Projekt - Podstawy baz danych

System wspomagania zarządzania konferencjami

Michał Maksoń
Rafał Gaweł

Spis treści

1. Wstęp

Celem naszego projektu jest stworzenie systemu bazodanowego wspomagającego działalność firmy organizującej konferencję. Rejestracja odbywa się za pomocą strony www. Klienci mogą uczestniczyć zarówno w konferencjach jak i związanych z nimi warsztatach. Dane uczestników konferencji muszą zostać umieszczone najpóźniej 2 tygodnie przed rozpoczęciem konferencji. System oprócz informacji dotyczących przebiegu konferencji przechowuje również informacje związane z rezerwacjami, opłatami oraz ich uczestnikami.

Administrator

- pełen dostęp do bazy danych oraz jej funkcjonalności

Pracownik organizujący konferencję

- dodawanie konferencji
- anulowanie konferencji
- dodawanie klientów
- dostęp do informacji o kliencie
- generowanie raportów
- generowanie list osobowych na każdy dzień konferencji i każdy warsztat

Pracownik firmy organizującej konferencję

- dodawanie uczestnika do konferencji
- dostęp do listy klientów
- dostęp do listy uczestników
- dostęp do listy konferencji

Klient jako firma

- rezerwacja wielu miejsc bez podawania danych osobowych
- opłacanie udziału wielu osób
- dostęp do informacji o konferencjach i terminarza

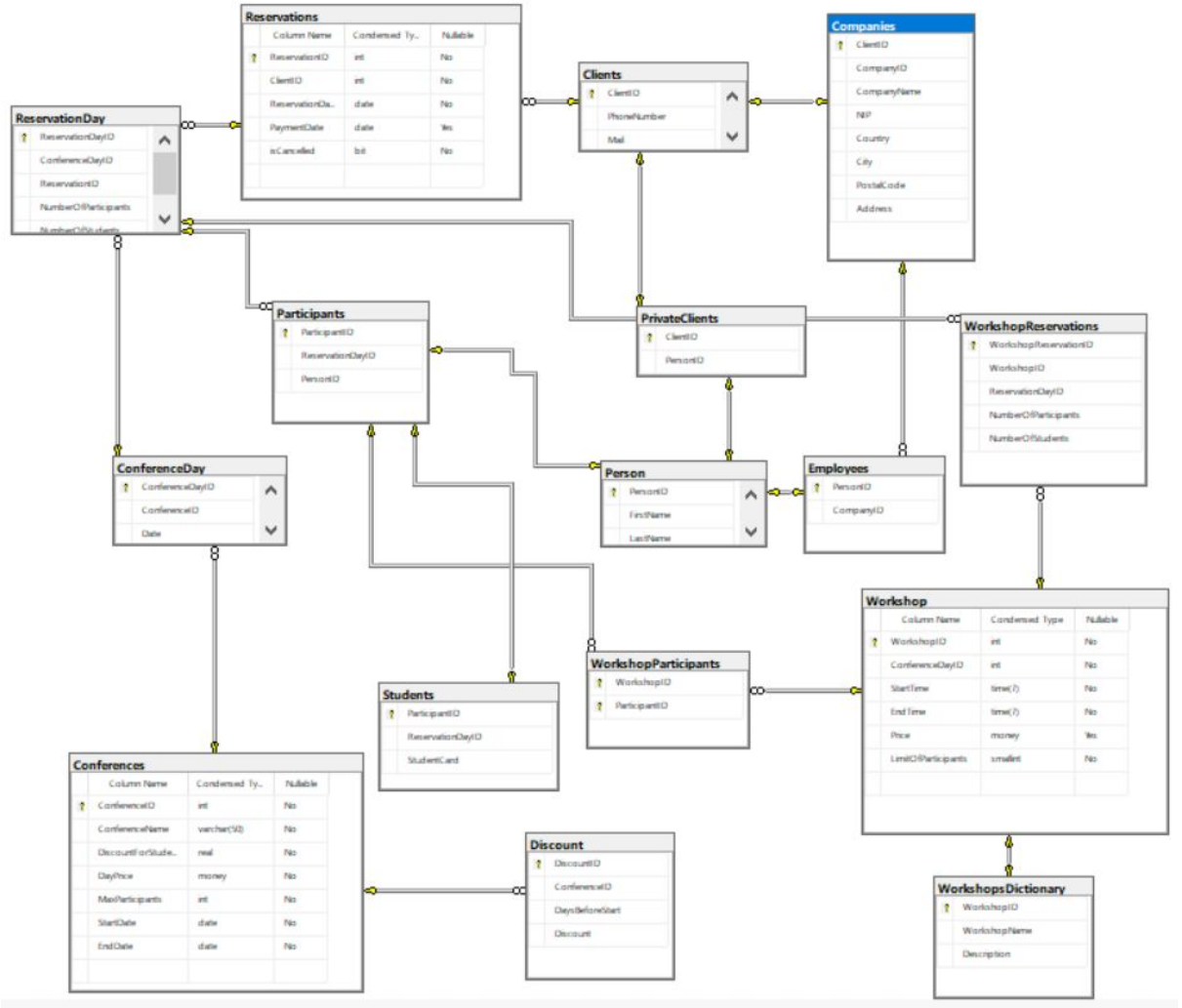
Klient jako osoba

- rezerwacja miejsca na konferencji
- dostęp do informacji o konferencjach i terminarza

Uczestnik konferencji

- zapisanie się na warsztat

2. Schemat bazy danych



3. Tabele

Tabela Clients

Tabela zawiera podstawowe dane kontaktowe klienta i stanowi wspólną część klientów indywidualnych oraz firmowych.

- ClientID - identyfikator klienta
- PhoneNumber - numer kontaktowy do klienta
- Mail - adres @ klienta

```
CREATE TABLE [dbo].[Clients](
	[ClientID] [int] IDENTITY(1000,1) NOT NULL,
	[PhoneNumber] [varchar](50) NOT NULL,
	[Mail] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Clients] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Clients]  WITH NOCHECK ADD  CONSTRAINT [CK_Clients]
CHECK  (([Mail] like '%@_%.%'))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [CK_Clients]
GO

ALTER TABLE [dbo].[Clients]  WITH NOCHECK ADD  CONSTRAINT [CK_Phone] CHECK
(([PhoneNumber] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Clients] CHECK CONSTRAINT [CK_Phone]
GO
```

Tabela Companies

Tabela bezpośrednio połączona z tabelą Customers. Zawiera informacje na temat klienta firmowego, dzięki czemu możemy pobrać dane na temat firmy.

- ClientID - identyfikator klienta
- CompanyID - identyfikator firmy
- CompanyName - nazwa firmy
- NIP - numer identyfikacyjny firmy
- Country - kraj, w którym znajduje się biuro firmy
- City - miasto, w którym znajduje się biuro firmy
- PostalCode - kod pocztowy
- Address - adres firmy

```
CREATE TABLE [dbo].[Companies](
	[ClientID] [int] NOT NULL,
	[CompanyID] [int] IDENTITY(1,1) NOT NULL,
	[CompanyName] [varchar](50) NOT NULL,
	[NIP] [int] NOT NULL,
	[Country] [varchar](50) NOT NULL,
	[City] [varchar](50) NOT NULL,
	[PostalCode] [varchar](50) NOT NULL,
	[Address] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Companies] PRIMARY KEY CLUSTERED
(
	[ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Companies]  WITH CHECK ADD  CONSTRAINT
[FK_Companies_Clients] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK_Companies_Clients]
GO
```

Tabela ConferenceDay

Tabela utworzona na każdy dzień konferencji wg dnia. Zawiera podstawowe informacje o dniu i służy jako pośrednie rozgałęzienie dla dalszej funkcjonalności bazy.

- ConferenceDayID - identyfikator poszczególnych dni konferencji
- ConferenceID - identyfikator konferencji, dla których są przyporządkowane dni
- Date - data konkretnego dnia konferencji8

```
CREATE TABLE [dbo].[ConferenceDay](
      [ConferenceDayID] [int] IDENTITY(1,1) NOT NULL,
      [ConferenceID] [int] NOT NULL,
      [Date] [date] NOT NULL,
 CONSTRAINT [PK_ConferenceDay] PRIMARY KEY CLUSTERED
(
      [ConferenceDayID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ConferenceDay]  WITH CHECK ADD  CONSTRAINT
[FK_ConferenceDay_Conferences] FOREIGN KEY([ConferenceID])
REFERENCES [dbo].[Conferences] ([ConferenceID])
GO

ALTER TABLE [dbo].[ConferenceDay] CHECK CONSTRAINT
[FK_ConferenceDay_Conferences]
GO
```

Tabela Conferences

Tabela zawierająca większość podstawowych danych na temat konferencji. Dzięki
niej możemy poznać ogólne informacje na temat danej konferencji. Wraz z
dołączoną tabelą Discounts umożliwia obliczenie ceny konferencji uwzględniając
progi cenowe i stanowi całość informacji na temat organizacji konferencji.

- ConferenceID - identyfikator konferencji
- ConferenceName - nazwa konferencji
- DiscountForStudents - wielkość zniżki studenckiej(od 0 do 1)
- DayPrice - cena jednego dnia konferencji
- MaxParticipants - maksymalna liczba uczestników na każdy z dni konfenrecji
- StartDate - data rozpoczęcia konferencji
- EndDate - data zakończenia konferencji

```
CREATE TABLE [dbo].[Conferences](
	[ConferenceID] [int] IDENTITY(1,1) NOT NULL,
	[ConferenceName] [varchar](50) NOT NULL,
	[DiscountForStudents] [real] NOT NULL,
	[DayPrice] [money] NOT NULL,
	[MaxParticipants] [int] NOT NULL,
	[StartDate] [date] NOT NULL,
	[EndDate] [date] NOT NULL,
 CONSTRAINT [PK_Conferences] PRIMARY KEY CLUSTERED
(
	[ConferenceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Conferences] ADD  CONSTRAINT [df_Discount1]  DEFAULT
((0)) FOR [DiscountForStudents]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[CK_Conferences] CHECK  (([StartDate]<[EndDate]))
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [CK_Conferences]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[CK_Conferences_1] CHECK  (([DayPrice]>=(0)))
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [CK_Conferences_1]
GO

ALTER TABLE [dbo].[Conferences]  WITH CHECK ADD  CONSTRAINT
[CK_Conferences_2] CHECK  (([MaxParticipants]>(0)))
GO

ALTER TABLE [dbo].[Conferences] CHECK CONSTRAINT [CK_Conferences_2]
GO
```

Tabela Discount

Tabela umożliwia obliczenie progów cenowych dla danej konferencji w zależności od
czasu.

- DiscountID – identyfikator zniżki
- ConferenceID – identyfikator konferencji
- DaysBeforeStart - liczba dni do dnia rozpoczęcia konferencji dla któychprzysługuje zniżka
- Discount - wartość zniżki

```
CREATE TABLE [dbo].[Discount](
      [DiscountID] [int] IDENTITY(1,1) NOT NULL,
      [ConferenceID] [int] NOT NULL,
      [DaysBeforeStart] [int] NOT NULL,
      [Discount] [real] NOT NULL,
 CONSTRAINT [PK_Discount] PRIMARY KEY CLUSTERED
(
      [DiscountID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Discount] ADD  CONSTRAINT [df_DiscountDisc]  DEFAULT
((0)) FOR [Discount]
GO

ALTER TABLE [dbo].[Discount]  WITH CHECK ADD  CONSTRAINT
[FK_Discount_Conferences] FOREIGN KEY([ConferenceID])
REFERENCES [dbo].[Conferences] ([ConferenceID])
GO

ALTER TABLE [dbo].[Discount] CHECK CONSTRAINT [FK_Discount_Conferences]
GO

ALTER TABLE [dbo].[Discount]  WITH CHECK ADD  CONSTRAINT [CK_Discount]
CHECK  (([DaysBeforeStart]>(0)))
GO

ALTER TABLE [dbo].[Discount] CHECK CONSTRAINT [CK_Discount]
GO
```

```
ALTER TABLE [dbo].[Discount]  WITH CHECK ADD  CONSTRAINT [CK_Discount_1]
CHECK  (([Discount]<(1)))
GO

ALTER TABLE [dbo].[Discount] CHECK CONSTRAINT [CK_Discount_1]
GO
```

Tabela Employees

Tabela przechowuje indendyfikator osoby i indendyfikator firmy dzięki czemu możemy uzyskac informacje na temat osoby i firmy

- PersonID - identyfikator osoby
- CompanyID - identyfikator firmy

```
CREATE TABLE [dbo].[Employees](
      [PersonID] [int] NOT NULL,
      [CompanyID] [int] NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
      [PersonID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[FK_Employees_Companies] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([CompanyID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Companies]
GO

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT
[FK_Employees_Person] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Person]
GO
```

Tabela Participants

Tabela zawierająca podstawowe dane wszystkich uczestników danego dniakonferencji.

- ParticipantID - identyfikator uczestnika
- DayReservationID - identyfikator dnia rezerwacji
- PersonID - identyfikator osoby

```
CREATE TABLE [dbo].[Participants](
      [ParticipantID] [int] NOT NULL,
      [DayReservationID] [int] NOT NULL,
      [PersonID] [int] NOT NULL,
 CONSTRAINT [PK_Participants] PRIMARY KEY CLUSTERED
(
      [ParticipantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Participants]  WITH CHECK ADD  CONSTRAINT
[FK_Participants_Person] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[Participants] CHECK CONSTRAINT [FK_Participants_Person]
GO

ALTER TABLE [dbo].[Participants]  WITH CHECK ADD  CONSTRAINT
[FK_Participants_ReservationDay] FOREIGN KEY([DayReservationID])
REFERENCES [dbo].[ReservationDay] ([ReservationDayID])
GO

ALTER TABLE [dbo].[Participants] CHECK CONSTRAINT
[FK_Participants_ReservationDay]
GO
```

Tabela Person

Tabela zawierająca podstawowe dane wszystkich uczestników osób.

- PersonID - identyfikator osoby
- FirstName - imię osoby
- LastName - nazwisko osoby

```
CREATE TABLE [dbo].[Person](
      [PersonID] [int] IDENTITY(10000,1) NOT NULL,
      [FirstName] [varchar](50) NOT NULL,
      [LastName] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Person] PRIMARY KEY CLUSTERED
(
      [PersonID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela PrivateClients

Tabela zawierająca uzupełnienie informacji o klientach indywidualnych. Jej funkcja
jest analogiczna do tabeli Companies.

- CulientID - identyfikator klienta
- PersonID - identyfikator osoby

```
CREATE TABLE [dbo].[PrivateClients](
      [ClientID] [int] NOT NULL,
      [PersonID] [int] NOT NULL,
 CONSTRAINT [PK_PrivateClients_1] PRIMARY KEY CLUSTERED
(
      [ClientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PrivateClients]  WITH CHECK ADD  CONSTRAINT
[FK_PrivateClients_Clients] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[PrivateClients] CHECK CONSTRAINT
[FK_PrivateClients_Clients]
GO

ALTER TABLE [dbo].[PrivateClients]  WITH CHECK ADD  CONSTRAINT
[FK_PrivateClients_Person] FOREIGN KEY([PersonID])
REFERENCES [dbo].[Person] ([PersonID])
GO

ALTER TABLE [dbo].[PrivateClients] CHECK CONSTRAINT
[FK_PrivateClients_Person]
GO
```

Tabela ReservationDay

Tabela przechowująca informacje na konkretne dni konferencji. Najważniejszym
zastosowaniem i cechą tabeli jest możliwość sprawdzenia ilości uczestników na
każdy dzień konferencji.

- ReservationDayID - idenfikator dnia rezerwacji
- ConferenceDayID - identyfikator dnia konferenccji
- ReservationID - identyfikator rezerwacji
- NumberOfParticipants - liczba rezerwacji w konkretnym dniu
- NumberOfStudents - liczba rezerwacji studentów w konkretnym dniu

```
CREATE TABLE [dbo].[ReservationDay](
      [ReservationDayID] [int] IDENTITY(1,1) NOT NULL,
      [ConferenceDayID] [int] NOT NULL,
      [ReservationID] [int] NOT NULL,
      [NumberOfParticipants] [smallint] NOT NULL,
      [NumberOfStudents] [smallint] NOT NULL,
 CONSTRAINT [PK_ReservationDay] PRIMARY KEY CLUSTERED
(
      [ReservationDayID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ReservationDay] ADD  CONSTRAINT [df_reservDayStud]
DEFAULT ((0)) FOR [NumberOfStudents]
GO

ALTER TABLE [dbo].[ReservationDay]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationDay_ConferenceDay] FOREIGN KEY([ConferenceDayID])
REFERENCES [dbo].[ConferenceDay] ([ConferenceDayID])
GO

ALTER TABLE [dbo].[ReservationDay] CHECK CONSTRAINT
[FK_ReservationDay_ConferenceDay]
GO

ALTER TABLE [dbo].[ReservationDay]  WITH CHECK ADD  CONSTRAINT
[FK_ReservationDay_Reservations] FOREIGN KEY([ReservationID])
REFERENCES [dbo].[Reservations] ([ReservationID])
GO
```

```
ALTER TABLE [dbo].[ReservationDay] CHECK CONSTRAINT
[FK_ReservationDay_Reservations]
GO

ALTER TABLE [dbo].[ReservationDay]  WITH CHECK ADD  CONSTRAINT
[CK_ReservationDay] CHECK  (([NumberOfParticipants]>(0)))
GO

ALTER TABLE [dbo].[ReservationDay] CHECK CONSTRAINT [CK_ReservationDay]
GO
```

Tabela Reservations

Tabela zawiera ogólne informacje na temat rezerwacji. Dzięki niej wiadomo, jakiej i kiedy rezerwacji dokonał klient, czy rezerwacja jest opłacona na kanwie daty w wymaganym czasie jednego tygodnia, a także czy została anulowana.

- ReservationID - identyfikator rezerwacji
- ClientID - identyfikator klienta
- ReservationDate - data rezerwacji
- PaymentDate - data dokonania płatności
- isCancelled - informacja, czy rezerwacja została anulowana

```
CREATE TABLE [dbo].[Reservations](
      [ReservationID] [int] IDENTITY(1,1) NOT NULL,
      [ClientID] [int] NOT NULL,
      [ReservationDate] [date] NOT NULL,
      [PaymentDate] [date] NULL,
      [isCancelled] [bit] NOT NULL,
 CONSTRAINT [PK_Reservations] PRIMARY KEY CLUSTERED
(
      [ReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Reservations]  WITH CHECK ADD  CONSTRAINT
[FK_Reservations_Clients] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([ClientID])
GO

ALTER TABLE [dbo].[Reservations] CHECK CONSTRAINT [FK_Reservations_Clients]
GO
```

Tabela Students

Tabela zawiera przede wszystkim numer legitymacji studenckiej i jest bezpośrednio połączoną z tabelą Participants.h.

- ParticipantID - identyfikator uczestnika
- ParticipantID - identyfikator rezerwacji
- StudentCard - numer legitymacji studenckiej

```
CREATE TABLE [dbo].[Students](
      [ParticipantID] [int] NOT NULL,
      [ReservationID] [int] NOT NULL,
      [StudentCard] [nchar](10) NULL,
 CONSTRAINT [PK_Students] PRIMARY KEY CLUSTERED
(
      [ParticipantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Students]  WITH CHECK ADD  CONSTRAINT
[FK_Students_Participants] FOREIGN KEY([ParticipantID])
REFERENCES [dbo].[Participants] ([ParticipantID])
GO

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [FK_Students_Participants]
GO
```

Tabela Workshop

Tabela zawiera większość informacji oraz podstawowych szczegółów o warsztacie.

- WorkShopID - identyfikator warsztatu
- ConferenceDayID - identyfikator dnia konferencji
- StartTime - godzina rozpoczęcia warsztatu
- EndTime - godzina zakończenia warsztatu
- Price - stała cena warsztatu
- LimitOfParticipants - limit uczestników w warsztacie

```
CREATE TABLE [dbo].[Workshop](
      [WorkshopID] [int] IDENTITY(1,1) NOT NULL,
      [ConferenceDayID] [int] NOT NULL,
      [StartTime] [time](7) NOT NULL,
      [EndTime] [time](7) NOT NULL,
      [Price] [money] NULL,
      [LimitOfParticipants] [smallint] NOT NULL,
 CONSTRAINT [PK_Workshop] PRIMARY KEY CLUSTERED
(
      [WorkshopID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Workshop] ADD  CONSTRAINT [df_Price1]  DEFAULT ((0)) FOR
[Price]
GO

ALTER TABLE [dbo].[Workshop]  WITH CHECK ADD  CONSTRAINT [CK_Workshop]
CHECK  (([LimitOfParticipants]>(0)))
GO

ALTER TABLE [dbo].[Workshop] CHECK CONSTRAINT [CK_Workshop]
GO

ALTER TABLE [dbo].[Workshop]  WITH CHECK ADD  CONSTRAINT [CK_Workshop_1]
CHECK  (([StartTime]<[EndTime]))
GO

ALTER TABLE [dbo].[Workshop] CHECK CONSTRAINT [CK_Workshop_1]
GO
```

```
ALTER TABLE [dbo].[Workshop]  WITH CHECK ADD  CONSTRAINT [CK_Workshop_2]
CHECK  (([Price]>=(0)))
GO

ALTER TABLE [dbo].[Workshop] CHECK CONSTRAINT [CK_Workshop_2]
GO
```

Tabela WorkshopParticipants

Tabela przechowująca identyfikator uczestnika warsztatu, dzięki czemu możemy otrzymać więcej informacji na temat danego uczestnika biorącego udział w konkretnym warsztacie.

- WorkshopID - identyfikator warsztatu
- ParticipantID - identyfikator uczestnika

```
CREATE TABLE [dbo].[WorkshopParticipants](
      [WorkshopID] [int] NOT NULL,
      [ParticipantID] [int] NOT NULL,
 CONSTRAINT [PK_WorkshopParticipants] PRIMARY KEY CLUSTERED
(
      [WorkshopID] ASC,
      [ParticipantID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WorkshopParticipants]  WITH CHECK ADD  CONSTRAINT
[FK_WorkshopParticipants_Participants] FOREIGN KEY([ParticipantID])
REFERENCES [dbo].[Participants] ([ParticipantID])
GO

ALTER TABLE [dbo].[WorkshopParticipants] CHECK CONSTRAINT
[FK_WorkshopParticipants_Participants]
GO

ALTER TABLE [dbo].[WorkshopParticipants]  WITH CHECK ADD  CONSTRAINT
[FK_WorkshopParticipants_Workshop] FOREIGN KEY([WorkshopID])
REFERENCES [dbo].[Workshop] ([WorkshopID])
GO

ALTER TABLE [dbo].[WorkshopParticipants] CHECK CONSTRAINT
[FK_WorkshopParticipants_Workshop]
GO
```

Tabela WorkshopReservations

Tabela przechowująca rezerwacje na konkretny warsztat. Zawiera przede wszystkimliczbę rezerwacji na konkretny warsztat.

- WorkshopReservationID - identyfikator rezerwacji warsztatu
- WorkshopID - identyfikator warsztatu
- DayReservationID - identyfikator dnia konfernecji
- NumberOfParticipants - liczba uczestników w danym warsztacie (dokonanych rezerwacji)
- NumberOfStudents- liczba studentów w danym warsztacie

```
CREATE TABLE [dbo].[WorkshopReservations](
      [WorkshopReservationID] [int] IDENTITY(1,1) NOT NULL,
      [WorkshopID] [int] NOT NULL,
      [ReservationDayID] [int] NOT NULL,
      [NumberOfParticipants] [smallint] NOT NULL,
      [NumberOfStudents] [smallint] NULL,
 CONSTRAINT [PK_WorkshopReservations] PRIMARY KEY CLUSTERED
(
      [WorkshopReservationID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WorkshopReservations] ADD  CONSTRAINT [df_WsRes]
DEFAULT ((0)) FOR [NumberOfStudents]
GO

ALTER TABLE [dbo].[WorkshopReservations]  WITH CHECK ADD  CONSTRAINT
[FK_WorkshopReservations_ReservationDay] FOREIGN KEY([ReservationDayID])
REFERENCES [dbo].[ReservationDay] ([ReservationDayID])
GO

ALTER TABLE [dbo].[WorkshopReservations] CHECK CONSTRAINT
[FK_WorkshopReservations_ReservationDay]
GO

ALTER TABLE [dbo].[WorkshopReservations]  WITH CHECK ADD  CONSTRAINT
[FK_WorkshopReservations_Workshop] FOREIGN KEY([WorkshopID])
REFERENCES [dbo].[Workshop] ([WorkshopID])
GO
```

```
ALTER TABLE [dbo].[WorkshopReservations] CHECK CONSTRAINT
[FK_WorkshopReservations_Workshop]
GO

ALTER TABLE [dbo].[WorkshopReservations]  WITH CHECK ADD  CONSTRAINT
[CK_WorkshopReservations] CHECK  (([NumberOfParticipants]>(0)))
GO

ALTER TABLE [dbo].[WorkshopReservations] CHECK CONSTRAINT
[CK_WorkshopReservations]
GO
```

Tabela WorkshopDictionary

Tabela ta zawiera nazwę oraz opis warsztatu.

- WorkshopID - identyfikator warsztatu
- WorkshopName - nazwa warsztatu
- Description - opis tekstowy warsztatu

```
CREATE TABLE [dbo].[WorkshopsDictionary](
      [WorkshopID] [int] NOT NULL,
      [WorkshopName] [varchar](100) NOT NULL,
      [Description] [varchar](1000) NULL,
 CONSTRAINT [PK_WorkshopsDictionary] PRIMARY KEY CLUSTERED
(
      [WorkshopID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[WorkshopsDictionary]  WITH CHECK ADD  CONSTRAINT
[FK_WorkshopsDictionary_Workshop] FOREIGN KEY([WorkshopID])
REFERENCES [dbo].[Workshop] ([WorkshopID])
GO

ALTER TABLE [dbo].[WorkshopsDictionary] CHECK CONSTRAINT
[FK_WorkshopsDictionary_Workshop]
GO
```

4. Warunki Integralnościowe

Tabela Conferences

Jeśli zniżka nie została podana to przyhjmujemy ją jako zero
DEFAULT ((0)) FOR [DiscountForStudents]

Konferencja nie może zacząć się później niż się kończy
CHECK (([StartDate]<=[EndDate]))

Cena musi być nieujemna
CHECK (([DayPrice]>=(0)))

Ilość miejsc konferencji musi być większa od zera
CHECK (([MaxParticipants]>(0)))

Wartość zniżki studenckiej musi mieścić się w przedziale [0,100%)
CHECK (([DiscountForStudents]<(1) AND [DiscountForStudents]>=(0)))

Tabela Clients

W skład maila musi wchodzić @(małpka), po niej conajmniej 1 znak oraz .
CHECK (([Mail] like '%@_%.%'))
Numer telefonu składa się z 9 cyfr
CHECK (([PhoneNumber] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
Mail musi być unikalny
Unique Mail
Numer telefonu musi być unikalny
Unique PhoneNumber

Tabela Companies

NIP musi składać się z 10 cyfr
CHECK (([NIP] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
ID Firmy musi być unikatowe dla firmy
Unique CompanyID
Nazwa firmy jest unikatowa
Unique CompanyName
Numer musi być unikalny dla każdej firmy

Unique NIP

Tabela Discount

Zniżka musi przyjmować wartości z przedziału [0;100%)

CHECK([Discount]<(1) AND [Discount]>=(0))

Znizka musi zostać opłacona dodatnią ilość dnia przed startem

CHECK (([DaysBeforeStart]>(0)))

Standardowo zniżka przyjmuje wartość 0

DEFAULT ((0)) FOR [Discount]

Tabela ReservationDay

Ilość zajmowanych miejsc jest nieujemna

CHECK (([NumberOfParticipants]>(0)))

Jeśli nie zostali zgłoszeni to przyjmujemy, że nie ma studentów

DEFAULT ((0)) FOR [NumberOfStudents]

Tabela Reservations

Rezerwacja przy tworzeniu jest ustawiana na "nieanulowaną"

DEFAULT ((0)) FOR [isCancelled]

Tabela Students

Każdy student ma unikalny numer karty studenckiej

Unique StudentCard

Tabela Workshop

Defaultowa cena warsztatu wynosi 0

DEFAULT ((0)) FOR [Price]

Ilość miejsc jest dodatnia

CHECK (([LimitOfParticipants]>(0)))

Warsztaty nie mogą kończyć się przed tym jak się zaczynały

CHECK (([StartTime]<[EndTime]))

Cena jest liczbą nieujemną

CHECK (([Price]>=(0)))

Tabela WorkshopReservations

Jeśli nie zostali zgłoszeni to przyjmujemy, że nie ma studentów
DEFAULT ((0)) FOR [NumberOfStudents]
Nie można złożyć rezerwacji na 0 miejsc
CHECK (([NumberOfParticipants]>(0)))

5. Widoki

Widok viewAllCompanies

Widok pokazujący rekordy wszystkich klientów firmowych.

```
CREATE VIEW [dbo].[viewAllCompanies]
AS
	SELECT
		Clients.ClientID, Companies.CompanyName, Companies.Address,
		Companies.Country, Companies.City, Companies.PostalCode,
		Clients.Mail, Clients.PhoneNumber, Companies.NIP
	FROM Clients
	JOIN Companies ON Clients.ClientID = Companies.ClientID
GO
```

Widok viewAllClients

Widok pokazujący wszystkich klientów.

```
CREATE VIEW [dbo].[viewAllClients]
AS
	SELECT Clients.ClientID, Person.FirstName + Person.LastName as
"Name", Clients.Mail, Clients.PhoneNumber, 'Private Client' as 'Status'
	FROM Clients
	JOIN PrivateClients ON Clients.ClientID = PrivateClients.ClientID
	JOIN Person ON PrivateClients.PersonID = Person.PersonID
	UNION
	SELECT Clients.ClientID, Companies.CompanyName as "Name",
Clients.Mail, Clients.PhoneNumber, 'Company' as 'Status'
	FROM Clients
	JOIN Companies ON Clients.ClientID = Companies.ClientID
GO
```

Widok viewAllPrivateClients

Widok pokazujący wszystkich indywidualnych klientów.

```
CREATE VIEW [dbo].[viewAllPrivateClients]
AS
      SELECT Clients.ClientID, Person.FirstName, Person.LastName,
Clients.Mail, Clients.PhoneNumber
      FROM Clients
      JOIN PrivateClients ON Clients.ClientID = PrivateClients.ClientID
      JOIN Person ON PrivateClients.PersonID = Person.PersonID
GO
```

Widok viewAllCancelledConferences

Widok pokazujący wszystkie odwołane konferencje

```
create view [dbo].[viewCancelledConferences]
as
SELECT Conferences.ConferenceID, Conferences.ConferenceName,
Conferences.StartDate, Conferences.EndDate
FROM Conferences
inner join ConferenceDay ON
ConferenceDay.ConferenceID=Conferences.ConferenceID
inner join ReservationDay ON
ConferenceDay.ConferenceDayID=ReservationDay.ConferenceDayID
inner join Reservations ON
Reservations.ReservationID=ReservationDay.ReservationID
where Reservations.isCancelled=1;
GO
```

Widok viewClientswithNumberofReservations

Widok pokazujący wszystkie klientów i ilość dokonanych przez nich rezerwacji

```
CREATE VIEW [dbo].[viewClientswithNumberofReservations]
AS
SELECT Clients.ClientID, Companies.CompanyName AS 'Name',
(SELECT count(*) FROM Reservations
WHERE (Clients.ClientID = Reservations.ClientID AND
Reservations.isCancelled = 0)) AS 'Number of reservations'
FROM Clients INNER JOIN
Companies ON Clients.ClientID = Companies.ClientID
UNION
SELECT Clients.ClientID, FirstName + ' ' + LastName AS 'Name',
(SELECT count(*) FROM Reservations
WHERE (Clients.ClientID = Reservations.ClientID AND
Reservations.isCancelled = 0)) AS 'Number of reservations'
FROM Clients INNER JOIN
PrivateClients ON PrivateClients.ClientID = Clients.ClientID INNER JOIN
Person ON PrivateClients.PersonID = Person.PersonID
GO
```

Widok viewTopClients

Widok pokazujący wszystkie klientów, którzy najczęśniej dokonywali rezerwacji

```
CREATE VIEW [dbo].[viewTopClients]
AS
SELECT TOP 10 *
FROM viewClientswithNumberofReservations
ORDER BY 'Number of reservations' DESC
GO
```

Widok viewConferencesPrices

Widok pokazujący wszystkie ceny konferencji z uwzględnionymi progami

```
create view [dbo].[viewConferencesPrices]
as
select top (100) PERCENT Conferences.ConferenceID,
Conferences.ConferenceName, Discount.DaysBeforeStart, Conferences.DayPrice
as 'Price without discount', Discount.DiscountID, Discount.Discount,
Conferences.DayPrice * (1 - Discount) as 'Price'
from Discount
JOIN Conferences ON Discount.ConferenceID = Conferences.ConferenceID
```

```
order by Conferences.ConferenceID, Discount.Discount
GO
```

Widok viewFreeAndBookedPlaces

Widok pokazujący wszystkich ile jest wolnych, a ile zajętych miejsc konferencji

```
CREATE VIEW [dbo].[viewFreeAndBookedPlaces]
AS
      SELECT ConferenceDay.ConferenceDayID, Conferences.ConferenceID,
Conferences.MaxParticipants,
      ISNULL(SUM(ReservationDay.NumberOfParticipants), 0) as 'BOOKED',
      Conferences.MaxParticipants -
ISNULL(SUM(ReservationDay.NumberOfParticipants), 0) AS 'FREE'
      FROM
      ConferenceDay
      JOIN Conferences ON
ConferenceDay.ConferenceID=Conferences.ConferenceID
      LEFT JOIN ReservationDay ON
ReservationDay.ConferenceDayID=ReservationDay.ConferenceDayID
      JOIN Reservations on
Reservations.ReservationID=ReservationDay.ReservationID and
Reservations.isCancelled=0
      GROUP BY ConferenceDay.ConferenceDayID, Conferences.ConferenceID,
Conferences.MaxParticipants
GO
```

Widok viewFreeAndBookedPlacesAtWorkshops

Widok pokazujący wszystkich ile jest wolnych, a ile zajętych miejsc warsztatu

```
CREATE VIEW [dbo].[viewFreeAndBookedPlacesAtWorkshops]
AS
SELECT Workshop.WorkshopID, Workshop.ConferenceDayID,
WorkshopsDictionary.WorkshopName, Workshop.LimitOfParticipants,
ISNULL(sum(WorkshopReservations.NumberOfParticipants),0) AS 'BOOKED
PLACES',
Workshop.LimitOfParticipants-ISNULL(sum(WorkshopReservations.NumberOfPartic
ipants),0) AS 'FREE PLACES'
FROM Workshop
```

```
inner join WorkshopsDictionary on
Workshop.WorkshopID=WorkshopsDictionary.WorkshopID
inner join WorkshopReservations on
WorkshopReservations.WorkshopID=Workshop.WorkshopID
GROUP BY Workshop.WorkshopID, Workshop.ConferenceDayID,
WorkshopsDictionary.WorkshopName, Workshop.LimitOfParticipants
GO
```

Widok viewInformationForIdentifiers

Widok pokazujący dane do identyfikatorów uczestników

```
CREATE VIEW [dbo].[viewInformationForIdentifiers]
AS
      SELECT Conferences.ConferenceID, Conferences.ConferenceName,
ParticipantID, FirstName, LastName, '' AS Company
      FROM Participants
      JOIN Person ON Participants.PersonID=Person.PersonID
      JOIN ReservationDay ON
Participants.ReservationDayID=ReservationDay.ReservationDayID
      JOIN Reservations ON
ReservationDay.ReservationID=Reservations.ReservationID and isCancelled=0
      JOIN ConferenceDay ON
ReservationDay.ConferenceDayID=ConferenceDay.ConferenceDayID
      JOIN Conferences ON
Conferences.ConferenceID=ConferenceDay.ConferenceID
      UNION
      SELECT Conferences.ConferenceID, Conferences.ConferenceName,
ParticipantID, FirstName, LastName, CompanyName AS Company
      FROM Participants
      JOIN Person ON Participants.PersonID=Person.PersonID
      JOIN Employees ON Person.PersonID=Employees.PersonID
      JOIN Companies ON Companies.CompanyID=Employees.CompanyID
      JOIN ReservationDay ON
Participants.ReservationDayID=ReservationDay.ReservationDayID
      JOIN Reservations ON
ReservationDay.ReservationID=Reservations.ReservationID and isCancelled=0
      JOIN ConferenceDay on
ReservationDay.ConferenceDayID=ConferenceDay.ConferenceDayID
      JOIN Conferences ON
Conferences.ConferenceID=ConferenceDay.ConferenceID
GO
```

Widok viewIdentifiers

Widok pokazujący identyfikatory uczestników uczestników

```
CREATE VIEW [dbo].[viewIdentifiers]
AS
      SELECT ConferenceID, ConferenceName, ParticipantID , FirstName + ' '
+ LastName + ' ' + Company as Identifier
      FROM viewInformationForIdentifiers
GO
```

Widok viewListOfParticipants

Widok pokazujący liste uczestników dni konferencji

```
Widok viewListOfParticipants_Workshop
CREATE VIEW [dbo].[viewListOfParticipants]
AS
      SELECT DISTINCT ConferenceDay.ConferenceDayID,
Conferences.ConferenceName, ConferenceDay.Date,
      Person.FirstName + ' ' + Person.LastName AS 'Name'
      FROM ConferenceDay
      JOIN Conferences ON
Conferences.ConferenceID=ConferenceDay.ConferenceID
      JOIN ReservationDay ON
ReservationDay.ConferenceDayID=ConferenceDay.ConferenceDayID
      JOIN Participants ON
Participants.ReservationDayID=ReservationDay.ReservationDayID
      JOIN Person ON Person.PersonID=Participants.PersonID
GO
```

Widok viewListOfParticipants_Workshop

Widok pokazujący listę uczestników warsztatów

```
CREATE VIEW [dbo].[viewListParticipants_Workshop]
AS
      SELECT DISTINCT Workshop.WorkshopID,
WorkshopsDictionary.WorkshopName,
```

```
	Person.FirstName + ' ' + Person.LastName as 'Name'
	FROM Workshop
	JOIN WorkshopsDictionary ON Workshop.WorkshopID=Workshop.WorkshopID
	JOIN WorkshopParticipants ON
Workshop.WorkshopID=WorkshopParticipants.WorkshopID
	JOIN Participants ON
Participants.ParticipantID=WorkshopParticipants.ParticipantID
	JOIN Person ON Person.PersonID=Participants.PersonID
GO
```

Widok viewListOfStudentCards

Widok pokazujący liste legitymacji studenckich studentów, którzy będą na konferencji

```
CREATE VIEW [dbo].[viewListOfStudentCards]
AS
select distinct ConferenceDay.ConferenceDayID, Conferences.ConferenceName,
ConferenceDay.Date, Students.StudentCard
from ConferenceDay
inner join Conferences on
Conferences.ConferenceID=ConferenceDay.ConferenceID
inner join ReservationDay on
ReservationDay.ConferenceDayID=ConferenceDay.ConferenceDayID
inner join Participants on
Participants.ReservationDayID=ReservationDay.ReservationDayID
inner join Students on Participants.ParticipantID=Students.ParticipantID
GO
```

Widok viewListOfStudentCards_Workshop

Widok pokazujący liste legitymacji studenckich studentów, którzy będą na warsztatach

```
CREATE VIEW [dbo].[viewListOfStudentCards_Workshop]
AS
	SELECT Workshop.WorkshopID, WorkshopsDictionary.WorkshopName,
Students.StudentCard
	FROM Workshop
	JOIN WorkshopsDictionary ON Workshop.WorkshopID=Workshop.WorkshopID
	JOIN WorkshopParticipants ON
Workshop.WorkshopID=WorkshopParticipants.WorkshopID
	JOIN Participants ON
Participants.ParticipantID=WorkshopParticipants.ParticipantID
	JOIN Students ON Students.ParticipantID=Participants.ParticipantID
GO
```

Widok viewWorkshopsPopularity

Widok pokazujący popularność warsztatów

```
create view [dbo].[viewWorkshopsPopularity]
as
select Workshop.WorkshopID, WorkshopsDictionary.WorkshopName,
sum(dbo.F_CountTakenPlacesInWokrshop(WorkshopReservationID)) as 'Ilość
zajętych miejsc'
from Workshop
join WorkshopsDictionary on Workshop.WorkshopID =
WorkshopsDictionary.WorkshopID
join WorkshopReservations on WorkshopReservations.WorkshopID =
Workshop.WorkshopID
group by Workshop.WorkshopID, WorkshopsDictionary.WorkshopName
GO
```

Widok viewTopWorkshop

Widok pokazujący najpopularniejsze warsztaty

```
create view [dbo].[viewTopWorkshop]
as
select top 10 *
from viewWorkshopsPopularity
order by 3 DESC
GO
```

Widok viewConferencesPopularity

Widok pokazujący popularność konferencji

```
create view [dbo].[viewConferencesPopularity]
as
select Conferences.ConferenceID, Conferences.ConferenceName,
sum(dbo.F_CountTakenPlacesInReservartionDay(ReservationDayID)) as 'Ilość
zajętych miejsc'
from ConferenceDay
JOIN Conferences on ConferenceDay.ConferenceID = Conferences.ConferenceID
JOIN ReservationDay on ReservationDay.ConferenceDayID =
ConferenceDay.ConferenceDayID
group by Conferences.ConferenceID, Conferences.ConferenceName
GO
```

Widok viewTopConferences

Widok pokazujący najpopularniejsze konferencje

```
create view [dbo].[viewTopConferences]
as
select top 10 *
from viewConferencesPopularity
order by 3 desc
GO
```

Widok viewUpcomingConferences

Widok pokazujący dziesięć konferencji które odbędą się w najbliższym czasie

```
CREATE VIEW [dbo].[viewUpcomingConferences]
AS
SELECT TOP 10 *
FROM Conferences
WHERE (StartDate > GETDATE())
ORDER BY StartDate
GO
```

Widok viewClientsWhoNotFulfilledDataReservations

Widok pokazujący firmy które nie wypełniły wymaganych danych

```
create view [dbo].[viewClientsWhoNotFulfilledDataReservations]
as
select Reservations.ReservationID, Clients.ClientID, Companies.CompanyName,
Clients.Mail, Clients.PhoneNumber
from Reservations
join Clients on Clients.ClientID=Reservations.ClientID
join Companies on Clients.ClientID = Companies.CompanyID
join ReservationDay on
ReservationDay.ReservationID=Reservations.ReservationID
join Participants on
Participants.ReservationDayID=ReservationDay.ReservationDayID
join Person on Person.PersonID = Participants.PersonID
where (Person.FirstName IS NULL and Person.LastName IS NULL)
GO
```

Widok viewUnpaidReservations

Widok pokazujący nieopłacone rezerwacje

```
CREATE VIEW [dbo].[viewUnpaidReservations]
AS
SELECT Reservations.ReservationID, Person.FirstName + Person.LastName AS
Name, 'Private Client' AS ClientType, Clients.PhoneNumber, Clients.Mail
FROM PrivateClients
JOIN Clients ON Clients.ClientID = PrivateClients.ClientID
JOIN Reservations ON Reservations.ClientID = Clients.ClientID
JOIN Person ON PrivateClients.PersonID = Person.PersonID
WHERE Reservations.PaymentDate is NULL
UNION
SELECT Reservations.ReservationID, Companies.CompanyName AS Name, 'Company'
AS ClientType, Clients.PhoneNumber, Clients.Mail
FROM Companies
JOIN Clients ON Clients.ClientID = Companies.ClientID
JOIN Reservations ON Reservations.ClientID = Clients.ClientID
WHERE Reservations.PaymentDate is NULL
GO
```

6. Procedury

Procedura P_Add_Client

Procedura dodająca klienta

```
CREATE PROCEDURE [dbo].[P_Add_Client]
@PhoneNumber varchar(50),
@Mail varchar(50),
@ClientID int OUTPUT

AS
BEGIN
	SET NOCOUNT ON

		INSERT INTO Clients(
		PhoneNumber,
		Mail
		)
		VALUES(
		@PhoneNumber,
		@Mail
		)
		SET @ClientID = @@IDENTITY
END
GO
```

Procedura P_Add_CompanyClient

Procedura dodaje klienta firmowego.

```
CREATE PROCEDURE [dbo].[P_Add_CompanyClient]
@CompanyName varchar(50),
@NIP int,
@Country varchar(50),
@City varchar(50),
@PostalCode varchar(50),
@Address varchar(50),
@PhoneNumber varchar(50),
@Mail varchar(50),
@ClientID int OUT

AS
BEGIN
	SET NOCOUNT ON
```

```
                        --DECLARE @ClientID int
                        EXECUTE  P_Add_Client @PhoneNumber, @Mail,
@ClientID = @ClientID OUT
                  INSERT INTO Companies(
                  ClientID,
                  CompanyName,
                  Address,
                  City,
                  NIP,
                  Country,
                  PostalCode
                  )
                  VALUES(
                  @ClientID,
                  @CompanyName,
                  @Address,
                  @City,
                  @NIP,
                  @Country,
                  @PostalCode
                  )
END
GO
```

Procedura P_Add_Conference

Procedura umożliwia dodanie konferencji

```
CREATE PROCEDURE [dbo].[P_Add_Conference]
@ConferenceName varchar(50),
@DiscountForStudents real,
@DayPrice money,
@MaxParticipants int,
@StartDate date,
@EndDate date,
@ConferenceID int OUTPUT
AS
BEGIN

      SET NOCOUNT ON

            INSERT INTO Conferences(
                  ConferenceName,
                  DiscountForStudents,
                  DayPrice,
                  MaxParticipants,
                  StartDate,
                  EndDate
```

```
                  )
                  VALUES(
                  @ConferenceName,
                  @DiscountForStudents,
                  @DayPrice,
                  @MaxParticipants,
                  @StartDate,
                  @EndDate

                  )
                  SET @ConferenceID = @@IDENTITY

                  DECLARE @i date = @StartDate

                  WHILE @i <= @EndDate
                  BEGIN
                        EXECUTE P_add_ConferenceDay @i,  @ConferenceID
                        SET @i = DATEADD(Day, 1, @i);
                  END
END
GO
```

Procedura P_Add_ConferenceDay

Procedura umożliwia dodanie dnia konferencji

```
CREATE PROCEDURE [dbo].[P_Add_ConferenceDay]
@Date date,
@ConferenceID int

AS
BEGIN

      SET NOCOUNT ON
            IF NOT EXISTS(
            SELECT * FROM Conferences
            WHERE Conferences.ConferenceID = @ConferenceID
            )
            BEGIN;
            THROW 50004, 'Nie ma konferencji o tym ID',1
            END
            IF EXISTS(
            SELECT * FROM ConferenceDay
            WHERE ConferenceDay.ConferenceID = @ConferenceID AND
ConferenceDay.Date = @Date)
            BEGIN;
            THROW 50004, 'Jest juz zarejestrowany dzien tej konferencji na
ta date', 1
```

```
            END
            INSERT INTO ConferenceDay(
                  Date,
                  ConferenceID
                  )
                  VALUES(
                  @Date,
                  @ConferenceID
                  )

END
GO
```

Procedura P_Add_Discount

Procedura dodająca zniżkę

```
CREATE PROCEDURE [dbo].[P_Add_Discount]
@ConferenceID int,
@Discount real,
@DaysBeforeStart smallint
AS
BEGIN
      SET NOCOUNT ON
      BEGIN TRY
      IF NOT EXISTS(
            SELECT * FROM Conferences
            WHERE ConferenceID = @ConferenceID
            )
            BEGIN;
            THROW 50004, 'Nie ma konferencji o podanym ID', 1
            END
      IF EXISTS(
            SELECT * FROM Discount
                  WHERE ConferenceID = @ConferenceID
                  AND DaysBeforeStart = @DaysBeforeStart
            )
            BEGIN;
            THROW 50004, 'Termin znizki dla tej konferencji zostal juz
zajety', 1
            END
            INSERT INTO Discount(
                  ConferenceID,
                  Discount,
                  DaysBeforeStart
                  )
                  VALUES(
                  @ConferenceID,
```

```
                  @Discount,
                  @DaysBeforeStart
                  )
      END TRY
      BEGIN CATCH
            DECLARE @ERROR varchar(50) = 'Nie udalo sie dodac znizki';
            THROW 50004, @ERROR, 1
      END CATCH
END
GO
```

Procedura P_Add_Participant

Procedura dodająca uczestnika

```
CREATE PROCEDURE [dbo].[P_Add_Participant](
@FirstName varchar(50),
@LastName varchar(50),
@ReservationDayID int,
@ParticipantID int OUTPUT
)
AS
BEGIN
      SET NOCOUNT ON

            DECLARE @PersonID int

            EXECUTE P_Add_Person @FirstName, @LastName, @PersonID =
@PersonID OUT

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

Procedura P_Add_Person

Procedura dodająca osobę

```
CREATE PROCEDURE [dbo].[P_Add_Person]
@FirstName varchar(50),
@LastName varchar(50),
@PersonID int OUTPUT

AS
BEGIN
      SET NOCOUNT ON

            INSERT INTO Person(
            FirstName,
            LastName
            )
            VALUES(
            @FirstName,
            @LastName
            )
            SET @PersonID = @@IDENTITY


END
GO
```

Procedura P_Add_PrivateClient

Procedura dodająca klienta indywidualnego

```
CREATE PROCEDURE [dbo].[P_Add_PrivateClient]
@PhoneNumber varchar(50),
@Mail varchar(50),
@FirstName varchar(50),
@LastName varchar(50),
@ClientID int OUT,
@PersonID int OUT

AS
BEGIN
      SET NOCOUNT ON

                  EXECUTE P_Add_Client @PhoneNumber, @Mail, @ClientID =
@ClientID OUT
```

```
                EXECUTE P_Add_Person @FirstName, @LastName, @PersonID =
@PersonID OUT
            --Przypisanie wartości odpowiednim kolumnom w PrivateClients
            INSERT INTO PrivateClients(
            ClientID,
            PersonID
            )
            VALUES(
            @ClientID,
            @PersonID
            )

END
GO
```

Procedura P_Add_Reservation

Procedura dodająca rezerwację

```
CREATE PROCEDURE [dbo].[P_Add_Reservation]
@ClientID int,
@ReservationID int OUTPUT
AS
BEGIN
      SET NOCOUNT ON
      DECLARE @ReservationDate date
      SET @ReservationDate = GETDATE();
      IF NOT EXISTS(
            SELECT * FROM Clients
            WHERE @ClientID = ClientID
            )
            BEGIN;
            THROW 50004, 'Nie ma takiego klienta',1;
            END
            INSERT INTO Reservations(
            ClientID,
            isCancelled,
            ReservationDate
                  )
                  VALUES(
            @ClientID,
            0,
            @ReservationDate
                  )
            SET @ReservationID = @@IDENTITY
END
GO
```

Procedura P_Add_ReservationDay

Procedura dodająca dzień rezerwacji

```
CREATE PROCEDURE [dbo].[P_Add_ReservationDay]
@ReservationID int,
@ConferenceDayID int,
@NumberOfParticipants int,
@NumberOfStudents int,
@ReservationDayID int OUTPUT
AS
BEGIN
	SET NOCOUNT ON
	IF NOT EXISTS(
		SELECT * FROM ConferenceDay
		WHERE @ConferenceDayID = ConferenceDayID
		)
		BEGIN;
		DECLARE @ERROR1 varchar(50) = 'Nie ma dnia konferencji z tym
ID';
		THROW 50001, @ERROR1, 1
		END
	IF NOT EXISTS(
		SELECT * FROM Reservations
		WHERE @ReservationID = ReservationID
		)
		BEGIN;
		DECLARE @ERROR2 varchar(50) = 'Nie ma rezerwacji z tym ID';
		THROW 50002, @ERROR2, 1
		END

		INSERT INTO ReservationDay(
		ReservationID,
		ConferenceDayID,
		NumberOfParticipants,
		NumberOfStudents
			)
			VALUES(
		@ReservationID,
		@ConferenceDayID,
		@NumberOfParticipants,
		@NumberOfStudents
			)
		SET @ReservationDayID = @@IDENTITY

END
```

```
GO
```

Procedura P_Add_StudentParticipant

Procedura dodająca uczestnika będącego studentem

```
CREATE PROCEDURE [dbo].[P_Add_StudentParticipant]
@FirstName varchar(50),
@LastName varchar(50),
@DayReservationID int,
@StudentCard varchar(50),
@ParticipantID int OUT
AS
BEGIN
      SET NOCOUNT ON
                  EXECUTE P_Add_Participant  @FirstName, @LastName,
@DayReservationID, @ParticipantID = @ParticipantID OUT

            --Przypisanie wartości odpowiednim kolumnom w Participants
            INSERT INTO Students(
            StudentCard,
            ReservationDayID,
            ParticipantID
            )
            VALUES(
            @StudentCard,
            @DayReservationID,
            @ParticipantID
            )

END
GO
```

Procedura P_Add_Workshop

Procedura dodająca warsztat.

```
CREATE PROCEDURE [dbo].[P_Add_Workshop]
@ConferenceDayID int,
@StartTime time,
@EndTime time,
@Price money,
@LimitOfParticipants smallint,
```

```
@WorkshopName nvarchar(50),
@Description nvarchar(50),
@WorkshopID int OUTPUT
AS
BEGIN
	SET NOCOUNT ON
		INSERT INTO Workshop(
			ConferenceDayID,
			StartTime,
			EndTime,
			Price,
			LimitOfParticipants
			)
			VALUES(
			@ConferenceDayID,
			@StartTime,
			@EndTime,
			@Price,
			@LimitOfParticipants
			)
			SET @WorkshopID = @@IDENTITY
			EXECUTE P_Add_WorkshopInfo @WorkshopName, @Description,
@WorkshopID

END
GO
```

Procedura P_Add_WorkshopInfo

Procedura dodająca słownik warsztatu

```
CREATE PROCEDURE [dbo].[P_Add_WorkshopInfo]
@WorkshopName varchar(50),
@Description varchar(50),
@WorkshopID int
AS
BEGIN
	SET NOCOUNT ON
		INSERT INTO WorkshopsDictionary(
			WorkshopName,
			Description,
			WorkshopID
			)
			VALUES(
			@WorkshopName,
			@Description,
			@WorkshopID
			)
```

```
END
GO
```

Procedura P_Add_WorkshopParticipant

Procedura dodająca uczestnika warsztatu

```
CREATE PROCEDURE [dbo].[P_Add_WorkshopParticipant]
@WorkshopID int,
@ParticipantID int
AS
BEGIN
      SET NOCOUNT ON

            INSERT INTO WorkshopParticipants(
            WorkshopID,
            ParticipantID
            )
            VALUES(
            @WorkshopID,
            @ParticipantID
            )

END
GO
```

Procedura P_Add_WorkshopReservation

Procedura dodająca rezerwacje warsztatu

```
CREATE PROCEDURE [dbo].[P_Add_WorkshopReservation]
@WorkshopID int,
@DayReservationID int,
@NumberOfParticipants int,
@NumberOfStudents int
AS
BEGIN
      SET NOCOUNT ON
      IF NOT EXISTS(
            SELECT * FROM Workshop
```

```
            WHERE @WorkshopID = WorkshopID
            )
            BEGIN;
            THROW 50004, 'Nie istnieje warsztat o tym ID', 1
            END
      IF NOT EXISTS(
            SELECT * FROM ReservationDay
            WHERE @DayReservationID = ReservationDayID
            )
            BEGIN;
            THROW 50004, 'Nie istnieje taki dzien rezerwacji', 1
            END

            INSERT INTO WorkshopReservations(
                  WorkshopID,
                  ReservationDayID,
                  NumberOfParticipants,
                  NumberOfStudents
                  )
                  VALUES(
                  @WorkshopID,
                  @DayReservationID,
                  @NumberOfParticipants,
                  @NumberOfStudents
                  )

END
GO
```

Procedura P_Cancel_Reservation

Procedura anuluje rezerwację

```
CREATE PROCEDURE [dbo].[P_Cancel_Reservation]
@ReservationID int
AS
BEGIN

      SET NOCOUNT ON

      IF NOT EXISTS(
            SELECT * FROM Reservations
            WHERE @ReservationID = Reservations.ReservationID
            )
            BEGIN;
            THROW 50004, 'Nie ma takiej rezerwacji',1;
            END

            UPDATE Reservations
```

```
            SET isCancelled = 0
END
GO
```

Procedura P_Change_Workshop_Description

Procedura umożliwia zmianę opisu warsztatu

```
CREATE PROCEDURE [dbo].[P_Change_Workshop_Description]
@WorkshopID int,
@NewDescription nvarchar(50)
AS
BEGIN
      SET NOCOUNT ON
            IF NOT EXISTS(
            SELECT * FROM WorkshopsDictionary
            WHERE WorkshopsDictionary.WorkshopID = @WorkshopID
            )
            BEGIN;
            THROW 50004, 'Nie ma Workshopu o tym ID', 1
            END

            UPDATE WorkshopsDictionary
            SET Description = @NewDescription
            WHERE WorkshopsDictionary.WorkshopID = @WorkshopID

END
GO
```

Procedura P_Change_Workshop_Limit

Procedura umożliwia zmianę limitu uczestników warsztatu

```
CREATE PROCEDURE [dbo].[P_Change_Workshop_Limit]
@WorkshopID int,
@NewLimit smallint
AS
BEGIN
      SET NOCOUNT ON
            IF NOT EXISTS(
            SELECT * FROM Workshop
            WHERE Workshop.WorkshopID = @WorkshopID
            )
            BEGIN;
            THROW 50004, 'Nie ma Workshopu o tym ID', 1
            END
            IF @NewLimit <= 0
```

```
            BEGIN;
            THROW 50004, 'Ujemna ilosc miejsc', 1
            END

            DECLARE @Taken int
            SET @Taken = dbo.F_WorkshopSlotsTaken(@WorkshopID)
            IF @Taken > @NewLimit
            BEGIN;
            THROW 50004, 'Zajeto wiecej miejsc niz nowy limit', 1
            END

            UPDATE Workshop
            SET LimitOfParticipants = @NewLimit
            WHERE Workshop.WorkshopID = @WorkshopID

END
GO
```

Procedura P_Delete_Conference

Procedura umożliwia usunięcie konferencji

```
CREATE PROCEDURE [dbo].[P_Delete_Conference]

@ConferenceID int

AS

BEGIN

      SET NOCOUNT ON

            IF NOT EXISTS(

            SELECT * from Conferences

            where Conferences.ConferenceID = @ConferenceID

            )

            BEGIN;

            THROW 50005, 'Nie ma konferencji o tym ID', 1

            END

            DELETE FROM Conferences

            where Conferences.ConferenceID = @ConferenceID

            DECLARE @i date = dbo.F_GetConferenceStartDate (@ConferenceID)
```

```
            DECLARE @end date = dbo.F_GetConferenceEndDate (@ConferenceID)

                  WHILE @i <= @end

                  BEGIN

                        DECLARE @confid int = dbo.F_GetDayID
(@ConferenceID, @i)

                        EXECUTE P_Delete_ConferenceDay @confid

                        SET @i = DATEADD(Day, 1, @i);

                  END

            DELETE FROM Conferences

            WHERE Conferences.ConferenceID = @ConferenceID


END

GO
```

Procedura P_Delete_ConferenceDay

Procedura usuwa dzień konferencji

```
CREATE PROCEDURE [dbo].[P_Delete_ConferenceDay]
@ConferenceDayID int

AS
BEGIN

      SET NOCOUNT ON
            IF NOT EXISTS(
            SELECT * FROM ConferenceDay
            WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID
            )
            BEGIN;
            THROW 50004, 'Nie ma dnia konferencji o tym ID',1
            END
            IF EXISTS(
            SELECT * From ReservationDay
            WHERE ReservationDay.ConferenceDayID = @ConferenceDayID
            )
            BEGIN;
            THROW 50004, 'Jest rezerwacja na ten dzien, wiec nie mozna
usunac',1
            END
```

```
            DELETE FROM ConferenceDay
            WHERE ConferenceDay.ConferenceDayID = @ConferenceDayID
END
GO
```

Procedura P_Delete_Discount

Procedura usuwa próg zniżki

```
CREATE PROCEDURE [dbo].[P_Delete_Discount]
@DiscountID int
AS
BEGIN
      SET NOCOUNT ON

      IF NOT EXISTS(
            SELECT * FROM Discount
            WHERE DiscountID = @DiscountID
            )
            BEGIN;
            THROW 50004, 'Nie ma znizki o podanym ID', 1
            END

            DELETE FROM Discount
            WHERE Discount.DiscountID = @DiscountID

END
GO
```

Procedura P_Delete_ReservationDay

Procedura usuwa rezerwacje dnia

```
CREATE PROCEDURE [dbo].[P_Delete_ReservationDay]
@ReservationDayID int
AS
BEGIN
      SET NOCOUNT ON

      IF NOT EXISTS(
            SELECT * FROM ReservationDay
            WHERE @ReservationDayID = ReservationDayID
            )
            BEGIN;
            DECLARE @ERROR2 varchar(50) = 'Nie ma rezerwacji na dzien z tym
ID';
            THROW 50002, @ERROR2, 1
            END
```

```
            DELETE FROM ReservationDay
            Where ReservationDayID = @ReservationDayID
END
GO
```

Procedura P_DeleteCancelled_Reservaions

Procedura usuwa rezerwacje, które zostały odwołane

```
CREATE PROCEDURE [dbo].[P_DeleteCancelled_Reservations]

AS
BEGIN
      SET NOCOUNT ON

      DELETE FROM Reservations
      WHERE Reservations.isCancelled = 1
END
GO
```

Procedura P_Update_Client_MailAndPhone

Procedura umożliwia aktualizację maila i telefonu klienta

```
CREATE PROCEDURE [dbo].[P_Update_Client_MailAndPhone]
@PhoneNumber varchar(50),
@Mail varchar(50),
@ClientID int
AS
BEGIN
      SET NOCOUNT ON

            IF NOT EXISTS(
            SELECT * From Clients
            WHERE Clients.ClientID = @ClientID
            )
            BEGIN;
            THROW 50001, 'Nie ma klienta o tym ID',1
            END

            UPDATE Clients
            SET Mail = @Mail,
            PhoneNumber = @PhoneNumber
            WHERE Clients.ClientID = @ClientID
END
GO
```

Procedura P_Update_Conference_Data

Procedura umożliwia zmianę informacji o konferencji

```
CREATE PROCEDURE [dbo].[P_Update_Conference_Data]
@ConferenceName varchar(50),
@DiscountForStudents real,
@DayPrice money,
@MaxParticipants int,
@StartDate date,
@EndDate date,
@ConferenceID int
AS
BEGIN

      SET NOCOUNT ON

            IF NOT EXISTS(
            SELECT * from Conferences
            where Conferences.ConferenceID = @ConferenceID
            )
            BEGIN;
            THROW 50005, 'Nie ma konferencji o tym ID', 1
            END
            INSERT INTO Conferences(
                  ConferenceName,
                  DiscountForStudents,
                  DayPrice,
                  MaxParticipants,
                  StartDate,
                  EndDate
                  )
                  VALUES(
                  @ConferenceName,
                  @DiscountForStudents,
                  @DayPrice,
                  @MaxParticipants,
                  @StartDate,
                  @EndDate
                  )

END
GO
```

Procedura P_Update_Discount

Procedura umożliwia zmianę progów zniżek

```
CREATE PROCEDURE [dbo].[P_Update_Discount]
@ConferenceID int,
@Discount real,
@DaysBeforeStart smallint,
@DiscountID int
AS
BEGIN
	SET NOCOUNT ON
	BEGIN TRY
	IF NOT EXISTS(
		SELECT * FROM Conferences
		WHERE ConferenceID = @ConferenceID
		)
		BEGIN;
		THROW 50004, 'Nie ma konferencji o podanym ID', 1
		END
	IF NOT EXISTS(
		SELECT * FROM Discount
		WHERE DiscountID = @DiscountID
		)
		BEGIN;
		THROW 50004, 'Nie ma znizki o podanym ID', 1
		END

		UPDATE Discount
		SET Discount = @Discount,
		DaysBeforeStart = @DaysBeforeStart
		WHERE Discount.DiscountID = @DiscountID
	END TRY
	BEGIN CATCH
		DECLARE @ERROR varchar(50) = 'Nie udalo sie zmienic znizki';
		THROW 50004, @ERROR, 1
	END CATCH
END
GO
```

Procedura P_Update_Participant_Data

Procedura umożliwia zmianę danych uczestnika

```
CREATE PROCEDURE [dbo].[P_Update_Participant_Data](
@FirstName varchar(50),
@LastName varchar(50),
@ReservationDayID int,
@ParticipantID int
)
AS
BEGIN
	SET NOCOUNT ON
```

```
            IF NOT EXISTS(
            SELECT * FROM Participants
            WHERE Participants.ParticipantID = @ParticipantID
            )
            BEGIN;
            THROW 50004, 'Nie ma takiego participanta',1
            END


            DECLARE @ThisPersonID int =
            (SELECT  PersonID FROM Participants
            WHERE Participants.ParticipantID = @ParticipantID
            )

            UPDATE Person
            SET FirstName = @FirstName,
            LastName = @LastName
            WHERE Person.PersonID = @ThisPersonID
END
GO
```

7. Triggery

Trigger checkIfDiscountsAreGettingLower

Trigger sprawdza czy

```
CREATE TRIGGER [dbo].[checkIfDiscountsAreGettingLower] ON [dbo].[Discount]
AFTER INSERT, UPDATE
AS
BEGIN

	IF EXISTS( SELECT inserted.discountID from inserted Join Discount on
Discount.ConferenceID = inserted.ConferenceID
	WHERE( inserted.DaysBeforeStart < Discount.DaysBeforeStart AND
inserted.Discount > Discount.Discount) OR
	(inserted.DaysBeforeStart > Discount.DaysBeforeStart AND
inserted.Discount < Discount.Discount))
	BEGIN;
	THROW 50004, 'Nie spelnia malejacych wymagan progow', 1
	END
END
GO

ALTER TABLE [dbo].[Discount] ENABLE TRIGGER
[checkIfDiscountsAreGettingLower]
GO
```

Trigger checkIfWorkshopHasMoreSlotsThanConference

Trigger uniemożliwia ustawienie limitu uczestników warsztatu większego niż konferencji

```
CREATE TRIGGER [dbo].[checkIfWorkshopHasMoreSlotsThanConference] ON
[dbo].[Workshop]
AFTER INSERT, UPDATE
AS
BEGIN
	IF EXISTS( SELECT inserted.LimitOfParticipants FROM inserted INNER
JOIN ConferenceDay ON inserted.ConferenceDayID =
ConferenceDay.ConferenceDayID
	INNER JOIN Conferences ON Conferences.ConferenceID =
ConferenceDay.ConferenceID
	WHERE(inserted.LimitOfParticipants > Conferences.MaxParticipants))
	BEGIN;
	THROW 50004, 'Nie moze miec wiecej miejsc niz konferencja', 1
	END
END
GO
```

```
ALTER TABLE [dbo].[Workshop] ENABLE TRIGGER
[checkIfWorkshopHasMoreSlotsThanConference]
GO
```

Trigger checkLimitOfWorkshopParticipants

Trigger uniemożliwia dodanie uczestników ponad limit

```
CREATE TRIGGER [dbo].[checkLimitOfWorkshopParticipants] ON
[dbo].[WorkshopParticipants]
AFTER INSERT, UPDATE
AS
BEGIN
	DECLARE @Participant int = (SELECT ParticipantID FROM inserted);
	DECLARE @Workshop int = (SELECT WorkshopID FROM inserted);
	DECLARE @Day int = (SELECT Workshop.ConferenceDayID from Workshop
where Workshop.WorkshopID = @Workshop)
	DECLARE @ResDay int = (
		SELECT ReservationDay.ReservationDayID
		FROM ReservationDay
		JOIN Participants ON ReservationDay.ReservationDayID =
Participants.ReservationDayID
		WHERE ReservationDay.ConferenceDayID = @Day AND
Participants.ParticipantID = @Participant
	);
	DECLARE @CurrentAmount int = (
		SELECT COUNT(*)
		FROM WorkshopParticipants
		JOIN Participants ON WorkshopParticipants.ParticipantID =
Participants.ParticipantID
		JOIN ReservationDay ON
ReservationDay.ReservationDayID=Participants.ReservationDayID
		WHERE ReservationDay.ReservationDayID = @ResDay
	);
	DECLARE @MaxPlaces int = (
		SELECT NumberOfParticipants
		FROM WorkshopReservations
		WHERE WorkshopID = @Workshop AND ReservationDayID = @ResDay
	);

	IF @CurrentAmount > @MaxPlaces
	BEGIN
		ROLLBACK TRANSACTION;
		THROW 50001, 'The participants limit has been reached', 1
	END
END
GO

ALTER TABLE [dbo].[WorkshopParticipants] ENABLE TRIGGER
[checkLimitOfWorkshopParticipants]
```

```
GO
```

Trigger isParticipatingInConferenceThisDay

Trigger uniemożliwia dodanie uczestników do warsztatu jeśli nie uczestniczy w tym dniu nie uczestniczy w konferencji

```
CREATE TRIGGER [dbo].[isParticipatingInConferenceThisDay] ON
[dbo].[WorkshopParticipants]
      AFTER INSERT, UPDATE
AS
BEGIN
      IF EXISTS (SELECT inserted.WorkshopID from inserted
      JOIN Workshop ON inserted.WorkshopID=Workshop.WorkshopID
      JOIN Participants ON inserted.ParticipantID =
Participants.ParticipantID
      JOIN ReservationDay ON ReservationDay.ConferenceDayID =
Participants.ReservationDayID
      WHERE ReservationDay.ConferenceDayID != Workshop.ConferenceDayID)
      BEGIN
            ROLLBACK TRANSACTION;
            THROW 50001, 'The person has not reserved a place at conference
this day', 1
      END
END
GO

ALTER TABLE [dbo].[WorkshopParticipants] ENABLE TRIGGER
[isParticipatingInConferenceThisDay]
GO
```

Trigger workshopOverlap

Trigger sprawdza czy uczestnikowi nie nachodzą warsztaty

```
CREATE TRIGGER [dbo].[workshopOverlap] ON [dbo].[WorkshopParticipants]
      AFTER INSERT, UPDATE
AS
BEGIN
      DECLARE @PartiID int = (SELECT inserted.ParticipantID FROM inserted)

      DECLARE @WorkiID int = (SELECT inserted.WorkshopID FROM inserted)

      DECLARE @StartTime time = (SELECT Workshop.StartTime FROM inserted
inner join Workshop on inserted.WorkshopID = Workshop.WorkshopID)

      DECLARE @EndTime time = (SELECT Workshop.EndTime FROM inserted inner
join Workshop on inserted.WorkshopID = Workshop.WorkshopID)
```

```
      IF EXISTS(SELECT * FROM WorkshopParticipants INNER JOIN Workshop ON
Workshop.WorkshopID = WorkshopParticipants.WorkshopID
      WHERE WorkshopParticipants.ParticipantID = @PartiID and
WorkshopParticipants.WorkshopID <> @WorkiID and ((@StartTime BETWEEN
Workshop.StartTime and Workshop.EndTime) OR (@EndTime BETWEEN
Workshop.StartTime and Workshop.EndTime)
      OR (Workshop.StartTime BETWEEN @StartTime and @EndTime) OR
(Workshop.EndTime BETWEEN @StartTime and @EndTime)))
      BEGIN;
      THROW 50004, 'Nachodzace warsztaty', 1
      END
END
GO

ALTER TABLE [dbo].[WorkshopParticipants] ENABLE TRIGGER [workshopOverlap]
GO
```

8. Funkcje

Funkcja F_ConfMaxParticipants

Funkcja zwraca limit uczestników konferencji

```
CREATE FUNCTION [dbo].[F_ConfMaxParticipants]
(
@conferenceID int
)
RETURNS int
AS
      BEGIN
      RETURN(ISNULL(0,(SELECT MaxParticipants FROM Conferences WHERE
ConferenceID = @conferenceID)))
      END
GO
```

Funkcja F_CountTakenPlaceInReservationDay

Funkcja zwraca ilość zajętych miejsc w danym dniu rezerwacji

```
CREATE FUNCTION [dbo].[F_CountTakenPlacesInReservartionDay]
(
@ReservationDayID int
)
RETURNS int
AS
BEGIN
RETURN ISNULL((SELECT SUM(NumberOfParticipants) + SUM(NumberOfStudents)
FROM ReservationDay
Where ReservationDayID = @ReservationDayID), 0)
END
GO
```

Funkcja F_CountTakenPlacesInWokrshop

Funkcja zwraca liczbę uczestników warsztatu

```
CREATE FUNCTION [dbo].[F_CountTakenPlacesInWorkshop]
(
@WorkshopReservationsID int
)
RETURNS int
AS
BEGIN
RETURN ISNULL((SELECT SUM(NumberOfParticipants)
```

```
FROM WorkshopReservations
Where WorkshopReservations.WorkshopReservationID =
@WorkshopReservationsID), 0)
END

GO
```

Funkcja F_GetConfDayFreeSlots

Funkcja zwraca ilość wolnych miejsc na dzień konferencji

```
CREATE FUNCTION [dbo].[F_GetConfDayFreeSlots]
(
@conferenceDayID int
)
RETURNS int
AS
BEGIN
RETURN ((SELECT Conferences.MaxParticipants FROM Conferences
INNER JOIN ConferenceDay on Conferences.ConferenceID =
ConferenceDay.ConferenceDayID)
-(SELECT SUM(ReservationDay.NumberOfParticipants) FROM ReservationDay INNER
JOIN ConferenceDay ON ReservationDay.ConferenceDayID =
ConferenceDay.ConferenceDayID))
END
GO
```

Funkcja F_GetConferenceEndDate

Funkcja zwraca datę końca konferencji

```
CREATE FUNCTION [dbo].[F_GetConferenceEndDate]
(
@ConferenceID int
)
RETURNS date
AS
BEGIN
RETURN(SELECT Conferences.EndDate
FROM Conferences
WHERE Conferences.ConferenceID = @ConferenceID)
END
GO
```

Funkcja F_GetConferenceID

Funkcja zwraca ID konferencji

```
CREATE FUNCTION [dbo].[F_GetConferenceID]
```

```
(
@ConferenceDayID int
)
RETURNS int
AS
BEGIN
RETURN(SELECT ConferenceID
FROM ConferenceDay
WHERE ConferenceDayID = @ConferenceDayID)
END
GO
```

Funkcja F_GetConferenceStartDate

Funkcja zwraca datę początku konferencji

```
CREATE FUNCTION [dbo].[F_GetConferenceStartDate]
(
@ConferenceID int
)
RETURNS date
AS
BEGIN
RETURN(SELECT Conferences.StartDate
FROM Conferences
WHERE Conferences.ConferenceID = @ConferenceID)
END
GO
```

Funkcja F_GetDayID

Funkcja zwraca ID dnia konferencji mając ID konferencji oraz jego datę

```
CREATE FUNCTION [dbo].[F_GetDayID]
(
@conferenceID int,
@date date
)
RETURNS int
AS
BEGIN
RETURN (Select ConferenceDayID
From ConferenceDay
WHERE ConferenceID = @conferenceID AND Date = @date)
END
GO
```

Funkcja F_GetDiscount

Funkcja zwraca zniżkę na podstawie ID konferencji i daty rezerwacji

```
CREATE FUNCTION [dbo].[F_GetDiscount]
(
@ConferenceID int,
@ReservationDate date
)
RETURNS real
AS
BEGIN
	RETURN(
	SELECT ISNULL(MAX(Discount), 0) FROM Discount as d inner join
Conferences as c on c.conferenceid = d.conferenceid
	WHERE c.startdate >= DATEADD(Day, d.daysbeforestart,
@reservationdate))
END
```

Funkcja F_NumberOfStudentsForEachDay

Funkcja zwraca liczbę studentów na każdy dzień

```
CREATE FUNCTION [dbo].[F_NumberOfStudentsForEachDay]
(
@ReservationDayID int
)
RETURNS smallint
AS
BEGIN
	RETURN(
	SELECT COUNT(*) FROM Students as s inner join Participants as p on
p.ParticipantID = s.ParticipantID
	WHERE p.DayReservationID = @ReservationDayID)
END
GO
```

Funkcja F_ReservDayNormal

Funkcja zwraca ilość uczestników niestudentów na dzień

```
CREATE FUNCTION [dbo].[F_ReservDayNormal]
(
@DayReservationID int
)
RETURNS int
AS
	BEGIN
	RETURN ISNULL(
	(SELECT NumberOfParticipants
	FROM ReservationDay
		WHERE ReservationDayID = @DayReservationID)
	- (SELECT NumberOfStudents
	FROM ReservationDay
		WHERE ReservationDayID = @DayReservationID), 0)
	END
GO
```

Funkcja F_ReservDayParticipants

Funkcja zwraca ilość uczestników danego dnia

```
CREATE FUNCTION [dbo].[F_ReservDayParticipants]
(
@DayReservationID int
)
RETURNS int
AS
	BEGIN
	RETURN ISNULL((SELECT NumberOfParticipants
	FROM ReservationDay
		WHERE ReservationDayID = @DayReservationID), 0)
	END
GO
```

Funkcja F_ReservDayStudent

Funkcja zwraca ilość studentów uczestniczących w danym dniu

```
CREATE FUNCTION [dbo].[F_ReservDayStudent]
(
@DayReservationID int
)
RETURNS int
AS
	BEGIN
	RETURN ISNULL((SELECT NumberOfStudents
	FROM ReservationDay
		WHERE ReservationDayID = @DayReservationID), 0)
	END
```

```
GO
```

Funkcja F_WorkshopSlots

Funkcja zwraca ilość miejsc na warsztat

```
CREATE FUNCTION [dbo].[F_WorkshopSlots]
(
@WorkshopID int
)
RETURNS int
AS
	BEGIN
	RETURN ISNULL((SELECT LimitOfParticipants FROM Workshop WHERE
WorkshopID = @WorkshopID),0)
END
GO
```

Funkcja F_WorkshopSlotsTaken

Funkcja zwraca ilość zajętych miejsc na warsztat

```
CREATE FUNCTION [dbo].[F_WorkshopSlotsTaken]
(
@WorkshopID int
)
RETURNS int
AS
	BEGIN
	RETURN ISNULL((SELECT NumberOfParticipants FROM WorkshopReservations
WHERE WorkshopID = @WorkshopID),0)
	END
GO
```

Funkcja F_WorkshopSlotsTakenByNotStudents

Funkcja zwraca ilość zajętych miejsc nie przez studentów

```
CREATE FUNCTION [dbo].[F_WorkshopSlotsTakenByNotStudents]
(
@WorkshopID int
)
RETURNS int
AS
	BEGIN
```

```
      RETURN ISNULL((SELECT NumberOfParticipants - NumberOfStudents FROM
WorkshopReservations WHERE WorkshopID = @WorkshopID),0)
      END
GO
```

Funkcja F_WorkshopSlotsTakenByStudents

Funkcja zwraca ilość zajętych miejsc przez studentów

```
CREATE FUNCTION [dbo].[F_WorkshopSlotsTakenByStudents]
(
@WorkshopID int
)
RETURNS int
AS
      BEGIN
      RETURN ISNULL((SELECT NumberOfStudents FROM WorkshopReservations
WHERE WorkshopID = @WorkshopID),0)
      END
```

Funkcja F_GetReservationCost

Funkcja zwraca koszt rezerwacji

```
CREATE FUNCTION [dbo].[F_GetReservationCost]
(
@ReservationID int
)
returns money
AS
BEGIN
DECLARE @cenanormalna MONEY =
dbo.F_GetReservDayNormalTicketPrice(@ReservationID)


DECLARE @znizkastudencka real =
(SELECT C.DiscountForStudents
FROM Reservations as R
JOIN ReservationDay as Rday ON R.reservationID = Rday.reservationid
JOIN ConferenceDay as Cday ON Cday.conferenceDayID = Rday.conferenceDayID
JOIN Conferences as C ON C.ConferenceID = Cday.ConferenceID
WHERE R.ReservationID = @ReservationID)



DECLARE @kosztrezerwacji MONEY =
(Select
sum(ReservationDay.NumberOfParticipants  * @cenanormalna) +
```

```
Sum(ReservationDay.NumberOfStudents) * @cenanormalna * (1 -
@znizkastudencka)
From ReservationDay WHERE ReservationDay.ReservationID = @ReservationID)

DECLARE @kosztwarsztatu MONEY =
(Select SUM(val)
From (Select (Select SUM(WorkshopReservations.NumberOfParticipants *
Workshop.Price) +
SUM(WorkshopReservations.NumberOfStudents * (1 - @znizkastudencka) *
Workshop.Price)
FROM WorkshopReservations
INNER JOIN Workshop
ON Workshop.WorkSHOPID = WorkshopReservations.WorkshopID
WHERE WorkshopReservations.ReservationDayID =
ReservationDay.ReservationDayID) as val
FROM ReservationDay WHERE ReservationDay.ReservationID = @ReservationID)
src)


RETURN (ISNULL(@kosztrezerwacji, 0) + ISNULL(@kosztwarsztatu, 0))
END

GO
```

Funkcja F_ReservDayNormalTicketPrice

Funkcja cenę normalnego biletu na podstawie progu zniżki

```
CREATE
FUNCTION [dbo].[F_ReservDayNormalTicketPrice]

(
@ReservationID int
)
RETURNS MONEY
AS
BEGIN
      DECLARE @cena MONEY =
(SELECT C.DayPrice * (1 - dbo.F_GetDiscount(C.ConferenceID,
R.ReservationDate))
FROM Reservations as R
JOIN ReservationDay as Rday ON R.reservationID = Rday.reservationid
JOIN ConferenceDay as Cday ON Cday.conferenceDayID = Rday.conferenceDayID
JOIN Conferences as C ON C.ConferenceID = Cday.ConferenceID
WHERE R.ReservationID = @ReservationID)

RETURN isnull (@cena,0)
END
GO
```

9. Uprawnienia

Administrator

Posiada pełny dostęp do wszystkich widoków, procedur, i funkcji

Pracownik organizujący konferencję

Posiada dostęp do widoków przedstawiających informacje o płatnościach, listach uczestników, generujących identyfikatory imienne, wyświetlających informacje o klientach, konferencjach i warsztatach oraz procedur i funkcji pozwalających na zarządzanie konferencjami i warsztatami (dodawanie nowych, edycja istniejących) oraz do procedury generującej fakturę

Pracownik firmy organizującej konferencję

Posiada dostęp do list klientów, uczestników, rezerwacji, a także procedur i funkcji odpowiedzialnych za dodawanie i edycję danych uczestników.

Klient jako firma

Posiada dostęp do widoków przedstawiających konferencje, warsztaty wraz z cenami, listy i informacje o pracownikach oraz procedur realizujących proces rezerwacji oraz zapisu na konferencje i warsztaty.

Klient jako osoba

Posiada dostęp do widoków przedstawiających konferencje i warsztaty wraz z cenami oraz procedur realizujących proces rezerwacji oraz zapisu na konferencje i warsztaty.

Uczestnik konferencji

Uczestnik posiada możliwość zapisania się na warsztat