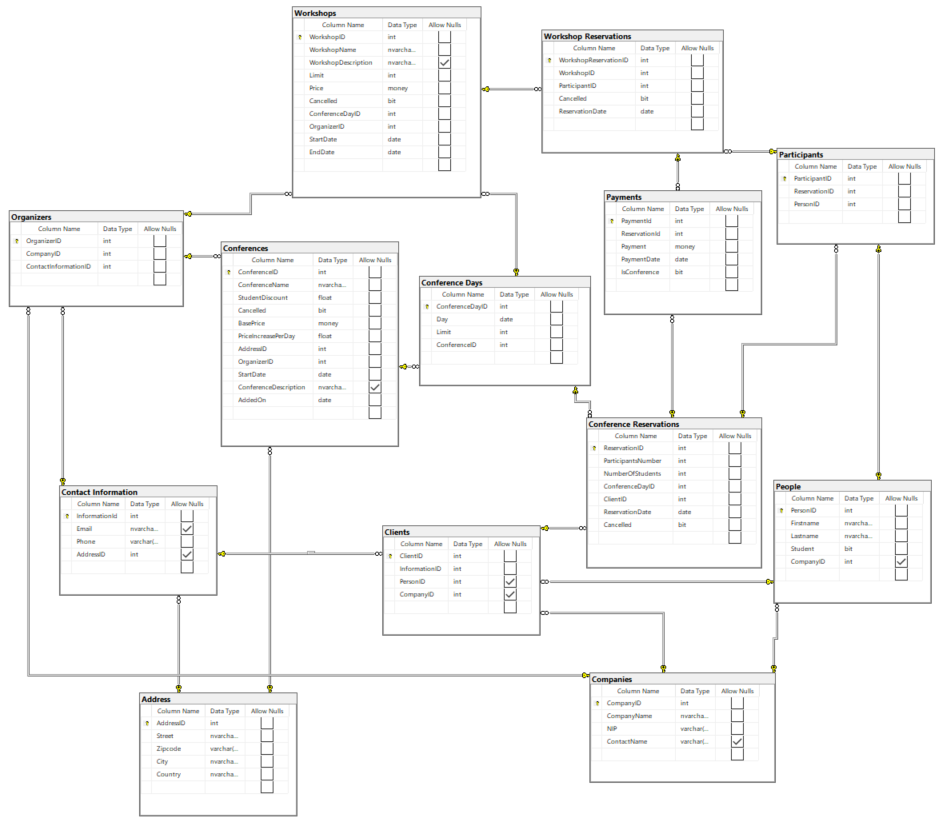
Mikołaj Zatorski

Dominik Jastrząb

1. **Schemat bazy danych**

****

1. **Opisy tabel i kod tworzący je:**
2. **Workshop Reservations**

Zawiera rezerwacje na warsztaty dokonywane przez uczestników konferencji.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Workshop Reservations](

[WorkshopReservationID] [int] NOT NULL,

[WorkshopID] [int] NOT NULL,

[ParticipantID] [int] NOT NULL,

[Cancelled] [bit] NOT NULL,

[ReservationDate] [date] NOT NULL,

CONSTRAINT [PK\_Workshop Reservations] PRIMARY KEY CLUSTERED

(

[WorkshopReservationID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Conference Reservations**

Zawiera rezerwacje miejsc na dany dzień konferencji, jeśli rezerwacja jest zbiorowa to ParticipantsNumber zawiera liczbę uczestników.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Conference Reservations](

[ReservationID] [int] NOT NULL,

[ParticipantsNumber] [int] NOT NULL,

[NumberOfStudents] [int] NOT NULL,

[ConferenceDayID] [int] NOT NULL,

[ClientID] [int] NOT NULL,

[ReservationDate] [date] NOT NULL,

[Cancelled] [bit] NOT NULL,

CONSTRAINT [PK\_Conference Resertvations] PRIMARY KEY CLUSTERED

(

[ReservationID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Participants**

Tabela wiążąca fizyczne osoby z ich rezerwacjami, każda osoba w tej tabeli musi mieć rezerwacje na dzień konferencji.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Participants](

[ParticipantID] [int] NOT NULL,

[ReservationID] [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, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Conferences**

Tabela zawierająca szczegóły konferencji.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Conferences](

[ConferenceID] [int] NOT NULL,

[ConferenceName] [nvarchar](100) NOT NULL,

[StudentDiscount] [float] NOT NULL,

[Cancelled] [bit] NOT NULL,

[BasePrice] [money] NOT NULL,

[PriceIncreasePerDay] [float] NOT NULL,

[AddressID] [int] NOT NULL,

[OrganizerID] [int] NOT NULL,

[StartDate] [date] NOT NULL,

[ConferenceDescription] [nvarchar](250) NULL,

[AddedOn] [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, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Conference Days**

Tabela zawierająca poszczególne dni konferencji wraz z limitami osób. Klienci dokonują rezerwacji na dany dzień konferencji .

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Conference Days](

[ConferenceDayID] [int] NOT NULL,

[Day] [date] NOT NULL,

[Limit] [int] NOT NULL,

[ConferenceID] [int] NOT NULL,

CONSTRAINT [PK\_Reservation Days] PRIMARY KEY CLUSTERED

(

[ConferenceDayID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Workshops**

Tabela zawierająca szczegóły warsztatów.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Workshops](

[WorkshopID] [int] NOT NULL,

[WorkshopName] [nvarchar](100) NOT NULL,

[WorkshopDescription] [nvarchar](250) NULL,

[Limit] [int] NOT NULL,

[Price] [money] NOT NULL,

[Cancelled] [bit] NOT NULL,

[ConferenceDayID] [int] NOT NULL,

[OrganizerID] [int] NOT NULL,

[StartDate] [date] NOT NULL,

[EndDate] [date] NOT NULL,

CONSTRAINT [PK\_Workshops] PRIMARY KEY CLUSTERED

(

[WorkshopID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Organizers**

Tabela ze opisująca szczegóły dotyczące organizatorów konferencji/warsztatów.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Organizers](

[OrganizerID] [int] NOT NULL,

[CompanyID] [int] NOT NULL,

[ContactInformationID] [int] NOT NULL,

CONSTRAINT [PK\_Organizers] PRIMARY KEY CLUSTERED

(

[OrganizerID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Payments**

Tabela zawierająca wpłaty dotyczące danych rezerwacji na konferencję/warsztat.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Payments](

[PaymentId] [int] NOT NULL,

[ReservationId] [int] NOT NULL,

[Payment] [money] NOT NULL,

[PaymentDate] [date] NOT NULL,

[IsConference] [bit] NOT NULL,

CONSTRAINT [PK\_Payments] PRIMARY KEY CLUSTERED

(

[PaymentId] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **People**

Tabela zawierająca dane poszczególnych osób fizycznych. Jeśli StudentCardID jest NULL to osoba nie jest studentem, jeśli CompanyID jest NULL to osoba nie jest zatrudniona w żadnej firmie.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[People](

[PersonID] [int] IDENTITY(1,1) NOT NULL,

[Firstname] [nvarchar](20) NOT NULL,

[Lastname] [nvarchar](30) NOT NULL,

[Student] [bit] NOT NULL,

[CompanyID] [int] NULL,

CONSTRAINT [PK\_People] PRIMARY KEY CLUSTERED

(

[PersonID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Clients**

Tabela z klientami, czyli osobami lub firmami, które dokonują rezerwacji na konferencję, opłacają ją, zostawiają swoje dane kontaktowe.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Clients](

[ClientID] [int] IDENTITY(1,1) NOT NULL,

[InformationID] [int] NOT NULL,

[PersonID] [int] NULL,

[CompanyID] [int] 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, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Companies**

Tabela przechowująca dane firm dokonujących rezerwacji na konferencje.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Companies](

[CompanyID] [int] NOT NULL,

[CompanyName] [nvarchar](50) NOT NULL,

[NIP] [varchar](15) NOT NULL,

[ContactName] [varchar](30) NULL,

CONSTRAINT [PK\_Companies] PRIMARY KEY CLUSTERED

(

[CompanyID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Contact Information**

Dane kontaktowe klientów oraz organizatorów. W niektórych przypadkach nie muszą zawierać adresu (klient może nie życzyć sobie udostępniać takich danych, a komunikacja z firmą organizującą konferencję odbywa się poprzez telefon/e-mail).

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Contact Information](

[InformationId] [int] IDENTITY(1,1) NOT NULL,

[Email] [nvarchar](40) NULL,

[Phone] [varchar](15) NOT NULL,

[AddressID] [int] NULL,

CONSTRAINT [PK\_Contact Information] PRIMARY KEY CLUSTERED

(

[InformationId] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Address**

Przechowuje zarówno adresy klientów jak i miejsca odbywania się konferencji.

SET ANSI\_NULLS ON

GO

SET QUOTED\_IDENTIFIER ON

GO

CREATE TABLE [dbo].[Address](

[AddressID] [int] NOT NULL,

[Street] [nvarchar](40) NOT NULL,

[Zipcode] [varchar](12) NOT NULL,

[City] [nvarchar](40) NOT NULL,

[Country] [nvarchar](20) NOT NULL,

CONSTRAINT [PK\_Address] PRIMARY KEY CLUSTERED

(

[AddressID] ASC

)WITH (PAD\_INDEX = OFF, STATISTICS\_NORECOMPUTE = OFF, IGNORE\_DUP\_KEY = OFF, ALLOW\_ROW\_LOCKS = ON, ALLOW\_PAGE\_LOCKS = ON, OPTIMIZE\_FOR\_SEQUENTIAL\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY]

GO

1. **Widoki**

**Cancelled Conference Reservations**

Spis anulowanych rezerwacji na konferencje.

CREATE VIEW [dbo].[Cancelled Conference Reservations]

AS

SELECT ReservationID, ParticipantsNumber, NumberOfStudents,

ClientID, ReservationDate, Cancelled, ConferenceDayID

FROM dbo.[Conference Reservations]

WHERE (Cancelled = 1)

GO

**Cancelled Conferences**

Spis anulowanych konferencji.

CREATE VIEW [dbo].[Cancelled Conferences]

AS

SELECT ConferenceID, ConferenceName, StudentDiscount, Cancelled,

BasePrice, PriceIncreasePerDay, AddressID, OrganizerID, StartDate

FROM dbo.Conferences

WHERE (Cancelled = 1)

GO

**Cancelled Workshop Reservations**

Spis anulowanych rezerwacji na warsztaty.

CREATE VIEW [dbo].[Cancelled Workshop Reservations]

AS

SELECT WorkshopReservationID, WorkshopID, ParticipantID, PaymentDeadline, Cancelled, ReservationDate

FROM dbo.[Workshop Reservations]

WHERE (Cancelled = 1)

GO

**Cancelled Workshops**

Spis anulowanych warsztatów.

CREATE VIEW [dbo].[Cancelled Workshops]

AS

SELECT WorkshopID, WorkshopName, WorkshopDescription,

Cancelled, Limit, Price, ConferenceDayID, OrganizerID, WorkshopDate

FROM dbo.Workshops

WHERE (Cancelled = 1)

GO

**Conf Reservations Payments Balance**

Spis rezerwacji na dni konferencji wraz ze statusami płatności (czy zapłacono już pełną kwotę, termin zapłaty itp.).

CREATE VIEW [dbo].[Conf Reservations Payments Balance]

AS

SELECT dbo.[Conference Reservations].ReservationID,

dbo.[Conference Reservations].ClientID,

dbo.[Conference Reservations].Cancelled,

dbo.CalculateFinalPrice(dbo.[Conference Reservations].ReservationID) AS FinalPrice,

SUM(dbo.Payments.Payment) AS PaymentsSum,

dbo.CalculateFinalPrice(dbo.[Conference Reservations].ReservationID) - SUM(dbo.Payments.Payment)

AS ToPay,

DATEADD(week, 1, dbo.[Conference Reservations].ReservationDate) AS PaymentDeadline

FROM dbo.[Conference Reservations]

INNER JOIN dbo.Payments

ON dbo.[Conference Reservations].ReservationID = dbo.Payments.ReservationId

WHERE (dbo.Payments.IsConference = 1)

GROUP BY dbo.[Conference Reservations].ReservationID,

dbo.[Conference Reservations].ClientID,

dbo.CalculateFinalPrice(dbo.[Conference Reservations].ReservationID),

DATEADD(week, 1, dbo.[Conference Reservations].ReservationDate),

dbo.[Conference Reservations].Cancelled

GO

**Conferences available spaces**

Spis dni konferencji wraz z liczbą wolnych miejsc na nie.

CREATE VIEW [dbo].[Conferences available spaces]

AS

SELECT dbo.[Conference Days].ConferenceDayID, dbo.[Conference Days].Limit,

SUM(dbo.[Conference Reservations].ParticipantsNumber) AS Reserved,

dbo.[Conference Days].Limit - SUM(dbo.[Conference Reservations].ParticipantsNumber) AS Free

FROM dbo.[Conference Days]

INNER JOIN dbo.[Conference Reservations]

ON dbo.[Conference Days].ConferenceDayID = dbo.[Conference Reservations].ConferenceDayID

WHERE (dbo.[Conference Reservations].Cancelled = 0)

GROUP BY dbo.[Conference Days].ConferenceDayID, dbo.[Conference Days].Limit

GO

**Incoming Conferences**

Nadchodzące konferencje.

CREATE VIEW [dbo].[Incoming Conferences]

AS

SELECT dbo.Conferences.ConferenceName, dbo.Conferences.BasePrice,

dbo.Conferences.Cancelled, dbo.Conferences.StudentDiscount,

dbo.Conferences.PriceIncreasePerDay, dbo.Conferences.StartDate,

dbo.[Conference Days].Day, dbo.[Conference Days].Limit,

dbo.Companies.CompanyName AS OrganizerName

FROM dbo.Conferences

INNER JOIN dbo.Organizers

ON dbo.Conferences.OrganizerID = dbo.Organizers.OrganizerID

INNER JOIN dbo.[Conference Days]

ON dbo.Conferences.ConferenceID = dbo.[Conference Days].ConferenceID

INNER JOIN dbo.Companies

ON dbo.Organizers.CompanyID = dbo.Companies.ClientID

WHERE (dbo.Conferences.StartDate > DATEADD(week, 2, GETDATE()))

GO

**Missing Client Data**

Spis rezerwacji na konferencję wraz z informacją czy podano dane wszystkich uczestników.

CREATE VIEW [dbo].[Missing Client Data]

AS

SELECT dbo.Conferences.ConferenceName,

dbo.[Conference Reservations].ParticipantsNumber,

COUNT(dbo.Participants.ParticipantID) AS SignedParticipants,

dbo.[Conference Reservations].ParticipantsNumber - COUNT(dbo.Participants.ParticipantID) AS MissingParticipants

FROM dbo.[Conference Reservations]

INNER JOIN dbo.[Conference Days]

ON dbo.[Conference Reservations].ConferenceDayID = dbo.[Conference Days].ConferenceDayID

INNER JOIN dbo.Conferences

ON dbo.[Conference Days].ConferenceID = dbo.Conferences.ConferenceID

INNER JOIN dbo.Clients

ON dbo.[Conference Reservations].ClientID = dbo.Clients.ClientID

INNER JOIN dbo.[Contact Information]

ON dbo.Clients.InformationID = dbo.[Contact Information].InformationId

INNER JOIN dbo.Participants

ON dbo.[Conference Reservations].ReservationID = dbo.Participants.ReservationID

GROUP BY dbo.Conferences.ConferenceName, dbo.[Conference Reservations].ParticipantsNumber

HAVING (COUNT(dbo.Participants.ParticipantID) < dbo.[Conference Reservations].ParticipantsNumber)

GO

**Workshop Reservations Payments Balance**

Spis rezerwacji na warsztaty wraz ze statusami płatności (czy zapłacono już pełną kwotę itp.).

CREATE VIEW [dbo].[Workshop Reservations Payments Balance]

AS

SELECT dbo.[Workshop Reservations].WorkshopReservationID,

dbo.[Workshop Reservations].ParticipantID,

dbo.[Workshop Reservations].Cancelled,

dbo.Workshops.Price, SUM(dbo.Payments.Payment) AS PaymentsSum,

dbo.Workshops.Price - SUM(dbo.Payments.Payment) AS ToPay

FROM dbo.[Workshop Reservations]

INNER JOIN dbo.Workshops

ON dbo.[Workshop Reservations].WorkshopID = dbo.Workshops.WorkshopID

INNER JOIN dbo.Payments

ON dbo.[Workshop Reservations].WorkshopReservationID = dbo.Payments.ReservationId

GROUP BY dbo.[Workshop Reservations].WorkshopReservationID,

dbo.[Workshop Reservations].ParticipantID,

dbo.Workshops.Price,

dbo.[Workshop Reservations].Cancelled

GO

**Workshops available spaces**

Spis warsztatów wraz z liczbą wolnych miejsc na nie.

CREATE VIEW [dbo].[Workshops Available Spaces]

AS

SELECT dbo.Workshops.WorkshopID,

dbo.Workshops.Limit,

COUNT(dbo.Participants.ParticipantID) AS Reserved,

dbo.Workshops.Limit - COUNT(dbo.[Workshop Reservations].ParticipantID) AS Free

FROM dbo.Workshops

INNER JOIN dbo.[Workshop Reservations]

ON dbo.Workshops.WorkshopID = dbo.[Workshop Reservations].WorkshopID

INNER JOIN dbo.Participants

ON dbo.[Workshop Reservations].ParticipantID = dbo.Participants.ParticipantID

WHERE (dbo.[Workshop Reservations].Cancelled = 0)

GROUP BY dbo.Workshops.WorkshopID, dbo.Workshops.Limit

GO

* 1. **Procedury z parametrem, spełniające funkcję widoków**

**ConferenceDayParticipantsList**

Spis uczestników danego dnia konferencji wraz z danymi do identyfikatora.

CREATE PROCEDURE [dbo].[ConferenceDayParticipantsList]

@ConferenceDayID int

AS

BEGIN

SET NOCOUNT ON;

SELECT P.PersonID, P.Firstname, P.Lastname, C.CompanyName

FROM dbo.[Conference Days] AS CD

INNER JOIN dbo.[Conference Reservations] AS CR

ON CD.ConferenceDayID = @ConferenceDayID AND CD.ConferenceDayID = CR.ConferenceDayID

INNER JOIN dbo.Participants AS PART

ON CR.Cancelled = 0 AND CR.ReservationID = PART.ReservationID

INNER JOIN PEOPLE AS P

ON PART.PersonID = P.PersonID

LEFT OUTER JOIN Companies AS C

ON P.CompanyID = C.ClientID

END

**ConfReservationDetails**

Szczegółowe dane pojedynczej rezerwacji.

ALTER PROCEDURE [dbo].[ConfReservationDetails]

@ReservationID int

AS

BEGIN

SET NOCOUNT ON;

SELECT CR.ReservationID, CR.ReservationDate, CR.Cancelled,

CR.ParticipantsNumber, CR.NumberOfStudents, CR\_BALANCE.FinalPrice,

CR\_BALANCE.ToPay, CR\_BALANCE.PaymentDeadline, CR.ClientID, CD.Day,

CD.Limit, C.ConferenceID, C.ConferenceName, C.Cancelled

FROM [Conference Reservations] AS CR

INNER JOIN [Conference Days] AS CD

ON CR.ReservationID = @ReservationID AND CR.ConferenceDayID = CD.ConferenceDayID

INNER JOIN Conferences AS C

ON CD.ConferenceID = C.ConferenceID

INNER JOIN [Conf Reservations Payments Balance] AS CR\_BALANCE

ON CR.ReservationID = CR\_BALANCE.ReservationID

END

**MyConferences**

Spis danych o konferencjach, dniach, na które jest zapisana dana osoba.

ALTER PROCEDURE [dbo].[MyConferences]

@PersonID int

AS

BEGIN

SET NOCOUNT ON;

SELECT C.ConferenceID, C.ConferenceName, C.ConferenceDescription, C.StartDate,

A.Street, A.Zipcode, A.City, COMP.CompanyName, C.Cancelled, CD.ConferenceDayID, CD.Day, CD.Limit

FROM People AS P

INNER JOIN Participants AS PART

ON P.PersonID = @PersonID and P.PersonID = PART.PersonID

INNER JOIN [Conference Reservations] AS CR

ON PART.ReservationID = CR.ReservationID AND CR.Cancelled = 0

INNER JOIN [Conference Days] AS CD

ON CR.ConferenceDayID = CD.ConferenceDayID

INNER JOIN Conferences AS C

ON CD.ConferenceID = C.ConferenceID

INNER JOIN Address as A

ON C.AddressID = A.AddressID

INNER JOIN Organizers AS O

ON C.OrganizerID = O.OrganizerID

INNER JOIN Companies AS COMP

ON O.CompanyID = COMP.ClientID

END

**MyConfReservationsBalance**

Spis rezerwacji na konferencje danego użytkownika wraz z informacją o płatnościach.

ALTER PROCEDURE [dbo].[MyConfReservationsBalance]

@ClientID int

AS

BEGIN

SET NOCOUNT ON;

SELECT R.ReservationID, R.Cancelled, R.FinalPrice, R.PaymentsSum, R.ToPay, R.PaymentDeadline

FROM [Conf Reservations Payments Balance] AS R

WHERE R.ClientID = @ClientID

END

**MyWorkshopReservationBalance**

Spis rezerwacji na warsztaty danego użytkownika wraz z informacją o płatnościach.

ALTER PROCEDURE [dbo].[MyWorkshopReservationsBalance]

@PersonID int

AS

BEGIN

SET NOCOUNT ON;

SELECT WR.WorkshopReservationID, WR.Cancelled, WR.Price, WR.PaymentsSum, WR.ToPay

FROM People AS P

INNER JOIN Participants AS PART

ON P.PersonID = @PersonID AND P.PersonID = PART.PersonID

INNER JOIN [Workshop Reservations Payments Balance] AS WR

ON PART.ParticipantID = WR.ParticipantID

END

**My Workshops**

Spis danych o warsztatach, na które jest zapisana dana osoba.

ALTER PROCEDURE [dbo].[MyWorkshops]

@PersonID int

AS

BEGIN

SET NOCOUNT ON;

SELECT W.WorkshopID, W.WorkshopName, W.WorkshopDescription, W.WorkshopDate,

COMP.CompanyName as 'Organizer', W.Cancelled, W.Limit, W.Price

FROM People AS P

INNER JOIN Participants AS PART

ON P.PersonID = @PersonID and P.PersonID = PART.PersonID

INNER JOIN [Workshop Reservations] AS WR

ON PART.ParticipantID = WR.ParticipantID AND WR.Cancelled = 0

INNER JOIN Workshops AS W

ON WR.WorkshopID= W.WorkshopID

INNER JOIN Organizers AS O

ON W.OrganizerID = O.OrganizerID

INNER JOIN Companies AS COMP

ON O.CompanyID = COMP.ClientID

END

**WorkshopParticipantsList**

Spis uczestników danego warsztatu wraz z danymi do identyfikatora.

ALTER PROCEDURE [dbo].[WorkshopParticipantsList]

@WorkshopID int

AS

BEGIN

SET NOCOUNT ON;

SELECT P.PersonID, P.Firstname, P.Lastname, C.CompanyName

FROM dbo.Workshops AS W

INNER JOIN dbo.[Workshop Reservations] WR

ON W.WorkshopID = @WorkshopID AND W.WorkshopID = WR.WorkshopID

INNER JOIN dbo.Participants AS PART

ON WR.Cancelled = 0 and WR.ParticipantID = PART.ParticipantID

INNER JOIN PEOPLE AS P

ON PART.PersonID = P.PersonID

LEFT OUTER JOIN Companies AS C

ON P.CompanyID = C.ClientID

END

**WorkshopsReservationDetails**

Szczegółowe informacje na temat danej rezerwacji na warsztat.

ALTER PROCEDURE [dbo].[WorkshopReservationDetails]

@ReservationID int

AS

BEGIN

SET NOCOUNT ON;

SELECT WR.WorkshopReservationID, WR.ReservationDate, WR.Cancelled, W.Price,

WR.PaymentDeadline, WR.ParticipantID, W.WorkshopID, W.WorkshopName,

W.Cancelled, W.Limit

FROM [Workshop Reservations] AS WR

INNER JOIN Workshops AS W

ON WR.WorkshopReservationID = @ReservationID AND WR.WorkshopID = W.WorkshopID

END

**TopCustomers**

Dla danego organizatora zwraca listę klientów najczęściej korzystających z jego usług.

ALTER PROCEDURE [dbo].[TopCustomers]

@OrganizerID int

AS

BEGIN

SET NOCOUNT ON;

SELECT CL.ClientID, COUNT(CR.ReservationID) as 'ReservationsNumber'

FROM Organizers AS O

INNER JOIN Conferences AS C

ON O.OrganizerID = @OrganizerID AND O.OrganizerID = C.OrganizerID

INNER JOIN [Conference Days] AS CD

ON C.ConferenceID = CD.ConferenceID

INNER JOIN [Conference Reservations] AS CR

ON CR.Cancelled = 0 AND CD.ConferenceDayID = CR.ConferenceDayID

INNER JOIN Clients AS CL

ON CR.ClientID = CL.ClientID

GROUP BY CL.ClientID

ORDER BY [ReservationsNumber] DESC

END

**ClientDetails // do poprawy**

Szczegółowe dane danego klienta.

ALTER PROCEDURE [dbo].[ClientDetails]

@ClientID int

AS

BEGIN

SET NOCOUNT ON;

SELECT C.ClientID, P.Firstname, P.Lastname, P.Student,

COMP.CompanyName, COMP.ContactName, COMP.NIP, I.Email,

I.Phone, A.Country, A.City, A.Street, A.Zipcode

FROM Clients AS C

INNER JOIN [Contact Information] AS I

ON C.ClientID = @ClientID AND C.InformationID = I.InformationId

LEFT OUTER JOIN Address AS A

ON I.AddressID = A.AddressID

LEFT OUTER JOIN Companies AS COMP

ON C.ClientID = COMP.ClientID

LEFT OUTER JOIN People AS P

ON C.ClientID = P.PersonID

END

1. **Funkcje ze skalarną wartością zwracaną**

**CalculateFinalPrice**

Dla danej rezerwacji na dzień konferencji oblicza łączną należność.

ALTER FUNCTION [dbo].[CalculateFinalPrice]

(

@ReservationID int

)

RETURNS money

AS

BEGIN

DECLARE @BasePrice money;

DECLARE @StudentDiscount float;

DECLARE @Students int;

DECLARE @Participants int;

DECLARE @PriceIncreasePerDay money;

DECLARE @ConfDate date;

DECLARE @ReservationDate date;

SELECT @BasePrice = C.BasePrice,

@StudentDiscount = C.StudentDiscount,

@PriceIncreasePerDay = C.PriceIncreasePerDay,

@ConfDate = C.AddedOn,

@Students = CR.NumberOfStudents,

@Participants = CR.ParticipantsNumber,

@ReservationDate =CR.ReservationDate

FROM [Conference Reservations] AS CR

INNER JOIN [Conference Days] AS CD

ON CR.ReservationID = @ReservationID

AND CR.ConferenceDayID = CD.ConferenceDayID

INNER JOIN Conferences AS C

ON CD.ConferenceID = C.ConferenceID

DECLARE @NormalPrice money = @BasePrice

+ DATEDIFF(day, @ConfDate, @ReservationDate)\*@PriceIncreasePerDay

DECLARE @StudentPrice money = @NormalPrice \* (1 - @StudentDiscount)

RETURN @NormalPrice\*(@Participants - @Students)

+ @StudentPrice \* @Students

END

**ShouldBeCancelled**

Dla danej rezerwacji, sprawdza czy przekroczono termin płatności i czy należy anulować rezerwację.

ALTER FUNCTION [dbo].[ShouldBeCancelled]

(

@ReservationID int

)

RETURNS bit

AS

BEGIN

DECLARE @Deadline date;

DECLARE @ToPay money;

SELECT @Deadline = CR.PaymentDeadline, @ToPay = CR.ToPay

FROM [Conf Reservations Payments Balance] AS CR

WHERE CR.ReservationID = @ReservationID

DECLARE @Result bit;

SET @Result =IIF((@ToPay > 0 and getdate() > @Deadline), 1, 0)

RETURN @Result

END

**CanRegisterForWorkshop**

Dla danego uczestnika konferencji sprawdza czy może ona zarezerwować miejsce na dany warsztat.

ALTER FUNCTION [dbo].[CanRegisterForWorkshop]

(

@ParticipantID int,

@WorkshopID int

)

RETURNS bit

AS

BEGIN

DECLARE @Result bit

DECLARE @ConferenceDayID int;

DECLARE @WorkshopStartDate date;

DECLARE @WorkshopEndDate date;

SELECT @ConferenceDayID = W.ConferenceDayID,

@WorkshopStartDate = W.StartDate,

@WorkshopEndDate = W.EndDate

FROM Workshops AS W

WHERE W.WorkshopID = @WorkshopID

IF @ParticipantID not in (

SELECT P.ParticipantID

FROM [Conference Days] AS CD

INNER JOIN [Conference Reservations] AS CR

ON CD.ConferenceDayID = @ConferenceDayID AND CD.ConferenceDayID = CR.ConferenceDayID

INNER JOIN Participants AS P

ON CR.ReservationID = P.ReservationID

)

BEGIN

RETURN 0;

END

DECLARE @count INT;

SELECT @count = COUNT(\*)

FROM Workshops W

INNER JOIN [Workshop Reservations] AS WR

ON W.WorkshopID = WR.WorkshopID

INNER JOIN Participants AS P

ON P.ParticipantID = @ParticipantID AND WR.ParticipantID = P.ParticipantID

WHERE W.StartDate <= @WorkshopEndDate AND W.EndDate >= @WorkshopStartDate

RETURN iif(@count > 0, 0, 1);

END

1. **Procedury**

**AddWorkshop**

CREATE PROCEDURE AddWorkshop

@WorkshopName NVARCHAR(100),

@WorkshopDescription NVARCHAR(250) = NULL,

@Limit INT,

@Price MONEY,

@OrganizerID INT,

@ConferenceDayID INT,

@StartDate DATE,

@EndDate DATE

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Workshops(

WorkshopName,

WorkshopDescription,

Limit,

Price,

Cancelled,

ConferenceDayID,

OrganizerID,

StartDate,

EndDate

)

VALUES(

@WorkshopName,

@WorkshopDescription,

@Limit,

@Price,

0,

@ConferenceDayID,

@OrganizerID,

@StartDate,

@EndDate

)

END

**AddPerson**

CREATE PROCEDURE AddPerson

@FirstName NVARCHAR(50),

@LastName VARCHAR(15),

@Student VARCHAR(30) = 0,

@CompanyID INT = NULL

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO People(Firstname, Lastname, Student, CompanyID)

VALUEs(@Firstname, @Lastname, @Student, @CompanyID)

END

**AddPayment**

Umożliwia dokonywanie płatności za rezerwacje.

CREATE PROCEDURE AddPayment

@ReservationID INT,

@Payment MONEY,

@PaymentDate DATE,

@IsConference BIT

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Payments(ReservationId, Payment, PaymentDate, IsConference)

VALUEs(@ReservationId, @Payment, @PaymentDate, @IsConference)

END

**AddAddress**

CREATE PROCEDURE AddAddress

@Street NVARCHAR(40),

@Zipcode VARCHAR(12),

@City NVARCHAR(40),

@Country NVARCHAR(20)

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Address(Street, Zipcode, City, Country)

VALUES(@Street, @Zipcode, @City, @Country)

EN

**AddCompany**

ALTER PROCEDURE AddCompany

@CompanyName NVARCHAR(50),

@NIP VARCHAR(15),

@ContactName VARCHAR(30) = NULL

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Companies(CompanyName, NIP, ContactName)

VALUEs(@CompanyName, @NIP, @ContactName)

END

**AddCompanyClient**

Tworzy klienta grupowego (firmę).

CREATE PROCEDURE AddCompanyClient

@CompanyName NVARCHAR(50),

@NIP VARCHAR(15),

@ContactName VARCHAR(30) = NULL,

@Email NVARCHAR(40) = NULL,

@Phone VARCHAR(15),

@Street NVARCHAR(40) = NULL,

@Zipcode VARCHAR(12) = NULL,

@City NVARCHAR(40) = NULL,

@Country NVARCHAR(20) = NULL

AS

BEGIN

SET NOCOUNT ON;

DECLARE @CompanyID INT;

DECLARE @InformationID INT;

DECLARE @AddressID INT;

IF @Street IS NOT NULL AND @Zipcode IS NOT NULL

AND @City IS NOT NULL AND @Country IS NOT NULL

BEGIN

SELECT @AddressID = A.AddressID

FROM Address AS A

WHERE A.City LIKE @City AND A.Country LIKE @Country

AND A.Zipcode LIKE @Zipcode AND A.Street LIKE @Street

IF @AddressID is NULL

BEGIN

EXEC dbo.AddAddress @Street, @Zipcode, @City, @Country

SELECT @AddressID = MAX(AddressID) FROM Address

END

END

SELECT @InformationID = InformationId

FROM [Contact Information]

WHERE Phone LIKE @Phone

IF @InformationID IS NULL

BEGIN

EXEC dbo.AddContactInformation @Phone, @Email, @AddressID

SELECT @InformationID = MAX(InformationID) FROM [Contact Information]

END

ELSE

BEGIN

UPDATE [Contact Information]

SET AddressID = @AddressID, Email = @Email

WHERE InformationId = @InformationID

END

SELECT @CompanyID = CompanyID

FROM Companies

WHERE NIP LIKE @NIP

IF @CompanyID IS NULL

BEGIN

EXEC dbo.AddCompany @CompanyName, @NIP, @ContactName

SELECT @CompanyID = MAX(CompanyID) FROM Companies

END

ELSE

BEGIN

UPDATE Companies

SET ContactName = @ContactName

WHERE CompanyID = @CompanyID

END

INSERT INTO Clients(CompanyID, PersonID, InformationID)

VALUES(@CompanyID, NULL, @InformationID)

END

**AddIndividualClient**

Tworzy klienta indywidualnego (osobę).

CREATE PROCEDURE AddIndividualClient

@FirstName NVARCHAR(20),

@LastName NVARCHAR(30),

@Student BIT = 0,

@CompanyID INT = NULL,

@Email NVARCHAR(40) = NULL,

@Phone VARCHAR(15),

@Street NVARCHAR(40) = NULL,

@Zipcode VARCHAR(12) = NULL,

@City NVARCHAR(40) = NULL,

@Country NVARCHAR(20) = NULL

AS

BEGIN

SET NOCOUNT ON;

DECLARE @PersonID INT;

DECLARE @InformationID INT;

DECLARE @AddressID INT;

IF @Street IS NOT NULL AND @Zipcode IS NOT NULL

AND @City IS NOT NULL AND @Country IS NOT NULL

BEGIN

SELECT @AddressID = A.AddressID

FROM Address AS A

WHERE A.City LIKE @City AND A.Country LIKE @Country

AND A.Zipcode LIKE @Zipcode AND A.Street LIKE @Street

IF @AddressID is NULL

BEGIN

EXEC dbo.AddAddress @Street, @Zipcode, @City, @Country

SELECT @AddressID = MAX(AddressID) FROM Address

END

END

SELECT @InformationID = InformationId

FROM [Contact Information]

WHERE Phone LIKE @Phone

IF @InformationID IS NULL

BEGIN

EXEC dbo.AddContactInformation @Phone, @Email, @AddressID

SELECT @InformationID = MAX(InformationID) FROM [Contact Information]

END

ELSE

BEGIN

UPDATE [Contact Information]

SET AddressID = @AddressID, Email = @Email

WHERE InformationId = @InformationID

END

EXEC dbo.AddPerson @FirstName, @LastName, @Student, @CompanyID

SELECT @PersonID = MAX(PersonID) FROM PEOPLE

INSERT INTO Clients(CompanyID, PersonID, InformationID)

VALUES(@CompanyID, NULL, @InformationID)

END

**AddConference**

CREATE PROCEDURE AddConference

@ConferenceName NVARCHAR(100),

@ConferenceDescription NVARCHAR(250) = NULL,

@StudentDiscount FLOAT = 0,

@BasePrice MONEY,

@PriceIncreasePerDay FLOAT = 0,

@OrganizerID INT,

@Street NVARCHAR(40),

@Zipcode VARCHAR(12),

@City NVARCHAR(40),

@Country NVARCHAR(20),

@StartDate DATE

AS

BEGIN

SET NOCOUNT ON;

DECLARE @AddressID INT;

SELECT @AddressID = A.AddressID

FROM Address AS A

WHERE A.City LIKE @City AND A.Country LIKE @Country

AND A.Zipcode LIKE @Zipcode AND A.Street LIKE @Street

IF @AddressID is NULL

BEGIN

EXEC dbo.AddAddress @Street, @Zipcode, @City, @Country

VALUES(@Street, @City, @Zipcode, @Country)

SELECT @AddressID = MAX(AddressID) FROM Address

END

INSERT INTO Conferences(

ConferenceName,

ConferenceDescription,

StudentDiscount,

Cancelled,

BasePrice,

PriceIncreasePerDay,

AddressID,

OrganizerID,

StartDate,

AddedOn

)

VALUES(

@ConferenceName,

@ConferenceDescription,

@StudentDiscount,

0,

@BasePrice,

@PriceIncreasePerDay,

@AddressID,

@OrganizerID,

@StartDate,

GETDATE()

)

END

**AddConferenceDay**

CREATE PROCEDURE AddConferenceDay

@ConferenceID INT,

@Day DATE,

@Limit INT = 0

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO [Conference Days](ConferenceID, Day, Limit)

VALUES(@ConferenceID, @Day, @Limit)

END

**AddContactInformation**

CREATE PROCEDURE AddContactInformation

@Phone VARCHAR(15),

@Email NVARCHAR(40) = NULL,

@AddressID INT = NULL

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO [Contact Information](Email, Phone, AddressID)

VALUES(@Email, @Phone, @AddressID)

END

**AddOrganizer**

CREATE PROCEDURE [dbo].[AddOrganizer]

@CompanyName NVARCHAR(50),

@NIP VARCHAR(15),

@ContactName VARCHAR(30) = NULL,

@Email NVARCHAR(40) = NULL,

@Phone VARCHAR(15),

@Street NVARCHAR(40) = NULL,

@Zipcode VARCHAR(12) = NULL,

@City NVARCHAR(40) = NULL,

@Country NVARCHAR(20) = NULL

AS

BEGIN

SET NOCOUNT ON;

DECLARE @CompanyID INT;

DECLARE @InformationID INT;

DECLARE @AddressID INT;

IF @Street IS NOT NULL AND @Zipcode IS NOT NULL

AND @City IS NOT NULL AND @Country IS NOT NULL

BEGIN

SELECT @AddressID = A.AddressID

FROM Address AS A

WHERE A.City LIKE @City AND A.Country LIKE @Country

AND A.Zipcode LIKE @Zipcode AND A.Street LIKE @Street

IF @AddressID is NULL

BEGIN

EXEC dbo.AddAddress @Street, @Zipcode, @City, @Country

SELECT @AddressID = MAX(AddressID) FROM Address

END

END

SELECT @InformationID = InformationId

FROM [Contact Information]

WHERE Phone LIKE @Phone

IF @InformationID IS NULL

BEGIN

EXEC dbo.AddContactInformation @Phone, @Email, @AddressID

SELECT @InformationID = MAX(InformationID) FROM [Contact Information]

END

ELSE

BEGIN

UPDATE [Contact Information]

SET AddressID = @AddressID, Email = @Email

WHERE InformationId = @InformationID

END

SELECT @CompanyID = CompanyID

FROM Companies

WHERE NIP LIKE @NIP

IF @CompanyID IS NULL

BEGIN

EXEC dbo.AddCompany @CompanyName, @NIP, @ContactName

SELECT @CompanyID = MAX(CompanyID) FROM Companies

END

ELSE

BEGIN

UPDATE Companies

SET ContactName = @ContactName

WHERE CompanyID = @CompanyID

END

INSERT INTO Organizers(CompanyID, ContactInformationID)

VALUES(@CompanyID, @InformationID)

END

**AddParticipants**

Dodanie osoby (uczestnika) do rezerwacji.

CREATE PROCEDURE AddParticipants

@PersonID INT,

@ReservationID INT

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO Participants(ReservationID, PersonID)

VALUES(@ReservationID, @PersonID)

END

**AddParticipantsToReservation**

Tworzy nową osobę i dodaje ją do rezerwacji.

CREATE PROCEDURE AddParticipantsToReservation

@ReservationID INT,

@FirstName NVARCHAR(50),

@LastName VARCHAR(15),

@Student VARCHAR(30) = 0,

@CompanyID INT = NULL

AS

BEGIN

SET NOCOUNT ON;

DECLARE @PersonID INT;

EXEC dbo.AddPerson @FirstName, @LastName, @Student, @CompanyID

SELECT @PersonID = MAX(PersonID) FROM People

EXEC dbo.AddParticipants @PersonID, @ReservationID

END

**AddPerson**

CREATE PROCEDURE AddPerson

@FirstName NVARCHAR(50),

@LastName VARCHAR(15),

@Student VARCHAR(30) = 0,

@CompanyID INT = NULL

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO People(Firstname, Lastname, Student, CompanyID)

VALUEs(@Firstname, @Lastname, @Student, @CompanyID)

END

**CancelConference**

CREATE PROCEDURE [dbo].[CancelConference]

@ConferenceID INT

AS

BEGIN

SET NOCOUNT ON;

UPDATE [Conferences]

SET Cancelled = 1

WHERE ConferenceID = @ConferenceID

UPDATE [Workshops]

SET Cancelled = 1

WHERE WorkshopID IN (

SELECT W.WorkshopID

FROM Conferences AS C

INNER JOIN [Conference Days] AS CD

ON C.ConferenceID = @ConferenceID AND C.ConferenceID = CD.ConferenceID

INNER JOIN Workshops AS W

ON CD.ConferenceDayID = W.ConferenceDayID

)

END

**CancelConfReservation**

CREATE PROCEDURE CancelConfReservation

@ReservationID int

AS

BEGIN

SET NOCOUNT ON;

UPDATE [Conference Reservations]

SET Cancelled = 1

WHERE ReservationID = @ReservationID

END

GO

**CancelUnpaidReservations**

Anuluje wszystkie rezerwacje na dni konferencji, które nie zostały opłacone w terminie.

CREATE PROCEDURE CancelUnpaidReservations

AS

BEGIN

SET NOCOUNT ON;

UPDATE [Conference Reservations]

SET Cancelled = 1

WHERE ReservationID in (

SELECT CR.ReservationID

FROM [Conference Reservations] AS CR

WHERE dbo.ShouldBeCancelled(Cr.ReservationID) = 1

)

END

**CancelWorkshop**

CREATE PROCEDURE CancelWorkshop

@WorkshopID int

AS

BEGIN

SET NOCOUNT ON;

UPDATE Workshops

SET Cancelled = 1

WHERE WorkshopID = @WorkshopID

END

GO

**CancelWorkshopReservation**

CREATE PROCEDURE CancelWorkshopReservation

@ReservationID int

AS

BEGIN

SET NOCOUNT ON;

UPDATE [Workshop Reservations]

SET Cancelled = 1

WHERE WorkshopReservationID = @ReservationID

END

GO

**ChangeConfDayLimit**

CREATE PROCEDURE ChangeConfDayLimit

@ConferenceDayID INT,

@NewLimit INT

AS

BEGIN

SET NOCOUNT ON;

UPDATE [Conference Days] SET Limit = @NewLimit

WHERE ConferenceDayID = @ConferenceDayID

END

**ChangeContactAddress**

Create PROCEDURE ChangeContactAddress

@InformationID INT,

@Street NVARCHAR(40),

@Zipcode NVARCHAR(12),

@City NVARCHAR(40),

@Country NVARCHAR(20)

AS

BEGIN

SET NOCOUNT ON;

DECLARE @AddressID INT;

EXEC dbo.AddAddress @Street, @Zipcode, @City, @Country

SELECT @AddressID = MAX(AddressID) FROM Address

UPDATE [Contact Information]

SET AddressID = @AddressID

WHERE InformationId = @InformationID

END

**ChangeParticipantsNumber**

CREATE PROCEDURE ChangeParticipantsNumber

@ReservationID INT,

@NewParticipantsNumber INT,

@NewStudentsNumber INT = NULL

AS

BEGIN

SET NOCOUNT ON;

IF @NewStudentsNumber IS NOT NULL

BEGIN

UPDATE [Conference Reservations]

SET NumberOfStudents = @NewStudentsNumber

WHERE ReservationID = @ReservationID

END

UPDATE [Conference Reservations]

SET ParticipantsNumber = @NewParticipantsNumber

WHERE ReservationID = @ReservationID

END

**ChangeWorkshopLimit**

CREATE PROCEDURE ChangeWorkshopLimit

@WorkshopID INT,

@NewLimit INT

AS

BEGIN

SET NOCOUNT ON;

UPDATE Workshops SET Limit = @NewLimit

WHERE WorkshopID = @WorkshopID

END

**MakeConfDayReservation**

CREATE PROCEDURE MakeConfDayReservation

@ConferenceDayID INT,

@ParticipantsNumber INT = 1,

@NumberOfStudents INT = 0,

@ClientID INT

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO [Conference Reservations](

ConferenceDayID,

ParticipantsNumber,

NumberOfStudents,

ClientID,

ReservationDate,

Cancelled)

VALUES(@ConferenceDayID,

@ParticipantsNumber,

@NumberOfStudents,

@ClientID,

GETDATE(),

0)

END

**MakeWorkshopReservation**

CREATE PROCEDURE MakeWorkshopReservation

@WorkshopID INT,

@ParticipantID INT

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO [Workshop Reservations](

WorkshopID,

ParticipantID,

Cancelled,

ReservationDate

)

VALUES(@WorkshopID,

@ParticipantID,

0,

GETDATE()

)

END

**UncancelConfReservation**

CREATE PROCEDURE UncancelConfReservation

@ReservationID int

AS

BEGIN

SET NOCOUNT ON;

UPDATE [Conference Reservations]

SET Cancelled = 0

WHERE ReservationID = @ReservationID

END

GO

**UncancelWorkshopReservation**

CREATE PROCEDURE UncancelWorkshopReservation

@ReservationID int

AS

BEGIN

SET NOCOUNT ON;

UPDATE [Workshop Reservations]

SET Cancelled = 0

WHERE WorkshopReservationID = @ReservationID

END

GO.

1. **Trigery**

**WatchDayLimit**

Przy zmianie limitu na dzień konferencji sprawdza, czy wszyscy do tej pory zajerestrowani uczestnicy wciąż „mieszczą” się w limicie.

CREATE TRIGGER dbo.WatchDayLimit

ON dbo.[Conference Days]

AFTER UPDATE

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS(SELECT 'YES'

FROM inserted as NEW\_CD

INNER JOIN [Conferences available spaces] AS CD

ON NEW\_CD.ConferenceDayID = CD.ConferenceDayID

WHERE NEW\_CD.Limit < CD.Reserved

)

BEGIN

RAISERROR('You can not reduce the limit

because there are already too much participants enrolled', 1, 1);

ROLLBACK TRANSACTION;

END

END

GO

**WatchWorkshopLimit**

Ta sama funkcjonalność co wyżej tylko dla warsztatów.

CREATE TRIGGER dbo.WatchWorkshopLimit

ON dbo.Workshops

AFTER UPDATE

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS(SELECT 'YES'

FROM inserted as NEW\_W

INNER JOIN [Workshops Available Spaces] AS OLD\_W

ON NEW\_W.WorkshopID = OLD\_W.WorkshopID

WHERE NEW\_W.Limit < OLD\_W.

)

BEGIN

RAISERROR('You can not reduce the limit

because there are already too much participants enrolled', 1, 1);

ROLLBACK TRANSACTION;

END

END

GO

**WatchDayReservationUpdate**

Pilnuje aby przy zmianie osób w rezerwacji nie przekroczyć liczby osób już zapisanych.

CREATE TRIGGER dbo.WatchDayReservationUpdate

ON dbo.[Conference Reservations]

AFTER UPDATE

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS(SELECT NEW\_RES.ReservationID

FROM inserted as NEW\_RES

INNER JOIN Participants AS P

ON NEW\_RES.ReservationID = P.ReservationID

GROUP BY NEW\_RES.ReservationID

HAVING COUNT(P.ParticipantID) > NEW\_RES.ParticipantsNumber

)

BEGIN

RAISERROR('You can not reduce Particiapnts Number

as you already added too much participants to reservation', 1, 1);

ROLLBACK TRANSACTION;

END

IF EXISTS(SELECT NEW\_RES.ReservationID

FROM inserted as NEW\_RES

INNER JOIN Participants AS PART

ON NEW\_RES.ReservationID = PART.ReservationID

INNER JOIN People AS P

ON PART.PersonID = P.PersonId AND P.Student = 1

GROUP BY NEW\_RES.ReservationID

HAVING COUNT(PART.ParticipantID) > NEW\_RES.NumberOfStudents

)

BEGIN

RAISERROR('You can not reduce NumberOfStudents

as you already added too much students to reservation', 1, 1);

ROLLBACK TRANSACTION;

END

END

GO

**WatchConfDayLimitExceed**

Sprawdza czy jest wystarczająca ilość wolnych miejsc na dzień konferencji.

CREATE TRIGGER dbo.WatchConfDayLimitExceed

ON dbo.[Conference Reservations]

AFTER INSERT, UPDATE

AS

BEGIN

SET NOCOUNT ON;

IF EXISTS(SELECT SUM(IIF(I.ReservationID IS NULL, CR.ParticipantsNumber, I.ParticipantsNumber))

FROM [Conference Reservations] as CR

LEFT OUTER JOIN inserted as I

ON CR.ReservationID = I.ReservationID

INNER JOIN [Conference Days] as CD

ON CR.ConferenceDayID = CD.ConferenceDayID

GROUP BY CD.ConferenceDayID, CD.Limit

HAVING SUM(IIF(I.ReservationID IS NULL, CR.ParticipantsNumber, I.ParticipantsNumber)) > CD.Limit

)

BEGIN

RAISERROR('You can not make a reservation because

there are no enough free spaces on this day', 1, 1);

ROLLBACK TRANSACTION;

END

END

GO

**WatchPayments**

Zapobiega dodawaniu płatności jeśli rezerwacja jest już całkowicie zapłacona.

CREATE TRIGGER dbo.WatchPayments

ON dbo.Payments

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

DECLARE @IsConference BIT;

SELECT @IsConference = IsConference FROM inserted

IF (@IsConference = 1)

BEGIN

IF EXISTS(SELECT 'YES'

FROM inserted as I

INNER JOIN [Conf Reservations Payments Balance] as CB

ON I.ReservationId = CB.ReservationID

WHERE I.Payment > 0 AND CB.ToPay < 0

)

BEGIN

RAISERROR('You can not pay for the reservation which

is already fully paid', 1, 1);

ROLLBACK TRANSACTION;

END

END

ELSE

BEGIN

IF EXISTS(SELECT 'YES'

FROM inserted as I

INNER JOIN [Workshop Reservations Payments Balance] as WB

ON I.ReservationId = WB.WorkshopReservationID

WHERE I.Payment > 0 AND WB.ToPay < 0

)

BEGIN

RAISERROR('You can not pay for the reservation which

is already fully paid', 1, 1);

ROLLBACK TRANSACTION;

END

END

END

GO

**WatchWorkshopReservation**

Sprawdza, czy participant może złożyć rezerwację na warsztat.

CREATE TRIGGER dbo.WatchWorkshopReservation

ON dbo.[Workshop Reservations]

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

DECLARE @WorkshopID INT;

DECLARE @ParticipantID INT;

SELECT @WorkshopID = WorkshopID, @ParticipantID = ParticipantID FROM inserted

IF dbo.CanRegisterForWorkshop(@ParticipantID, @WorkshopID) = 0

BEGIN

RAISERROR('Participant can not register for this workshop', 1, 1);

ROLLBACK TRANSACTION;

END

IF @ParticipantID in (SELECT WR.ParticipantID

FROM Workshops AS W

INNER JOIN [Workshop Reservations] AS WR

ON W.WorkshopID = @WorkshopID AND W.WorkshopID = WR.WorkshopID

WHERE WR.Cancelled = 0

)

BEGIN

RAISERROR('Participant is already registred for workshop', 1, 1);

ROLLBACK TRANSACTION;

END

IF EXISTS( SELECT 'YES'

FROM [Workshops Available Spaces] as W

WHERE W.WorkshopID = @WorkshopID AND W.Free > 0

)

BEGIN

RAISERROR('Workshop is full', 1, 1);

ROLLBACK TRANSACTION;

END

END

GO

**WatchAddedParticipants**

Sprawdza czy można dodać osobę jako uczestnika do rezerwacji.

CREATE TRIGGER dbo.WatchAddedParticipants

ON dbo.Participants

AFTER INSERT

AS

BEGIN

SET NOCOUNT ON;

DECLARE @PersonID INT;

DECLARE @ReservationID INT;

DECLARE @ConferenceDayID INT;

SELECT @PersonID = PersonID, @ReservationID = ReservationID

FROM inserted

SELECT @ConferenceDayID = ConferenceDayID

FROM [Conference Reservations] as CR

WHERE CR.ReservationID = @ReservationID

IF @PersonID in (SELECT PART.PersonID

FROM [Conference Days] as CD

INNER JOIN [Conference Reservations] as CR

ON CD.ConferenceDayID = @ConferenceDayID

AND CD.ConferenceDayID = CR.ConferenceDayID

INNER JOIN Participants as PART

ON CR.Cancelled = 0 AND CR.ReservationID = PART.ReservationID

)

BEGIN

RAISERROR('Person is already registred for this conference day', 1, 1);

ROLLBACK TRANSACTION;

END

IF EXISTS(SELECT 'YES'

FROM [Conference Reservations] AS CR

INNER JOIN Participants as PART

ON CR.ReservationID = @ReservationID AND CR.ReservationID = PART.ReservationID

GROUP BY CR.ReservationID, CR.ParticipantsNumber

HAVING CR.ParticipantsNumber <= COUNT(PART.ParticipantID)

)

BEGIN

RAISERROR('All participants are already added to reservation', 1, 1);

ROLLBACK TRANSACTION;

END

IF EXISTS(SELECT 'YES'

FROM [Conference Reservations] AS CR

INNER JOIN Participants as PART

ON CR.ReservationID = @ReservationID AND CR.ReservationID = PART.ReservationID

INNER JOIN People as P

ON PART.PersonID = P.PersonID AND P.Student = 1

GROUP BY CR.ReservationID, CR.ParticipantsNumber

HAVING CR.NumberOfStudents <= COUNT(PART.ParticipantID)

)

BEGIN

RAISERROR('All participants are already added to reservation', 1, 1);

ROLLBACK TRANSACTION;

END

END

GO

1. **Indeksy**
2. **Abc**
3. **Role**

Proponowany przez nas rozkład ról w systemie bazy danych:

* **Administrator** – pełen dostęp do bazy (wszystkie tabele, widoki, procedury)
* **Pracownik** – dostęp do wszystkich widokow oraz procedur i funkcji
* **Organizator** – dostęp do procedur tworzących i edytujących dane organizatora / dane konferencji i dni konferencji. Ponadto dostęp do widoków i procedur pokazujących stan płatności klientów oraz raporty klientów, którzy najczęściej korzystają z jego usług.
* **Klient** – Może tworzyć/anulować rezerwacje na konferencję, dodawać swoje informacje kontaktowe (tworzyć profil klienta). Może wyświetlić status jego rezerwacji, płatności, zalegających należności, może zobaczyć nadchodzące i aktualne konferencje, podać dane uczestników konferencji do rezerwacji.
* **Uczestnik konferencji** – Może sprawdzać na co jest zapisany, rezerwować warsztaty, oglądać status jego płatności za warsztaty, edytować własne dane. Może także stać się klientem (wtedy dochodzą wszystkie funkcje klienta).