**Podstawy Baz Danych**

**Projekt: System zarządzania konferencjami**

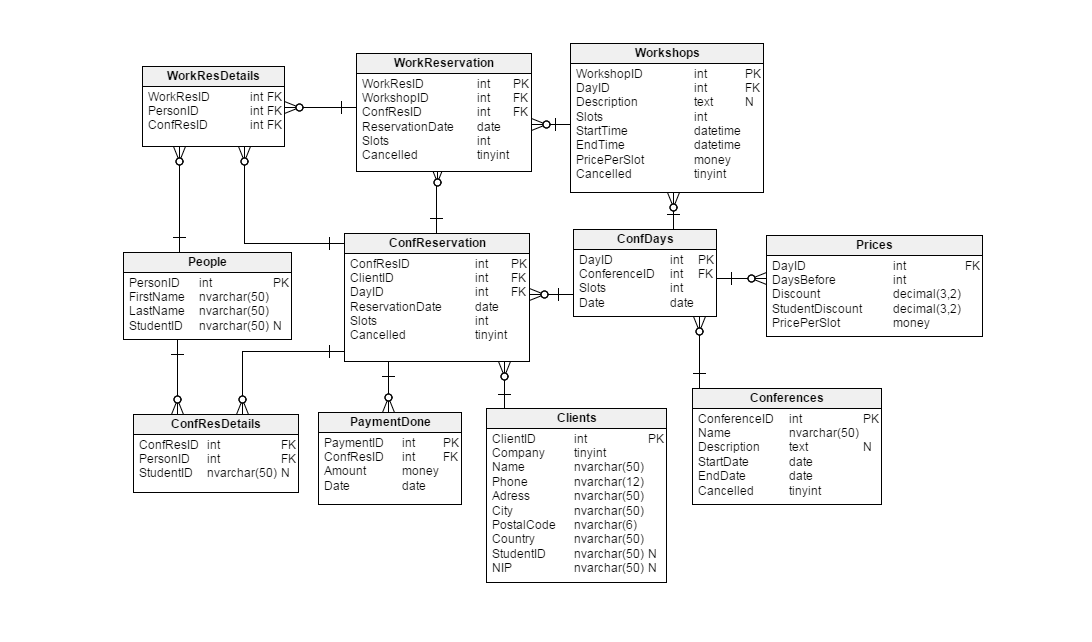
Artur Czopek

Mateusz KASznia

1. Opis:

Firma organizuje konferencje, które mogą być jedno­ lub kilkudniowe. Klientami mogą być zarówno indywidualne osoby jak i firmy, natomiASt uczestnikami konferencji są osoby (firma nie musi podawać od razu przy rejestracji listy uczestników ­ może zarezerwować odpowiednią ilość miejsc na określone dni oraz na warsztaty, natomiASt na 2 tygodnie przed rozpoczęciem musi te dane uzupełnić). Dla konferencji kilkudniowych, uczestnicy mogą rejestrować na dowolne z tych dni, dowolną liczbę osób. Klient może zmienić liczbę osób na rezerwacji, lub całkiem ją anulować (do 2 tygodni przed konferencją).

2. Schemat:



3. Opis tabel:

* **Clients** ­

Tabela reprezentująca klientów w bazie danych. Każdy klient posiada dane: (Company, Name, Phone, Adress, City, PostalCode, Country, StudentID, NIP). Jeśli klient jest firmą to pole (Company) ustawione jest na 1, a w polu (Name) znajduje się nazwa firmy, dla klienta będącego osobą prywatną (Firma) ustawiona jest na 0, telefon może składać się tylko ze znaków numerycznych (cyfry i znaki typu +/­), kod musi być typu \*\*­-\*\*\* (gdzie \* to cyfra).

CREATE TABLE Clients (

ClientID int NOT NULL IDENTITY(1,1),

Company tinyint NOT NULL CHECK(Clients.Company LIKE '[0-1]'),

Name nvarchar(50) NOT NULL,

Phone nvarchar(50) NOT NULL CHECK(Clients.Phone LIKE

'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),

Adress nvarchar(50) NOT NULL,

City nvarchar(50) NOT NULL,

PostalCode nvarchar(50) NOT NULL CHECK(Clients.PostalCode LIKE

'[0-9][0-9]-[0-9][0-9][0-9]'),

Country nvarchar(50) NOT NULL,

StudentID nvarchar(50) NULL,

NIP nvarchar(50) NULL,

CONSTRAINT Clients\_pk PRIMARY KEY (ClientID)

);

* **Conferences**

Tabela do której wpisywane są konferencje. Konferencja posiada dane:

(Name, Description, StartDate, EndDate, Cancelled). Konferencja może zostać anulowana (pole ‘Cancelled’ ustawiane wtedy na 1).

CREATE TABLE Conferences (

ConferenceID int NOT NULL IDENTITY(1,1),

Name nvarchar(50) NOT NULL,

Description varchar(255) NULL,

StartDate date NOT NULL CHECK (StartDate > GETDATE()),

EndDate date NOT NULL CHECK (EndDate > GETDATE()),

Cancelled tinyint NOT NULL DEFAULT 0,

CONSTRAINT Conferences\_pk PRIMARY KEY (ConferenceID),

CONSTRAINT StartDateEndDate\_C CHECK (StartDate <= EndDate)

);

* **ConfDays**

Tabela do której wpisywane są poszczególne dni konferencji. Dzień konferencji odwołuje się do konkretnej konferencji (ConferenceID), ma swoją datę (Date), oraz ilość miejsc (Slots). Warunki: ilość miejsc musi być większa niż 0.

CREATE TABLE ConfDays (

DayID int NOT NULL IDENTITY(1,1),

ConferenceID int NOT NULL,

Slots int CHECK (Slots > 0) NOT NULL,

Date date NOT NULL,

CONSTRAINT ConfDays\_pk PRIMARY KEY (DayID)

);

* **Prices**

Tabela reprezentująca różne progi cenowe na dni konferencji. Cena przypisana jest do dnia konferencji (przez DayID). Cena jednego biletu na dany dzień jest w polu (PricePerSlot). Tabela ta zawiera również zniżkę (Discount) jeśli dokonamy rezerwacji odpowiednio wcześniej (DaysBefore) oraz zniżkę dla studentów (StudentDiscount) Warunki: Cena, zniżka, oraz ilość dni przed nie mogą być ujemne.

CREATE TABLE Prices (

DayID int NOT NULL,

DaysBefore int NOT NULL CHECK (DaysBefore >= 0),

Discount decimal(3,2) CHECK (Discount >= 0) NOT NULL,

StudentDiscount decimal(3,2) CHECK(StudentDiscount >= 0) NOT NULL,

PricePerSlot decimal(15,2) CHECK (PricePerSlot >= 0) NOT NULL

);

* **People**

Tabela reprezentująca osoby. Posiada pola: (FirstName, LAStName, StudentID). Jeśli osoba nie jest studentem, pole StudentID ustawiamy na NULL.

CREATE TABLE People (

PersonID int NOT NULL IDENTITY(1,1),

FirstName nvarchar(50) NOT NULL,

LAStName nvarchar(50) NOT NULL,

StudentID nvarchar(50) NULL,

CONSTRAINT People\_pk PRIMARY KEY (PersonID)

);

* **ConfReservation**

Tabela reprezentująca rezerwacje klientów na dany dzień konferencji. Podpięta jest pod dzień konferencji (DayID), dotyczy klienta (ClientID), i rezerwuje liczbę miejsc (Slots). Posiada datę rezerwacji, oraz pole (Cancelled) ustawione odpowiednio na 0 lub 1. Warunki: pole Slots nie może być ujemne.

CREATE TABLE ConfReservation (

ConfResID int NOT NULL IDENTITY(1,1),

ClientID int NOT NULL,

DayID int NOT NULL,

ReservationDate DATETIME NOT NULL,

Slots int CHECK(Slots > 0) NOT NULL,

Cancelled tinyint NOT NULL DEFAULT 0,

CONSTRAINT ConfReservation\_pk PRIMARY KEY (ConfResID)

);

* **Workshops**

­Tabela reprezentująca warsztat. Warsztat posiada opis (Description), odbywa się w przedziale czASowym (StartTime, EndTime), w dniu konferencji (DayID), maksymalnie może w nim uczestniczyć ilość osób, określona przez (Slots). Koszt uczestnictwa w warsztacie to (PricePerSlot). Warsztat może zostać anulowany (Cancelled).

Warunki: Ilość miejsc nie może być ujemna, data rozpoczęcia i zakończenia późniejsza od obecnej daty, cena nie może być ujemna, data zakończenia późniejsza niż data rozpoczęcia.

CREATE TABLE Workshops (

WorkshopID int NOT NULL IDENTITY(1,1),

DayID int NOT NULL,

Description varchar(255) NULL,

Slots int NOT NULL CHECK (Slots > 0),

StartTime datetime NOT NULL CHECK (StartTime > GETDATE()),

EndTime datetime NOT NULL CHECK (EndTime > GETDATE()),

PricePerSlot decimal(15,2) CHECK (PricePerSlot >= 0) NOT NULL,

Cancelled tinyint NOT NULL DEFAULT 0,

CONSTRAINT Workshops\_pk PRIMARY KEY (WorkshopID),

CONSTRAINT StartTimeEndTime\_W CHECK (StartTime < EndTime)

);

* **WorkReservation**

Tabela reprezentująca rezerwacje klientów na dany warsztat. Podpięta jest pod rezerwację konferencji (ConfResID). Dotyczy warsztatu (WorkshopID), i rezerwuje liczbę miejsc (Slots). Rezerwacja warsztatu może zostać anulowana (Cancelled).

Warunki: Liczba miejsc nie może być ujemna.

CREATE TABLE WorkReservation (

WorkResID int NOT NULL IDENTITY(1,1),

WorkshopID int NOT NULL,

ConfResID int NOT NULL,

ReservationDate DATETIME NOT NULL,

Slots int NOT NULL CHECK (Slots > 0),

Cancelled tinyint NOT NULL DEFAULT 0,

CONSTRAINT WorkReservation\_pk PRIMARY KEY (WorkResID)

);

* **PaymentDone**

Tabela zawierająca informacje o wpłatach. Zawiera (ConfResID) której dotyczy, oraz wartość wpłaty (Amount) i datę (Date). Warunki: Wartość wpłaty nie może być ujemna.

CREATE TABLE PaymentDone (

PaymentID int NOT NULL IDENTITY(1,1),

ConfResID int NOT NULL,

Amount decimal(15,2) CHECK (PaymentDone.Amount > 0) NOT NULL,

Date date NOT NULL,

CONSTRAINT PaymentDone\_pk PRIMARY KEY (PaymentID)

);

* **ConfResDetails**

Tabela łącząca konkretne osoby (PersonID) z poszczególną rezerwacją (ConfResID).

CREATE TABLE ConfResDetails (

ConfResID int NOT NULL,

PersonID int NOT NULL,

StudentID nvarchar(50) NULL

);

* **WorkResDetails**

Tabela łącząca konkretne osoby (PersonID) z poszczególnej rezerwacji (ConfResID) z rezerwacją warsztatu (WorkResID).

CREATE TABLE WorkResDetails (

WorkResID int NOT NULL,

PersonID int NOT NULL,

ConfResID int NOT NULL

);

4. Widoki:

* **Niepełne zgłoszenia**

Pokazuje zgłoszenia które jeszcze nie są wypełnione

CREATE VIEW Niepelne\_zgloszenia

AS

SELECT cr.ConfResID, c.Name AS [Nazwa klienta], c.Phone,

conf.Name AS [Nazwa Konferencji], cd.Date AS [Data Konferencji],

cr.Slots AS [Ile powinno być],

cr.Slots - (SELECT COUNT(\*) FROM ConfResDetails AS crd WHERE

crd.ConfResID = cr.ConfResID) AS [Ilu brakuje]

FROM ConfReservation AS cr

JOIN Clients AS c

ON cr.ClientID = c.ClientID

JOIN ConfDays AS cd

ON cr.DayID = cd.DayID

JOIN Conferences AS conf

ON cd.ConferenceID = conf.ConferenceID

WHERE cr.Slots - (SELECT COUNT(\*) FROM ConfResDetails AS crd

WHERE crd.ConfResID = cr.ConfResID) >0

AND cr.Cancelled = 0

GO

* **Najpopularniejsze konferencje (top 20)**

CREATE VIEW Najpopularniejsze\_konferencje

AS

SELECT TOP 20 conf.ConferenceID, conf.Name, COUNT(\*) AS [Ilosc chetnych]

FROM Conferences AS conf

LEFT JOIN ConfDays AS cd

ON conf.ConferenceID = cd.ConferenceID

LEFT JOIN ConfReservation AS cr

ON cr.DayID = cd.DayID

JOIN ConfResDetails AS crd

ON crd.ConfResID = cr.ConfResID

GROUP BY conf.ConferenceID, conf.Name

GO

* **Najpopularniejsze warsztaty (top 20)**

CREATE VIEW Najpopularniejsze\_warsztaty

AS

SELECT TOP 20 work.WorkshopID, work.Description, work.StartTime, work.EndTime, COUNT(\*) AS [Ilosc chetnych]

FROM Workshops AS work

LEFT JOIN WorkReservation AS wr

ON work.WorkshopID = wr.WorkshopID

JOIN WorkResDetails AS wrd

ON wr.WorkResID = wrd.WorkResID

GROUP BY work.WorkshopID, work.Description, work.StartTime, work.EndTime

GO

* **Najczęściej korzystający z usług**

CREATE VIEW Najczesciej\_korzystajacy\_z\_uslug

AS

SELECT Name AS [Client],

(SELECT COUNT(\*)

FROM ConfReservation

WHERE (Clients.ClientID = ClientID) AND ConfReservation.Cancelled = 0) AS [How many times:]

FROM Clients

GROUP BY ClientID, Name

GO

* **Anulowane konferencje**

CREATE VIEW Anulowane\_konferencje

AS

SELECT \*

FROM Conferences AS c

WHERE c.Cancelled=1

GO

* **Anulowane rezerwacje na konferencje**

CREATE VIEW Anulowane\_konf\_rezerwacje

AS

SELECT \*

FROM ConfReservation AS cr

WHERE cr.Cancelled=1

GO

* **Anulowane warsztaty**

CREATE VIEW Anulowane\_warsztaty

AS

SELECT \*

FROM Workshops AS w

WHERE w.Cancelled=1

GO

* **Anulowane rezerwacje na warsztaty**

CREATE VIEW Anulowane\_wars\_rezerwacje

AS

SELECT \*

FROM WorkReservation AS wr

WHERE wr.Cancelled=1

GO

* **Klienci prywatni**

CREATE VIEW Klienci\_prywatni

AS

SELECT \*

FROM Clients AS c

WHERE c.Company=0

GO

* **Firmy**

CREATE VIEW Klienci\_firmowi

AS

SELECT \*

FROM Clients AS c

WHERE c.Company=1

GO

5. Funkcje, Procedury i Triggery:

* **Czy rezerwacja konferencji jest ok**

Sprawdza czy ilość osób nie została przekroczona

CREATE TRIGGER Czy\_konf\_rez\_ok

ON ConfReservation

AFTER INSERT,UPDATE AS

BEGIN

IF EXISTS (SELECT [TAK]

FROM inserted

JOIN ConfDays AS cd

ON inserted.DayId =cd.DayId

JOIN ConfReservation AS cr ON cd.DayID = cr.DayID

WHERE inserted.DayID = cr.DayID

GROUP BY cr.DayID, cd.Slots

HAVING SUM(cr.Slots) > cd.Slots

)

BEGIN

RAISERROR ('Nie mozna wpisac wiecej osob na ta konferencje. Brak miejsc', 16, 1)

ROLLBACK TRANSACTION

END

END

* **Czy rezerwacja warsztatu jest ok**

Sprawdza czy ilość osób nie została przekroczona

CREATE TRIGGER Czy\_wars\_rez\_ok

ON WorkReservation

AFTER INSERT,UPDATE AS

BEGIN

IF EXISTS (SELECT [TAK]

FROM inserted

JOIN Workshops AS w

ON inserted.WorkshopID = w.WorkshopID

JOIN WorkReservation AS wr ON wr.WorkshopID = w.WorkshopID

WHERE inserted.WorkshopID = wr.WorkshopID

GROUP BY wr.WorkshopID, w.Slots

HAVING SUM(wr.Slots) > w.Slots

)

BEGIN

RAISERROR ('Nie mozna wpisac wiecej osob na ten warsztat. Brak miejsc', 16, 1)

ROLLBACK TRANSACTION

END

END

* **Czy osoba jest już na konferencji**

Sprawdza czy dodawana osoba nie jest już zapisana na konferencje

CREATE TRIGGER Czy\_osoba\_jest\_konf

ON ConfResDetails

AFTER INSERT,UPDATE AS

BEGIN

IF (SELECT COUNT(\*)

FROM inserted

JOIN ConfReservation AS cr

ON inserted.ConfResID = cr.ConfResID

JOIN ConfResDetails AS crd

ON crd.ConfResID = cr.ConfResID

WHERE inserted.PersonID = crd.PersonID

)>1

BEGIN

RAISERROR ('Osoba juz jest na konferencji', 16, 1)

ROLLBACK TRANSACTION

END

END

* **Czy osoba zapisująca się na warsztat jest na konferencji**

Sprawdza czy dodawana do warsztatu osoba jest zapisana na odpowiednią konferencję

CREATE TRIGGER czy\_osoba\_wars\_jest\_konf

ON WorkResDetails

AFTER INSERT,UPDATE AS

BEGIN

IF (SELECT COUNT(\*)

FROM inserted AS wrdi

JOIN WorkReservation AS ws

ON wrdi.WorkResID = ws.WorkResID

JOIN

Workshops AS w ON w.WorkshopID = ws.WorkshopID

JOIN

ConfReservation AS cr ON cr.ConfResID=ws.ConfResID

JOIN

ConfResDetails AS crd ON cr.ConfResID = crd.ConfResID

WHERE wrdi.PersonID=crd.PersonID

)=0

BEGIN

RAISERROR ('Osoba nie jest zapisana na konferencje na ktorej jest ten warsztat.', 16, 1)

ROLLBACK TRANSACTION

END

END

GO

* **Czy osoba chcąca zapisać się na warsztat nie bierze w tym czasie udziału w innym**

CREATE TRIGGER Czy\_nie\_koliduja\_warsztaty

ON WorkResDetails

AFTER INSERT,UPDATE AS

BEGIN

IF( SELECT COUNT(\*)

FROM inserted AS iwrd

JOIN WorkReservation AS wr

ON wr.WorkResID = iwrd.WorkResID

JOIN Workshops AS w

ON w.WorkshopID = wr.WorkshopID

JOIN WorkResDetails AS wrd

ON iwrd.PersonID = wrd.PersonID

JOIN WorkReservation AS wr2

ON wrd.WorkResID = wr2.WorkResID

JOIN Workshops AS w2

ON w2.WorkshopID = wr2.WorkshopID

WHERE (w.StartTime < w2.EndTime AND w.EndTime > w2.StartTime)

)>1

BEGIN

RAISERROR ('Ta osoba bierze już udział w innym warsztacie w tych godzinach.', 16, 1)

ROLLBACK TRANSACTION

END

END

* **Osoby zapisane na dany dzień konferencji**

CREATE PROCEDURE Osoby\_dzien\_konf

(

@dayid int

)

AS

SELECT p.PersonID, p.FirstName, p.LAStName, p.StudentID FROM People AS p

JOIN ConfResDetails AS crd

ON p.PersonID = crd.PersonID

JOIN ConfReservation AS cr

ON cr.ConfResID = crd.ConfResID

WHERE cr.DayID = @dayid AND cr.Cancelled = 0

GO

* **Osoby zapisane na dany warsztat**

CREATE PROCEDURE Osoby\_warsztat

(

@dayid int

)

AS

SELECT p.PersonID, p.FirstName, p.LAStName, p.StudentID FROM People AS p

JOIN ConfResDetails AS crd

ON p.PersonID = crd.PersonID

JOIN WorkResDetails AS wrd

ON crd.ConfResID = wrd.WorkResID

JOIn WorkReservation AS wr

ON wrd.WorkResID = wr.WorkResID

WHERE wr.WorkShopID = @dayid AND wr.Cancelled = 0

GO

* **Generuj ID**

Generuje ID dla osób na dany dzień konferencji

CREATE PROCEDURE Generuj\_id

(

@dayid int

)

AS

SELECT p.FirstName, p.LAStName,

CAST(cr.DayID AS nvarchar(5)) + '-' + CAST(cr.ConfResID AS nvarchar(5)) + '-' + CAST(p.PersonID AS nvarchar(5)) + SUBSTRING(p.FirstName, 0,5) + SUBSTRING(p.LAStName,0,5) + '-' + SUBSTRING(c.Name, 0,4) AS [Identyfikator]

FROM People AS p

JOIN ConfResDetails AS crd

ON p.PersonID = crd.PersonID

JOIN ConfReservation AS cr

ON crd.ConfResID = cr.ConfResID

JOIN Clients AS c

ON cr.ClientID = c.ClientID

WHERE cr.DayID = @dayid

GO

* **Moje Konferencje**

Generuje dla danej osoby listę

CREATE PROCEDURE Moje\_konferencje

(

@personid int

)

AS

SELECT p.FirstName AS [Imie], p.LAStName AS [Nazwisko], c.Name AS [Nazwa konferencji]

FROM People AS p

JOIN ConfResDetails AS crd

ON p.PersonId= crd.PersonID

JOIN ConfReservation AS cr

ON crd.ConfResID = cr.ConfResID

JOIN ConfDays AS cd

ON cr.DayID = cd.DayID

JOIN Conferences AS c

ON cd.ConferenceID = c.ConferenceID

WHERE p.PersonID = @personid

GO

* **Moje Warsztaty**

CREATE PROCEDURE Moje\_warsztaty

(

@personid int

)

AS

SELECT p.FirstName AS [Imie], p.LAStName AS [Nazwisko], w.Description AS [Nazwa Warsztatu]

FROM People AS p

JOIN WorkResDetails AS wrd

ON p.PersonID = wrd.PersonID

JOIN WorkReservation AS wr

ON wrd.WorkResId = wr.WorkResId

JOIN Workshops AS w

ON wr.WorkshopID = w.WorkshopID

WHERE p.PersonID = @personid

GO

* **Płatności konferencja**

Wyświetla płatności dla danej konferencji

CREATE PROCEDURE Platnosci\_konferencja

(

@conferenceid int

)

AS

SELECT c.Name AS [Nazwa klienta], conf.Name AS [Nazwa konferencji], cd.Date AS [Dzień konferencji], SUM(pd.Amount) AS [Płatności Dokonane]

FROM Conferences AS conf

JOIN ConfDays AS cd

ON conf.ConferenceId = cd.ConferenceID

JOIN ConfReservation AS cr

ON cd.DayID = cr.DayID

JOIN Clients AS c

ON cr.ClientID = c.ClientID

JOIN PaymentDone AS pd

ON cr.ConfResID = pd.ConfResID

WHERE conf.ConferenceID = @ conferenceid AND cr.Cancelled = 0

GROUP BY c.Name, conf.Name, cd.Date

GO

* **Płatności rezerwacja**

Wyświetla płatności dla danej rezerwacji

CREATE PROCEDURE Platnosci\_rezerwacja

(

@reservationid int

)

AS

SELECT c.Name AS [Nazwa klienta], conf.Name AS [Nazwa konferencji], cd.Date AS [Dzień konferencji], SUM(pd.Amount) AS [Płatności Dokonane]

FROM Conferences AS conf

JOIN ConfDays AS cd

ON conf.ConferenceId = cd.ConferenceID

JOIN ConfReservation AS cr

ON cd.DayID = cr.DayID

JOIN Clients AS c

ON cr.ClientID = c.ClientID

JOIN PaymentDone AS pd

ON cr.ConfResID = pd.ConfResID

WHERE cr.ConfResID = @reservationid ANd cr.Cancelled = 0

GROUP BY c.Name, conf.Name, cd.Date

GO

* **Progi cenowe**

Wyświetla progi cenowe dla danej konferencji

CREATE PROCEDURE Progi\_cenowe

(

@confid int

)

AS

SELECT pr.DayID, pr.DaysBefore, FORMAT(pr.PricePerSlot\*(1-pr.Discount), 'N', 'en-us') AS [Cena normalna],

FORMAT(pr.PricePerSlot\*(1-pr.StudentDiscount), 'N', 'en-us') AS [Cena studencka] FROM Prices AS pr

JOIN ConfDays AS cd

ON pr.DayID = cd.DayID

WHERE cd.ConferenceID = @confid

GO

* **Osoby z danej firmy/osoby prywatnej**

Wyświetla osoby z firmy o podanym ID

CREATE PROCEDURE Osoby\_firma

(

@clientid int

)

AS

SELECT p.PersonID, p.FirstName, p.LAStName, p.StudentID

FROM People AS p

JOIN ConfResDetails AS crd

ON p.PersonID = crd.PersonID

JOIN ConfReservation AS cr

ON crd.ConfResID = cr.ConfResID

JOIN Clients AS c

ON cr.ClientID = c.ClientID

WHERE c.ClientID = @clientid

GO

* **Dodaj prywatnego klienta**

Imię, tel, adres, miASto, kod, kraj, studentid

CREATE PROCEDURE Dodaj\_klient\_prywatny

(

@name nvarchar(50),

@phone nvarchar(12),

@adres nvarchar(50),

@city nvarchar(50),

@postcode nvarchar(6),

@country nvarchar(50),

@studentid nvarchar(50)

)

AS

INSERT INTO Clients VALUES(0, @name, @phone, @adres, @city, @postcode, @country, @studentid, NULL)

GO

* **Dodaj firmę**

Nazwa, tel, adres, miASto, kod, kraj, nip

CREATE PROCEDURE Dodaj\_klient\_firma

(

@name nvarchar(50),

@phone nvarchar(12),

@adres nvarchar(50),

@city nvarchar(50),

@postcode nvarchar(6),

@country nvarchar(50),

@nip nvarchar(50)

)

AS

INSERT INTO Clients VALUES(1, @name, @phone, @adres, @city, @postcode, @country, NULL, @nip)

GO

* **Zmień dane klienta prywatnego**

CREATE PROCEDURE Zmien\_dane\_prywatny

(

@id int,

@name nvarchar(50)=NULL,

@phone nvarchar(12)=NULL,

@adres nvarchar(50)=NULL,

@city nvarchar(50)=NULL,

@postcode nvarchar(6)=NULL,

@country nvarchar(50)=NULL,

@studentid nvarchar(50)=NULL

)

AS

IF @name IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Name = @name

WHERE Clients.ClientID = @id AND Clients.Company = 0

END

IF @phone IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Phone = @phone

WHERE Clients.ClientID = @id AND Clients.Company = 0

END

IF @adres IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Adress = @adres

WHERE Clients.ClientID = @id AND Clients.Company = 0

END

IF @city IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.City = @city

WHERE Clients.ClientID = @id AND Clients.Company = 0

END

IF @postcode IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.PostalCode = @postcode

WHERE Clients.ClientID = @id AND Clients.Company = 0

END

IF @country IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Country = @country

WHERE Clients.ClientID = @id AND Clients.Company = 0

END

IF @studentid IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.StudentID = @studentid

WHERE Clients.ClientID = @id AND Clients.Company = 0

END

GO

* **Zmień dane firmy**

CREATE PROCEDURE Zmien\_dane\_firma

(

@id int,

@name nvarchar(50)=NULL,

@phone nvarchar(12)=NULL,

@adres nvarchar(50)=NULL,

@city nvarchar(50)=NULL,

@postcode nvarchar(6)=NULL,

@country nvarchar(50)=NULL,

@nip nvarchar(50)=NULL

)

AS

IF @name IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Name = @name

WHERE Clients.ClientID = @id AND Clients.Company = 1

END

IF @phone IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Phone = @phone

WHERE Clients.ClientID = @id AND Clients.Company = 1

END

IF @adres IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Adress = @adres

WHERE Clients.ClientID = @id AND Clients.Company = 1

END

IF @city IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.City = @city

WHERE Clients.ClientID = @id AND Clients.Company = 1

END

IF @postcode IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.PostalCode = @postcode

WHERE Clients.ClientID = @id AND Clients.Company = 1

END

IF @country IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.Country = @country

WHERE Clients.ClientID = @id AND Clients.Company = 1

END

IF @nip IS NOT NULL

BEGIN

UPDATE Clients

SET Clients.NIP = @nip

WHERE Clients.ClientID = @id AND Clients.Company = 1

END

GO

* **Dodaj osobę**

Imię, nazwisko, studentid

CREATE PROCEDURE Dodaj\_osoba

(

@firstname nvarchar(50),

@sname nvarchar(50),

@studentid nvarchar(50)

)

AS

INSERT INTO People VALUES(@firstname, @sname, @studentid)

GO

* **Zmień studentID**

IDosoby, IDstudenta

CREATE PROCEDURE Zmien\_student

(

@id int,

@studentid nvarchar(50)

)

AS

UPDATE People

SET People.StudentID = @studentid

WHERE People.PersonID = @id

GO

* **Dodaj konferencję**

Nazwa, opis, start, koniec, ilość miejsc

CREATE PROCEDURE Dodaj\_konferencje

(

@name nvarchar(50),

@desc text,

@start date,

@end date,

@slots int

)

AS

INSERT INTO Conferences VALUES (@name, @desc, @start, @end,0);

DECLARE @days int = DATEDIFF(DAY, @start, @end) + 1;

DECLARE @cnt int = 0;

DECLARE @currentDate date = @start;

DECLARE @confId int = (SELECT ConferenceID FROM Conferences WHERE Name = @name AND StartDate = @start AND EndDate = @end);

WHILE @cnt < @days

BEGIN

INSERT INTO ConfDays VALUES(@confId, @slots, @currentDate);

SET @currentDate = DATEADD(DAY, 1, @currentDate);

SET @cnt = @cnt + 1;

END;

GO

* **Zmień ilość miejsc na konferencji**

CREATE PROCEDURE zmien\_ilosc\_miejsc\_konf

(

@dayid int,

@slots int

)

AS

UPDATE ConfDays

SET ConfDays.Slots = @slots

WHERE ConfDays.DayID = @dayid

GO

* **Dodaj rezerwacje na konferencję**

KlientID, DayID, Miejsca

CREATE PROCEDURE Dodaj\_rezerwacja\_konf

(

@clientid int,

@dayid int,

@slots int

)

AS

INSERT INTO ConfReservation VALUES (@clientid, @dayid, GETDATE(), @slots, 0)

GO

* **Dodaj osobę do rezerwacji na konferencję**

IDKonferencji, IDOsoby

CREATE PROCEDURE Dodaj\_osoba\_do\_rez\_konf

(

@confresid int,

@personid int

)

AS

INSERT INTO ConfResDetails VALUES(@confresid, @personid, (SELECT StudentID FROM People WHERE PersonID = @personid))

GO

* **Dodaj próg cenowy**

IDDnia, ile dni przed, zniżka studencka, cena normalna

CREATE PROCEDURE Dodaj\_prog\_cenowy

(

@dayid int,

@daysbefore int,

@disc decimal(3,2),

@studentdisc decimal(3,2),

@price money

)

AS

INSERT INTO Prices VALUES(@dayid, @daysbefore, @disc, @studentdisc, @price)

GO

* **Dodaj warsztat**

IDDnia, opis, miejsca, start, koniec, cena

CREATE PROCEDURE Dodaj\_warsztat

(

@dayid int,

@desc text,

@slots int,

@starttime datetime,

@endtime datetime,

@price money

)

AS

INSERT INTO Workshops VALUES(@dayid, @desc, @slots, @starttime, @endtime, @price, 0)

GO

* **Dodaj rezerwacje na warsztat**

IDwarsztatu, IDRezerwacji na konferencję , Miejsca

CREATE PROCEDURE Dodaj\_rezerwacja\_wars

(

@workid int,

@confresid int,

@slots int

)

AS

INSERT INTO WorkReservation VALUES (@workid, @confresid, GETDATE(), @slots, 0)

GO

* **Zmień ilość miejsc na warsztacie**

IDWarsztatu, miejsca

CREATE PROCEDURE Zmien\_miejsca\_wars

(

@workid int,

@slots int

)

AS

UPDATE Workshops

SET Workshops.Slots = @slots

WHERE WorkshopID = @workid

GO

* **Dodaj osobę do rezerwacji na warsztat**

IDRezerwacji, IDOsoby

CREATE PROCEDURE Dodaj\_osoba\_do\_rez\_wars

(

@workresid int,

@personid int

)

AS

INSERT INTO WorkResDetails VALUES(@workresid, @personid, (SELECT ConfResID FROM WorkReservation WHERE WorkResId = @workresid))

GO

* **Anuluj konferencję**

CREATE PROCEDURE Anuluj\_konferencja

(

@confid int

)

AS

--konferencja

UPDATE Conferences

SET Conferences.Cancelled = 1

WHERE ConferenceID = @confid

--rezerwacje konferencja

UPDATE cr

SET cr.Cancelled =1

FROM ConfReservation AS cr

JOIN ConfDays AS cd

ON cr.DayID = cd.DayID

JOIN Conferences AS c

ON cd.ConferenceID = c.ConferenceID

WHERE c.ConferenceID = @confid

--warsztaty

UPDATE w

SET w.Cancelled = 1

FROM WorkShops AS w

JOIN ConfDays AS cd

ON w.DayID = cd.DayID

JOIN Conferences AS c

ON cd.ConferenceID = c.ConferenceID

WHERE c.ConferenceID = @confid

--rezerwacje warsztaty

UPDATE wr

SET wr.Cancelled = 1

FROM WorkReservation AS wr

JOIN ConfReservation AS cr

ON wr.ConfResID = cr.ConfResID

JOIN ConfDays AS cd

ON cr.DayID = cd.DayID

JOIN Conferences AS c

ON cd.ConferenceID = c.ConferenceID

WHERE c.ConferenceID = @confid

GO

* **Anuluj rezerwację na konferencję**

CREATE PROCEDURE Anuluj\_rezerwacja\_konf

(

@confresid int

)

AS

--konferencja rezerwacja

UPDATE ConfReservation

SET ConfReservation.Cancelled = 1

WHERE ConfResID = @confresid

--warsztat rezerwacja

UPDATE WorkReservation

SET WorkReservation.Cancelled = 1

WHERE ConfResID = @confresid

GO

* **Anuluj warsztat**

CREATE PROCEDURE Anuluj\_warsztat

(

@workid int

)

AS

--warsztat

UPDATE Workshops

SET Workshops.Cancelled = 1

WHERE WorkshopID = @workid

--workreservation

UPDATE WorkReservation

SET WorkReservation.Cancelled = 1

WHERE WorkshopID = @workid

GO

* **Anuluj rezerwację na warsztat**

CREATE PROCEDURE Anuluj\_rezerwacja\_warsztat

(

@workresid int

)

AS

UPDATE WorkReservation

SET WorkReservation.Cancelled = 1

WHERE WorkResID = @workresid

GO

* **Lista osób z danej rezerwacji na konferencję**

CREATE PROCEDURE Pokaz\_rezerwacja\_konf

(

@confresid int

)

AS

SELECT p.FirstName, p.LAStName, p.StudentID, cr.Cancelled

FROM People AS p

JOIN ConfResDetails AS crd

ON p.PersonID = crd.PersonID

JOIN ConfReservation AS cr

ON crd.ConfResID = cr.ConfResID

WHERE cr.ConfResID = @confresid

GO

* **Lista osób z danej rezerwacji na warsztat**

CREATE PROCEDURE Pokaz\_rezerwacja\_wars

(

@workresid int

)

AS

SELECT p.FirstName, p.LAStName, p.StudentID, wr.Cancelled

FROM People AS p

JOIN WorkResDetails AS wrd

ON p.PersonID = wrd.PersonID

JOIN WorkReservation AS wr

ON wrd.WorkResID = wr.WorkResID

WHERE wr.WorkResID = @workresid

GO

6. Generator danych:

7. Role:

* ­**Administrator** **bazy**
* **Pracownicy obsługi** (dostęp do widoków i procedur informacyjnych, dostęp do funkcji, obsługa ma udzielać pomocy właścicielowi) ­
* **Właściciel** (może tworzyć konferencje i warsztaty, usuwać/anulować włASne wydarzenia, modyfikować ich dane, dostęp do wszystkich widoków i procedur związanych z utworzonymi konferencjami + wszystko to co klient) ­
* **Klient** (może robić/anulować rezerwacje na konferencje i warsztaty, zgłASzać dowolną ilość osób na nie, korzystać z funkcji mówiącej o jego płatnościach, użyć funkcji zwracającej informację ile jeszcze pozostało do zapłaty za daną konferencję + klient może być uczestnikiem) ­
* **Uczestnik** (może sprawdzać na co jest zapisany, może stać się klientem, korzystać z procedury do edycji włASnych danych)