Podstawy Baz Danych

Projekt: System zarządzania konferencjami

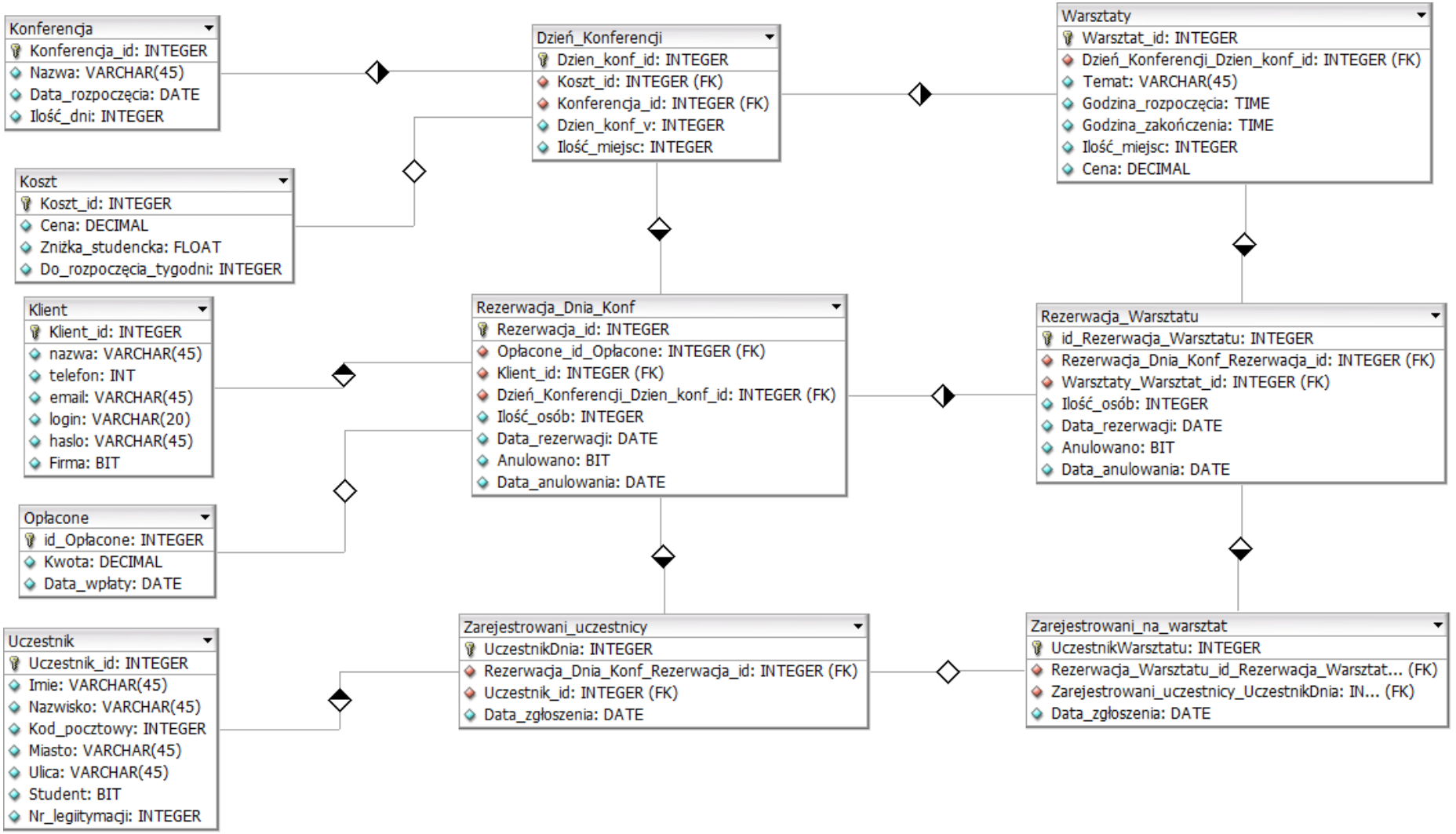
Dokumentacja

Wiktor Łęczyński, Dawid Wełna

1.Opis systemu

System obsługi rezerwacji konferencji . Konferencje mogą być kilkudniowe . Klientami mogą być firmy jak i osoby .Uczestnikami sa osoby i studenci .Klient nie musi przy rezerwacji podawać Ilosci osób , jednak musi je uzupełnić 2 tyg przed terminem.
Opłata musi być zapłacona maksymalnie tydzień przed terminem.
Każda konferencja posiada wyarsztaty . Kilka wykładów mogą trwać jednoczesnie .

2.Schemat


Konferencja
Konferencja_id: INTEGER
Nazwa: VARCHAR(45)
Data_rozpoczęcia: DATE
Ilość_dni: INTEGER
Dzień_Konferencji
Dzien_konf_id: INTEGER
Koszt_id: INTEGER (FK)
Konferencja_id: INTEGER (FK)
Dzien_konf_v: INTEGER
Ilość_miejsc: INTEGER
Warsztaty
Warsztat_id: INTEGER
Dzień_Konferencji_Dzien_konf_id: INTEGER (FK)
Temat: VARCHAR(45)
Godzina_rozpoczęcia: TIME
Godzina_zakończenia: TIME
Ilość_miejsc: INTEGER
Cena: DECIMAL
Koszt
Koszt_id: INTEGER
Cena: DECIMAL
Zniżka_studencka: FLOAT
Do_rozpoczęcia_tygodni: INTEGER
Klient
Klient_id: INTEGER
nazwa: VARCHAR(45)
telefon: INT
email: VARCHAR(45)
login: VARCHAR(20)
haslo: VARCHAR(45)
Firma: BIT
Rezerwacja_Dnia_Konf
Rezerwacja_id: INTEGER
Opłacone_id_Opłacone: INTEGER (FK)
Klient_id: INTEGER (FK)
Dzień_Konferencji_Dzien_konf_id: INTEGER (FK)
Ilość_osób: INTEGER
Data_rezerwacji: DATE
Anulowano: BIT
Data_anulowania: DATE
Rezerwacja_Warsztatu
id_Rezerwacja_Warsztatu: INTEGER
Rezerwacja_Dnia_Konf_Rezerwacja_id: INTEGER (FK)
Warsztaty_Warsztat_id: INTEGER (FK)
Ilość_osób: INTEGER
Data_rezerwacji: DATE
Anulowano: BIT
Data_anulowania: DATE
Opłacone
id_Opłacone: INTEGER
Kwota: DECIMAL
Data_wpłaty: DATE
Uczestnik
Uczestnik_id: INTEGER
Imie: VARCHAR(45)
Nazwisko: VARCHAR(45)
Kod_pocztowy: INTEGER
Miasto: VARCHAR(45)
Ulica: VARCHAR(45)
Student: BIT
Nr_legitymacji: INTEGER
Zarejestrowani_uczestnicy
UczestnikDnia: INTEGER
Rezerwacja_Dnia_Konf_Rezerwacja_id: INTEGER (FK)
Uczestnik_id: INTEGER (FK)
Data_zgłoszenia: DATE
Zarejestrowani_na_warsztat
UczestnikWarsztatu: INTEGER
Rezerwacja_Warsztatu_id_Rezerwacja_Warsztat... (FK)
Zarejestrowani_uczestnicy_UczestnikDnia: IN... (FK)
Data_zgłoszenia: DATE

3.Opis

Klient- tabela rezprezentująca klientów w bazie danych. Każdy klient posiada dane(Nazwa,telefon,email,login,haslo,firma)
Mail i Login musza być unikalne , telefon musi się składac z 9 cyfr .Email musi mieć forme emailu

```
CREATE TABLE Klient (
  Klient_id INTEGER NOT NULL Primary key IDENTITY(1,1),
  nazwa VARCHAR(45) unique  NOT NULL,
  telefon INTEGER NULL Check(telefon>100000000 and telefon<999999999),
  email VARCHAR(45) NOT NULL Unique CHECK (email LIKE '%_@_%._%'),
  login VARCHAR(20) NOT NULL Unique ,
  haslo VARCHAR(45) NOT NULL ,
  Firma BIT not null default 0,

);
```

Rezerwacja_Dnia_Konf – tabela reprezentująca rezerwacje na jakiś dzień konferencji.

Połączona jest z tablica rezerwacji , oplacone , Klient oraz Dzień Konferencji_Dzien_konf_id.

W tabeli tej znajduja się kolumny ilosc osob , wartosc ta musi być wieksza od 0

Data_rezerwacji która zawiera date rezerwacji, Anulowano jeśli się rowna 1 to w Data anulowano znajduje się data anulowania

```
CREATE TABLE Rezerwacja_Dnia_Konf (
  Rezerwacja_id INTEGER   NOT NULL Primary key IDENTITY(1,1),
  Opłacone_id_Opłacone Integer null,
  Klient_id INTEGER  NOT NULL ,--
  Dzień_Konferencji_Dzien_konf_id INTEGER  NOT NULL, --
  Ilość_osób INTEGER NOT NULL check(ilość_osób>0),
  Data_rezerwacji DATE NOT NULL,
  Anulowano BIT NOT NULL default 0,
  Data_anulowania DATE  NULL,

);
```

Oplacone – tabela reprezentująca opłate , w kolumnie Kwota znajduje się kwota wieksza od 0 , Data wpłaty znajduje się data

```
CREATE TABLE Opłacone (
  id_Opłacone INTEGER  NOT NULL Primary key IDENTITY(1,1),
  Kwota DECIMAL NOT NULL CHECK (Kwota > 0),
  Data_wpłaty DATE NOT NULL,
);
```

Zarejestrowani_uczestnicy-tabela reprezentująca zarejestrowanych uczestików . Połączona z tablica RezerwacjaDniaKonf_Rezerwacja , Uczestnik . Zawiera Date zgloszenia.

```
CREATE TABLE Zarejestrowani_uczestnicy (
  Uczestnik_Dnia INTEGER NOT NULL Primary key IDENTITY(1,1),
  Rezerwacja_Dnia_Konf_Rezerwacja_id INTEGER  NOT NULL, --
  Uczestnik_id INTEGER  NOT NULL ,--
  Data_zgłoszenia DATE NULL, );
```

Uczestnik-tabela reprezentująca uczestników . Połączona z tablicą Student .

Tabela zawiera kolumny Imie ,Nazwisko, Kod Pocztowy, Miasto , Ulica oraz Student.

Jeśli student jest rowne 1 to w polu Nr_legitymacji jest nr legitymacji studenta

```
CREATE TABLE Uczestnik (
  Uczestnik_id INTEGER NOT NULL Primary key IDENTITY(1,1),
  Imie VARCHAR(45) NOT NULL,
  Nazwisko VARCHAR(45) NOT NULL,
  Kod_pocztowy INTEGER  NULL Check(Kod_pocztowy>10000 and Kod_pocztowy<99999),
  Miasto VARCHAR(45)  NULL,
  Ulica VARCHAR(45)  NULL,
  Student BIT NOT NULL default 0,
  Nr_legitymacji INTEGER NULL Check(Nr_legitymacji>0 )

);
```

Zarejestrowani_na_warsztat-tablica reprezentująca zarejestrowanych na warsztat .

Połączona z tablica Rezerwacja_Warsztatu oraz Zarejstowani_uczestnicy . Zawiera Date zgloszenia

```
CREATE TABLE Zarejestrowani_na_warsztat (
  Uczestnik_Warsztatu INTEGER NOT NULL Primary key IDENTITY(1,1),
  Rezerwacja_Warsztatu_id_Rezerwacja_Warsztatu INTEGER  NOT NULL, --
  Zarejestrowani_uczestnicy_Uczestnik_Dnia INTEGER  NOT NULL , --
  Data_zgłoszenia DATE NULL,

);
```

Rezerwacja_Warsztatu- tablica reprezentująca rezerwacje warszatu . Połączona z tabela Warsztatów i Rezerwacja Warsztów . Zawiera kolumny Ilosc osob (wieksze od 0) , Anulowano , Jeśli Anulowano =1 to Data anulowania zawiera Date anulowania.

```
CREATE TABLE Rezerwacja_Warsztatu (
  id_Rezerwacja_Warsztatu INTEGER NOT NULL Primary key IDENTITY(1,1),
  Rezerwacja_Dnia_Konf_Rezerwacja_id INTEGER NOT NULL,
  Warsztaty_Warsztat_id INTEGER  NOT NULL, --
  Ilość_osób INTEGER NOT NULL check(Ilość_osób>0),
  Data_rezerwacji DATE NOT NULL,
  Anulowano BIT NOT NULL default 0,
  Data_anulowania DATE NULL,
);
```

Warszaty –tablica reprezentująca warstaty . Polaczona jest z tablica Dzien Konferencji . Zawiera Temat , Godzine rozpoczecia , Godzine zakończenia , Ilość miejsc (wieksze od 0) oraz Cene

```
CREATE TABLE Warsztaty (
  Warsztat_id INTEGER NOT NULL Primary key IDENTITY(1,1),
  Dzień_Konferencji_Dzien_konf_id Integer  NOT NULL,
  Temat VARCHAR(45) NOT NULL,
  Godzina_rozpoczęcia TIME NOT NULL,
```

```
  Godzina_zakończenia TIME NOT NULL,
  Ilość_miejsc INTEGER  NULL check(Ilość_miejsc>0),
  Cena DECIMAL NULL check (Cena>0),

);
```

Dzień_Konferencji – tablica reprezentująca Dni konferencji . Połączona jest z tablica Koszt oraz Konferencja . Zawiera Dzien konferencji oraz ilość miejsc

```
CREATE TABLE Dzień_Konferencji (
  Dzien_konf_id INTEGER   NOT NULL Primary key IDENTITY(1,1),
  Konferencja_id INTEGER  NOT NULL,
  Koszt_id INTEGER NOT NULL,
  dzien_konf_v INTEGER NOT NULL check(dzien_konf_v > 0 and dzien_konf_v < 10),
  ilość_miejsc INTEGER NOT NULL check(ilość_miejsc>0),

);
```

Konferencja- tablica reprezentujaca konferencje . Zawiera nazwe , Date rozpoczecia , Ilość dni

```
CREATE TABLE Konferencja (
  Konferencja_id INTEGER  NOT NULL Primary key IDENTITY(1,1),
  Nazwa VARCHAR(45) Not NULL,
  Data_rozpoczęcia DATE NOT NULL,
  Ilość_dni INTEGER  NOT NULL check(ilość_dni>0),

);
```

Koszt-tablica zawierajaca koszty . Zawiera Cena , zniżke studencka , do rozpoczecia

```
CREATE TABLE Koszt (
  Koszt_id INTEGER   NOT NULL Primary key IDENTITY(1,1),
  Cena DECIMAL NOT NULL CHECK (Cena >= 0),
  Zniżka_studencka FLOAT NOT NULL CHECK(Zniżka_studencka >= 0 and Zniżka_studencka <=
100),
  Do_rozpoczęcia_tygodni INTEGER  NULL,

);
```

4. Procedury

DodajKlienta – procedura dodająca Klienta

```
CREATE PROCEDURE [dbo].[DodajKlienta]
       @nazwa VARCHAR(45),
       @telefon VARCHAR(45),
       @email VARCHAR(45),
       @login VARCHAR(45),
       @haslo VARCHAR(45),
       @Firma BIT
AS
BEGIN
       INSERT INTO [Klient](nazwa, telefon, email, login, haslo, Firma)
       VALUES (@nazwa, @telefon, @email, @login, @haslo, @Firma)
END
```

DodajKonferencje – procedura dodająca Konferencje

```
CREATE PROCEDURE [dbo].[DodajKonferencje]
	@nazwa VARCHAR(45),
	@dataRoz DATE,
	@iloscDni INTEGER
AS
BEGIN
	INSERT INTO [Konferencja](nazwa, Data_rozpoczęcia, Ilość_dni)
	VALUES (@nazwa, @dataRoz, @iloscDni)
END
```

DodajDzienKonferencji – procedura dodajaca dzien konferencji

```
CREATE PROCEDURE [dbo].[DodajDzienKonf]
	@KonferencjaId INTEGER,
	@dzienKonf INTEGER,
	@iloscMiejsc INTEGER
AS
BEGIN
	INSERT INTO [Dzień_Konferencji](Konferencja_Id, dzien_konf_v, ilość_miejsc)
	VALUES (@KonferencjaId, @dzienKonf, @iloscMiejsc)
END
```

DodajUczestnika – procedura dodajaca uczestnika

```
CREATE PROCEDURE [dbo].[DodajUczestnika]
	@Imie VARCHAR(45),
	@Nazwisko VARCHAR(45),
	@Kod_pocztowy INTEGER,
	@Miasto VARCHAR(45),
	@Ulica VARCHAR(45),
	@Student BIT,
	@Nr_leg INTEGER
AS
BEGIN
	INSERT INTO [Uczestnik](Imie, Nazwisko, Kod_pocztowy, Miasto, Ulica, Student,
Nr_legitymacji)
	VALUES (@Imie, @Nazwisko, @Kod_pocztowy, @Miasto, @Ulica, @Student, @Nr_leg)
END
```

dodajOplate - procedura dodaje informacje o uiszczonej opłacie

```
CREATE PROCEDURE [dbo].[dodajOplate]
	@Kwota DECIMAL
AS
BEGIN
	INSERT INTO [Opłacone](Kwota, Data_wpłaty)
	VALUES(@Kwota, getDate())
```

```
END
```

dodajKoszt - procedura dodaje informacje o kosztach i progach cenowych w zależności od czasu rejestracji.

```
CREATE PROCEDURE [dbo].[dodajKoszt]
	@Cena DECIMAL(5,1),
	@Zniżka_studencka FLOAT,
	@do_rozpoczecia INTEGER
AS
BEGIN
	INSERT INTO [Koszt](Cena, Zniżka_studencka, Do_rozpoczęcia_tygodni)
	VALUES(@Cena, @Zniżka_studencka, @do_rozpoczecia)
END
```

dodajWarsztat – dodaje informacje o warsztatach

```
CREATE PROCEDURE [dbo].[dodajWarsztat]
	@Warsztat_id INTEGER,
    @Dzien_konf_id INTEGER,
    @Temat VARCHAR(45),
    @Godzina_rozpoczęcia TIME,
    @Godzina_zakończenia TIME,
    @Ilość_miejsc INTEGER,
    @Cena DECIMAL
AS
BEGIN
	INSERT INTO [Warsztaty](Warsztat_id, Dzień_Konferencji_Dzien_konf_id, Temat,
Godzina_rozpoczęcia, Godzina_zakończenia,
                            Ilość_miejsc, Cena)
	VALUES (@Warsztat_id, @Dzien_konf_id, @Temat, @Godzina_rozpoczęcia,
@Godzina_zakończenia, @Ilość_miejsc, @Cena)
END
```

ZarejestrujUczestnikaNaDzien – procedura umożliwiająca rejestracje uczestnika na dzień konferencji

```
CREATE PROCEDURE [dbo].[zarejestrujUczestnikaNaDzien]
	@Uczestnik_id INTEGER,
	@Rezerwacja_id INTEGER
AS
BEGIN

	INSERT INTO [Zarejestrowani_uczestnicy](Rezerwacja_Dnia_Konf_Rezerwacja_id,
Uczestnik_id, Data_zgłoszenia)
	VALUES (@Rezerwacja_id, @Uczestnik_id, getDate())
END
```

zarejestrujUczestnikaNaWarsztat - procedura umożliwiająca rejestracje uczestnika na warsztat

```
CREATE PROCEDURE [dbo].[zarejestrujUczestnikaNaWarsztat]
	@Uczestnik_id INTEGER,
	@idRezerwacjaWarsztatu INTEGER
AS
BEGIN
	DECLARE @UczestnikDnia INT = (SELECT Uczestnik_Dnia FROM
Zarejestrowani_uczestnicy WHERE (@Uczestnik_id = Uczestnik_id))
```

```
	IF (@UczestnikDnia = NULL)
	BEGIN
	  RAISERROR('Ten uczestnik nie jest jeszcze zarejestrowany na konferencje. Nie
można zarejestrować go na wybrany warsztat.', 16, -1);
	END

	INSERT INTO
[Zarejestrowani_na_warsztat](Rezerwacja_Warsztatu_id_Rezerwacja_Warsztatu,
Zarejestrowani_uczestnicy_Uczestnik_Dnia, Data_zgłoszenia)
	VALUES (@idRezerwacjaWarsztatu, @UczestnikDnia, getDate())
END
```

zarezerwujMiejscaNaDzien – procedura dzięki której dokonujemy rezerwacji dnia konferencji.

```
CREATE PROCEDURE [dbo].[zarezerwujMiejscaNaDzien]
	@IloscOs INTEGER,
	@idKlienta INTEGER,
	@idDnia	INTEGER
AS
BEGIN


	INSERT INTO [Rezerwacja_Dnia_Konf](Klient_id, Dzień_Konferencji_Dzien_konf_id,
Ilość_osób, Data_rezerwacji)
	VALUES (@idKlienta, @idDnia, @IloscOs, getDate())
END
```

zarezerwujMiejscaNaWarsztat - procedura dzięki której dokonujemy rezerwacji warsztatu.

```
CREATE PROCEDURE [dbo].[zarezerwujMiejscaNaWarsztat]
	@IloscOs INTEGER,
	@idWar INTEGER,
	@rezDniaid INTEGER
AS
BEGIN
	INSERT INTO [Rezerwacja_Warsztatu](Warsztaty_Warsztat_id, Ilość_osób,
Rezerwacja_Dnia_Konf_Rezerwacja_id)
	VALUES(@IloscOs, @idWar, @rezDniaid)
END
```

anulowanieRezerwacjiWarsztatu – procedura za pomocą której zostaje anulowana rezerwacja warszatu

```
CREATE PROCEDURE [dbo].[anulowanieRezerwacjiWarsztatu]
	@Rezerwacja_id INTEGER
AS
BEGIN


	if( (SELECT Anulowano FROM Rezerwacja_Warsztatu WHERE @Rezerwacja_id =
id_Rezerwacja_Warsztatu) = 0)
	BEGIN
		UPDATE Rezerwacja_Warsztatu
		SET Anulowano = 1,
			Data_anulowania = GETDATE()
			WHERE id_Rezerwacja_Warsztatu = @Rezerwacja_id
	END
	ELSE RAISERROR('Warsztat był już anulowany.', 16, -1);
END
```

anulowanieRezerwacjiDnia - procedura za pomocą której zostaje anulowana rezerwacja na dzień konferencji.

```
CREATE PROCEDURE [dbo].[anulowanieRezerwacjiDnia]
	@Rezerwacja_id INTEGER
AS
BEGIN
	if( (SELECT Anulowano FROM Rezerwacja_Dnia_Konf WHERE @Rezerwacja_id =
Rezerwacja_id) = 0)
	BEGIN
		UPDATE Rezerwacja_Dnia_Konf
		SET Anulowano = 1,
			Data_anulowania = GETDATE()
			WHERE Rezerwacja_id = @Rezerwacja_id
	END
	ELSE RAISERROR(‘Rezerwacja była już anulowana.', 16, -1);
END
```

analujNieoplaconeRezerwacje – procedura służąca do anulowania nieopłaconych jeszcze rezerwacji.

```
CREATE PROCEDURE [dbo].[anulujNieoplaconeRezerwacje]
AS
BEGIN
	DECLARE tab  CURSOR LOCAL FOR (SELECT rdk.Data_rezerwacji, rdk.Rezerwacja_id
FROM Rezerwacja_Dnia_Konf as rdk
LEFT JOIN Opłacone as o ON rdk.Opłacone_id_Opłacone = o.id_Opłacone
WHERE rdk.Opłacone_id_Opłacone is NULL)

	DECLARE @Rezerwacja_id INTEGER, @dataRezerwacji date
	OPEN tab
	FETCH NEXT FROM tab INTO  @dataRezerwacji, @Rezerwacja_id
		WHILE @@FETCH_STATUS = 0
		BEGIN
			IF(DATEDIFF(day,@dataRezerwacji,GETDATE()) > 7)
			BEGIN
```

```
                        exec anulowanieRezerwacjiDnia @Rezerwacja_id
                END
                FETCH NEXT FROM tab INTO @Rezerwacja_id, @dataRezerwacji
        END
    CLOSE tab
    DEALLOCATE tab
END
```

5.Triggery

ZabronRejestracjiNaDwaTygPrzed - trigger, który sprawdza czy data rezerwacji nie jest na mniej niż 2 tygodnie przed rozpoczeciem

```
CREATE TRIGGER [dbo].[ZabronRejestracjiNaDwaTygPrzed]
  ON [dbo].[Rezerwacja_Dnia_Konf]
  AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @Rezerwacja_id INTEGER = (SELECT Rezerwacja_id FROM Inserted)
    DECLARE @DataRez DATE = (SELECT Data_rezerwacji FROM Inserted)
    DECLARE @DataRozp DATE = (SELECT Data_rozpoczęcia FROM Konferencja as k
                                                    JOIN Dzień_Konferencji as dk ON
k.Konferencja_id = dk.Konferencja_id
                                                    JOIN Rezerwacja_Dnia_Konf as rdk
ON rdk.Dzień_Konferencji_Dzien_konf_id = dk.Dzien_konf_id
                                                    WHERE rdk.Rezerwacja_id =
@Rezerwacja_id)

    IF((DAY(@DataRozp) - DAY(@DataRez)) < 14)
    BEGIN
        RAISERROR ('Za poźno aby zarezerwować.', 16, -1);
        ROLLBACK TRANSACTION
    END
END
```

placesOfWorkshopLessThanConfDay - czy liczba miejsc na warsztaty tego dnia nie jest wieksza niz na konferencje odbywającą się tego dnia

```
CREATE TRIGGER [dbo].[placesOfWorkshopLessThanConfDay]
    ON [dbo].[Warsztaty]
    AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @miejscaWar INTEGER = (SELECT Ilość_miejsc FROM INSERTED);
    DECLARE @DzienKonfId INTEGER = (SELECT Dzień_Konferencji_Dzien_konf_id FROM
inserted)
    DECLARE @miejscaKonf INTEGER = (SELECT Ilość_miejsc FROM Dzień_Konferencji WHERE
                                                    (@DzienKonfId =
Dzień_Konferencji.Dzien_konf_id))
    IF( @miejscaWar > @miejscaKonf )
    BEGIN
        RAISERROR ('W warsztacie nie może uczestniczyć więcej osób niż w
konferencji.', 16, -1);
        ROLLBACK TRANSACTION
    END
END
```

nieMożnaRezerwowaćnaWarsztaty - TRIGGER SPRAWDZAJACY CZY CZY LICZBA OSOB NA WARSZTAT NIE PRZEKRACZA LICZBY OSOB NA DZIEN KONFERENCJI

```
CREATE TRIGGER [dbo].[nieMożnaRezerwowaćnaWarsztaty]
	ON [dbo].[Rezerwacja_Warsztatu]
	AFTER INSERT, UPDATE
AS
BEGIN
	DECLARE @RezerwacjaKonfId INTEGER = (SELECT Rezerwacja_Dnia_Konf_Rezerwacja_id
FROM INSERTED)
	DECLARE @liczbaOsDzien INTEGER = (SELECT rdk.Ilość_osób FROM
Rezerwacja_Dnia_Konf as rdk WHERE (@RezerwacjaKonfId = rdk.Rezerwacja_id))
	DECLARE @liczbaOsWar INTEGER = (SELECT Ilość_osób FROM INSERTED)


	if(@liczbaOsWar > @liczbaOsDzien)
	BEGIN
		RAISERROR ('Nie można zarezerwoać więcej miejsc na warsztat niż na dany
dzień konferencji', 16, -1);
		ROLLBACK TRANSACTION
	END
END
```

nieMoznaRejestrowacNaWarsztatyRownoczesne - uczestnik nie powinien rejestrowac sie na 2 warsztaty trwajace w tym samym czasie.

```
CREATE TRIGGER [dbo].[nieMoznaRejestrowacNaWarsztatyRownoczesne]
	ON [dbo].[Zarejestrowani_na_warsztat]
	AFTER INSERT, UPDATE
AS
BEGIN
	DECLARE @isNull INTEGER = (SELECT Zarejestrowani_uczestnicy_Uczestnik_Dnia
FROM INSERTED
JOIN Rezerwacja_Warsztatu as rw1 ON rw1.id_Rezerwacja_Warsztatu
=INSERTED.Rezerwacja_Warsztatu_id_Rezerwacja_Warsztatu
JOIN Warsztaty as war ON rw1.Warsztaty_Warsztat_id = war.Warsztat_id
WHERE Zarejestrowani_uczestnicy_Uczestnik_Dnia IN (SELECT
Zarejestrowani_uczestnicy_Uczestnik_Dnia
FROM Zarejestrowani_na_warsztat AS znw JOIN Rezerwacja_Warsztatu as rw2 ON
znw.Rezerwacja_Warsztatu_id_Rezerwacja_Warsztatu = rw2.id_Rezerwacja_Warsztatu
JOIN Warsztaty as war2 ON rw2.Warsztaty_Warsztat_id = war2.Warsztat_id
WHERE (Zarejestrowani_uczestnicy_Uczestnik_Dnia =
znw.Zarejestrowani_uczestnicy_Uczestnik_Dnia
```

```
and rw2.Warsztaty_Warsztat_id <> rw1.Warsztaty_Warsztat_id and (
(war.Godzina_rozpoczęcia < war2.Godzina_zakończenia and war.Godzina_zakończenia >
war2.Godzina_rozpoczęcia) OR (war.Godzina_rozpoczęcia < war2.Godzina_rozpoczęcia and
war.Godzina_zakończenia > war2.Godzina_zakończenia) OR (war.Godzina_rozpoczęcia <
war2.Godzina_rozpoczęcia and war.Godzina_zakończenia < war2.Godzina_zakończenia) OR
(war.Godzina_rozpoczęcia > war2.Godzina_rozpoczęcia and war.Godzina_zakończenia <
war2.Godzina_zakończenia)
))))

	IF(@isNull is not NULL or @isNull != 0)
	BEGIN
		RAISERROR ('Nie można zarejestrowac tego uczestnika, gdyż on już jest
zarejestrowany na warsztat odbywający się w tym samym czasie', 16, -1);
		ROLLBACK TRANSACTION
	END
END
```

godzinaRozpMniejszaOdGodzinyZak - trigger sprawdzający czy godzina rozpoczęcia warsztatu jest przed godziną zakończenia oraZ czy warsztat trwa przynajmniej 30 minut

```
CREATE TRIGGER [dbo].[godzinaRozpMniejszaOdGodzinyZak]
	ON [dbo].[Warsztaty]
	AFTER INSERT, UPDATE
AS
BEGIN
	DECLARE @rozp TIME = (SELECT Godzina_rozpoczęcia FROM INSERTED)
	DECLARE @zak TIME = (SELECT Godzina_zakończenia FROM INSERTED)

	IF(DATEDIFF(MINUTE, @ROZP, @ZAK) < 30)
	BEGIN
		RAISERROR ('Godzina rozpoczęcia musi byc mniejsza od godziny zakończenia
oraz warsztat powinien
				trwać przynajmniej 30 minut.', 16, -1);
		ROLLBACK TRANSACTION
	END
END
```

6.Widoki

Najdrozszy_Warsztat- pokazuje najdrozszy wyklad

```
Create view Najdrozszy_Warsztat as
Select Top 1 * from Warsztaty
order by Cena
```

Najdrozsza_Konferencja-pokazuje najdrozsza konferencje

```
Create view Najdrozsza_Konferencja as
Select Top 1 * from Koszt
order by Cena desc
```

Napopularniejszy_Dzien-pokazuje najpopularniejszy warsztat

```
Create view Napopularniejszy_Dzien as
Select TOP 1 * from Rezerwacja_Warsztatu
order by Ilość_osób desc
```

Najblizsza_Konf-pokazuje najblizsza konferencje

```
Create view Najblizsza_Konf as
```

```
select TOP 1 * from Konferencja
where Data_rozpoczęcia>GETDATE()
order by Data_rozpoczęcia

Anulowane_Warsztaty-pokazuje anulowane warsztaty
Create view Anulowane_Warsztaty as
Select * from Rezerwacja_Warsztatu
where Anulowano=1


Anulowane_Dni-pokazuje anulowane dni konferencje
Create view Anulowane_Dni as
Select * from Rezerwacja_Dnia_Konf
where Anulowano=1

Najdluzsza_konf-pokazuje najdluzsza konferencje
Create view Najdluzsza_konf as
select top 1 * from Konferencja
order by Ilość_dni desc

Uczestnik_nieStudent-pokazuje uczestnikow ,którzy nie sa studentami
Create view Uczestnik_nieStudent as
Select * from Uczestnik
where Student=0

Uczestnik_Student-pokazuje studentow
Create view Uczestnik_Student as
Select * from Uczestnik
where Student=1

Klient_Firma-pokazuje Firme
Create view Klient_Firma as
Select * from Klient
where Firma=1

Klient_nieFirma-pokazuje klientów którzy nie są firmami
Create view Klient_nieFirma as
Select * from Klient
where Firma=0

Nie_Oplacone-pokazuje wszystkie nie oplacone konferencje
Create view Nie_Oplacone as
Select * from Rezerwacja_Dnia_Konf
where Opłacone_id_Opłacone is null
GO
```

7.Generator

```
Uzytkownik uzytkownik = new Uzytkownik(zapis);
  uzytkownik.generuj();
        for (int i = 1; i < 73; i++) {
```

```
Konferencje konf = new Konferencje(i, new Date(data * 1000), zapis);
ileDni = konf.generuj();
for (int j = 1; j < ileDni + 1; j++) {
        List<Integer> lista = new ArrayList<>();
        iloscWarsztatow = generator.nextInt(3) + 2;

        Koszt koszt = new Koszt(dzienKonf, zapis);
        kwota = koszt.generuj();

        Dzień dzien = new Dzień(i, 200, j, zapis, dzienKonf);
        dzienKonf = dzien.generuj();

        Oplacone oplacone = new Oplacone(rezerwacjanr, kwota, new Date(
                data * 1000), zapis);
        oplacone.generuj();

        Rezerwacja_dzien rezerwacja_dzien = new Rezerwacja_dzien(dzienKonf-1,
                data, zapis, rezerwacjanr);

        datarezerwaci = rezerwacja_dzien.generuj();
        rezerwacjanr = rezerwacja_dzien.generujnr();

        Zarejestrowani_uzytkownicy zarejestrowani = new Zarejestrowani_uzytkownicy(
                (rezerwacjanr-1), datarezerwaci, UczestnikDnia, zapis);
        lista = zarejestrowani.generuj();

        UczestnikDnia = UczestnikDnia + lista.size();

        for (int k = 1; k < iloscWarsztatow + 1; k++) {

                Warsztaty warsztaty = new Warsztaty(dzienKonf-1, Warsztatyid,
                        zapis);
                Warsztatyid = warsztaty.generuj();

                Rezerwacja_warsztatu rezerwacja_warsztatu = new Rezerwacja_warsztatu(
                        Warsztatyid-1, datarezerwaci, (rezerwacjanr-1), zapis);
                ileOsob = rezerwacja_warsztatu.generuj();

                for (int l = 1; l < ileOsob; l++) {
                        Zarejestrowani_warsztat zare = new Zarejestrowani_warsztat(
                                Warsztatyid-1, lista, datarezerwaci,
                                UczestnikWarsztatu, zapis);
                        UczestnikWarsztatu = zare.generator();

                }

        }

}
dni = generator.nextInt(5) + 12;
data = data + dni * 24 * 3600;
```

}

Generator zostal wygenerowany w jezyku java.

8.Role

-administrator-posiada pełny dotęp do bazy

-obsluga-dostęp do widoków i procedur informacyjnych

-Klient-może dodac Konferencje/warsztat oraz anulować je

-Uczestnik-może się dodac na dana konferencje /warsztat