Karol Hamielec, Grzegorz Gackowski

Podstawy Baz Danych

Projekt i implementacja systemu bazodanowego

System zarządzania konferencjami

SPIS TREŚCI:

Sformułowanie zadania projektowego 4

Dodatkowe wymagania systemowe na podstawie wywiadu 4

5. Funkcje systemu w oparciu o rodzaje użytkowników (Use Case): 6

6. Scenariusze przypadków użycia: 7

7. Model konceptualny bazy danych 10

10. Widoki 24

v_rezerwacjeKlientow
Wyświetla listę wszystkich klientów (indywidualnych jak i firmowych) wraz z liczbą rezerwacji przez nich dokonanych. 24

v_iloscDniKonferencjiUczestnikow
Wyświetla listę wszystkich uczestników wraz z ilością dni konferencji na które zapisany jest dany uczestnik 24

v_iloscWarsztatowUczestnikow 24

v_iloscWarsztatowDniKonferencjiUczestnikow 25

v_liczbaZamowienPodsumowanieKlientow 25

v_nadchodzaceWarsztaty 25

v_top10ZajetychDniKonferencji 26

zajetoscDniKonferencji 26

v_dniKonferencjiMozliweDoZapisania 26

v_nadchodzaceKonferencje 27

v_zajetoscWarsztatow 27

v_topWarsztaty10 27

v_konferencjeWarsztaty 27

10. Funkcje 28

f_nazwiskaUczestnikowDzienKonferencji 28

f_nazwiskaUczestnikowWarsztatow
Funkcja zwracająca tabelę z imionami i nazwiskami uczestników zapisanych na dany warsztat. 29

f_uczestnicyDzienKonferencji
Funkcja zwracająca tabelę z danymi uczestników zapisanych na dany dzień konferencji. 30

f_ilosc_std_w_zam_konf 30

f_liczbaNieprzypisanychUczestnikowDzienKonf
Funkcja zwraca liczbę uczestników danego dnia konferencji, dla których nie zostały jeszcze wprowadzone dane. 31
f_liczbaNieprzypisanychUczestnikowWarsztat 32
f_wolneMiejscaDzienKonferencji 32
f_wolneMiejscaWarsztat
Funkcja zwracająca liczbę wolnych miejsc na dany warsztat. 33
f_zam_konf_cena 34
f_zam_warsz_cena 35
f_daneZamowienia 36
11. Procedury 38
prod_dodaj_klienta_firmowego 38
prod_dodaj_klienta_indywidualnego 39
prod_dodaj_konferencje 40
prod_dodaj_progi_cenowe 40
prod_dodaj_prowadzacego 41
prod_dodaj_std_do_zam_konf 42
prod_dodaj_studenta_firmowego 42
prod_dodaj_uczestnika_firmowego 43
prod_dodaj_warsztat 44
prod_dodaj_warsztatOPIS 45
prod_dodaj_zam_konf_firmowe 45
prod_dodaj_zam_konf_ind 46
prod_dodaj_zam_warsztatowe_firmowe 47
prod_dodaj_zam_warsztatowe_ind 49
prod_dodaj_zamowienie 50
prod_oplac_zamowienie 50
prod_podaj_dane_studenta 51
prod_usun_nieoplacone_zam 51
prod_usun_zam 52
prod_usun_zam_konf 53
prod_usun_zam_warsztat 53
prod_zapisz_na_konferencje 54
prod_zapisz_na_warsztat 55
13. Triggery 56
1. trig_zamawianieWarsztat 56
2. trig_czyNieZaDuzoMiejscNaWarsztat 57
3. trig_zam_na_konf 57

1. Sformułowanie zadania projektowego

   Celem projektu jest stworzenie bazy danych wspomagającej działalność firmy organizującej konferencję.

   Informacje wstępne

   Firma zajmuje się organizowaniem konferencji. W czasie konferencji trwają warsztaty. Osoby indywidualne oraz firmy mogą zapisać się na dany dzień konferencji. Po zapisaniu się na dany dzień jest możliwość zapisania się w tym dniu na warsztaty, o ile pula wolnych miejsc na dany warsztat nie została wyczerpana. Warsztaty mogą być darmowe lub płatne. Firma może zapisać na konferencję oraz warsztat więcej niż jedną osobę. Firma średnio organizuje 2 konferencje w miesiącu, każda z nich trwa około 3 dni. Każdego dnia odbywają się średnio 4 warsztaty. Na konferencje zapisuje się średnio 200 osób.

2. Dodatkowe wymagania systemowe na podstawie wywiadu
   3. Czas trwania konferencji jest wielokrotnością pełnego dnia.
   4. Konferencje mogą na siebie nachodzić
   5. Warsztaty mogą być darmowe
   6. Osoba indywidualna może zarezerwować miejsce jedynie dla siebie
   7. Każda konferencja ma inner progi zniżkowe
   8. Nie można anulować rezerwacji ani dokonać częściowej płatności
   9. Nieopłacone rezerwacje trzymane są tydzień

3. Identyfikacja typów/rodzajów użytkowników (Aktorzy)
   1.1. Organizator konferencji
   1.2. Klient indywidualny
   1.3. Klient firmowy
   1.4. Automat systemowy
   1.5. Pracownik działu sprzedaży
   1.6. Administrator

4. User stories

1. Klient indywidualny
   1.1. Jako klient indywidualny chciałbym zarejestrować się na określone dni konferencji.
   1.2. Jako klient indywidualny chciałbym zarejestrować się na dany warsztat.
   1.3. Jako klient indywidualny chciałbym mieć możliwość zobaczenia listy wszystkich nadchodzących konferencji wraz z warsztatami.
   1.4. Jako klient indywidualny chciałbym dokonać płatności od razu podczas zapisywania się na konferencję lub w późniejszym terminie.
   1.5. Jako klient indywidualny chciałbym mieć możliwość skorzystania ze zniżki studenckiej, z różnych progów cenowych dla danego dnia konferencji.
2. Klient firmowy
   2.1. Jako klient firmowy chciałbym zarejestrować określoną liczbę uczestników na wybrane dni danej konferencji.
   2.2. Jako klient firmowy chciałbym zarejestrować określoną liczbę uczestników na dany warsztat.
   2.3. Jako klient firmowy chciałbym uzupełnić dane o zarejestrowanych osobach w późniejszym terminie niż zapisy.
   2.4. Jako klient firmowy chciałbym rejestrować różne osoby na różne dni..
   2.5. Jako klient firmowy chciałbym dokonać płatności od razu po zapisaniu uczestników lub w późniejszym terminie.
   2.6. Jako klient firmowy chciałbym mieć możliwość zobaczenia listy wszystkich nadchodzących konferencji wraz z warsztatami.
   2.7. Jako klient firmowy chciałbym mieć możliwość skorzystania ze zniżki studenckiej, z różnych progów cenowych dla danego dnia konferencji dla każdej osoby zapisywanej na daną konferencję / warsztat.
3. Organizator konferencji
   3.1. Jako organizator konferencji chciałbym tworzyć nową konferencję.
   3.2. Jako organizator konferencji chciałbym dodawać warsztaty do konferencji.
   3.3. Jako organizator konferencji chciałbym móc odwołać dany warsztat / konferencję / dzień konferencji

3.4. Jako organizator konferencji chciałbym mieć możliwość zobaczenia listy wszystkich nadchodzących konferencji oraz warsztatów.
3.5. Jako organizator konferencji chciałbym móc korzystać z wbudowanej bazy warsztatów przy tworzeniu nowego warsztatu w celu oszczędzenia czasu na podawaniu wielokrotnie tych samych informacji o warsztacie.
3.6. Jako organizator konferencji chciałbym mieć możliwość generowania listy osobowej uczestników na każdy dzień konferencji i na każdy warsztat, a także informacji o płatnościach klientów.
3.7. Jako organizator konferencji chciałbym sprawdzać informacje o klientach, którzy najczęściej korzystają z usług firmy.
3.8. Jako organizator konferencji chciałbym mieć możliwość edytowania danych na temat konferencji oraz warsztatu.
3.9. Jako organizator konferencji chciałbym otrzymywać informacje na 2 tygodnie przed daną konferencją o firmach, które nie wypełniły potrzebnych danych o zarejestrowanych osobach w celu telefonicznego ustalenia tych danych.
3.10. Jako organizator konferencji chciałbym dodać i usunąć prowadzącego z listy dostępnych prowadzących

5. Funkcje systemu w oparciu o rodzaje użytkowników (Use Case):
    1. Administrator systemu:
        1.1. wykonuje kopie zapasowe systemu
        1.2. dba o wprowadzanie aktualizacji i poprawek bezpieczeństwa do systemu
    2. Organizator konferencji:
        2.1. dodanie nowej konferencji
        2.2. zapisanie informacji o warsztacie do bazy warsztatów
        2.3. dodanie warsztatu do istniejącej konferencji
        2.4. możliwość przeglądania statystyk rezerwacji konferencji z podziałem na klientów indywidualnych i firmowych
        2.5. generowanie list uczestników każdych warsztatów/wykładów wraz informacją o dokonanej płatności
        2.6. anulować zaplanowaną konferencję, jej pojedynczy dzień lub warsztat

2.7. dodanie prowadzącego do listy dostępnych prowadzących
2.8. usunięcie prowadzącego z listy dostępnych prowadzących

3. Klient indywidualny:
3.1. Przeglądanie listy nadchodzących konferencji
3.2. Przeglądanie listy nadchodzących warsztatów
3.3. Zapisanie się na dany dzień konferencji
3.4. Zapisanie się na wybrane warsztaty w danym dniu konferencji
3.5. Wykonanie płatności za zamówienie

4. Klient firmowy:
4.1. Założenie konta firmowego
4.2. Przeglądanie listy nadchodzących konferencji
4.3. Przeglądanie listy nadchodzących warsztatów
4.4. Zapis firmy na wybrany dzień konferencji i rezerwacja miejsc dla określonej liczby uczestników
4.5. Zapis firmy na wybrane warsztaty i rezerwacja miejsc dla określonej liczby uczestników
4.6. Wprowadzenie danych zapisanych uczestników
4.7. Wykonanie płatności za zamówienie (otrzymanie danych do faktury)

5. Pracownik działu sprzedaży
5.1. Wprowadzenie informacji o dokonanej płatności

6. Automat systemowy:
6.1. przeglądanie bazy raz na dobę i anulowanie rezerwacji, które nie zostały opłacone 2 tygodnie przed rozpoczęciem konferencji
6.2. sumuje wartość zamówienia

6. Scenariusze przypadków użycia:

1. Rezerwacja miejsca na warsztaty przez klient firmowego
1.1. Klient firmowy wyświetla listę warsztatów na które może zapisywać uczestników. Aby dany warsztat pojawił się na liście klient firmowy musi zarezerwować wcześniej miejsca na dzień konferencji, w którym odbywa się warsztat, oraz warsztat powinien posiadać co najmniej 1 miejsce wolne.

1.2. Po wybraniu warsztatu klient firmowy podaje spodziewaną liczbę osób (nie większą niż liczba osób zapisana przez nią na dany dzień konferencji).

1.3. System sprawdza, czy suma wszystkich osób z tej firmy zapisanych na warsztaty trwające w danym czasie jest nie większa od liczby osób zapisanych przez tą firmę na dany dzień konferencji.

1.4. Klient firmowy ma możliwość zapłaty od razu, lub w późniejszym terminie. Generowanie faktury przez system.

1.5. Zarejestrowany klient firmowy po przejściu do panelu rezerwacji i wybraniu jednej z konferencji na które jest zapisany ma możliwość przeglądnięcia rezerwacji.

2. Rezerwacja miejsca na konferencję przez klient firmowego

2.1. Klient firmowy przegląda dostępne konferencje i wybiera konferencję oraz dni tej konferencji na które chciałby zapisać swoją firmę (pojawiają się tylko konferencje z wolnymi miejscami).

2.2. Zostaje przekierowany do panelu gdzie wpisuje liczbę rezerwowanych miejsc oraz określa dni konferencji. System sprawdza czy liczba rezerwowanych miejsc jest nie większa od wolnych miejsc na dany dzień konferencji.

2.3. Klient indywidualny ma do wyboru opcję zapłaty teraz lub później.

2.4. Po wybraniu opcji zapłaty teraz klient przekierowywany jest na stronę operatora płatności, który zwraca informację do systemu o powodzeniu transakcji i rezerwacja zapisywana jest do bazy danych. Po wybraniu opcji zapłaty później rezerwacja zapisywana jest do bazy danych wraz z informacją o braku wpłaty.

3. Rezerwacja miejsca na konferencji przez klienta indywidualnego

3.1. Zarejestrowany klient po przeglądnięciu dostępnych konferencji zaznacza na którą konferencję oraz na jakie dni tej konferencji chce się zapisać.

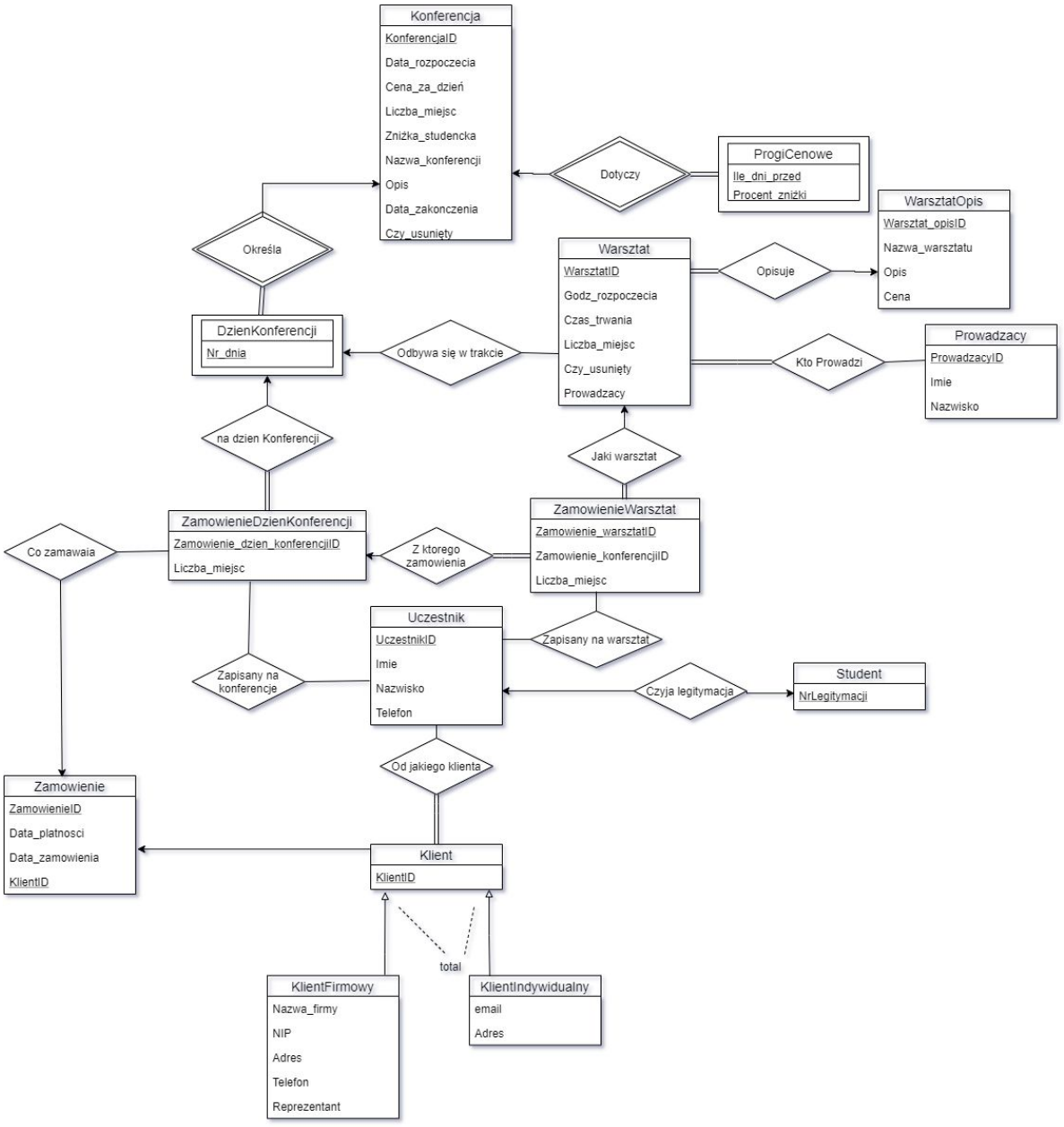
3.2. Po potwierdzeniu chęci rezerwacji klient ma do wyboru opcję zapłaty teraz lub później.

3.3. Po wybraniu opcji zapłaty teraz klient przekierowywany jest na stronę operatora płatności, który zwraca informację do systemu o powodzeniu transakcji i rezerwacja zapisywana jest do bazy danych.

3.4. Po wybraniu opcji zapłaty później rezerwacja zapisywana jest do bazy danych wraz z informacją o braku wpłaty.

4. Rejestracja klienta firmowego
   4.1. Klient firmowy w panelu rejestracji wprowadza: nazwę firmy, adres, NIP, a następnie zatwierdza.
5. Rejestracja klienta indywidualnego
   5.1. Klient indywidualny w panelu rejestracji wprowadza: Imię, Nazwisko, Adres, Adres E-mail, telefon , a następnie zatwierdza.
6. Dodanie nowej konferencji:
   6.1. Po zalogowaniu do systemu organizator konferencji wprowadza nazwę, opis, cenę, godzinę rozpoczęcia, godzinę zakończenia, liczbę dni i limit miejsc na konferencję
7. Dodanie warsztatów do istniejącej konferencji:
   7.1. Po zalogowaniu do systemu organizator wybiera konferencję której będzie dotyczył warsztat. Następnie wprowadza opis warsztatu, cenę, prowadzącego, liczbę miejsc oraz terminy warsztatu.
8. Anulowanie warsztatu
   8.1. Organizator po zalogowaniu ma możliwość przeglądnięcia zaplanowanych przez niego konferencji oraz warsztatów. Po wybraniu zaplanowanego już warsztatu ma możliwość anulowania warsztatu
9. Anulowanie zaplanowanej konferencji
   9.1. Organizator po zalogowaniu ma możliwość przeglądnięcia zaplanowanych przez niego konferencji. Następnie poprzez wybór może anulować wybrany dzień konferencji lub całą konferencję (anulowane są wtedy również wszystkie warsztaty odbywające się w danym dniu konferencji)

7. Model konceptualny bazy danych



8. Model logiczny w SQL Serwerze
   a) Kod generujący relacje

Tabela DzienKonferencji jest tabelą służącą do modelowania możliwości zapisywania się na wybrane dni konferencji. Pola tej tabeli to KonferencjaID - klucz obcy do tabeli Konferencja oraz Nr_dnia - indeksowany od 1 numer dnia danej konferencji.

```
CREATE TABLE DzienKonferencji (
    KonferencjaID int  NOT NULL,
    Nr_dnia int  NOT NULL,
    CONSTRAINT DzienKonferencji_pk PRIMARY KEY  (KonferencjaID,Nr_dnia)
);

ALTER TABLE DzienKonferencji ADD CONSTRAINT Okresla_konferencje
    FOREIGN KEY (KonferencjaID)
    REFERENCES Konferencja (KonferencjaID);

create nonclustered index DzienKonfKonfInd on
DzienKonferencji(KonferencjaID)
```

Tabela Klient posiada jedno pole będące kluczem głównym - KlientID. Służy ona ujednoliceniu numerów ID klientów indywidualnych i firmowych.

```
CREATE TABLE Klient (
    KlientID int  NOT NULL IDENTITY,
    CONSTRAINT Klient_pk PRIMARY KEY  (KlientID)
);
```

Tabela KlientFirmowy przechowuje informacje o firmie dokonującej rezerwacji. Jej pola to KlientID - klucz główny, NIP firmy oraz telefon firmowy w celu ewentualnego kontaktu z firmą.

```
CREATE TABLE KlientFirmowy (
    KlientID int  NOT NULL,
    NIP char(10)  NOT NULL,
    telefon_firmowy varchar(9)  NOT NULL,
    CONSTRAINT KlientFirmowy_pk PRIMARY KEY  (KlientID)
);
```

```
ALTER TABLE KlientFirmowy ADD CONSTRAINT KlientFirmowy_Klient
    FOREIGN KEY (KlientID)
    REFERENCES Klient (KlientID);

--telefon klienta firmowego tylko cyfry

alter table KlientFirmowy with check add constraint NrTelCyfry
check (telefon_firmowy not like '%[^0-9]%')

alter table KlientFirmowy add constraint NIPunique unique (NIP)

alter table KlientFirmowy check constraint NrTelCyfry

--nip 10 cyfr

alter table KlientFirmowy with check add constraint NIPdlugosc
check (len(NIP) = 10)

alter table KlientFirmowy check constraint NIPdlugosc
```

Tabela KlientIndywidualny przechowuje informacje na temat klienta indywidualnego. Jej pola to klucz główny KlientID, adres klienta oraz jego email.

```
CREATE TABLE KlientIndywidualny (
    KlientID int  NOT NULL,
    adres nvarchar(128)  NOT NULL,
    email nvarchar(64)  NOT NULL,
    CONSTRAINT KlientIndywidualny_pk PRIMARY KEY  (KlientID)
);
ALTER TABLE KlientIndywidualny ADD CONSTRAINT Klient_KlientIndywidualny
    FOREIGN KEY (KlientID)
    REFERENCES Klient (KlientID);

--email

alter table KlientIndywidualny with check add constraint EmailConstr
check (email like '%_@__%.__%')

alter table KlientIndywidualny check constraint EmailConstr
```

```
ALTER TABLE KlientIndywidualny ADD CONSTRAINT emailUniq UNIQUE(email)

alter table KlientIndywidualny check constraint emailUniq
```

Tabela Konferencja jest tabelą przechowującą informacje o pojedynczej konferencji. Zawiera informacje o nazwie konferencji, opłaty za dzień, liczbie przewidzianych miejsc, zniżce studenckiej, opisie oraz przechowuje daty rozpoczęcia oraz zakończenia konferencji. Przy pomocy pola czy_usuniety oznacza się konferencje usunięte.

```
CREATE TABLE Konferencja (
    Nazwa_Konfernecji nvarchar(128)  NOT NULL,
    KonferencjaID int  NOT NULL IDENTITY,
    Cena_za_dzien money  NOT NULL,
    Liczba_miejsc int  NOT NULL,
    Znizka_studencka decimal(4,2)  NOT NULL,
    Opis nvarchar(300)  NOT NULL,
    Data_rozpoczecia datetime  NOT NULL,
    Data_zakonczenia datetime  NOT NULL,
    czy_usuniety bit  NOT NULL,
    CONSTRAINT Konferencja_pk PRIMARY KEY  (KonferencjaID)
);

--znizka pomiedzy 0 a 1
alter table Konferencja with check add constraint Znizka01
check (Znizka_studencka <= 1 and Znizka_studencka >= 0)

alter table Konferencja check constraint Znizka01

--data rozpoczęcia przed datą końca
alter table Konferencja with check add constraint KonferencjaDaty
check (Data_rozpoczecia <= Data_zakonczenia)

alter table Konferencja check constraint KonferencjaDaty

--cena nieujemna
alter table Konferencja with check add constraint CenaNieujemna
check (Cena_za_dzien >= 0)

alter table Konferencja check constraint CenaNieujemna
```

```
--liczba miejsc nieujemna
alter table Konferencja with check add constraint LiczbaMiejscNieujemna
check (Liczba_miejsc >= 0)

alter table Konferencja check constraint LiczbaMiejscNieujemna

--nazwa konferencji niepusta
alter table Konferencja with check add constraint NazwaNiepusta
check (len(Nazwa_Konferencji) > 0)

alter table Konferencja check constraint NazwaNiepusta

--domyślnie nieusunięta

alter table Konferencja with check add constraint NieUsunietaKonfDomyslnie
default 0 for czy_usuniety
```

W tabeli ProgiCenowe można definiować wysokość i ilość progów cenowych dla danej konferencji. Tabela posiada kolumny KonferencjaID - klucz obcy konferencji, Ile_dni_przed - ile dni przed rozpoczęciem konferencji stosuje się dany próg cenowy, Procent_znizki - wartość zniżki w danym progu cenowym.

```
CREATE TABLE ProgiCenowe (
    KonferencjaID int  NOT NULL,
    Ile_dni_przed int  NOT NULL,
    Procent_znizki decimal(4,2)  NOT NULL,
    CONSTRAINT ProgiCenowe_pk PRIMARY KEY  (KonferencjaID,Ile_dni_przed)
);

ALTER TABLE ProgiCenowe ADD CONSTRAINT Konferencja_ProgCenowy
    FOREIGN KEY (KonferencjaID)
    REFERENCES Konferencja (KonferencjaID);

--ilość dni przed > 0 dla progu cenowego

alter table ProgiCenowe with check add constraint IleDniDodatnie
check (ile_dni_przed > 0)

create nonclustered index ProgiKonfInd on ProgiCenowe(KonferencjaID)
```

```
alter table ProgiCenowe check constraint IleDniDodatnie
```

Tabela Prowadzacy zawiera informacje o prowadzących warsztaty. Relacja przechowuje imię, nazwisko oraz ID prowadzącego (klucz główny).

```
CREATE TABLE Prowadzacy (
    Imie nvarchar(32)  NOT NULL,
    Nazwisko nvarchar(32)  NOT NULL,
    ProwadzacyID int  NOT NULL IDENTITY,
    CONSTRAINT Prowadzacy_pk PRIMARY KEY  (ProwadzacyID)
);

--imię tylko znaki i niepuste

alter table Prowadzacy with check add constraint ImieProw
check (Imie like '%[^0-9]%' and len(Imie) > 0)

alter table Prowadzacy check constraint ImieProw

--nazwisko tylko znaki i > 0

alter table Prowadzacy with check add constraint NazwiskoProw
check (Nazwisko like '%[^0-9]%' and len(Nazwisko) > 0)

alter table Prowadzacy check constraint NazwiskoProw
```

Tabela student zawiera informacje o numerach legitymacji uczestników konferencji oraz warsztatów. Jej pola to numer legitymacji oraz ID uczestnika (klucz obcy).

```
CREATE TABLE Student (
    Nr_legitymacji int  NOT NULL,
    UczestnikID int  NOT NULL,
    CONSTRAINT Student_pk PRIMARY KEY  (Nr_legitymacji)
);

ALTER TABLE Student ADD CONSTRAINT Student_Uczestnik
    FOREIGN KEY (UczestnikID)
    REFERENCES Uczestnik (Uczestnik_ID);
```

```
ALTER TABLE Student ADD CONSTRAINT legitymacjaUniq
UNIQUE(Nr_legitymacji)

create nonclustered index StudentUczestInd on Student(UczestnikID)
```

Tabela Uczestnik przedstawia konkretnego uczestnika warsztatu lub konferencji. Dane przechowywane o nim w tabeli to ID, imię, nazwisko, telefon oraz ID klienta, który wykupił miejsce uczestnikowi.

```
CREATE TABLE Uczestnik (
    Uczestnik_ID int  NOT NULL IDENTITY,
    Imie nvarchar(32)  NOT NULL,
    Nazwisko nvarchar(32)  NOT NULL,
    Telefon varchar(9)  NOT NULL,
    KlientID int  NOT NULL,
    CONSTRAINT Uczestnik_pk PRIMARY KEY  (Uczestnik_ID)
);

ALTER TABLE Uczestnik ADD CONSTRAINT Uczestnik_Klient
    FOREIGN KEY (KlientID)
    REFERENCES Klient (KlientID);

--telefon uczestnika tylko cyfry

alter table Uczestnik with check add constraint NrTelCyfryUczestnik
check (Telefon not like '%[^0-9]%')

alter table Uczestnik check constraint NrTelCyfryUczestnik

--imię co najmniej 1 znak i tylko litery

alter table Uczestnik with check add constraint ImieConstr
check (Imie like '%[^0-9]%' and len(Imie) > 0)

alter table Uczestnik check constraint ImieConstr

create nonclustered index UczestnikKlientInd on Uczestnik(KlientID)

--nazwisko co najmniej 1 znak i tylko litery
```

```
alter table Uczestnik with check add constraint NazwConstr
check (Nazwisko like '%[^0-9]%' and len(Nazwisko) > 0)

alter table Uczestnik check constraint NazwConstr
```

UczestnikKonferencji to tabela służąca do stworzenia relacji wiele do wielu pomiędzy uczestnikami a zamówieniami na konferencję. Tabela składa się z klucza głównego oraz kluczy obcych wspomnianych relacji.

```
CREATE TABLE UczestnikKonferencji (
    Zamowienie_konferencjaID int  NOT NULL,
    UczestnikID int  NOT NULL,
    Uczestnik_konferencjiID int  NOT NULL IDENTITY,
    CONSTRAINT UczestnikKonferencji_pk PRIMARY KEY
(Uczestnik_konferencjiID)
);

ALTER TABLE UczestnikKonferencji ADD CONSTRAINT
UczestnikKonferencji_Uczestnik
    FOREIGN KEY (UczestnikID)
    REFERENCES Uczestnik (Uczestnik_ID);


ALTER TABLE UczestnikKonferencji ADD CONSTRAINT
ZamowienieKonferencja_UczestnikKonferencji
    FOREIGN KEY (Zamowienie_konferencjaID)
    REFERENCES ZamowienieKonferencja (Zamowienie_konferencjaID);

create nonclustered index UczKonfUczInd on UczestnikKonferencji(UczestnikID)

create nonclustered index UczKonfZamKInd on
UczestnikKonferencji(Zamowienie_konferencjaID)
```

Tabela UczestnikWarsztatu jest tabelą służącą do zamodelowania relacji wiele do wielu pomiędzy tabelami ZamowienieWarsztat a Uczestnik. Składa się z klucza głównego oraz kluczy obcych wspomnianych tabel.

```
CREATE TABLE UczestnikWarsztatu (
    UczestnikID int  NOT NULL,
```

```
    Zamowienie_warsztatID int  NOT NULL,
    Uczestnik_wartsztatuID int  NOT NULL IDENTITY,
    CONSTRAINT UczestnikWarsztatu_pk PRIMARY KEY  (Uczestnik_wartsztatuID)
);

ALTER TABLE UczestnikWarsztatu ADD CONSTRAINT Uczestnik_UczestnikWarsztatu
    FOREIGN KEY (UczestnikID)
    REFERENCES Uczestnik (Uczestnik_ID);

create nonclustered index UczWarszUczesInd on
UczestnikWarsztatu(UczestnikID)

create nonclustered index UczWarszZamowienWarInd on
UczestnikWarsztatu(Zamowienie_warsztatID)


ALTER TABLE UczestnikWarsztatu ADD CONSTRAINT
UczestnikWarsztatu_ZamowienieWarsztat
    FOREIGN KEY (Zamowienie_warsztatID)
    REFERENCES ZamowienieWarsztat (Zamowienie_warsztatID);
```

Tabela warsztat opisuje konkretny warsztat odbywający się w trakcie określonej konferencji podanego dnia. Baza przechowuje informacje o godzinie rozpoczęcia warsztatu, czasie trwania, liczbie przewidzianych miejsc, konferencji, na której odbywa się warsztat oraz numerze dnia tej konferencji. Przy pomocy flagi czy_usuniety można dany warsztat uznać za nieaktywny. Tabela odwołuje się do relacji WarsztatOpis, gdzie znajdują się szczegółowe informacje na temat warsztatu.

```
CREATE TABLE Warsztat (
    WarsztatID int  NOT NULL IDENTITY,
    Godz_rozpoczecia time(24)  NOT NULL,
    Czas_trwania int  NOT NULL,
    Liczba_miejsc int  NOT NULL,
    Czy_usuniety bit  NOT NULL DEFAULT 0,
    Warsztat_opis_ID int  NOT NULL,
    KonferencjaID int  NOT NULL,
    Nr_dnia int  NOT NULL,
    CONSTRAINT Warsztat_pk PRIMARY KEY  (WarsztatID)
);
```

```
ALTER TABLE Warsztat ADD CONSTRAINT Opisuje
    FOREIGN KEY (Warsztat_opis_ID)
    REFERENCES WarsztatOpis (Warsztat_opis_ID);

ALTER TABLE Warsztat ADD CONSTRAINT Warsztat_DzienKonferencji
    FOREIGN KEY (KonferencjaID,Nr_dnia)
    REFERENCES DzienKonferencji (KonferencjaID,Nr_dnia);

create nonclustered index WarszWarszOpisInd on Warsztat(Warsztat_opis_ID)

create nonclustered index WarszKonfInd on Warsztat(KonferencjaID)

--czas trwania większy od 0

alter table Warsztat with check add constraint CzasTrwaniaWarsz
check (Godz_zakonczenia >= Godz_rozpoczecia)

alter table Warsztat check constraint CzasTrwaniaWarsz

--default czy usunięty

alter table Warsztat with check add constraint DefaultCzyUsu
default 0 for czy_usuniety
```

Tabela zawierająca szczegółowe informacje o danym typie warsztatu. Ponieważ te same warsztaty mogą odbywać się wielokrotnie, posiadanie osobnej tabeli na opisy warsztatów przyczynia się do zmniejszenia używania pamięci. Kolumny tej tabeli zawierają informacje o nazwie warsztatu, ID, opisie warsztatu oraz cenie za warsztat.

```
CREATE TABLE WarsztatOpis (
    Nazwa_warsztatu nvarchar(128)  NOT NULL,
    Opis nvarchar(512)  NOT NULL,
    Warsztat_opis_ID int  NOT NULL IDENTITY,
    Cena money  NOT NULL,
    CONSTRAINT WarsztatOpis_pk PRIMARY KEY  (Warsztat_opis_ID)
);
```

WarsztatProwadzacy to tabela łącząca tabele Warsztat z Prowadzacy. Pozwala na istnienie relacji wiele do wielu międzdzy wspomnianymi tabelami. Jej pola to klucze obce wymienionych relacji.

```
CREATE TABLE WarsztatProwadzacy (
    WarsztatID int  NOT NULL,
    ProwadzacyID int  NOT NULL,
    CONSTRAINT WarsztatProwadzacy_pk PRIMARY KEY  (WarsztatID,ProwadzacyID)
);

ALTER TABLE WarsztatProwadzacy ADD CONSTRAINT WarsztatProwadzacy_Prowadzacy
    FOREIGN KEY (ProwadzacyID)
    REFERENCES Prowadzacy (ProwadzacyID);

ALTER TABLE WarsztatProwadzacy ADD CONSTRAINT Warsztat_WarsztatProwadzacy
    FOREIGN KEY (WarsztatID)
    REFERENCES Warsztat (WarsztatID);
```

Tabela zamówienie przechowuje informacje o ID zamówienia, statusie płatności oraz jej dacie i numerze klienta, który dokonuje zamówienia.

```
CREATE TABLE Zamowienie (
    ZamowienieID int  NOT NULL IDENTITY,
    Data_platnosci datetime  NULL,
    Data_zamowienia datetime  NOT NULL,
    KlientID int  NOT NULL,
    CONSTRAINT Zamowienie_pk PRIMARY KEY  (ZamowienieID)
);

ALTER TABLE Zamowienie ADD CONSTRAINT Klient_Zamowienie
    FOREIGN KEY (KlientID)
    REFERENCES Klient (KlientID);

--data zamówienia domyślnie aktualna

alter table Zamowienie with check add constraint DomyslnaDataZamKonf
default getDate() for Data_zamowienia

create nonclustered index ZamowKlientInd on Zamowienie(KlientID)
```

```
--data płatności domyslnie null

alter table Zamowienie with check add constraint DataPlatnosciDomyslna
default null for Data_platnosci
```

ZamówienieKonferencja jest tabelą, w której znajduje się informacja o dniu konferencji, na jakie składane jest zamówienie oraz o ilości osób, które na ten dzień zostało zapisanych w określonym zamówieniu.

```
CREATE TABLE ZamowienieKonferencja (
    Zamowienie_konferencjaID int  NOT NULL IDENTITY,
    Liczba_miejsc int  NOT NULL,
    KonferencjaID int  NOT NULL,
    Nr_dnia int  NOT NULL,
    ZamowienieID int  NOT NULL,
    CONSTRAINT ZamowienieKonferencja_pk PRIMARY KEY
(Zamowienie_konferencjaID)
);

ALTER TABLE ZamowienieKonferencja ADD CONSTRAINT
Zamowienie_ZamowienieKonferencja
    FOREIGN KEY (ZamowienieID)
    REFERENCES Zamowienie (ZamowienieID);

ALTER TABLE ZamowienieKonferencja ADD CONSTRAINT
DzienKonferencji_ZamowienieKonferencja
    FOREIGN KEY (KonferencjaID,Nr_dnia)
    REFERENCES DzienKonferencji (KonferencjaID,Nr_dnia);

create nonclustered index ZamKonfKonfInd on
ZamowienieKonferencja(KonferencjaID)

create nonclustered index ZamKonfZamInd on
ZamowienieKonferencja(ZamowienieID)

--ilość zamawianych miejsc na konferencje > 0

alter table ZamowienieKonferencja with check add constraint MiejscaKonfDod
check (Liczba_miejsc > 0)

alter table ZamowienieKonferencja check constraint MiejscaKonfDod
```

ZamowienieWarsztat jest tabelą przechowującą informacje o liczbie osób zapisanych w danym zamówieniu na określony warsztat. Pola tej tabeli to ID, liczba miejsc, ID warsztatu oraz ID zamówienia na dzień konferencji, w którym odbywa się warsztat.

```
CREATE TABLE ZamowienieWarsztat (
    Zamowienie_warsztatID int  NOT NULL IDENTITY,
    Liczba_miejsc int  NOT NULL,
    WarsztatID int  NOT NULL,
    Zamowienie_konferencjaID int  NOT NULL,
    CONSTRAINT ZamowienieWarsztat_pk PRIMARY KEY  (Zamowienie_warsztatID)
);

ALTER TABLE ZamowienieWarsztat ADD CONSTRAINT Warsztat_ZamowienieWarsztat
    FOREIGN KEY (WarsztatID)
    REFERENCES Warsztat (WarsztatID);

ALTER TABLE ZamowienieWarsztat ADD CONSTRAINT
ZamowienieWarsztat_ZamowienieKonferencja
    FOREIGN KEY (Zamowienie_konferencjaID)
    REFERENCES ZamowienieKonferencja (Zamowienie_konferencjaID);

create nonclustered index ZamWarsztWarsztInd on
ZamowienieWarsztat(WarsztatID)

create nonclustered index ZamWarsztZamKonfInd on
ZamowienieWarsztat(Zamowienie_konferencjaID)


--liczba miejsc zamówionych na warsztat > 0

alter table ZamowienieWarsztat with check add constraint LiczbaMiejscWarsz
check (Liczba_miejsc > 0)

alter table ZamowienieWarsztat check constraint LiczbaMiejscWarsz
```

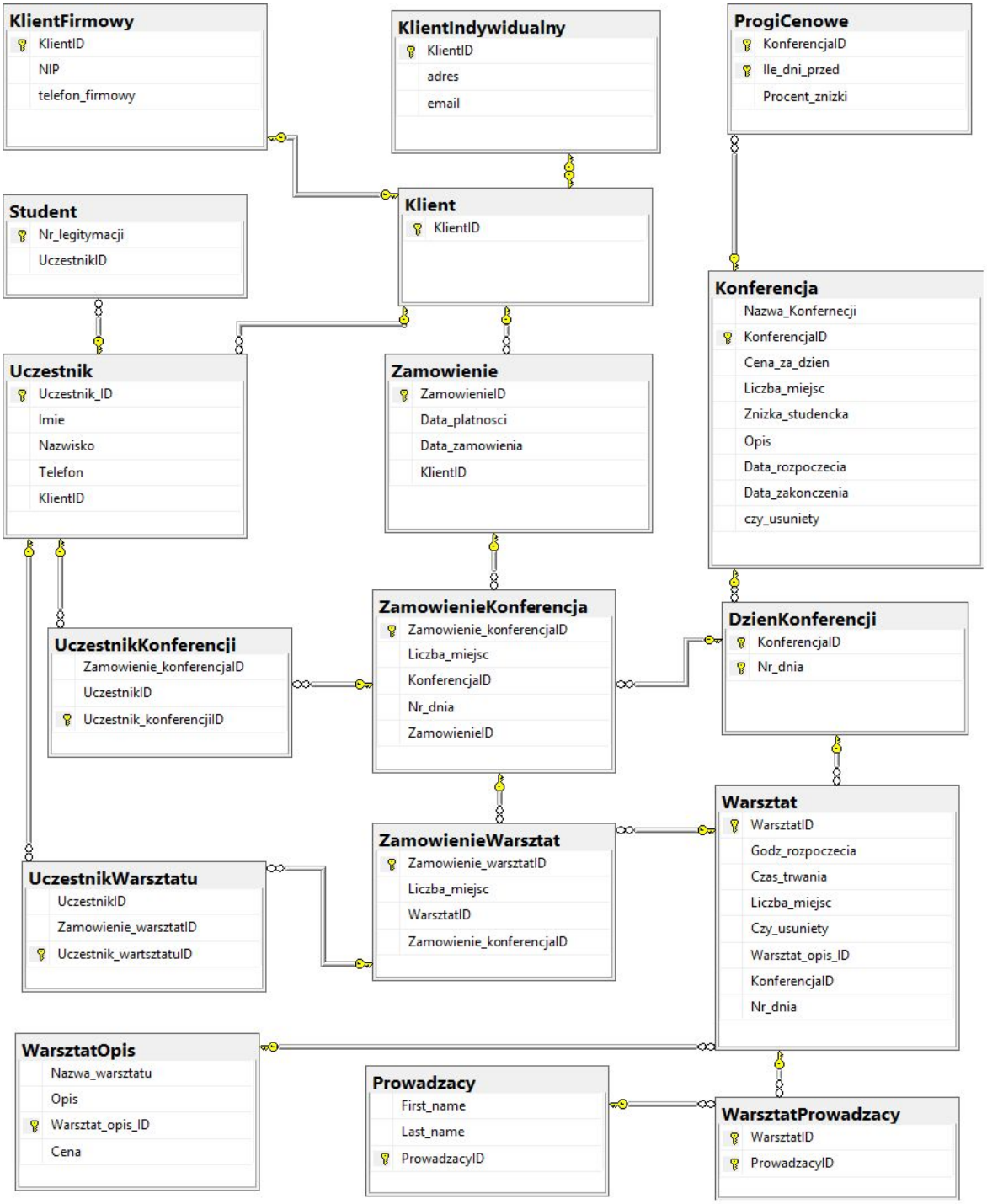
b) Spis warunków integralnościowych

check:
KlientFirmowy telefon
KlientFirmowy NIP
KlientIndywidualny email
Konferencja znizka
Konferencja daty
Konferencja cena
Konferencja miejsca
Konferencja nazwa
ProgiCenowe ilosc_dni
Prowadzacy nazwisko
Prowadzacy imię
Uczestnik telefon
Uczestnik imię
Uczestnik nazwisko
Warsztat czasy
ZamowienieKonferencja ilość miejsc
ZamowienieWarsztat ilość miejsc

unique:
KlientFirmowy NIP
KlientIndywidualny email
Student Nr legitymacji

default:
Konferencja czy_usuniety
Warsztat czy_usuniety
Zamowienie data
Zamowienie data płatności

c) Diagram bazy wykonany w SQL Serwerze



10. Widoki

1. v_rezerwacjeKlientow

   Wyświetla listę wszystkich klientów (indywidualnych jak i firmowych) wraz z liczbą rezerwacji przez nich dokonanych.

```
create view v_rezerwacjeKlientow as
SELECT k.KlientID, COUNT(zk.Liczba_miejsc) as number
FROM Klient AS k
INNER JOIN Zamowienie AS z ON k.KlientID = z.KlientID
INNER JOIN ZamowienieKonferencja AS zk on z.ZamowienieID = zk.ZamowienieID
GROUP BY k.KlientID
```

2. v_iloscDniKonferencjiUczestnikow

   Wyświetla listę wszystkich uczestników wraz z ilością dni konferencji na które zapisany jest dany uczestnik

```
create view v_iloscDniKonferencjiUczestnikow as
select u.Uczestnik_ID as 'ID uczestnika', count(Zamowienie_konferencjaID) as
'liczba dni konferencji' from Uczestnik u
left outer join UczestnikKonferencji uk
on uk.UczestnikID = u.Uczestnik_ID
group by(u.Uczestnik_ID)
```

3. v_iloscWarsztatowUczestnikow

   Wyświetla listę wszystkich uczestników wraz z ilością warsztatów na które zapisany jest dany uczestnik.

```
create view v_iloscWarsztatowUczestnikow as
select u.Uczestnik_ID as 'ID uczestnika', count(Zamowienie_WarsztatID) as
'liczba warsztatow' from Uczestnik u
```

```
left outer join UczestnikWarsztatu uk
on uk.UczestnikID = u.Uczestnik_ID
group by(u.Uczestnik_ID)
```

4. ## v_iloscWarsztatowDniKonferencjiUczestnikow

   Wyświetla listę wszystkich uczestników wraz z ilością warsztatów oraz dni konferencji na które zapisany jest dany uczestnik.

```
create view v_iloscWarsztatowDniKonferencjiUczestnikow as
select a.[id uczestnika] as 'ID Uczestnika' , a.[liczba dni konferencji] as
'ilosc', 'dni konferencji' as 'opis' from v_iloscDniKonferencjiUczestnikow a
union
select a.[id uczestnika] as 'ID Uczestnika' , a.[liczba warsztatow] as
'ilosc', 'warsztatów' as 'opis' from v_iloscWarsztatowUczestnikow a
```

5. ## v_liczbaZamowienPodsumowanieKlientow

   Wyświetla liczbę wszystkich zamówień każdego z klientów. Służy do generowania raportu.

```
create view v_liczbaZamowienPodsumowanieKlientow as
select KlientID, count(*) as 'Liczba zamowien' from Zamowienie
group by KlientID
```

6. ## v_nadchodzaceWarsztaty

   Wyświetla warsztaty, które jeszcze się nie odbyły.

```
create view v_nadchodzaceWarsztaty as
select WarsztatID, Nazwa_warsztatu, op.Opis, cena, Godz_rozpoczecia,
Nazwa_Konferencji,
Data_rozpoczecia + Nr_dnia - 1 as 'Data',
dbo.f_wolneMiejscaWarsztat(WarsztatID) as 'Liczba wolnych miejsc'
from WarsztatOpis op inner join
Warsztat w on
op.Warsztat_opis_ID = w.Warsztat_opis_ID
inner join Konferencja k on
```

```
k.KonferencjaID = w.KonferencjaID
where w.czy_usuniety <> 1 and Data_rozpoczecia + Nr_dnia - 1 > getDate()
```

7. v_top10ZajetychDniKonferencji

Wyświetla 10 dni konferencji o najmniejszym procencie wolnych miejsc.

```
create view v_top10ZajetychDniKonferencji as
select top 10 * from zajetoscDniKonferencji
order by [Zajętość] desc
```

8. zajetoscDniKonferencji

Wyświetla procent zajętych miejsc dla każdego dnia konferencji.

```
create view zajetoscDniKonferencji as
select k.konferencjaID, Nazwa_konferencji, Data_rozpoczecia + dk.Nr_Dnia - 1
as 'Data',
convert (varchar, convert (float, (k.Liczba_miejsc -
dbo.f_wolneMiejscaDzienKonferencji(k.KonferencjaID, dk.Nr_dnia))) /
k.Liczba_miejsc) + '%' as 'Zajętość'
from Konferencja k
inner join DzienKonferencji dk
on k.KonferencjaID = dk.KonferencjaID
```

9. v_dniKonferencjiMozliweDoZapisania

Widok prezentuje dni konferencji, na które istnieje możliwość zapisywania się. Tzn te dni konferencji, które jeszcze się nie odbyły, które posiadają jeszcze wolne miejsca i nie są usunięte.

```
create view v_dniKonferencjiMozliweDoZapisania as
select k.konferencjaID, Data_rozpoczecia + dk.Nr_dnia - 1 as 'Data',
Nazwa_Konferencji,
Opis, dbo.f_wolneMiejscaDzienKonferencji(k.KonferencjaID, dk.Nr_dnia) as
'Liczba wolnych miejsc' from Konferencja k
inner join DzienKonferencji dk
on k.KonferencjaID = dk.KonferencjaID
where dbo.f_wolneMiejscaDzienKonferencji(k.KonferencjaID, dk.Nr_dnia) > 0
and czy_usuniety <> 1 and Data_rozpoczecia + dk.Nr_dnia -1 > getDate()
```

10. v_nadchodzaceKonferencje

Prezentuje wszystkie nadchodzące konferencje.

```
create view v_nadchodzaceKonferencje as
select KonferencjaID, Nazwa_Konferencji, Opis, Data_rozpoczecia,
Data_zakonczenia from Konferencja
where czy_usuniety <> 1 and Data_rozpoczecia > getdate()
```

11. v_zajetoscWarsztatow

Widok przedstawia procent miejsc zajętych dla wszystkich warsztatów.

```
create view v_zajetoscWarsztatow as
select w.WarsztatID, Nazwa_warsztatu, Nazwa_Konferencji,
Data_rozpoczecia + Nr_dnia - 1 as 'Data warsztatu',
convert(varchar, convert( float, ((w.Liczba_miejsc -
dbo.f_wolneMiejscaWarsztat(w.WarsztatID)))/w.Liczba_miejsc * 100)) + '%' as
'Zajętość'
from WarsztatOpis op
inner join Warsztat w
on w.Warsztat_opis_ID = op.Warsztat_opis_ID
left outer join ZamowienieWarsztat zw
on zw.WarsztatID = w.WarsztatID
left outer join Konferencja k
on k.KonferencjaID = w.KonferencjaID
```

12. v_topWarsztaty10

Widok prezentuje 10 warsztatów o największej procentowo liczbie miejsc zajętych

```
create view v_topWarsztaty10 as
select top 10 * from v_zajetoscWarsztatow
order by [Zajętość] desc
```

13. v_konferencjeWarsztaty

Przedstawia wszystkie konferencje wraz z odbywającymi się w ich trakcie warsztatami.

```
create view v_konferencjeWarsztaty as
```

```
SELECT        k.KonferencjaID, k.Nazwa_Konferencji, k.Opis AS [konferencja
opis], dk.Nr_dnia, k.Data_rozpoczecia, wo.Nazwa_warsztatu, wo.Opis AS
[warsztat opis], wo.Cena AS [Cena warsztatu],
dbo.f_wolneMiejscaWarsztat(w.WarsztatID)
                        AS [Wolne miejsca na warsztat],
dbo.f_wolneMiejscaDzienKonferencji(k.KonferencjaID, dk.Nr_dnia) AS [Wolne
miejsca na dzień konferencji], w.Godz_rozpoczecia, k.Cena_za_dzien,
k.Znizka_studencka
FROM            dbo.Konferencja AS k LEFT OUTER JOIN
                        dbo.DzienKonferencji AS dk ON dk.KonferencjaID =
k.KonferencjaID LEFT OUTER JOIN
                        dbo.Warsztat AS w ON w.KonferencjaID =
k.KonferencjaID AND w.Nr_dnia = dk.Nr_dnia LEFT OUTER JOIN
                        dbo.WarsztatOpis AS wo ON w.Warsztat_opis_ID =
wo.Warsztat_opis_ID
```

10. Funkcje

1. f_nazwiskaUczestnikowDzienKonferencji

Funkcja zwracająca tabelę z imionami i nazwiskami uczestników zapisanych na dany dzień konferencji.

```
create function f_nazwiskaUczestnikowDzienKonferencji
(
	@KonferencjaID int,
	@NrDnia int
)
returns @tabela table (
	UczestnikID int,
	Imie nvarchar(32),
	Nazwisko nvarchar(32)
)
as
begin
	insert @tabela
		select ucz.Uczestnik_ID, ucz.Imie, ucz.Nazwisko
```

```
		from ZamowienieKonferencja zk
		inner join UczestnikKonferencji uk
		on zk.Zamowienie_konferencjaID = uk.Zamowienie_konferencjaID
		inner join Uczestnik ucz
		on ucz.Uczestnik_ID = uk.UczestnikID
		where zk.KonferencjaID = @KonferencjaID
			and zk.Nr_dnia = @NrDnia
	return
end
```

2. f_nazwiskaUczestnikowWarsztatow

   Funkcja zwracająca tabelę z imionami i nazwiskami uczestników zapisanych na dany warsztat.

```
create function f_nazwiskaUczestnikowWarsztatow
(
	@WarsztatID int
)
returns @tabela table (
	UczestnikID int,
	Imie nvarchar(32),
	Nazwisko nvarchar(32)
)
as
begin
	insert @tabela
		select ucz.Uczestnik_ID, ucz.Imie, ucz.Nazwisko
		from Uczestnik ucz
		inner join UczestnikWarsztatu uw
		on uw.UczestnikID = ucz.Uczestnik_ID
		inner join ZamowienieWarsztat zw
		on zw.Zamowienie_warsztatID = uw.Zamowienie_warsztatID
		where zw.WarsztatID = @WarsztatID
	return
end
```

3. f_uczestnicyDzienKonferencji

   Funkcja zwracająca tabelę z danymi uczestników zapisanych na dany dzień konferencji.

```
create function f_uczestnicyDzienKonferencji
(
	@KonferencjaID int,
	@NrDnia int
)
returns @tabela table (
	UczestnikID int,
	Imie nvarchar(32),
	Nazwisko nvarchar(32),
	Telefon nvarchar(9),
	CzyStudent bit
)
as
begin
	insert @tabela
		select ucz.Uczestnik_ID, ucz.Imie, ucz.Nazwisko, ucz.Telefon,
		IIF(ucz.Uczestnik_ID in (
			select s.UczestnikID from Student s
		), 1, 0)
		from ZamowienieKonferencja zk
		inner join UczestnikKonferencji uk
		on zk.Zamowienie_konferencjaID = uk.Zamowienie_konferencjaID
		inner join Uczestnik ucz
		on ucz.Uczestnik_ID = uk.UczestnikID
		where zk.KonferencjaID = @KonferencjaID
			and zk.Nr_dnia = @NrDnia
	return
end
```

4. f_ilosc_std_w_zam_konf

   Funkcja zwraca liczbę studentów, którzy są zapisani w danym zamówieniu.

```
create function dbo.f_ilosc_std_w_zam_konf(@zamKonfID int)
```

```
returns int
    as
    begin
        return (select count(s.Nr_legitymacji) from ZamowienieKonferencja zk
            inner join  UczestnikKonferencji uk on
zk.Zamowienie_konferencjaID = uk.Zamowienie_konferencjaID
            inner join uczestnik u on uk.UczestnikID = u.Uczestnik_ID
            inner join Student s on u.Uczestnik_ID = s.UczestnikID
            where zk.Zamowienie_konferencjaID = @zamKonfID
            )
    end
```

5. f_liczbaNieprzypisanychUczestnikowDzienKonf

   Funkcja zwraca liczbę uczestników danego dnia konferencji, dla których nie zostały jeszcze wprowadzone dane.

```
create function f_liczbaNieprzypisanychUczestnikowDzienKonf
(
       @Zamowienie_konferencjaID int
)
returns int
as
begin
       declare @przyp int
       set @przyp = (
              select count(*) from Uczestnik u
              inner join UczestnikKonferencji uk
              on u.Uczestnik_ID = uk.UczestnikID
              inner join ZamowienieKonferencja zk
              on uk.Zamowienie_konferencjaID = zk.Zamowienie_konferencjaID
              where u.Imie is null
       )
       return @przyp
end
```

6. f_liczbaNieprzypisanychUczestnikowWarsztat

Funkcja zwraca liczbę uczestników danego warsztatu dla których nie zostały jeszcze podane dane osobowe.

```
create function f_liczbaNieprzypisanychUczestnikowWarsztat
(
	@ZamowienieWarsztatID int
)
returns int
as
begin
	declare @przyp int
	set @przyp = (
		select count(*) from Uczestnik u
		inner join UczestnikWarsztatu uw
		on uw.UczestnikID = u.Uczestnik_ID
		where u.Imie is null and uw.Zamowienie_warsztatID =
@ZamowienieWarsztatID

	)
	return @przyp
end
```

7. f_wolneMiejscaDzienKonferencji

Funkcja zwracająca liczbę wolnych miejsc na dany dzień konferencji.

```
	create function f_wolneMiejscaDzienKonferencji
(
	@KonferencjaID int,
	@NrDnia int
)
returns int
as
begin
```

```
	declare @calosc int
	set @calosc = (
		select Liczba_miejsc from Konferencja where KonferencjaID =
@KonferencjaID
	)
	declare @zajete int
	set @zajete = (
		select sum(zk.Liczba_miejsc) from ZamowienieKonferencja zk
		inner join DzienKonferencji dk
		on dk.KonferencjaID = zk.KonferencjaID and
		dk.Nr_dnia = zk.Nr_dnia
	)
	return (@calosc - @zajete)
end
```

8. f_wolneMiejscaWarsztat

   Funkcja zwracająca liczbę wolnych miejsc na dany warsztat.

```
create function f_wolneMiejscaWarsztat
(
	@WarsztatID int
)
returns int
as
begin
	declare @calosc int
	set @calosc = (
		select Liczba_miejsc from Warsztat where WarsztatID =
@WarsztatID
	)
	declare @zajete int
	set @zajete = (
		select sum(zw.Liczba_miejsc) from ZamowienieWarsztat zw
		where zw.WarsztatID = @WarsztatID
	)
	return (@calosc - @zajete)
end
```

9. f_zam_cena

Funkcja zwraca cenę za dane zamówienie.

```
CREATE FUNCTION dbo.f_zam_cena (@ZamowienieID int)
RETURNS money
    as
    begin
        declare @konf_cena money;
        SET @konf_cena = (SELECT
SUM(dbo.f_zam_konf_cena(zk.Zamowienie_konferencjaID)) from
ZamowienieKonferencja zk
        where zk.ZamowienieID = @ZamowienieID)
        return (
            dbo.f_zam_warsz_cena(@zamowienieID) + @konf_cena
            )
    end
```

10. f_zam_konf_cena

Funkcja oblicza i zwraca cenę za dni konferencji w danym zamówieniu.

```
CREATE FUNCTION dbo.f_zam_konf_cena (@ZamowienieKonferencjaID int)
RETURNS money
AS
    BEGIN

        return (
            SELECT
k.Cena_za_dzien*(1-ISNULL(pc.Procent_znizki,0))*(zk.Liczba_miejsc-
dbo.f_ilosc_std_w_zam_konf(zk.Zamowienie_konferencjaID))
            +
k.Cena_za_dzien*(1-ISNULL(pc.Procent_znizki,0))*dbo.f_ilosc_std_w_zam_konf(z
k.Zamowienie_konferencjaID)*(1-k.Znizka_studencka)
            from ZamowienieKonferencja zk
            left join zamowienie  z on zk.ZamowienieID = z.ZamowienieID
            left join DzienKonferencji DK on zk.KonferencjaID =
DK.KonferencjaID and zk.Nr_dnia = DK.Nr_dnia
            left join Konferencja K on DK.KonferencjaID = K.KonferencjaID
            left join ProgiCenowe PC on K.KonferencjaID = PC.KonferencjaID
and
```

```
                                        pc.Ile_dni_przed = (SELECT top 1
pc.Ile_dni_przed from ProgiCenowe pc
            where DATEADD(day, -pc.Ile_dni_przed, CONVERT(DATE,
k.data_rozpoczecia)) >= z.data_zamowienia
            order by 1 desc)
            where zk.Zamowienie_konferencjaID = @ZamowienieKonferencjaID)
    end
```

11. f_zam_warsz_cena

Funkcja oblicza i zwraca cenę za warsztaty w danym zamówieniu.

```
CREATE FUNCTION [dbo].[f_zam_warsz_cena] (@zamowienieID int)
RETURNS money
AS
    BEGIN
        return (
            SELECT SUM(isnull(wo.cena, 0) * zk.Liczba_miejsc)  from
ZamowienieKonferencja zk
            left join ZamowienieWarsztat zw on zk.Zamowienie_konferencjaID =
zw.Zamowienie_konferencjaID
            left join Warsztat w on zw.WarsztatID = w.WarsztatID
            left join WarsztatOpis wo on w.Warsztat_opis_ID =
wo.Warsztat_opis_ID
            where zk.ZamowienieID = @zamowienieID
            )
    end


create function f_warsztatyKonferencji ( @KonferencjaID int)
returns @warsztaty table (
       WarsztatID int,
       Nazwa_warsztatu nvarchar(128),
       Opis nvarchar(128),
       Data_rozpoczenia_konf datetime,
       Nr_dnia int
)
as
begin
insert @warsztaty
```

```
        select WarsztatID, Nazwa_warsztatu, WO.Opis, K.Data_rozpoczecia,
Nr_dnia
        from Warsztat W inner join WarsztatOpis WO on
        W.Warsztat_opis_ID = WO.Warsztat_opis_ID
        inner join Konferencja K on
        K.KonferencjaID = W.KonferencjaID
        where K.KonferencjaID = @KonferencjaID
        return
end
```

12. f_daneZamowienia

Funkcja zwraca informacje o danym zamówieniu.

```
create function f_daneZamowienia
(
        @ZamowienieID int
)
returns @tabela table (
        typ varchar(64),
        nazwa varchar(128),
        liczba_miejsc_stud int,
        liczba_miejsc_norm int,
        cena money
)
as
begin
        insert @tabela
                select 'dzień konferencji' as 'rodzaj', k.Nazwa_konferencji as 'nazwa',
                dbo.f_ilosc_std_w_zam_konf(zk.Zamowienie_konferencjaID)
                as 'liczba miejsc studenckich',
                zk.Liczba_miejsc - dbo.f_ilosc_std_w_zam_konf(
                zk.Zamowienie_konferencjaID) as 'liczba miejsc normalnych',
                dbo.f_zam_konf_cena(zk.Zamowienie_konferencjaID) as 'cena'
                from ZamowienieKonferencja zk
                inner join Konferencja k on
                zk.KonferencjaID = k.KonferencjaID
                where zk.ZamowienieID = @ZamowienieID
                union
                select 'warsztat' as 'rodzaj', wo.nazwa_warsztatu as 'nazwa',
                null as 'liczba miejsc studenckich', zw.Liczba_miejsc as
```

```
'liczba miejsc normalnych', dbo.f_zam_warsz_cena(zk.ZamowienieID)
as 'cena'
from ZamowienieKonferencja zk inner join
ZamowienieWarsztat zw on
zw.Zamowienie_konferencjaID = zk.Zamowienie_konferencjaID
inner join Warsztat w on
w.WarsztatID = zw.WarsztatID
inner join WarsztatOpis wo on
wo.Warsztat_opis_ID = w.Warsztat_opis_ID
where zk.ZamowienieID = @ZamowienieID

        return
    end
```

13. f_generujFakturaIndywidualna

Funkcja generuje fakturę do zamówienia klienta indywidualnego

```
create function dbo.f_generujFakturaIndywidualna
(
    @ZamowienieID int
)
returns @tabela table (
    tekst varchar(1024)
)
as
begin insert @tabela

    select concat('Faktura za zamowienie nr ', convert(varchar, @ZamowienieID), ' Data zamówienia: '
    , convert(varchar, (
        select data_zamowienia from Zamowienie where zamowienieID = @ZamowienieID
    ), 0)) as 'Faktura'
    union all
    select concat('typ: ', typ, ' nazwa: ', nazwa, ' liczba miejsc studenckich: ', isnull(liczba_miejsc_stud, 0), ' liczba miejsc normalnych: ', liczba_miejsc_norm, ' cena netto: ',
    cast((cast(cena as decimal(10,2)) / 1.23) as decimal(10,2)), ' cena brutto: ', cena) as 'Faktura'
    from dbo.f_daneZamowienia(@ZamowienieID)
    union all
    select concat('dane klienta indywidualnego: Adres ', adres, ', E-mail ', email) from Zamowienie inner join KlientIndywidualny on Zamowienie.KlientID = KlientIndywidualny.KlientID
    where Zamowienie.ZamowienieID = @ZamowienieID
```

```
return
end
```

14. f_generujFakturaFirmowa

Funkcja generuje fakturę do zamówienia klienta firmowego.

```
alter function dbo.f_generujFakturaFirmowa
(
	@ZamowienieID int
)
returns @tabela table (
	tekst varchar(1024)
)
as
begin insert @tabela

	select concat('Faktura za zamowienie nr ', convert(varchar, @ZamowienieID), ' Data zamówienia: '
	, convert(varchar, (
		select data_zamowienia from Zamowienie where zamowienieID = @ZamowienieID
	), 0)) as 'Faktura'
	union all
	select concat('typ: ', typ, ' nazwa: ', nazwa, ' liczba miejsc studenckich: ', isnull(liczba_miejsc_stud, 0), ' liczba miejsc normalnych: ', liczba_miejsc_norm, ' cena netto: ',
	cast((cast(cena as decimal(10,2)) / 1.23) as decimal(10,2)), ' cena brutto: ', cena) as 'Faktura'
	from dbo.f_daneZamowienia(@ZamowienieID)
	union all
	select concat('dane klienta firmowego: NIP ', NIP, ', Telefon firmowy ', telefon_firmowy) from Zamowienie inner join KlientFirmowy on Zamowienie.KlientID = KlientFirmowy.KlientID
	where Zamowienie.ZamowienieID = @ZamowienieID

return
end
go
```

15. f_data_warsztatu

funkcja oblicza date warsztatu dla zadanageo warsztatID

```
CREATE FUNCTION f_data_warsztatu(@warsztatID int)
returns date
```

```
  as
  begin
     DECLARE @konfData date = CONVERT( date, (SELECT k.Data_rozpoczecia FROM Warsztat w inner
join Konferencja k on k.KonferencjaID = w.KonferencjaID where w.WarsztatID = @warsztatID))
     DECLARE @nrDnia int = (SELECT Nr_dnia from Warsztat w where  w.WarsztatID = @warsztatID)
     return DATEADD(day, @nrDnia, @konfData);
  end
go
```

11. Procedury

prod_dodaj_klienta_firmowego

procedura dodaje klienta firmowego do bazy

```
CREATE procedure prod_dodaj_klienta_firmowego
  @NIP char(10),
  @telefon_firmowy varchar(9),
  @klientID int OUTPUT
as
begin
  SET NOCOUNT ON;
  BEGIN TRY;
     BEGIN TRAN;
        insert into Klient default values
        SET @KlientID = SCOPE_IDENTITY();
        insert into KlientFirmowy (KlientID, NIP, telefon_firmowy)
        VALUES (@KlientID, @NIP, @telefon_firmowy)
     COMMIT TRAN;
  END TRY
  BEGIN CATCH
     ROLLBACK TRAN
     declare @blad nvarchar(1024) = 'blad dodania klienta firmowego ' + ERROR_MESSAGE()
     ;throw 51000, @blad , 1
  end catch

end
go
```

prod_dodaj_klienta_indywidualnego

procedura dodaje klienta indywidualnego do bazy

```
CREATE procedure prod_dodaj_klienta_indywidualnego @Imie nvarchar(32), @Nazwisko nvarchar(32),
@Telefon varchar(9),
                                  @adres nvarchar(128), @email nvarchar(64), @student_nr int
  as
```

```
  begin
    SET nocount ON;
    begin try
      begin tran;
        insert into Klient default values
        Declare @KlientID int = SCOPE_IDENTITY()
        insert into KlientIndywidualny (KlientID, adres, email)
        VALUES(@KlientID, @adres, @email)
        INSERT INTO Uczestnik (Imie, Nazwisko, Telefon, KlientID)
        VALUES (@Imie, @Nazwisko, @Telefon, @KlientID)
        DECLARE @UczestnikID int = SCOPE_IDENTITY()
        IF(@student_nr is not null )
        begin
          insert into Student (Nr_legitymacji, UczestnikID)
          VALUES (@student_nr, @UczestnikID)
        end
      commit tran;
    end try
  BEGIN CATCH
    ROLLBACK TRAN
    declare @blad nvarchar(1024) = 'blad dodania klienta indywidualnego ' + ERROR_MESSAGE()
    ;throw 51000, @blad , 1
  end catch

  end
go
```

1.

prod_dodaj_konferencje

procedura dodaje konferencję do bazy oraz automatycznie tworzy krotki w tabeli z dniami konferencji na podstawie dat rozpoczęcia i zakończenia

```
CREATE PROCEDURE prod_dodaj_konferencje @Nazwa nvarchar(128), @cena_za_dzien money,
@liczba_miejsc int,
                        @Znizka_studencka decimal(4, 2), @Opis nvarchar(300),
                        @data_rozpoczecia datetime, @data_zakonczenia datetime
as
  begin
    begin try
      begin tran;
        IF(@data_rozpoczecia > @data_zakonczenia)
        begin
          throw 51000, 'data rozpoczecia nie moze byc pozniejsza niz data zakonczenia', 1;
        end
        IF(@Znizka_studencka < 0 OR @Znizka_studencka > 1)
        begin
          throw 51000, 'znizka studencka nie moze byc mniejsza 0 i wieksza od 1', 1;
        end
        INSERT INTO Konferencja (Nazwa_Konfernecji, Cena_za_dzien, Liczba_miejsc, Znizka_studencka,
Opis, Data_rozpoczecia, Data_zakonczenia, czy_usuniety)
        VALUES (@Nazwa, @cena_za_dzien, @liczba_miejsc, @Znizka_studencka, @Opis,
@data_rozpoczecia, @data_zakonczenia, 0)
```

```
        declare @konfID int = SCOPE_IDENTITY();
        DECLARE @cnt INT = 1;
        DECLARE @il_dni INT = DATEDIFF(day, @data_rozpoczecia, @data_zakonczenia) + 1;
        while @cnt <= @il_dni
          begin
            INSERT INTO DzienKonferencji (KonferencjaID, Nr_dnia) VALUES(@konfID, @cnt);
            SET @cnt = @cnt+1;
          end
      commit tran;
    end try
    begin catch
        ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania konferencji ' + ERROR_MESSAGE()
      ;throw 51000, @blad , 1
    end catch

  end
go
```

prod_dodaj_progi_cenowe

procedura dodaje progi cenowe do konferencji. Sprawdza czy konferencja istnieje i czy progi zachowują porządek

```
create procedure [dbo].[prod_dodaj_progi_cenowe] @konferencjaID int , @prog_cenowy decimal(4,2),
@ile_dni int
as
  begin
    declare @prog_przed_dni int = (SELECT top 1 Ile_dni_przed from ProgiCenowe where KonferencjaID
= @konferencjaID and @ile_dni > Ile_dni_przed order by 1 desc)
    declare @prog_przed_procent decimal(4,2) = (SELECT top 1 Procent_znizki from ProgiCenowe
where KonferencjaID = @konferencjaID and @ile_dni > Ile_dni_przed order by Ile_dni_przed desc)

    declare @prog_po_dni int = (SELECT top 1  Ile_dni_przed from ProgiCenowe where KonferencjaID =
@konferencjaID and @ile_dni < Ile_dni_przed order by 1 asc);
    declare @prog_po_procent decimal(4,2) = (SELECT top 1  Procent_znizki from ProgiCenowe where
KonferencjaID = @konferencjaID and @ile_dni < Ile_dni_przed order by Ile_dni_przed asc);
    if(@prog_przed_dni is not null and @prog_przed_procent > @prog_cenowy)
      begin
        ;throw 51000, 'blad progu', 1
      end
    if(@prog_po_dni is not null and @prog_po_procent < @prog_cenowy)
      begin
        ;throw 51000, 'blad progu 2', 1
      end
    IF not exists(SELECT * FROM Konferencja where KonferencjaID = @konferencjaID)
    begin
      throw 51000, 'taka konferencja nie istnieje', 1
    end
    IF (@ile_dni <= 0)
    begin
      throw 51000, 'ilosc dni nie moze byc mniejsza lub rowna zero', 1
    end
    if(@prog_cenowy <= 0)
    begin
```

```
        throw 51000, 'prog cenowy nie moze byc mniejszy lub rowny zero', 1
    end
    insert into ProgiCenowe (KonferencjaID, Ile_dni_przed, Procent_znizki)
    VALUES (@konferencjaID, @ile_dni, @prog_cenowy)
  end
```

prod_dodaj_prowadzacego

Procedura dodaje prowadzącego do bazy danych

```
CREATE PROCEDURE prod_dodaj_prowadzacego @Imie nvarchar(32), @Nazwisko nvarchar(32),
@prowadzacyID int OUTPUT
as
  begin
    begin try
      begin tran;
        INSERT INTO Prowadzacy (First_name, Last_name) VALUES (@Imie, @Nazwisko)
        set @prowadzacyID = SCOPE_IDENTITY();
      commit tran;
    end try
    begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania prowadzacego ' + ERROR_MESSAGE()
      ;throw 51000, @blad , 1
    end catch
  end
go
```

prod_dodaj_std_do_zam_konf

procedura dodaje studenta do zamówienia konferencyjnego. Procedura tworzy uczestnika z pustymi atrybutami oprócz legitymacji a następnie zapisuje go na dzień konferencji

```
CREATE PROCEDURE prod_dodaj_std_do_zam_konf @Zamowienie_konferencjaID int, @nr_legitymacji
int
as
  begin
    begin try
      begin tran;
        IF not exists (SELECT * FROM ZamowienieKonferencja where Zamowienie_konferencjaID =
@Zamowienie_konferencjaID)
          begin
            ;throw 51000, 'takie zamowienie na konferencje nie istnieje ',1
          end

        declare @klientID int = dbo.f_jaki_klient_zam_konf(@Zamowienie_konferencjaID)
        declare @uczestnikID int;
        IF not exists (SELECT * FROM Student where Nr_legitymacji = @nr_legitymacji)
          begin
            exec dbo.prod_dodaj_studenta_firmowego @klientID, @nr_legitymacji, @uczestnikID =
@uczestnikID OUTPUT
          end
        ELSE
```

```
            begin
            SET @uczestnikID = (SELECT Uczestnikid  from Student where nr_legitymacji =
@nr_legitymacji)
            end

         exec dbo.prod_zapisz_na_konferencje @uczestnikID, @Zamowienie_konferencjaID
      commit tran;
   end try
   begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania dodania studenta do zamowienia konferencyjnego '
+ ERROR_MESSAGE()
      ;throw 51000, @blad , 1
   end catch
 end
go
```

prod_dodaj_studenta_firmowego

procedura dodaje studenta-uczestnika do bazy danych przypisanego do konkretnego klienta. Jest to procedura systemowa i nie powinna być używana przez programistę aplikacji

```
CREATE PROCEDURE prod_dodaj_studenta_firmowego @KlientID int, @nr_legitymacji int,
@uczestnikID int OUTPUT
as
 begin
   begin try
      begin tran;
         IF not exists (SELECT * FROM KlientFirmowy k where k.KlientID = @KlientID)
         begin
            ;throw 51000, 'podany klient firmowy nie istnieje',1
         end
         IF exists (SELECT * FROM Student where Nr_legitymacji = @nr_legitymacji)
            begin
               ;throw 51000, 'podany numer legitymacji jest juz w bazie danych',1
            end
         INSERT INTO Uczestnik (Imie, Nazwisko, Telefon, KlientID)
         VALUES (null, null, null, @KlientID)

         set @uczestnikid = SCOPE_IDENTITY()
         INSERT INTO Student (Nr_legitymacji, UczestnikID)
         VALUES (@nr_legitymacji, @uczestnikid)

      commit tran;
   end try
   begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania studenta firmowego ' + ERROR_MESSAGE()
      ;throw 51000, @blad , 1
   end catch
 end
go
```

prod_dodaj_uczestnika_firmowego

procedura dodaje uczestnika przypisanego do klienta firmowego

```
ALTER PROCEDURE [dbo].[prod_dodaj_uczestnika_firmowego] @Imie nvarchar(32), @Nazwisko
nvarchar(32), @Telefon varchar(9), @KlientID int, @student_nr int
as
   begin
      begin try
         begin tran
            IF not exists (SELECT * FROM KlientFirmowy k where k.KlientID = @KlientID)
            begin
               ;throw 51000, 'podany klient firmowy nie istnieje',1
            end
            if @Imie is null
            begin
               ;throw 5100, 'Imie nie moze byc nullem',1
            end
            if @Nazwisko is null
            begin
               ;throw 5100, 'Nazwisko nie moze byc nullem',1
            end
            insert into Uczestnik (Imie, Nazwisko, Telefon, KlientID)
            VALUES (@imie, @Nazwisko, @Telefon, @KlientID)
            DECLARE @UczestnikID int = SCOPE_IDENTITY()
            IF(@student_nr is not null )
            begin
               insert into Student (Nr_legitymacji, UczestnikID)
               VALUES (@student_nr, @UczestnikID)
            end
         commit tran;
      end try
      begin catch
         ROLLBACK TRAN
         declare @blad nvarchar(1024) = 'blad dodania klienta indywidualnego ' + ERROR_MESSAGE()
         ;throw 51000, @blad , 1
      end catch
   end
```

prod_dodaj_warsztat

procedura dodaje warsztat do bazy danych

```
CREATE PROCEDURE prod_dodaj_warsztat @godz_rozp time, @godz_zakon time, @liczba_miejsc int,
@WarsztatOpisID int, @KonferencjaID int, @Nr_dnia int
as
  begin
     begin try
        begin tran;
           IF @godz_rozp > @godz_zakon
              begin
                 ;throw 51000, 'godzina zakonczenia nie moze byc wczesniejsza niz godz rozpczecia',1
              end
           IF @liczba_miejsc <= 0
```

```
            begin
               ;throw 51000, 'liczba miejsc nie moze byc ujemna',1
            end
          if @liczba_miejsc > (SELECT Liczba_miejsc from Konferencja where KonferencjaID =
          @KonferencjaID)
            begin
               ;throw 51000, 'liczba miejsc na warsztat nie moze byc wieksza niz na konferencje',1
            end

          IF not exists( SELECT * FROM WarsztatOpis where Warsztat_opis_ID = @WarsztatOpisID )
            begin
               ;throw 51000, 'nie istnieje taki opis warsztatu ',1
            end
          iF not exists( SELECT * FROM Konferencja where KonferencjaID = @KonferencjaID)
            begin
               ;throw 51000, 'taka konferencja nie istnieje',1
            end
          if not exists(SELECT * FROM DzienKonferencji where KonferencjaID =@KonferencjaID and Nr_dnia
= @Nr_dnia)
            begin
               throw 51000, 'ta konferencja nie odbywa się w ten dzien',1
            end
          insert into Warsztat (Godz_rozpoczecia, Godz_zakonczenia, Liczba_miejsc, Warsztat_opis_ID,
KonferencjaID, Nr_dnia)
          VALUES (@godz_rozp, @godz_zakon, @liczba_miejsc, @WarsztatOpisID, @KonferencjaID,
@Nr_dnia)
        commit tran;
    end try
    begin catch
       ROLLBACK TRAN
       declare @blad nvarchar(1024) = 'blad dodania warsztatu' + ERROR_MESSAGE()
       ;throw 51000, @blad , 1
    end catch
  end
go
```

prod_dodaj_warsztatOPIS

procedura dodaje opis typu warsztatu do bazy danych

```
CREATE PROCEDURE prod_dodaj_warsztatOPIS @Nazwa_warsztatu nvarchar(128), @Opis
nvarchar(512), @Cena money
as
  begin
    begin try
       begin tran;
       insert into WarsztatOpis (Nazwa_warsztatu, Opis, Cena)
       VALUES (@Nazwa_warsztatu, @Opis, @Cena)
       commit tran;
    end try
    begin catch
       ROLLBACK TRAN
```

```
        declare @blad nvarchar(1024) = 'blad dodania opisu warsztatu ' + ERROR_MESSAGE()
        ;throw 51000, @blad , 1
    end catch
  end
go
```

dodajProwadzacegoNaWarsztat

procedura łączy warsztat z prowadzącym.

```
create procedure dbo.dodajProwadzacegoNaWarsztat @ProwadzacyID int, @WarsztatID int
as
  begin
          DECLARE @dataWarsztatu date = dbo.f_data_warsztatu (@warsztatId);
          DECLARE @godzRozpoczecia time = (SELECT  Godz_rozpoczecia from Warsztat where
WarsztatID = @warsztatId);
           DECLARE @godzZakonczenica time = (SELECT  Godz_zakonczenia from Warsztat where
WarsztatID = @warsztatId);
        IF exists(
          SELECT * FROM WarsztatProwadzacy inner join Warsztat w2 on
WarsztatProwadzacy.WarsztatID = w2.WarsztatID
            where dbo.f_data_warsztatu(w2.WarsztatID) = @dataWarsztatu
             and (((@godzRozpoczecia <= Godz_rozpoczecia AND @godzZakonczenica >=
Godz_rozpoczecia) OR
                  (@godzRozpoczecia <= Godz_zakonczenia AND @godzZakonczenica >=
Godz_zakonczenia))) AND w2.WarsztatID != @warsztatId)
        begin
          ;throw 51000, 'ten prowadzacy prowadzi inny warsztat w tym czasie',1
        end

    insert into WarsztatProwadzacy (WarsztatID, ProwadzacyID)
    VALUES (@WarsztatID, @ProwadzacyID)
  end
go
```

prod_dodaj_zam_konf_firmowe

procedura dodaje zamówienie na dzień konferencji do zamówienia złożonego przez klienta firmowego

```
create procedure [dbo].[prod_dodaj_zam_konf_firmowe] @liczba_miejsc int, @KonferencjaID int,
@Nr_dnia int, @zamowienieID int
as
   begin
      begin try
         begin tran
           IF not exists (SELECT * FROM Zamowienie z inner join KlientFirmowy kf on kf.KlientID =
z.KlientID where ZamowienieID = @zamowienieID)
              begin
                ;throw 51000, 'to zamowienie nie zostalo zlozone przez klienta firmowego ',1
```

```
            end
        DECLARE @konfZamowiona int = (SELECT TOP 1 KonferencjaID FROM
ZamowienieKonferencja where ZamowienieID = @zamowienieID);
        IF(@konfZamowiona != @KonferencjaID)
            begin
                ;throw 51000, 'w jednym zamowieniu moga byc tylko zamowienia dotyczace jednej
konferencji',1
            end
        if (SELECT Data_rozpoczecia from Konferencja where KonferencjaID = @KonferencjaID) <
(SELECT Data_zamowienia from Zamowienie where ZamowienieID = @zamowienieID)
            begin
                ;throw 51000, 'ta konferencja juz sie odbyla',1
            end
        IF(@liczba_miejsc <= 0)
        begin
            throw 51000, 'liczba zamowionych miejsc musi byc wieksza od zera', 1
        end
        IF not exists( SELECT * FROM Konferencja where KonferencjaID = @KonferencjaID)
        begin
            throw 51000, 'taka konferencja nie istnieje',1
        end
        if not exists(SELECT * FROM DzienKonferencji where KonferencjaID = @KonferencjaID and
Nr_dnia = @Nr_dnia)
        begin
            throw 51000, 'ta konferencja nie odbywa sie w ten dzien', 1
        end
        if exists(SELECT * FROM Konferencja where KonferencjaID = @KonferencjaID and czy_usuniety
= 1 )
        begin
            throw 51000, 'ta konferencja jest odwolana',1
        end
        if dbo.f_wolneMiejscaDzienKonferencji(@KonferencjaID, @Nr_dnia) < @liczba_miejsc
        begin
            throw 51000, 'nie ma juz tylu wolnych miejsc na ten dzien konferencji',1
        end


        insert into ZamowienieKonferencja (Liczba_miejsc, KonferencjaID, Nr_dnia, ZamowienieID)
        VALUES (@liczba_miejsc, @KonferencjaID, @Nr_dnia, @zamowienieID)
      commit tran
    end try
    begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania zamowienia na konferencje ' + ERROR_MESSAGE()
      ;throw 51000, @blad , 1
    end catch
  end
```

prod_dodaj_zam_konf_ind

procedura dodaje zamówienia złożonego przez klienta indywidualnego zamówienie na dzień konferencji

```
create procedure [dbo].[prod_dodaj_zam_konf_ind] @KonferencjaID int, @Nr_dnia int, @zamowienieID
int
as
   begin
      begin try
         begin tran
            declare @liczba_miejsc int = 1;
            IF not exists (SELECT * FROM Zamowienie z inner join KlientIndywidualny ki on ki.KlientID =
z.KlientID where ZamowienieID = @zamowienieID)
               begin
                  ;throw 51000, 'to zamowienie nie zostalo zlozone przez klienta indywidualnego ',1
               end
            IF not exists( SELECT * FROM Konferencja where KonferencjaID = @KonferencjaID)
            begin
               throw 51000, 'taka konferencja nie istnieje',1
            end
            if (SELECT Data_rozpoczecia from Konferencja where KonferencjaID = @KonferencjaID) <
(SELECT Data_zamowienia from Zamowienie where ZamowienieID = @zamowienieID)
               begin
                  ;throw 51000, 'ta konferencja juz sie odbyla',1
               end
            if not exists(SELECT * FROM DzienKonferencji where KonferencjaID = @KonferencjaID and
Nr_dnia = @Nr_dnia)
            begin
               throw 51000, 'ta konferencja nie odbywa sie w ten dzien', 1
            end
            if exists(SELECT * FROM Konferencja where KonferencjaID = @KonferencjaID and czy_usuniety
= 1 )
            begin
               throw 51000, 'ta konferencja jest odwolana',1
            end
            if dbo.f_wolneMiejscaDzienKonferencji(@KonferencjaID, @Nr_dnia) < @liczba_miejsc
            begin
               throw 51000, 'nie ma juz wolnych miejsc na ten dzien konferencji',1
            end
            insert into ZamowienieKonferencja (Liczba_miejsc, KonferencjaID, Nr_dnia, ZamowienieID)
            VALUES (@liczba_miejsc, @KonferencjaID, @Nr_dnia, @zamowienieID)
            DECLARE @zamowienieKonferencjaID int = SCOPE_IDENTITY();
            DECLARE @uczestnikID int = (SELECT u.Uczestnik_ID from uczestnik u where KlientID =
(SELECT KlientID from zamowienie z where z.ZamowienieID = @zamowienieID))
            exec dbo.prod_zapisz_na_konferencje @uczestnikID, @zamowienieKonferencjaID
         commit tran
      end try
      begin catch
         ROLLBACK TRAN
         declare @blad nvarchar(1024) = 'blad dodania zamowienia na konferencje ' + ERROR_MESSAGE()
         ;throw 51000, @blad , 1
      end catch
   end
```

prod_dodaj_zam_warsztatowe_firmowe

procedura dodaje zamówienie na warsztat do zamówienia złożonego przez klienta firmowego

```
create PROCEDURE [dbo].[prod_dodaj_zam_warsztatowe_firmowe] @liczba_miejsc int, @warsztatID int,
@ZamowienieKonferencjaID int
as
begin
   begin try
      begin tran;
      IF (@liczba_miejsc <= 0)
         begin
            ;throw 51000, 'liczba miejsc nie moze byc ujemna',1
         end
      IF not exists(SELECT * FROM Warsztat where WarsztatID = @warsztatID)
         begin
            ;throw 51000,'taki warsztat nie istnieje',1
         end
      DECLARE @dataWarsztatu date = dbo.f_data_warsztatu(@warsztatID);
      DECLARE @dataZamowienia date = CONVERT(date, (SELECT Data_zamowienia
                              from ZamowienieKonferencja
                                    inner join zamowienie z on ZamowienieKonferencja.ZamowienieID =
z.ZamowienieID
                              where Zamowienie_konferencjaID = @ZamowienieKonferencjaID))
      IF @dataWarsztatu < @dataZamowienia
         begin
            ;throw 51000, 'ten warsztat juz sie odbyl',1
         end
      if not exists(SELECT * FROM ZamowienieKonferencja where Zamowienie_konferencjaID =
@ZamowienieKonferencjaID)
         begin
            ;throw 51000, 'takie zamowienie na konferencje nie istnieje',1
         end
      IF (SELECT w.czy_usuniety from Warsztat w where WarsztatID = @warsztatID) = 1
         begin
            ;throw 51000, 'niestety ten warsztat zostal anulowany',1
         end
      declare @dzien_z_zam_konf int = (SELECT zk.nr_dnia
                               from ZamowienieKonferencja zk
                                     inner join Zamowienie z on zk.ZamowienieID = z.ZamowienieID
                                     inner join DzienKonferencji DK
                                           on zk.KonferencjaID = DK.KonferencjaID and zk.Nr_dnia =
DK.Nr_dnia
                               where Zamowienie_konferencjaID = @ZamowienieKonferencjaID)
      declare @konf_z_zam_konf int = (SELECT DK.KonferencjaID
                               from ZamowienieKonferencja zk
                                     inner join Zamowienie z on zk.ZamowienieID = z.ZamowienieID
                                     inner join DzienKonferencji DK
                                           on zk.KonferencjaID = DK.KonferencjaID and zk.Nr_dnia = DK.Nr_dnia
```

```
                        where Zamowienie_konferencjaID = @ZamowienieKonferencjaID)
      declare @dzien_z_warsz int = (SELECT nr_dnia from Warsztat where WarsztatID = @warsztatID)
      declare @konf_z_warsz int = (SELECT KonferencjaID from Warsztat where WarsztatID =
@warsztatID)
      declare @liczba_miejsc_z_zam_konf int = (SELECT liczba_miejsc
                                    from ZamowienieKonferencja
                                    where Zamowienie_konferencjaID = @ZamowienieKonferencjaID)
      if (@liczba_miejsc > @liczba_miejsc_z_zam_konf)
         begin
            ;throw 51000, 'nie mozesz zarezerwowac wiecej miesjc niz masz na zarezerwowanych na
konferencje ',1
         end
      if not (@konf_z_warsz = @konf_z_zam_konf AND @dzien_z_warsz = @dzien_z_zam_konf)
         begin
            ;throw 51000, 'musisz byc zapisany na dzien konferencji podczas ktorego sie odbywa
warsztat',1
         end
      if (dbo.f_wolneMiejscaWarsztat(@warsztatID) < @liczba_miejsc)
         begin
            ;throw 51000, 'na ten warsztat nie ma juz tylu wolnych miejsc',1
         end

      insert into ZamowienieWarsztat (Liczba_miejsc, WarsztatID, Zamowienie_konferencjaID)
      VALUES (@liczba_miejsc, @warsztatID, @ZamowienieKonferencjaID)
      commit tran;
   end try
   begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania zamowienia na warsztat ' + ERROR_MESSAGE();throw
51000, @blad , 1
   end catch
end


      insert into ZamowienieWarsztat (Liczba_miejsc, WarsztatID, Zamowienie_konferencjaID)
      VALUES (@liczba_miejsc, @warsztatID, @ZamowienieKonferencjaID)
      commit tran;
   end try
   begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania zamowienia na warsztat ' + ERROR_MESSAGE();throw
51000, @blad , 1
   end catch
end
go
```

prod_dodaj_zam_warsztatowe_ind

Procedura dodaje zamówienie na warsztat do zamówienia złożonego przez klienta indywidualnego

```
create PROCEDURE [dbo].[prod_dodaj_zam_warsztatowe_ind] @warsztatID int,
@ZamowienieKonferencjaID int
as
begin
   begin try
      begin tran;
      declare @liczba_miejsc int = 1
      declare @klientID int = dbo.f_jaki_klient_zam_konf (@ZamowienieKonferencjaID);
      IF not exists ( SELECT * FROM KlientIndywidualny where KlientID = @klientID)
         begin
            throw 51000, 'to zamowienie nie zostalo zlozne przez klienta indywidualnego',1
         end
      IF not exists(SELECT * FROM Warsztat where WarsztatID = @warsztatID)
         begin
            ;throw 51000,'taki warsztat nie istnieje',1
         end
      DECLARE @dataWarsztatu date = dbo.f_data_warsztatu(@warsztatID);
      DECLARE @dataZamowienia date = CONVERT(date, (SELECT Data_zamowienia
                              from ZamowienieKonferencja
                                    inner join zamowienie z on ZamowienieKonferencja.ZamowienieID =
z.ZamowienieID
                              where Zamowienie_konferencjaID = @ZamowienieKonferencjaID))
      IF @dataWarsztatu < @dataZamowienia
         begin
            ;throw 51000, 'ten warsztat juz sie odbyl',1
         end
      if not exists(SELECT * FROM ZamowienieKonferencja where Zamowienie_konferencjaID =
@ZamowienieKonferencjaID)
         begin
            ;throw 51000, 'takie zamowienie na konferencje nie istnieje',1
         end
      IF (SELECT w.czy_usuniety from Warsztat w where WarsztatID = @warsztatID) = 1
         begin
            ;throw 51000, 'niestety ten warsztat zostal anulowany',1
         end
      declare @dzien_z_zam_konf int = (SELECT zk.nr_dnia
                              from ZamowienieKonferencja zk
                                    inner join Zamowienie z on zk.ZamowienieID = z.ZamowienieID
                                    inner join DzienKonferencji DK
                                          on zk.KonferencjaID = DK.KonferencjaID and zk.Nr_dnia =
DK.Nr_dnia where Zamowienie_konferencjaID = @ZamowienieKonferencjaID)
      declare @konf_z_zam_konf int = (SELECT DK.KonferencjaID
                        from ZamowienieKonferencja zk
                              inner join Zamowienie z on zk.ZamowienieID = z.ZamowienieID
                              inner join DzienKonferencji DK
                                    on zk.KonferencjaID = DK.KonferencjaID and zk.Nr_dnia = DK.Nr_dnia
where Zamowienie_konferencjaID = @ZamowienieKonferencjaID)
      declare @dzien_z_warsz int = (SELECT nr_dnia from Warsztat where WarsztatID = @warsztatID)
      declare @konf_z_warsz int = (SELECT KonferencjaID from Warsztat where WarsztatID =
@warsztatID)

      if not (@konf_z_warsz = @konf_z_zam_konf AND @dzien_z_warsz =  @dzien_z_zam_konf)
         begin
```

```
            ;throw 51000, 'musisz byc zapisany na dzien konferencji podczas ktorego sie odbywa
warsztat',1
         end
      if (dbo.f_wolneMiejscaWarsztat(@warsztatID) < @liczba_miejsc)
         begin
            ;throw 51000, 'na ten warsztat nie ma juz tylu wolnych miejsc',1
         end

      insert into ZamowienieWarsztat (Liczba_miejsc, WarsztatID, Zamowienie_konferencjaID)
      VALUES (@liczba_miejsc, @warsztatID, @ZamowienieKonferencjaID)
      declare @zamWarszID int = SCOPE_IDENTITY();
      declare @uczestnikID int = (SELECT Uczestnik_ID from Uczestnik where KlientID = @KlientID);
      exec dbo.prod_zapisz_na_warsztat @uczestnikID, @zamWarszID;
      commit tran;
   end try
   begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania zamowienia na warsztat ' + ERROR_MESSAGE();throw
51000, @blad , 1
   end catch
end
```

```
      insert into ZamowienieWarsztat (Liczba_miejsc, WarsztatID, Zamowienie_konferencjaID)
      VALUES (@liczba_miejsc, @warsztatID, @ZamowienieKonferencjaID)
      declare @zamWarszID int = SCOPE_IDENTITY();
      declare @uczestnikID int = (SELECT Uczestnik_ID from Uczestnik where KlientID = @KlientID);
      exec dbo.prod_zapisz_na_warsztat @uczestnikID, @zamWarszID;
      commit tran;
   end try
   begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad dodania zamowienia na warsztat ' + ERROR_MESSAGE();throw
51000, @blad , 1
   end catch
end
go
```

prod_dodaj_zamowienie

procedura dodaje zamówienie

```
CREATE procedure prod_dodaj_zamowienie @data_platnosci datetime, @data_zamowienia datetime,
@klientID int
as
   begin
      IF(@data_zamowienia is null)
         begin
            set @data_zamowienia = CONVERT( date, GETDATE());
         end
      IF(@data_platnosci is not null AND @data_platnosci < @data_zamowienia )
      begin
```

```
        throw 51000, 'data platnosci nie moze byc wczesniejsza niz data zamowienia',1
      end
IF not exists(SELECT * FROM Klient where KlientID = @klientID)
begin
  throw 51000, 'taki klient nie istnieje', 1
end

      INSERT INTO zamowienie  (Data_platnosci, Data_zamowienia, KlientID)
      VALUES (@data_platnosci, @data_zamowienia, @klientID)

  end
go
```

prod_oplac_zamowienie

procedura pozwala opłacić konkretne zamówienie

```
create PROCEDURE [dbo].[prod_oplac_zamowienie] @zamowienieID int, @Data datetime
as
begin
   begin try
      begin tran;
      IF (@data is null)
         begin
            set @data = CONVERT(date, GETDATE());
         end
         IF not exists (SELECT * FROM zamowienie where ZamowienieID = @zamowienieID)
            begin
               ;throw 51000, 'nie ma takiego zamowienia ',1
            end
         IF (SELECT Data_zamowienia FROM zamowienie where ZamowienieID = @zamowienieID) > @data
            begin
               ;throw 51000, 'data platnosci nie moze byc wczesniejsza niz data zamowienia',1
            end
         UPDATE zamowienie SET Data_platnosci = @data where ZamowienieID = @zamowienieID
      commit tran;
   end try
   begin catch
      ROLLBACK TRAN
      declare @blad nvarchar(1024) = 'blad oplacenia zamowienia ' + ERROR_MESSAGE();
      throw 51000, @blad , 1
   end catch
end
```

prod_podaj_dane_studenta

procedura pozwala uzupełnić dane studenta o konkretnym ID uczestnika

```
CREATE PROCEDURE prod_podaj_dane_studenta @uczestnikID int, @Imie nvarchar(32), @Nazwisko
nvarchar(32), @telefon varchar(9)
as
begin
  begin try
    begin tran;
      IF not exists (SELECT * FROM Uczestnik where Uczestnik_ID = @uczestnikID)
        begin
          ;throw 51000, 'nie ma takiego uczestnika ',1
        end
      UPDATE uczestnik SET Imie = @Imie, Nazwisko = @Nazwisko, Telefon = @telefon where
Uczestnik_ID = @uczestnikID
    commit tran;
  end try
  begin catch
    ROLLBACK TRAN
    declare @blad nvarchar(1024) = 'blad dodania_danych studenta ' + ERROR_MESSAGE();
    throw 51000, @blad , 1
  end catch
end
go
```

prod_usun_nieoplacone_zam

procedura usuwa nieopłacone zamówienia korzystając z procedur systemowych i widoku nieopłaconych rezerwacji

```
CREATE PROCEDURE prod_usun_nieoplacone_zam
as
begin
  begin try
    begin tran;
    declare @tmpZam int;
    while (SELECT count(*) from dbo.v_nieoplaconeZamowienia) > 0
      begin
        SET @tmpZam = (SELECT TOP 1 ZamowienieID from dbo.v_nieoplaconeZamowienia);
        exec dbo.prod_usun_zam @tmpZam;
      end
    commit tran;
  end try
  begin catch
    ROLLBACK TRAN
    declare @blad nvarchar(1024) = 'blad usuwania zamowien ' + ERROR_MESSAGE();
    throw 51000, @blad , 1
  end catch
end
go
```

prod_usun_zam

procedura usuwa zamówienie korzystając z procedur systemowych

```
CREATE PROCEDURE prod_usun_zam @ZamowienieID int
as
begin
  begin try
    begin tran;
    if not exists(SELECT * FROM Zamowienie where ZamowienieID = @ZamowienieID)
      begin
        ;throw 51000, ' takie zamowienie nie istnieje ',1
      end
    declare @tmpid int;
    while dbo.f_ilosc_zam_na_warsztat_w_zam(@ZamowienieID) > 0
      begin
        set @tmpid = (
        SELECT top 1 Zamowienie_warsztatID
        from ZamowienieWarsztat zw
              inner join ZamowienieKonferencja zk
                    on zw.Zamowienie_konferencjaID = zk.Zamowienie_konferencjaID
        where ZamowienieID = @ZamowienieID)
        exec dbo.prod_usun_zam_warsztat @tmpid
      end
    while dbo.f_ilosc_zam_na_konf_w_zam(@ZamowienieID) > 0
      begin
        set @tmpid = (
        SELECT top 1 Zamowienie_konferencjaID from ZamowienieKonferencja
        where ZamowienieID = @ZamowienieID)
        exec dbo.prod_usun_zam_konf @tmpid
      end

    delete from Zamowienie where ZamowienieID = @ZamowienieID
    commit tran;
  end try
  begin catch
    ROLLBACK TRAN
    declare @blad nvarchar(1024) = 'blad usuwania zamowienia ' + ERROR_MESSAGE();
    throw 51000, @blad , 1
  end catch
end
go
```

prod_usun_zam_konf

procedura usuwa zamówienie na konferencje oraz usuwa uczestników, którzy byli zapisani tylko na tą konferencję

```
CREATE PROCEDURE prod_usun_zam_konf @Zamowienie_konferencjaID int
as
begin
  begin try
     begin tran;
     if not exists(SELECT * FROM zamowieniekonferencja where Zamowienie_konferencjaID =
@Zamowienie_konferencjaID)
        begin
           ;throw 51000, 'takie zamowienie na konferencje nie istnieje',1
        end

     delete from UczestnikKonferencji where Zamowienie_konferencjaID = @Zamowienie_konferencjaID
     delete
     from Uczestnik
     where dbo.f_ilosc_dni_konferencji_uczestnika(Uczestnik_ID) = 0 and imie is null and Nazwisko is null
     delete from ZamowienieKonferencja where Zamowienie_konferencjaID = @Zamowienie_konferencjaID
     commit tran;
  end try
  begin catch
     ROLLBACK TRAN
     declare @blad nvarchar(1024) = 'blad usuwania zamowienia konferencyjnego  ' +
ERROR_MESSAGE();
     throw 51000, @blad , 1
  end catch
end
go
```

prod_usun_zam_warsztat

procedura usuwa zamówienie warsztatowe

```
CREATE PROCEDURE usun_zam_warsztat @zamowienie_warsztatID int
as
begin
  begin try
     begin tran;
     IF not exists(SELECT * FROM ZamowienieWarsztat where Zamowienie_warsztatID =
@zamowienie_warsztatID)
        begin
           ;throw 51000, ' taki zamowienie nie istnieje',1
        end
     delete from UczestnikWarsztatu where Zamowienie_warsztatID = @zamowienie_warsztatID
     delete from ZamowienieWarsztat where Zamowienie_warsztatID = @zamowienie_warsztatID
     commit tran;
  end try
  begin catch
     ROLLBACK TRAN
     declare @blad nvarchar(1024) = 'blad usuwania zamowienia  ' + ERROR_MESSAGE();
     throw 51000, @blad , 1
  end catch
end
go
```

prod_zapisz_na_konferencje

procedura zapisuje na konferencję uczestnika o zadanym ID

```
CREATE procedure prod_zapisz_na_konferencje @UczestnikID int, @zamowienieKonferencjaId int
as
  begin
    IF not exists(SELECT *
                  FROM Uczestnik
                  where Uczestnik_ID = @UczestnikID
                    and KlientID = (SELECT klientID
                                    from ZamowienieKonferencja zk
                                           inner join Zamowienie on zk.ZamowienieID = Zamowienie.ZamowienieID
                                    where Zamowienie_konferencjaID = @zamowienieKonferencjaId))
    begin
      throw 51000, 'taki uczestnik nie istnieje lub nie jest przypisany do klienta zamawiajacego', 1
    end
    IF not exists(SELECT * FROM ZamowienieKonferencja where Zamowienie_konferencjaID =
@zamowienieKonferencjaId)
    begin
      throw 51000, 'takie zamowienie konferencji nie istnieje', 1
    end
    declare @konfID int = (SELECT KonferencjaID from  ZamowienieKonferencja where
Zamowienie_konferencjaID = @zamowienieKonferencjaId)
    declare @nrDnia int = (SELECT Nr_dnia from ZamowienieKonferencja where
Zamowienie_konferencjaID = @zamowienieKonferencjaId)

    IF exists(SELECT * FROM UczestnikKonferencji uk inner join zamowienieKonferencja zk on
uk.Zamowienie_konferencjaID = zk.Zamowienie_konferencjaID inner join DzienKonferencji DK on
zk.KonferencjaID = DK.KonferencjaID and zk.Nr_dnia = DK.Nr_dnia where UczestnikID = @UczestnikID)
    begin
      throw 51000, 'ten uczestnik jest juz zapisany na ten dzien konferencji', 1
    end
    IF (SELECT k.czy_usuniety from ZamowienieKonferencja zk inner join Konferencja K on
zk.KonferencjaID = K.KonferencjaID where Zamowienie_konferencjaID = @zamowienieKonferencjaId) = 1
      begin
        ;throw 51000, 'niestety ta konferencja zostala anulowana ',1
      end
    DECLARE @zam_miejsca int = (SELECT Liczba_miejsc from ZamowienieKonferencja where
Zamowienie_konferencjaID = @zamowienieKonferencjaId)
    DECLARE @zapisani int = dbo.f_ilosc_zapisanych_z_zam_konf(@zamowienieKonferencjaId);
    if(@zapisani = @zam_miejsca)
      begin
        ;throw 51000,'nie mozesz zapisac wiecej osob niz zarezerowales miejsc',1
      end
    insert into UczestnikKonferencji (Zamowienie_konferencjaID, UczestnikID)
    VALUES (@zamowienieKonferencjaId, @UczestnikID)

  end
go
```

prod_zapisz_na_warsztat

procedura pozwala zapisać uczestnika na warsztat

```
CREATE PROCEDURE prod_zapisz_na_warsztat @UczestnikID int, @zamowienie_warsztatID int
as
begin
  begin try
    begin tran;
    if not exists(SELECT *
              FROM Uczestnik
              where Uczestnik_ID = @UczestnikID
                and KlientID = (SELECT KlientID
                        from ZamowienieWarsztat zw
                            inner join ZamowienieKonferencja ZK
                                    on zw.Zamowienie_konferencjaID = ZK.Zamowienie_konferencjaID
                            inner join zamowienie z on ZK.ZamowienieID = z.ZamowienieID
                        where Zamowienie_warsztatID = @zamowienie_warsztatID))
      begin
        ;throw 51000, 'taki uczestnik nie istnieje lub nie jest przypisany do klienta zamawiajcego ',1
      end
    IF not exists(SELECT * FROM ZamowienieWarsztat where Zamowienie_warsztatID =
@zamowienie_warsztatID)
      begin
        ;throw 51000, ' taki zamowienie nie istnieje',1
      end
    declare @warsztatId int = (SELECT zw.WarsztatID
                    from ZamowienieWarsztat zw
                        inner join warsztat w on zw.WarsztatID = w.WarsztatID
                    where zw.Zamowienie_warsztatID = @zamowienie_warsztatID)

    if not exists(SELECT Uczestnikid
              from ZamowienieWarsztat zw
                  inner join warsztat w on zw.WarsztatID = w.WarsztatID
                  inner join DzienKonferencji DK
                      on w.KonferencjaID = DK.KonferencjaID and w.Nr_dnia = DK.Nr_dnia
                  inner join ZamowienieKonferencja
                      on DK.KonferencjaID = ZamowienieKonferencja.KonferencjaID and
                        DK.Nr_dnia = ZamowienieKonferencja.Nr_dnia
                  inner join UczestnikKonferencji on ZamowienieKonferencja.Zamowienie_konferencjaID
=
                                          UczestnikKonferencji.Zamowienie_konferencjaID
              where UczestnikID = @UczestnikID)
      begin
        ;throw 51000, 'ten uczestnik musi byc zapisany na konferencje w dniu warsztatu', 1
      end
    if exists(SELECT *
            FROM UczestnikWarsztatu uw
                inner join ZamowienieWarsztat zw on uw.Zamowienie_warsztatID =
zw.Zamowienie_warsztatID
            where UczestnikID = @uczestnikID
              and zw.WarsztatID = @warsztatId)
```

```
        begin
           throw 51000, 'ten uczestnik jest juz zapisany na ten warsztat ', 1
        end
     DECLARE @dataWarsztatu date = dbo.f_data_warsztatu (@warsztatId);
     DECLARE @godzRozpoczecia time = (SELECT  Godz_rozpoczecia from Warsztat where WarsztatID =
@warsztatId);
     DECLARE @godzZakonczenica time = (SELECT  Godz_zakonczenia from Warsztat where WarsztatID
= @warsztatId);
     IF exists(SELECT *
           FROM UczestnikWarsztatu inner join ZamowienieWarsztat on
UczestnikWarsztatu.Zamowienie_warsztatID = ZamowienieWarsztat.Zamowienie_warsztatID
            inner join Warsztat W2 on ZamowienieWarsztat.WarsztatID = W2.WarsztatID
           where dbo.f_data_warsztatu(w2.WarsztatID) = @dataWarsztatu
            and (((@godzRozpoczecia <= Godz_rozpoczecia AND @godzZakonczenica >=
Godz_rozpoczecia) OR
                 (@godzRozpoczecia <= Godz_zakonczenia AND @godzZakonczenica >=
Godz_zakonczenia))) AND w2.WarsztatID != @warsztatId and UczestnikWarsztatu.UczestnikID =
@UczestnikID)
        begin
           ;throw 51000, 'ten uczestnik nie moze uczestniczyc w tym warsztacie, poniewaz w tym terminie
uczestniczy w innym warsztacie',1
        end




     IF (SELECT w.czy_usuniety
        from ZamowienieWarsztat zw
              inner join warsztat w on zw.WarsztatID = w.WarsztatID
        where zw.Zamowienie_warsztatID = @zamowienie_warsztatID) = 1
        begin
           ;throw 51000, 'niestety ten warsztat zostal anulowany',1
        end
     insert into UczestnikWarsztatu (UczestnikID, Zamowienie_warsztatID)
     VALUES (@UczestnikID, @zamowienie_warsztatID)
     commit tran;
  end try
  begin catch
      ROLLBACK TRAN
     declare @blad nvarchar(1024) = 'blad zapisu na warsztat ' + ERROR_MESSAGE();
     throw 51000, @blad , 1
  end catch
end
go
```

prod_dodaj_n_std_do_zam_konf

procedura dodaje studentów do zamówienia konferencyjnego

```
CREATE PROCEDURE prod_dodaj_n_std_do_zam_konf @Zamowienie_konferencjaID int, @numery
numeryLegitymacji READONLY
as
```

```
  begin
    begin try
      begin tran tr4;
        declare @id int;

        set @id = (SELECT min( Nr_legitymacji ) from @numery)

        while @id is not null
        begin
          exec prod_dodaj_std_do_zam_konf @Zamowienie_konferencjaID, @id
          set @id = (SELECT min( Nr_legitymacji ) from @numery where Nr_legitymacji > @id)
        end
      commit tran tr4;
    end try
    begin catch
      ROLLBACK TRAN tr4
      declare @blad nvarchar(1024) = 'blad dodania studentow do zamowienia konferencyjnego ' +
ERROR_MESSAGE()
      ;throw 51000, @blad , 1
    end catch
  end
go
```

13. Triggery

1. trig_zamawianieWarsztat

Trigger uruchamia się podczas dodawania nowego zamówienia na warsztat. Jeżeli wszystkie miejsca są zajęte operacja zakończy się błędem.

```
CREATE TRIGGER trig_zamawianieWarsztat ON ZamowienieWarsztat
AFTER INSERT
AS
BEGIN
IF(dbo.f_wolneMiejscaWarsztat(
(SELECT WarsztatID FROM inserted)
) < 0)
BEGIN
RAISERROR('Przekroczono liczbę wolnych miejsc na warsztat', 16, 1)
ROLLBACK TRANSACTION
END
END
```

2. trig_czyNieZaDuzoMiejscNaWarsztat

Trigger sprawdza, czy liczba miejsc dodawanego do dnia konferencji warsztatu nie przekroczył dostępnych miejsc na dany dzień konferencji.

```
create trigger trig_czyNieZaDuzoMiejscNaWarsztat on Warsztat
after insert
as
begin
declare @miejscaWarsztat int;
declare @miejscaDzienKonf int;
set @miejscaWarsztat = (select Liczba_miejsc from inserted)
set @miejscaDzienKonf = (select Liczba_miejsc from Konferencja where (select
KonferencjaID from inserted) = Konferencja.KonferencjaID)
IF(@miejscaDzienKonf < @miejscaWarsztat)
begin
raiserror('Przekroczono limit miejsc na dany dzień konferencji (%d)', 16, 1,
@miejscaDzienKonf)
rollback transaction
end
end
```

3. trig_zam_na_konf

Trigger uruchamia się po dodaniu nowego zamówienia na konferencję. Jeżeli podjęto próbę zamówienia miejsc na dzień konferencji na który nie ma już miejsc transakcja jest cofana

```
CREATE TRIGGER trig_zam_na_konf ON ZamowienieKonferencja
	AFTER INSERT
	AS
	BEGIN
	DECLARE @zamowione int = (SELECT Liczba_miejsc from inserted)
	DECLARE @wolne int = dbo.f_wolneMiejscaDzienKonferencji((SELECT
KonferencjaID FROM INSERTED),
                                                                (SELECT Nr_dnia
FROM INSERTED)) + @zamowione


	IF(@wolne < @zamowione)
	begin
		RAISERROR ('tylko %d miejsc jest dostepnych, a probujesz zamowic %d',
15, 1, @wolne, @zamowione)
		ROLLBACK TRANSACTION
```

```
    end

    end
```

Generator Danych

Dane zostały wygenerowane przy pomocy strony https://www.mockaroo.com do plików csv następnie przy pomocy skryptu w pythonie przepisane zostały do poleceń SQL wykonujących procedury dodawania danych do bazy

kod w python:

```
import csv
from random import random, seed, randint
seed(213)

def klientIndywidualny():
   csvfile = open('KlientIndywidualny.csv', newline = '', encoding
= 'utf-8-sig')
   sqlfile = open('KlientIndywidualny.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_klienta_indywidualnego '" +
row[0] + "', '" + row[1].replace("'", " ") + "', '" + row[2][1:] +
"', '" + row[3] + "', '" + row[4] + "', null"
       string  = string + ';'
       string = string + '\r\n'
       sqlfile.write(string)


def klientIndywidualnyStudent():
   csvfile = open('KlientIndywidualnyStudent.csv', newline = '',
encoding = 'utf-8-sig')
   sqlfile = open('KlientIndywidualnyStudent.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_klienta_indywidualnego '" +
row[0] + "', '" + row[1].replace("'", " ") + "', '" + row[2][1:] +
"', '" + row[3] + "', '" + row[4]+ "', " + row[5]
       string  = string + ';'
       string = string + '\r\n'
       sqlfile.write(string)

def klientFirmowy():
   name = 'KlientFirmowy'
```

```
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_klienta_firmowego '" + row[0]
+ "', '" + row[1] + "', " + "null"
       string  = string + ';'
       string = string + '\r\n'
       sqlfile.write(string)

def konferencje():
   name = 'Konferencje'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       if(row[10][1] == ":"):
           row[10] = "0" + row[10]
       if(row[13][1] == ":"):
           row[13] = "0" + row[13]
       string = "exec dbo.prod_dodaj_konferencje '" +
row[5].replace("'", " ") + "', " + row[6] + ", " + row[7] + ", " +
row[8] + ", '" + row[11].replace("'", " ") + "', '" + row[9] + "T"
+ row[10] + "', '" + row[12] + "T" + row[13] + "'"
       string  = string + ';'
       string = string + '\r\n'
       sqlfile.write(string)

def WarsztatOPIS():
   name = 'WarsztatOPIS'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_WarsztatOPIS '" +
row[3].replace("'", " ") + "', '" + row[4].replace("'", " ") + "',
" + row[5]
       string  = string + ';'
       string = string + '\r\n'
       sqlfile.write(string)

def Warsztat():
   name = 'Warsztat2'
```

```
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_Warsztat '" + row[0] + "', '"
+ row[2] + "', " + row[3] + ", " + row[4] + ", " + row[5] + ", " +
row[6]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)


def Progi():
   name = 'Progi'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_progi_cenowe " + row[0] + ",
" + row[2] + ", " + row[1]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def Prowadzacy():
   name = 'Prowadzacy'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_prowadzacego '" +
row[0].replace("'", " ") + "', '" + row[1].replace("'", " ") + "'"
+ ', null'
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def konferencjePrzyszle():
   name = 'KonferencjePrzyszle'
```

```
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       if(row[12][1] == ":"):
           row[12] = "0" + row[12]
       if(row[15][1] == ":"):
           row[15] = "0" + row[15]
       string = "exec dbo.prod_dodaj_konferencje '" +
row[7].replace("'", " ") + "', " + row[8] + ", " + row[9] + ", " +
row[10] + ", '" + row[13].replace("'", " ") + "', '" + row[11] +
"T" + row[12] + "', '" + row[14] + "T" + row[15] + "'"
       string  = string + ';'
       string = string + '\r\n'
       sqlfile.write(string)

def WarsztatPrzyszly():
   name = 'WarsztatPrzyszly'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_Warsztat '" + row[0] + "', '"
+ row[1] + "', " + row[2] + ", " + row[3] + ", " + row[4] + ", " +
row[5]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def Zamowienia():
   name = 'Zamowienia'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       if(row[4][1] == ":"):
           row[4] = "0" + row[4]
       if(row[1][1] == ":"):
           row[1] = "0" + row[1]
       data_platnosci = "'" + row[3] + "T" + row[4] + "'"
       if(row[6] == ''):
           data_platnosci = "null"
```

```
        string = "exec dbo.prod_dodaj_zamowienie " + data_platnosci
+ ", '" + row[0] + "T" + row[1] + "', " + row[5]
        string  = string + ';'
        string = string + '\r\n'
        string = string + 'go \r\n'
        sqlfile.write(string)

def ZamKonfFirma():
   name = 'ZamowienieKonferencjaFirma3'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_zam_konf_firmowe " + row[0] +
", " + row[1] + ", " + row[2] + ", " + row[3]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def ZamKonfInd():
   name = 'ZamowienieKonferencjaIndywidualne2'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_zam_konf_ind " + row[0] + ",
" + row[1] + ", " + row[2]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def StudentFirmowy():
   name = 'StudentFirmowy2'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_std_do_zam_konf " + row[1] +
", " + row[0]
       string  = string + ';'
       string = string + '\r\n'
```

```
        string = string + 'go \r\n'
        sqlfile.write(string)

def UczestnikFirma():
   name = 'UczestnikFirma'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   klienciFile = open("klienciFirmowi.txt", "r")
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       line = klienciFile.readline()
       if(row[4] == ''):
           row[4] = "null"
       if(line == ''):
           klienciFile.seek(0)
       string = "exec dbo.prod_dodaj_uczestnika_firmowego '" +
row[0].replace("'", " ") + "', '" + row[1].replace("'", " ") + "'
," + row[2] + ", " + line.replace("\n",'') + ", " + row[4]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def ZamowienieWarsztatFirmowe():
   name = 'ZamowienieWarsztatFirmowe'
   csvfile = open(name + '.csv', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_zam_warsztatowe_firmowe " +
row[0] + ", " + row[1]  + ", " + row[2]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def ZamWarszFirm():
   zkf = open("zamKonfFirm.txt")
   warsz = open("Warsztaty.txt")
   res = open("zamWarszFirm.sql", "w+")
   for zkfl in zkf:
       for warszl in warsz:
```

```
            string = "exec dbo.prod_dodaj_zam_warsztatowe_firmowe "
+ str(randint(1,10)) + ", " + str(int(warszl))  + ", " +
str(int(zkfl))
            string  = string + ';'
            string = string + '\r\n'
            string = string + 'go \r\n'
            res.write(string)

def ZamowienieWarsztatFirmowe3():
   name = 'Warsztaty'
   csvfile = open(name + '.txt', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '3'+ '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_zam_warsztatowe_firmowe " +
str(randint(1,5)) + ", " + row[2]  + ", " + row[0]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)


def ZamowienieWarsztatIndywidualne():
   name = 'Warsztaty'
   csvfile = open(name + '.txt', newline = '', encoding =
'utf-8-sig')
   sqlfile = open(name + '3'+ '.sql', 'w+')
   spamreader = csv.reader(csvfile, delimiter = ',')
   for row in spamreader:
       string = "exec dbo.prod_dodaj_zam_warsztatowe_ind " +
row[2]  + ", " + row[0]
       string  = string + ';'
       string = string + '\r\n'
       string = string + 'go \r\n'
       sqlfile.write(string)

def ZapiszNaKonfe():
   uf = open("uczestnikZamKonfFirm.txt")
   res = open("zapiszNaKonfe.sql", "w+")
   spamreader = csv.reader(uf, delimiter = ',')
   for row in spamreader:
           string = "exec dbo.prod_zapisz_na_konferencje " +
row[0]  + ", " + row[1]
           string  = string + ';'
           string = string + '\r\n'
```

```
            string = string + 'go \r\n'
            res.write(string)

def ZapiszNaWarsz():
   uf = open("UczestnikZamWarszFirm.txt")
   res = open("zapiszNaWarsz.sql", "w+")
   spamreader = csv.reader(uf, delimiter = ',')
   for row in spamreader:
            string = "exec dbo.prod_zapisz_na_warsztat " + row[0]
+ ", " + row[1]
            string  = string + ';'
            string = string + '\r\n'
            string = string + 'go \r\n'
            res.write(string)
ZapiszNaKonfe()
ZapiszNaWarsz()
```