorkshopID from WorkshopReservation where WorkshopID=@WorkshopID and
                ConferenceReservationID = @ConferenceReservationID) is null and
                @ReservationSlots <= (select SlotsLeft from Workshops where WorkshopID = @WorkshopID))
                begin
                        if (@Canceled <> 1) --na potrzeby generatora
                        begin
                                update Workshops
                                set SlotsLeft = SlotsLeft - @ReservationSlots
                                where WorkshopID = @WorkshopID
                                begin
                                        insert into WorkshopReservation (WorkshopID,
                                        ConferenceReservationID, ReservationSlots, ReservationPrice,
                                        Canceled)
                                        values (@WorkshopID, @ConferenceReservationID,
                                        @ReservationSlots, @ReservationSlots * (select SlotPrice from
                                        Workshops where WorkshopID = @WorkshopID), @Canceled)
                                end
                        end
                        else
                        begin
                                insert into WorkshopReservation (WorkshopID, ConferenceReservationID,
                                ReservationSlots, ReservationPrice, Canceled)
                                values (@WorkshopID, @ConferenceReservationID, @ReservationSlots, 0,
                                @Canceled)
                        end
                end
                else
                begin
                        THROW 51000, 'Company has to be registered to the conferences first or is
                        already registered or there is not enough slots left', 1
                end
        end
```

AddConferenceParticipantNotification

Procedura dodająca uczestnika na konferencję przyjmująca wszystkie argumenty dla tabeli ConferenceParticipantNotification. Sprawdza, czy on już przypadkiem nie jest wcześniej zarejestrowany na ten dzień konferencji.

```
create procedure AddConferenceParticipantNotification
	@ParticipantID int,
	@ConferenceReservationID int
	as
	begin
		if ((select ConferenceParticipantNotificationID from ConferenceParticipantNotification
		where ParticipantID = @ParticipantID and ConferenceReservationID =
		@ConferenceReservationID) is null)
		begin
			insert into ConferenceParticipantNotification (ParticipantID,
			ConferenceReservationID)
			values (@ParticipantID, @ConferenceReservationID)
		end
		else
		begin
			THROW 51000, 'Participant is already registered on that day', 1
		end
	end
```

AddWorkshopParticipantNotification

Procedura dodająca uczestnika na warsztat przyjmująca wszystkie argumenty dla tabeli ConferenceParticipantNotification. Sprawdza, czy jest zarejestrowany wcześniej na konferencję, a następnie czy przypadkiem nie jest już wcześniej zarejestrowany.

```
create procedure AddWorkshopParticipantNotification
	@WorkshopReservationID int,
	@ConferenceParticipantNotificationID int
	as
	begin
		if ((select ParticipantID from ConferenceParticipantNotification where
		ConferenceParticipantNotificationID=@ConferenceParticipantNotificationID) is not null)
		begin
			if ((select WorkshopParticipantNotificationID from
			WorkshopParticipantNotification where WorkshopReservationID =
			@WorkshopReservationID and ConferenceParticipantNotificationID =
			@ConferenceParticipantNotificationID) is null)
			begin
				insert into WorkshopParticipantNotification (WorkshopReservationID,
				ConferenceParticipantNotificationID)
				values (@WorkshopReservationID, @ConferenceParticipantNotificationID)
			end
			else
			begin
				THROW 51000, 'Participant is already registered on that workshop', 1
			end
		end
		else
		begin
			THROW 51000, 'Participant has to be registered to a conference first', 1
		end
	end
```

AddPayment

Procedura dodająca płatność.

```
create procedure AddPayment
	@ConferenceReservationID int,
	@Amount money,
	@Date date
	as
	begin
		insert into Payments (ConferenceReservationID, Amount, Date)
		values (@ConferenceReservationID, @Amount, @Date)
	end
```

ChangeReservationSlots

Procedura zmieniająca liczbę zarezerwowanych miejsc, uaktualniająca cenę wiążącą się ze zmianą, a także sprawdzająca czy dana zmiana może mieć miejsce ze względu na dostępną ilość, czy dostępną informację o cenach.

```
create procedure ChangeReservationSlots
	@ConferenceReservationID int,
	@ReservationSlots int,
	@ReservationStudents int,
	@ReservationDate date
	as
	begin
		declare @ConferenceDayID int = (select ConferenceDayID from ConferenceReservation where
		ConferenceReservationID = @ConferenceReservationID)
		if (((select sum(ReservationSlots) from ConferenceReservation where ConferenceDayID =
		@ConferenceDayID)+@ReservationSlots)<=(select Slots from ConferenceDay where
		ConferenceDayID = @ConferenceDayID))
			begin
				declare @SlotPrice int = (select top 1 SlotPrice from PricesInfo where
				ConferenceDayID=@ConferenceDayID and Expires >= @ReservationDate order
				by Expires),
				@Discount int = (select top 1 StudentDiscount from PricesInfo where
				ConferenceDayID=@ConferenceDayID and Expires >= @ReservationDate)
				if (@SlotPrice is null and @Discount is null)
				begin
					THROW 51000, 'Theres is no info about prices connected', 1
				end
				else
				begin
					update ConferenceReservation
					set ReservationSlots = @ReservationSlots, ReservationStudents
					= @ReservationStudents,
					ReservationPrice = @SlotPrice*(@ReservationSlots-
					@ReservationStudents)+@ReservationStudents*(1-@Discount*0.01),
					ReservationDate = @ReservationDate
					where ConferenceReservationID = @ConferenceReservationID
				end
			end
		else
		begin
			THROW 51000, 'Conference day has not enough slots for your change request', 1
		end
	end
```

DeleteUnpaidReservations

Procedura usuwająca niezapłacone rezerwacje po 7 dniach rezrwacji. Resztę pracy wykonają za nas triggery, które po usunięciu rezerwacji automatycznie usuną powiązane rezerwacje warsztatów i uczestników związanych z nimi i konferencjami.

```
create procedure DeleteUnpaidReservations
        as
        begin
                declare cur cursor local for (select cr.ConferenceReservationID from
                ConferenceReservation cr
                left outer join Payments p on cr.ConferenceReservationID=p.ConferenceReservationID
                left outer join ConferenceDay cd on cr.ConferenceDayID = cd.ConferenceDayID
                where Amount is null and datediff(day, cr.ReservationDate, cd.Date) > 7)
                declare @UnpaidReservationID int
                open cur
                        fetch next from cur into @UnpaidReservationID
                        while @@FETCH_STATUS = 0
                        begin
                                delete ConferenceReservation
                                where ConferenceReservationID = @UnpaidReservationID
                                fetch next from cur into @UnpaidReservationID
                        end
                close cur
                deallocate cur
        end
```

CancelConferenceReservation

Procedura anulująca rezerwacje na konferencję, a co za tym idzie także powiązane z nimi rezerwacje na warsztat, co wykonuje procedura CancelWorkshopReservation. Następnie są usuwani uczestnicy konferencji

```
create proc CancelConferenceReservation
        @ConferenceReservationID int
        as
        begin
                if (select Canceled from ConferenceReservation where ConferenceReservationID =
                @ConferenceReservationID) <> 1
                        begin
                                update ConferenceReservation
                                set Canceled = 1
                                where ConferenceReservationID = @ConferenceReservationID
                                declare cur cursor local for (select WorkshopReservationID from
                                WorkshopReservation where ConferenceReservationID =
                                @ConferenceReservationID)
                                declare @WorkshopReservationID int
                                open cur
                                while @@FETCH_STATUS = 0
                                begin
                                        fetch next from cur into @WorkshopReservationID
                                        exec CancelWorkshopReservation @WorkshopReservationID
                                end
                                close cur
                                deallocate cur
                                delete ConferenceParticipantNotification
                                where ConferenceReservationID = @ConferenceReservationID
                        end
                else
                begin
                        THROW 51000, 'Reservation is already canceled', 1
                end
        end
```

CancelWorkshopReservation

Podobnie jak wyżej procedura jednak anuluje nie samą rezerwację konferencję, lecz na warsztat i usuwa oczywiście uczestników do niej przypisanych.

```
create proc CancelWorkshopReservation
	@WorkshopReservationID int
	as
	begin
		if (select Canceled from WorkshopReservation where WorkshopReservationID =
		@WorkshopReservationID) <> 1
			begin
				update WorkshopReservation
				set Canceled = 1
				where WorkshopReservationID = @WorkshopReservationID
				delete WorkshopParticipantNotification
				where WorkshopReservationID = @WorkshopReservationID
				declare @WorkshopID int = (select WorkshopID from WorkshopReservation
				where WorkshopReservationID = @WorkshopReservationID)
				declare @ReservationSlots int = (select ReservationSlots from
				WorkshopReservation where WorkshopReservationID =
				@WorkshopReservationID)
				update Workshops
				set SlotsLeft = SlotsLeft + @ReservationSlots
			end
	end
```

5) Triggery

*** warto zaznaczyć, że wiele sprawdzeń poprawności dodawanych danych jest już zawartych w procedurach przez co nie tworzyłem dla nich odrębnych triggerów**

Remove ConferenceConnectedComponents

Trigger wyzwalany przy usuwaniu rezerwacji na konferencję, usuwający powiązane z nią warsztaty (warsztaty są ściśle związane z konferencjami) wraz z ich uczestnikami, a następnie uczestników konferencji. Wszystko w kolejności pozwalającej bezpieczne usunięcie bez powodowania konfliktów z kluczami obcymi. Dopełnia procedure DeleteUnpaidReservations

```
create trigger RemoveConferenceConnectedComponents
	on ConferenceReservation
	after delete
	as
	begin
		declare @DeletedConferenceReservationID int = (select ConferenceReservationID from
		deleted)
		delete WorkshopReservation
		where ConferenceReservationID = @DeletedConferenceReservationID
		delete ConferenceParticipantNotification
		where ConferenceReservationID = @DeletedConferenceReservationID
	end
```

RemoveWorkshopConnectedComponents

Trigger wyzwalany przy usunięciu rejestracji na warsztat, który usuwa również uczestników powiązanych z tą rejestracją. Również jest dopełnieniem procedury DeleteUnpaidReservations czy procedur z serii Cancel.

```
create trigger RemoveWorkshopConnectedComponents
        on WorkshopReservation
        after delete
        as
        begin
                declare @DeletedWorkshopReservationID int = (select WorkshopReservationID from deleted)
                delete WorkshopParticipantNotification
                where WorkshopReservationID = @DeletedWorkshopReservationID
        end
```

SeparateConferences

Trigger sprawdzający, czy dodana konferencja nie koliduje z inną konferencją, nie mogą na siebie nachodzić.

```
create trigger SeparatedConferences
        on Conferences
        after insert, update
        as
        begin
                declare @Start date = (select StartDate from inserted)
                declare @End date = (select EndDate from inserted)
                declare @ConferenceID int = (select ConferenceID from Inserted)
                if ((select ConferenceID from Conferences where (@Start between StartDate and EndDate
                or @End between StartDate and EndDate) and ConferenceID <> @ConferenceID) is not null)
                begin
                        THROW 51000, 'Conferences cannot collide', 1
                end
        end
```

WorkshopCollideParticipant

Trigger czuwający, czy dodawany przez nas uczestnik warsztatu nie uczestnicy w innym warsztacie w tym samym czasie (specyfikacja projektu tego wymaga).

```
create trigger WorkshopCollideParticipant
        on WorkshopParticipantNotification
        after insert
        as
        begin
                declare @WorkshopReservationID int = (select WorkshopReservationID from inserted)
                declare @ConferenceParticipantNotificationID int = (select
                ConferenceParticipantNotificationID from inserted)
                delete WorkshopParticipantNotification
                where WorkshopReservationID = @WorkshopReservationID and
                ConferenceParticipantNotificationID = @ConferenceParticipantNotificationID
                declare @WorkshopIDinserted int = (select WorkshopID from WorkshopReservation where
                WorkshopReservationID = @WorkshopReservationID)
                declare @Start datetime = (select StartTime from Workshops where WorkshopID =
                @WorkshopIDinserted)
                declare @End datetime = (select EndTime from Workshops where WorkshopID =
                @WorkshopIDinserted)
                if (select ParticipantID from ConferenceParticipantNotification cpn
```

```
                        inner join WorkshopParticipantNotification wpn on
                        cpn.ConferenceParticipantNotificationID =
                        wpn.ConferenceParticipantNotificationID
                        inner join WorkshopReservation wr on wpn.WorkshopReservationID =
                        wr.WorkshopReservationID
                        inner join Workshops w on wr.WorkshopID = w.WorkshopID
                        where @ConferenceParticipantNotificationID =
                        cpn.ConferenceParticipantNotificationID and @Start <> w.StartTime and @End <>
                        w.EndTime and
                        (@Start between w.StartTime and w.EndTime or @End between w.StartTime and
                        w.EndTime))is not null
                        begin
                                THROW 51000, 'Participant cannot attend 2 workshops in same time', 1
                        end
                        else
                        begin
                                insert into WorkshopParticipantNotification (WorkshopReservationID,
                                ConferenceParticipantNotificationID)
                                values (@WorkshopReservationID, @ConferenceParticipantNotificationID)
                        end
        end
go
```

PricesInfoExpire

Trigger czuwający, aby cennik miał sens, tj. daty wygaśnięcia nie byłby starsze niż data samego dnia konferencji z nim związanego.

```
create trigger PricesInfoExpire
        on PricesInfo
        after insert, update
        as
        begin
                declare @Expire date = (select Expires from inserted)
                declare @ConferenceDayID int = (select ConferenceDayID from inserted)
                if ((select Date from ConferenceDay where ConferenceDayID = @ConferenceDayID) <
@Expire)
                begin
                        THROW 51000, 'Wrong expire date', 1
                end
        end
```

6) Funkcje

HowMuchShouldIPay

Funkcja zwraca całkowity koszt rezerwacji (wliczając w to warsztaty) dla podanego numeru rezerwacji.

```
create function HowMuchSHouldIPay
        (
                @ClientReservationID int
        )
        returns money
        as
        begin
```

```
                return (select ReservationPrice from ConferenceReservation where
                ConferenceReservationID = @ClientReservationID)
                 + (select isnull(sum(ReservationPrice),0) from WorkshopReservation where
                ConferenceReservationID = @ClientReservationID)
        end
```

ParticipantsSlotsLeftConference

Funkcja zwraca ile uczestników pozostało klientowi do zarejestrowania na daną rezerwacje konferencji.

```
create function ParticipantsSlotsLeftConference
        (
                @ClientConferenceReservationID int
        )
        returns int
        as
        begin
                return (select ReservationSlots from ConferenceReservation where
                ConferenceReservationID = @ClientConferenceReservationID)
                 - (select count(ConferenceParticipantNotifictionID) from
                ConferenceParticipantNotifiction where ConferenceReservationID =
                @ClientConferenceReservationID)
        end
```

ParticipantsSlotsLeftWorkshop

Odpowiednik wcześniejszej funkcji dla warsztatu

```
create function ParticipantsSlotsLeftWorkshop
        (
                @ClientWorkshopReservationID int
        )
        returns int
        as
        begin
                return (select ReservationSlots from WorkshopReservation where ConferenceReservationID
                = @ClientWorkshopReservationID)
                - (select count(WorkshopParticipantNotifictionID) from WorkshopParticipantNotifiction
                where WorkshopReservationID = @ClientWorkshopReservationID)
        end
```

ConferenceSlots

Funkcja zwraca całkowitą liczbę miejsc dla danej konferencji (względy marktingowe np.)

```
create function ConferenceSlots
        (
                @ConferenceID int
        )
        returns int
        as
        begin
                return (select sum(Slots) from ConferenceDay where ConferenceID = @ConferenceID)
        end
```

HowManyConferenceDays

Funkcja zwracajaca liczbę dni konferencji

```
create function HowManyConferenceDays
        (
                @ConferenceID int
        )
        returns money
        as
        begin
                return (select count(ConferenceDayID) from ConferenceDay where ConferenceID =
                @ConferenceID)
        end
```

7) Widoki

ReservationsAndPayments

Widok pozwalający firmie zobaczyć informacje o płatnościach na rezerwacje przez ich klientów.

```
create view ReservationsAndPayments
        as
        select CompanyName, ClientFirstname, ClientLastname, cr.ConferenceReservationID,
        ReservationDate, Amount, Date from ConferenceReservation cr
        left join Payments p  on cr.ConferenceReservationID = p.ConferenceReservationID
        inner join Clients c on cr.ClientID = c.ClientID
```

Top10Clients

Widok pozwalający firmie zobaczyć jej najczęściej korzystajacych z jej usług klientów.

```
create view Top10Clients
        as
        select top 10 CompanyName, ClientFirstname, ClientLastname, sum(ReservationPrice) as 'Całkowita
        liczba rezerwacji' from ConferenceReservation cr
        inner join Clients c on cr.ClientID = c.ClientID
        group by CompanyName, ClientFirstname, ClientLastname
        order by 4 desc
```

ConferenceParticipants

Widok pozwalający zobaczyć uczestników zarejestrowanych na daną konferencję.

```
create view ConferenceParticipants
        as
        select p.ParticipantFirstname, p.ParticipantLastname, c.Name, c.StartDate, c.EndDate from
        ConferenceParticipantNotification cnp
        inner join Participants p on cnp.ParticipantID = p.ParticipantID
        inner join ConferenceReservation cr on cnp.ConferenceReservationID = cr.ConferenceReservationID
        inner join ConferenceDay cd on cd.ConferenceDayID = cr.ConferenceDayID
        inner join Conferences c on c.ConferenceID = cd.ConferenceID
```

WorkshopParticipants

Odpowiednik wcześniejszego tylko tym razem dla warsztatów.

```
create view WorkshopParticipants
        as
        select p.ParticipantFirstname, p.ParticipantLastname, w.Title, w.StartTime, w.EndTime from
        WorkshopParticipantNotification wpn
        inner join ConferenceParticipantNotification cpn on wpn.ConferenceParticipantNotificationID =
        cpn.ConferenceParticipantNotificationID
        inner join Participants p on cpn.ParticipantID = p.ParticipantID
        inner join WorkshopReservation wr on wpn.WorkshopReservationID = wr.WorkshopReservationID
        inner join Workshops w on w.WorkshopID = wr.WorkshopID
```

WorkshopDuringConference

Widok pozwalający wyróżnić jakie warsztaty odbywają się podczas konferencji

```
create view WorkshopDuringConference
        as
        select w.Title as 'Workshop title', c.Name as 'Conference name', 'At days from
        '+cast(c.StartDate as varchar(50))+' to '+cast(c.EndDate as varchar(50))
        as 'Conference lasts' from Workshops w
        inner join ConferenceDay cd on cd.ConferenceDayID = w.ConferenceDayID
        inner join Conferences c on c.ConferenceID = cd.ConferenceID
```

TheMostActivesParticipant

Widok ukazujący najczęściej uczestniczących uczestników konferencji i warsztatów(możemy np. nagradzać ich dodatkowymi bonusami)

```
create view WorkshopParticipants
        as
        select top 10 p.ParticipantFirstname, p.ParticipantLastname, count(cpn.ParticipantID) as
        'Liczba odbytych konferencji wraz z warsztatami' from Participants p
        inner join ConferenceParticipantNotification cpn on cpn.ParticipantID = p.ParticipantID
        inner join ConferenceReservation cr on cpn.ConferenceReservationID = cr.ConferenceReservationID
        where Canceled <> 1
        group by p.ParticipantFirstname, p.ParticipantLastname
        order by 3 desc
```

8) Role

- **Administrator bazy** – dostęp do wszystkiego w celu obsługi bazy i weryfikacji potencjalnych uchybień
- **Pracownik firmy** – dostęp do widoków, procedur informacyjnych w celu wykonywania swojej pracy, np sporządzania konkretnych statystyk, czy także rejestracji klienta, bądź uczestnika za klienta itp.
- **Właściciel –** dostęp do najbardziej interesujących go rzeczy podumowywujących, czyli właśnie np widoki top, widoki z płatnościami, czy też niektóre funkcje dedykowane klientowi, które mogą go zainteresować
- **Klient** – dostęp do funkcji, które dedykowane są dla niego, czyli HowMuchShouldIPay itp. dostęp do procedur rezerwujących, czy też dopiero co instalujących się na stronie www, czyli AddClientPrivate/Company
- **Uczestnik** – dostęp do procedur rezerwujących na strone www

9) Generator

Generator został stworzony w **pythonie** i korzysta z wszystkich procedur z przedrostkiem Add. Łączy się z bazą danych przy pomocy biblioteki **pypyodbc** i za pomocą wcześniej wspomnianych procedur dodaje dane do bazy. Koliduje z constraintami badgen w tabelach (co zostało wcześniej wspomniane) z względów oczywistych. Generator korzysta w generowaniu niektórych danych losowych takich jak imiona, nazwiska, maile, czy hasła itp. z bibilioteki **faker**. Dane wygenerowane przez generator są spójne i spokojnie przechodzą przez „strażników” w procedurach, czy triggerach, co świadczy o dobrej jakości generatora.

Program generuje na bieżąco losowe dane, a następnie umieszcza je w bazie oraz zapisuje w pliku, który może być użyty jako plik wykonawczy sql.

Został podzielony na cześci

- Najpierw dodające klientów i uczestników.
- Później dni konferencji wraz z warsztatami
- Następnie rezerwacje na konferencje, a dalej warsztaty i wpłaty
- Na końcu dodawanie uczestników

Tak jak zostało w specyfikacji wspomniane generuje on 3-letnią działalność firmy, co daje około ~300 klientów, ~3000 uczestników, ~70 konferencji, ~150 dni, ~300 rezerwacji, ~400

informacji o cenach, ~300 rezerwacji na konferencje, ~200 rezerwacji na warsztat, ~5500 uczestników konferencji, ~400 uczestników warsztatów, ~200 wpłat.

Plik data.py zawiera klasy, funtions.py funkcje programu, a main.py jest głównym plikiem wykonawczym, który startuje cały generator.

Generator tworzy także plik tekstowy o nazwie log_generator, który obrazuje to co wcześniej wykonał. Został on załączony w folderze i może być użyty jako plik wykonawczy sql.