

# Podstawy Baz Danych 2023/2024

Projekt: Informatyka, Semestr 3

## Autorzy

Imię i nazwisko	Wkład pracy
Maksymilian Katolik	33.33%
Wiktor Dybalski	33.33%
Mikołaj Nietupski	33.33%

# Spis treści

<b>Autorzy</b>	<b>2</b>
<b>Spis treści</b>	<b>3</b>
<b>Opis funkcji systemu</b>	<b>5</b>
<b>Schemat bazy danych</b>	<b>6</b>
<b>Tabele</b>	<b>7</b>
CenaDlaOsobyZewn	7
Firmy	8
Frekwencja	9
Języki	10
Kursy	11
LimityMiejsc	13
LimityMiejscNaZajecia	14
Linki	15
Miejsca	16
Odrobienia	17
Płatności	18
Praktyki	19
PraktykiFrekwencja	20
Produkty	21
Prowadzacy	23
Studia	24
Tlumacze	25
Tlumaczenia	26
Uczestnicy	27
UczestnicyZmiany	29
Webinary	31
Zajecia	32
Zapisy	33
ZapisyPojedynczeZajecia	34
<b>Widoki</b>	<b>35</b>
ListaDluznikow	35
ListaUczestnikowNaZajeciach	36
ListaUczestnikowZapisanychNaPrzyszleWydarzenia	37
PrzychodyZProduktu	38
PrzychodyZTypuProduktu	39
RaportFrekwencji	40
WyswietlSylabus	42
RaportBilokacji	43
ObecnoscStudentow	44
<b>Procedury</b>	<b>45</b>
Add_Webinar	45

AddCourse	47
AddStudies	49
AddParticipant	51
AddInstructor	52
RemoveWebinar	53
RemoveCourse	54
RemoveStudies	55
RemoveParticipant	56
RemoveInstructor	57
AddPayment	58
ModifyParticipant	59
ModifyProductPrice	61
SModifyOutsiderPrice	62
TakeAttendanceOfAStudent	63
<b>Funkcje</b>	<b>65</b>
GetClassByID	65
GetInstructorByID	66
GetParticipantByID	67
GetPaymentByID	68
GetPaymentsBetweenDates	69
GetProductByID	70
GetProductsAboveXPrice	71
GetProductsBetweenDates	72
GetRegistrationsByDate	73
GetLanguageForClass	74
GetLocationForClass	75
GetProductPriceByID	76
GetSlotsLimit	77
<b>Triggery</b>	<b>78</b>
RemoveWebinarLinkAfter30Days	78
<b>Indeksy</b>	<b>79</b>
<b>Uprawnienia</b>	<b>80</b>
Dyrektor Szkoły	80
Wykładowca	80
Uczestnik	80

## Opis funkcji systemu

- Wypisanie uczestników danych zajęć
- Widok przychodów z produktu lub łącznych przychodów z danego typu produktu
- Generowanie raportów: frekwencja i bilokacja
- Wyświetlanie sylabusu danego kierunku studiów
- Dodawanie/usuwanie użytkowników systemu oraz zmiana ich danych
- Sprawdzanie obecności w trakcie zajęć poprzez dodanie wpisu do frekwencji
- Przechowywanie historii płatności i zapisów



## Tabele

Poniższe sekcje zawierają krótkie opisy poszczególnych tabel oraz kody odpowiedzialne za ich generowanie.

### CenaDlaOsobyZewn

Cena, jaką musi zapłacić za uczestnictwo w zajęciach osoba, która nie jest zapisana na kurs lub studia, którego częścią są te zajęcia.

- id\_zajec (int) – klucz główny
- cena (money) – cena zapisu na dane zajęcia dla osoby spoza kursu/studiów

Warunki integralności

- cena  $\geq$  0

```
CREATE TABLE [dbo].[CenaDlaOsobyZewn](
    [id_zajec] [int] NOT NULL,
    [cena] [money] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[CenaDlaOsobyZewn] ADD CONSTRAINT
[PK_CenaDlaOsobyZewn] PRIMARY KEY CLUSTERED
(
    [id_zajec] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[CenaDlaOsobyZewn] WITH CHECK ADD CONSTRAINT
[FK_CenaDlaOsobyZewn_Zajecia] FOREIGN KEY([id_zajec])
REFERENCES [dbo].[Zajecia] ([id_zajec])
GO
ALTER TABLE [dbo].[CenaDlaOsobyZewn] CHECK CONSTRAINT
[FK_CenaDlaOsobyZewn_Zajecia]
GO
ALTER TABLE [dbo].[CenaDlaOsobyZewn] WITH CHECK ADD CONSTRAINT
[Valid_CenaDlaOsobyZewn] CHECK (([cena]>=(0)))
GO
ALTER TABLE [dbo].[CenaDlaOsobyZewn] CHECK CONSTRAINT
[Valid_CenaDlaOsobyZewn]
GO
```

## Firmy

Dostępne firmy, których studenci (rozumiani jako uczestnicy zapisani na produkty kategorii "studia") mogą odbywać praktyki.

- id\_firmy (int) – klucz główny
- nazwa\_firmy (varchar(50)) – nazwa firmy, w której organizowane są praktyki

```
CREATE TABLE [dbo].[Firmy](
    [id_firmy] [int] NOT NULL,
    [nazwa_firmy] [varchar](50) NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Firmy] ADD CONSTRAINT [PK_Firmy] PRIMARY KEY
CLUSTERED
(
    [id_firmy] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```



## Frekwencja

Lista uczestników zajęć, które się odbyły z zaznaczeniem, czy dany uczestnik był obecny.

- id\_wpisu (int) – klucz główny
- id\_uczestnika (int) – numer uczestnika
- id\_zajec (int) – zajęcia, których dotyczy wpis do frekwencji
- czy\_obecny (bit) – 1 jeśli dany uczestnik był obecny, 0 jeśli był nieobecny

```
CREATE TABLE [dbo].[Frekwencja](
    [id_wpisu] [int] NOT NULL,
    [id_uczestnika] [int] NOT NULL,
    [id_zajec] [int] NOT NULL,
    [czy_obecny] [bit] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Frekwencja] ADD CONSTRAINT [PK_Frekwencja] PRIMARY
KEY CLUSTERED
(
    [id_wpisu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Frekwencja] WITH CHECK ADD CONSTRAINT
[FK_Frekwencja_Uczestnicy] FOREIGN KEY([id_uczestnika])
REFERENCES [dbo].[Uczestnicy] ([id_uczestnika])
GO
ALTER TABLE [dbo].[Frekwencja] CHECK CONSTRAINT
[FK_Frekwencja_Uczestnicy]
GO
ALTER TABLE [dbo].[Frekwencja] WITH CHECK ADD CONSTRAINT
[FK_Frekwencja_Zajecia] FOREIGN KEY([id_zajec])
REFERENCES [dbo].[Zajecia] ([id_zajec])
GO
ALTER TABLE [dbo].[Frekwencja] CHECK CONSTRAINT [FK_Frekwencja_Zajecia]
GO
```

## Języki

Dostępne języki, w których mogą odbywać się zajęcia lub którymi posługują się prowadzący lub tłumacze.

- id\_jezyka (int) – klucz główny
- nazwa (varchar(50)) – język, którego dotyczy dane ID

```
CREATE TABLE [dbo].[Języki](
    [id_jezyka] [int] NOT NULL,
    [nazwa] [varchar](50) NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Języki] ADD CONSTRAINT [PK_Języki] PRIMARY KEY
CLUSTERED
(
    [id_jezyka] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```

## Kursy

Tabela dostępnych kursów.

- id\_produktu (int) – klucz główny
- typ\_modulu (varchar(50)) – dostępne są następujące typy:
  - stacjonarny
  - online-synchroniczny
  - online-asynchroniczny
  - hybrydowy
- id\_prowadzacego (int) – numer identyfikacyjny osoby prowadzącej dany kurs
- wymagana\_zaliczka (money) – cena, którą należy zapłacić przy zapisie na kurs, ale przed zapłaceniem pełnej ceny

Warunki integralności:

- wymagana\_zaliczka > 0

```
CREATE TABLE [dbo].[Kursy](
    [id_produktu] [int] NOT NULL,
    [typ_modulu] [varchar](50) NOT NULL,
    [id_prowadzacego] [int] NOT NULL,
    [wymagana_zaliczka] [money] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Kursy] ADD CONSTRAINT [PK_Kursy] PRIMARY KEY
CLUSTERED
(
    [id_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Kursy] WITH CHECK ADD CONSTRAINT
[FK_Kursy_Produkty] FOREIGN KEY([id_produktu])
REFERENCES [dbo].[Produkty] ([id_produktu])
GO
ALTER TABLE [dbo].[Kursy] CHECK CONSTRAINT [FK_Kursy_Produkty]
GO
ALTER TABLE [dbo].[Kursy] WITH CHECK ADD CONSTRAINT
[FK_Kursy_Prowadzacy] FOREIGN KEY([id_prowadzacego])
REFERENCES [dbo].[Prowadzacy] ([id_prowadzacego])
GO
ALTER TABLE [dbo].[Kursy] CHECK CONSTRAINT [FK_Kursy_Prowadzacy]
GO
ALTER TABLE [dbo].[Kursy] WITH CHECK ADD CONSTRAINT
[Valid_wymagana_zaliczka_kursy] CHECK (([wymagana_zaliczka]>(0)))
GO
```

```
ALTER TABLE [dbo].[Kursy] CHECK CONSTRAINT  
[Valid_wymagana_zaliczka_kursy]  
GO
```

## LimityMiejsc

Liczba dostępnych miejsc dla danego produktu.

- id\_produktu (int) – klucz główny
- limit\_miejsc (int) – liczba dostępnych miejsc

Warunki integralności

- limit\_miejsc > 0

```
CREATE TABLE [dbo].[LimityMiejsc](
    [id_produktu] [int] NOT NULL,
    [limit_miejsc] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[LimityMiejsc] ADD CONSTRAINT [PK_LimityMiejsc]
PRIMARY KEY CLUSTERED
(
    [id_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[LimityMiejsc] WITH CHECK ADD CONSTRAINT
[FK_LimityMiejsc_Produkty] FOREIGN KEY([id_produktu])
REFERENCES [dbo].[Produkty] ([id_produktu])
GO
ALTER TABLE [dbo].[LimityMiejsc] CHECK CONSTRAINT
[FK_LimityMiejsc_Produkty]
GO
ALTER TABLE [dbo].[LimityMiejsc] WITH NOCHECK ADD CONSTRAINT
[Valid_limit_miejsc] CHECK (([limit_miejsc]>(0)))
GO
ALTER TABLE [dbo].[LimityMiejsc] CHECK CONSTRAINT [Valid_limit_miejsc]
GO
```

## LimityMiejscNaZajecia

Liczba dostępnych miejsc dla pojedynczych zajęć.

- id\_zajec (int) – klucz główny 1
- limit\_miejsc (int) – klucz główny 2 liczba dostępnych miejsc

Warunki integralności:

- limit\_miejsc > 0

```
CREATE TABLE [dbo].[LimityMiejscNaZajecia](
    [id_zajec] [int] NOT NULL,
    [limit_miejsc] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[LimityMiejscNaZajecia] ADD CONSTRAINT
[PK_LimityMiejscNaZajecia] PRIMARY KEY CLUSTERED
(
    [id_zajec] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[LimityMiejscNaZajecia] WITH CHECK ADD CONSTRAINT
[FK_LimityMiejscNaZajecia_Zajecia] FOREIGN KEY([id_zajec])
REFERENCES [dbo].[Zajecia] ([id_zajec])
GO
ALTER TABLE [dbo].[LimityMiejscNaZajecia] CHECK CONSTRAINT
[FK_LimityMiejscNaZajecia_Zajecia]
GO
ALTER TABLE [dbo].[LimityMiejscNaZajecia] WITH CHECK ADD CONSTRAINT
[Valid_Limit_Miejsc_Na_Zajecia] CHECK (([limit_miejsc]>(0)))
GO
ALTER TABLE [dbo].[LimityMiejscNaZajecia] CHECK CONSTRAINT
[FK_LimityMiejscNaZajecia_Zajecia]
GO
```

## Linki

Link do spotkania oraz link do nagrania dla odpowiednich zajęć.

- id\_zajec (int) – klucz główny; zajęcia, których dotyczą podane linki
- link\_do\_spotkania (text) – link, którego należy użyć, aby uczestniczyć w wydarzeniu na żywo
- link\_do\_nagrania (text) – link, którego należy użyć, aby obejrzeć zarejestrowane nagranie z wydarzenia

```
CREATE TABLE [dbo].[Linki](
    [id_zajec] [int] NOT NULL,
    [link_do_spotkania] [text] NOT NULL,
    [link_do_nagrania] [text] NOT NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[Linki] ADD CONSTRAINT [PK_Link_i] PRIMARY KEY
CLUSTERED
(
    [id_zajec] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Linki] WITH CHECK ADD CONSTRAINT [FK_Link_i_Zajecia]
FOREIGN KEY([id_zajec])
REFERENCES [dbo].[Zajecia] ([id_zajec])
GO
ALTER TABLE [dbo].[Linki] CHECK CONSTRAINT [FK_Link_i_Zajecia]
GO
```

## Miejsca

Miejsca, w których mogą odbywać się zajęcia ("online" dla zajęć zdalnych).

- id\_miejsca (int) – klucz główny
- nazwa\_miejsca (varchar(50)) – jednoznaczne określenie miejsca odbywania się zajęć, np. poprzez podanie identyfikatora budynku i sali (lub po prostu "online")

```
CREATE TABLE [dbo].[Miejsca](
    [id_miejsca] [int] NOT NULL,
    [nazwa_miejsca] [varchar](50) NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Miejsca] ADD CONSTRAINT [PK_Miejsca] PRIMARY KEY
CLUSTERED
(
    [id_miejsca] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```



## Odrobienia

Informacja na temat uczestnictwa w zajęciach w celu odrobienia nieobecności.

- id\_wpisu (int) – klucz główny; wpis do frekwencji z nieobecnością, której dotyczy odrobienie
- id\_zajec\_zastepczych (int) – zajęcia, w których brał udział uczestnik w celu odrobienia nieobecności

```
CREATE TABLE [dbo].[Odrobienia](
    [id_wpisu] [int] NOT NULL,
    [id_zajec_zastepczych] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Odrobienia] ADD CONSTRAINT [PK_Odrobienia] PRIMARY
KEY CLUSTERED
(
    [id_wpisu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Odrobienia] WITH CHECK ADD CONSTRAINT
[FK_Odrobienia_Frekwencja] FOREIGN KEY([id_wpisu])
REFERENCES [dbo].[Frekwencja] ([id_wpisu])
GO
ALTER TABLE [dbo].[Odrobienia] CHECK CONSTRAINT
[FK_Odrobienia_Frekwencja]
GO
```

## Płatności

Historia dokonanych przez uczestników płatności.

- id\_platnosci (int) – klucz główny
- id\_uczestnika (int) – uczestnik, którego dotyczy płatność
- wartosc (money) – jaka kwota została zapłacona
- id\_produkту (int) – za jaki produkt zapłacono
- data\_dokonania\_opłaty (datetime) – kiedy miała miejsce płatność

```
CREATE TABLE [dbo].[Platnosci](
    [id_platnosci] [int] NOT NULL,
    [id_uczestnika] [int] NOT NULL,
    [wartosc] [money] NOT NULL,
    [id_produkту] [int] NOT NULL,
    [data_dokonania_opłaty] [datetime] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Platnosci] ADD CONSTRAINT [PK_Platnosci] PRIMARY KEY
CLUSTERED
(
    [id_platnosci] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Platnosci] WITH CHECK ADD CONSTRAINT
[FK_Platnosci_Produkty] FOREIGN KEY([id_produkту])
REFERENCES [dbo].[Produkty] ([id_produkту])
GO
ALTER TABLE [dbo].[Platnosci] CHECK CONSTRAINT [FK_Platnosci_Produkty]
GO
ALTER TABLE [dbo].[Platnosci] WITH CHECK ADD CONSTRAINT
[FK_Platnosci_Uczestnicy] FOREIGN KEY([id_uczestnika])
REFERENCES [dbo].[Uczestnicy] ([id_uczestnika])
GO
ALTER TABLE [dbo].[Platnosci] CHECK CONSTRAINT [FK_Platnosci_Uczestnicy]
GO
```

## Praktyki

Informacja na temat praktyk, które muszą odbyć uczestnicy zapisani na studia.

- id\_praktyki (int) – klucz główny
- id\_uczestnika (int) – uczestnik, który odbywał dane praktyki
- id\_firmy (int) – firma, w której miały miejsce praktyki
- id\_studiów (int) – studia, w ramach których odbyły się praktyki

```
CREATE TABLE [dbo].[Praktyki](
    [id_praktyki] [int] NOT NULL,
    [id_uczestnika] [int] NOT NULL,
    [id_firmy] [int] NOT NULL,
    [id_studiów] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Praktyki] ADD CONSTRAINT [PK_Praktyki] PRIMARY KEY
CLUSTERED
(
    [id_praktyki] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Praktyki] WITH CHECK ADD CONSTRAINT
[FK_Praktyki_Firmy] FOREIGN KEY([id_firmy])
REFERENCES [dbo].[Firmy] ([id_firmy])
GO
ALTER TABLE [dbo].[Praktyki] CHECK CONSTRAINT [FK_Praktyki_Firmy]
GO
ALTER TABLE [dbo].[Praktyki] WITH CHECK ADD CONSTRAINT
[FK_Praktyki_Studia] FOREIGN KEY([id_studiów])
REFERENCES [dbo].[Studia] ([id_produkty])
GO
ALTER TABLE [dbo].[Praktyki] CHECK CONSTRAINT [FK_Praktyki_Studia]
GO
ALTER TABLE [dbo].[Praktyki] WITH CHECK ADD CONSTRAINT
[FK_Praktyki_Uczestnicy] FOREIGN KEY([id_uczestnika])
REFERENCES [dbo].[Uczestnicy] ([id_uczestnika])
GO
ALTER TABLE [dbo].[Praktyki] CHECK CONSTRAINT [FK_Praktyki_Uczestnicy]
GO
```

## PraktykiFrekwencja

Informacja na temat obecności studentów na praktykach.

- id\_praktyki (int) – klucz główny 1; której praktyki dotyczy wpis do frekwencji
- nr\_dnia (int) – klucz główny 2; numer dnia danych praktyk (od 1 do 14)
- czy\_zaliczony (bit) – informacja, czy dany dzień praktyk został zaliczony przez studenta

Warunki integralności:

- 1 >= nr\_dnia
- nr\_dnia <= 14

```
CREATE TABLE [dbo].[PraktykiFrekwencja](
    [id_praktyki] [int] NOT NULL,
    [nr_dnia] [int] NOT NULL,
    [czy_zaliczony] [bit] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[PraktykiFrekwencja] ADD CONSTRAINT
[PK_PraktykiFrekwencja] PRIMARY KEY CLUSTERED
(
    [id_praktyki] ASC,
    [nr_dnia] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[PraktykiFrekwencja] WITH CHECK ADD CONSTRAINT
[FK_PraktykiFrekwencja_Praktyki] FOREIGN KEY([id_praktyki])
REFERENCES [dbo].[Praktyki] ([id_praktyki])
GO
ALTER TABLE [dbo].[PraktykiFrekwencja] CHECK CONSTRAINT
[FK_PraktykiFrekwencja_Praktyki]
GO
ALTER TABLE [dbo].[PraktykiFrekwencja] WITH NOCHECK ADD CONSTRAINT
[Valid_nr_dnia] CHECK (((1)<=[nr_dnia] AND [nr_dnia]<=(14)))
GO
ALTER TABLE [dbo].[PraktykiFrekwencja] CHECK CONSTRAINT [Valid_nr_dnia]
GO
```

## Produkty

Informacja na temat wszystkich produktów, składająca się z cech, które są wspólne dla webinarów, kursów i studiów (nazwa, data rozpoczęcia, data zakończenia, cena i język).

- id\_produktu (int) – klucz główny
- typ (varchar(50)) – webinar, kurs lub studia
- nazwa (varchar(50)) – pełna nazwa produktu
- data\_start (datetime) – data rozpoczęcia
- data\_koniec (datetime) – data zakończenia
- cena (money) – pełna kwota, jaką należy zapłacić za uczestnictwo
- id\_jezyka (int) – w jakim języku prowadzone są zajęcia w ramach danego produktu

Warunki integralności:

- data\_start < data\_koniec
- cena >= 0

```
CREATE TABLE [dbo].[Produkty](
    [id_produktu] [int] NOT NULL,
    [typ] [varchar](50) NOT NULL,
    [nazwa] [varchar](50) NOT NULL,
    [data_start] [datetime] NOT NULL,
    [data_koniec] [datetime] NOT NULL,
    [cena] [money] NOT NULL,
    [id_jezyka] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Produkty] ADD CONSTRAINT [PK_Produkty] PRIMARY KEY
CLUSTERED
(
    [id_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Produkty] WITH CHECK ADD CONSTRAINT
[FK_Produkty_Jezyki] FOREIGN KEY([id_jezyka])
REFERENCES [dbo].[Jezyki] ([id_jezyka])
GO
ALTER TABLE [dbo].[Produkty] CHECK CONSTRAINT [FK_Produkty_Jezyki]
GO
ALTER TABLE [dbo].[Produkty] WITH CHECK ADD CONSTRAINT [Valid_cena]
CHECK (([cena]>=(0)))
GO
ALTER TABLE [dbo].[Produkty] CHECK CONSTRAINT [Valid_cena]
GO
ALTER TABLE [dbo].[Produkty] WITH CHECK ADD CONSTRAINT [ValidDate]
```

```
CHECK ([data_koniec]>[data_start]))  
GO  
ALTER TABLE [dbo].[Produkty] CHECK CONSTRAINT [ValidDate]  
GO
```

## Prowadzacy

Informacje na temat prowadzących.

- id\_prowadzacego (int) – klucz główny
- imie (varchar(50)) – imię prowadzącego
- nazwisko (varchar(50)) – nazwisko prowadzącego
- id\_jezyka (int) – język, którym posługuje się prowadzący
- adres\_email (varchar(50)) – adres e-mail przeznaczony do kontaktu z prowadzącym

Warunki integralności:

- adres\_email: zawiera '@' i jest unikatowy

```
CREATE TABLE [dbo].[Prowadzacy](
    [id_prowadzacego] [int] NOT NULL,
    [imie] [varchar](50) NOT NULL,
    [nazwisko] [varchar](50) NOT NULL,
    [id_jezyka] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Prowadzacy] ADD CONSTRAINT [PK_Prowadzacy] PRIMARY
KEY CLUSTERED
(
    [id_prowadzacego] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Prowadzacy] ADD CONSTRAINT [Unique_Instructor_Email]
UNIQUE NONCLUSTERED
(
    [adres_email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Prowadzacy] WITH CHECK ADD CONSTRAINT
[FK_Prowadzacy_Jezyki] FOREIGN KEY([id_jezyka])
REFERENCES [dbo].[Jezyki] ([id_jezyka])
GO
ALTER TABLE [dbo].[Prowadzacy] CHECK CONSTRAINT [FK_Prowadzacy_Jezyki]
GO
```

## Studia

Dostępne kierunki studiów wraz z informacją na temat zaliczki i sylabusem.

- id\_produktu (int) – klucz główny
- wymagana\_zaliczka (money) – cena, którą należy zapłacić przy zapisie na studia przed zapłaceniem pełnej ceny
- sylabus (text) – informacja na temat przedmiotów, które odbywają się w ramach danego kierunku studiów

Warunki integralności:

- wymagana\_zaliczka > 0

```
CREATE TABLE [dbo].[Studia](
    [id_produktu] [int] NOT NULL,
    [wymagana_zaliczka] [money] NOT NULL,
    [sylabus] [text] NOT NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[Studia] ADD CONSTRAINT [PK_Studia] PRIMARY KEY
CLUSTERED
(
    [id_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Studia] WITH CHECK ADD CONSTRAINT
[FK_Studia_Produkty] FOREIGN KEY([id_produktu])
REFERENCES [dbo].[Produkty] ([id_produktu])
GO
ALTER TABLE [dbo].[Studia] CHECK CONSTRAINT [FK_Studia_Produkty]
GO
ALTER TABLE [dbo].[Studia] WITH CHECK ADD CONSTRAINT
[Valid_Wymagana_Zaliczka] CHECK (([wymagana_zaliczka]>(0)))
GO
ALTER TABLE [dbo].[Studia] CHECK CONSTRAINT [Valid_Wymagana_Zaliczka]
GO
```



## Tłumacze

Osoby odpowiedzialne za tłumaczenia zajęć na inny język.

- id\_tlumacza (int) – klucz główny
- imie (varchar(50)) – imię tłumacza
- nazwisko (varchar(50)) – nazwisko tłumacza
- id\_jezyka (int) – z jakiego języka tłumaczy dana osoba (na język polski)

```
CREATE TABLE [dbo].[Tlumacze](
    [id_tlumacza] [int] NOT NULL,
    [imie] [varchar](50) NOT NULL,
    [nazwisko] [varchar](50) NOT NULL,
    [id_jezyka] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Tlumacze] ADD CONSTRAINT [PK_Tlumacze] PRIMARY KEY
CLUSTERED
(
    [id_tlumacza] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Tlumacze] WITH CHECK ADD CONSTRAINT
[FK_Tlumacze_Jezyki] FOREIGN KEY([id_jezyka])
REFERENCES [dbo].[Jezyki] ([id_jezyka])
GO
ALTER TABLE [dbo].[Tlumacze] CHECK CONSTRAINT [FK_Tlumacze_Jezyki]
GO
```

## Tłumaczenia

Informacje na temat zajęć, które zostały przetłumaczone z jednego języka na inny.

- id\_zajec (int) – klucz główny
- id\_tlumacza (int) – osoba odpowiedzialna za przetłumaczenie zajęć na język polski

```
CREATE TABLE [dbo].[Tlumaczenia](
    [id_zajec] [int] NOT NULL,
    [id_tlumacza] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Tlumaczenia] ADD CONSTRAINT [PK_Tlumaczenia] PRIMARY
KEY CLUSTERED
(
    [id_zajec] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Tlumaczenia] WITH CHECK ADD CONSTRAINT
[FK_Tlumaczenia_Tlumacze] FOREIGN KEY([id_tlumacza])
REFERENCES [dbo].[Tlumacze] ([id_tlumacza])
GO
ALTER TABLE [dbo].[Tlumaczenia] CHECK CONSTRAINT
[FK_Tlumaczenia_Tlumacze]
GO
ALTER TABLE [dbo].[Tlumaczenia] WITH CHECK ADD CONSTRAINT
[FK_Tlumaczenia_Zajecia] FOREIGN KEY([id_zajec])
REFERENCES [dbo].[Zajecia] ([id_zajec])
GO
ALTER TABLE [dbo].[Tlumaczenia] CHECK CONSTRAINT
[FK_Tlumaczenia_Zajecia]
GO
```

## Uczestnicy

Dane osobowe uczestników i ich status aktywności.

- id\_uczestnika (int) – klucz główny
- imie (varchar(50)) – imię uczestnika
- nazwisko (varchar(50)) – nazwisko uczestnika
- adres\_pocztowy (varchar(50)) – adres pocztowy uczestnika, który może się przydać m.in. w wysyłaniu certyfikatu ukończenia kursu
- adres\_email (varchar(50)) – adres e-mail przeznaczony do kontaktu z uczestnikiem
- data\_ur (date) – data urodzenia
- czy\_aktywny (bit) – 1 jeśli uczestnik jest aktywny, 0 jeśli przerwał lub zakończył swoją aktywność

Warunki integralności:

- adres\_email zawiera '@' i unikatowy

```
CREATE TABLE [dbo].[Uczestnicy](
    [id_uczestnika] [int] NOT NULL,
    [imie] [varchar](50) NOT NULL,
    [nazwisko] [varchar](50) NOT NULL,
    [adres_pocztowy] [varchar](50) NOT NULL,
    [adres_email] [varchar](50) NOT NULL,
    [data_ur] [date] NOT NULL,
    [czy_aktywny] [bit] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Uczestnicy] ADD CONSTRAINT [PK_Uczestnicy] PRIMARY
KEY CLUSTERED
(
    [id_uczestnika] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
SET ANSI_PADDING ON
GO
ALTER TABLE [dbo].[Uczestnicy] ADD CONSTRAINT [Unique_Email] UNIQUE
NONCLUSTERED
(
    [adres_email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Uczestnicy] WITH CHECK ADD CONSTRAINT [Valid_email]
```

```
CHECK ([adres_email] like '%@%'))  
GO  
ALTER TABLE [dbo].[Uczestnicy] CHECK CONSTRAINT [Valid_email]  
GO
```

## UczestnicyZmiany

Historia zmian dotyczących danych osobowych uczestników. Jeśli pole nie zostało zmienione, jego stara i nowa wartość pozostają takie same.

- id\_zmiany (int) – klucz główny
- id\_uczestnika (int) – uczestnik, którego dotyczy zmiana
- data\_zmiany (datetime) – data, w której dokonano zmiany
- stare\_imie (varchar(50)) – imię przed zmianą
- stare\_nazwisko (varchar(50)) – nazwisko przed zmianą
- stary\_adres\_pocztowy (varchar(50)) – adres pocztowy przed zmianą
- stary\_adres\_email (varchar(50)) – adres e-mail przed zmianą
- stary\_status\_czy\_aktywny (bit) – status aktywności przed zmianą
- nowe\_imie (varchar(50)) – imię po zmianie
- nowe\_nazwisko (varchar(50)) – nazwisko po zmianie
- nowy\_adres\_pocztowy (varchar(50)) – adres pocztowy po zmianie
- nowy\_adres\_email (varchar(50)) – adres e-mail po zmianie
- nowy\_status\_czy\_aktywny (bit) – status aktywności po zmianie

Warunki integralności:

- stary\_adres\_email zawiera '@' i unikatowy
- nowy\_adres\_email zawiera '@' i unikatowy

```
CREATE TABLE [dbo].[UczestnicyZmiany](
    [id_zmiany] [int] NOT NULL,
    [id_uczestnika] [int] NOT NULL,
    [data_zmiany] [datetime] NOT NULL,
    [stare_imie] [varchar](50) NOT NULL,
    [stare_nazwisko] [varchar](50) NOT NULL,
    [stary_adres_pocztowy] [varchar](50) NOT NULL,
    [stary_adres_email] [varchar](50) NOT NULL,
    [stary_status_czy_aktywny] [bit] NOT NULL,
    [nowe_imie] [varchar](50) NOT NULL,
    [nowe_nazwisko] [varchar](50) NOT NULL,
    [nowy_adres_pocztowy] [varchar](50) NOT NULL,
    [nowy_adres_email] [varchar](50) NOT NULL,
    [nowy_status_czy_aktywny] [bit] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[UczestnicyZmiany] ADD CONSTRAINT
[PK_UczestnicyZmiany] PRIMARY KEY CLUSTERED
(
    [id_zmiany] ASC
) WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
```

```
SET ANSI_PADDING ON
GO
ALTER TABLE [dbo].[UczestnicyZmiany] ADD CONSTRAINT [Unique_New_Email]
UNIQUE NONCLUSTERED
(
    [nowy_adres_email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
SET ANSI_PADDING ON
GO
ALTER TABLE [dbo].[UczestnicyZmiany] ADD CONSTRAINT [Unique_Old_Email]
UNIQUE NONCLUSTERED
(
    [stary_adres_email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[UczestnicyZmiany] WITH CHECK ADD CONSTRAINT
[FK_UczestnicyZmiany_Uczestnicy] FOREIGN KEY([id_uczestnika])
REFERENCES [dbo].[Uczestnicy] ([id_uczestnika])
GO
ALTER TABLE [dbo].[UczestnicyZmiany] CHECK CONSTRAINT
[FK_UczestnicyZmiany_Uczestnicy]
GO
ALTER TABLE [dbo].[UczestnicyZmiany] WITH CHECK ADD CONSTRAINT
[valid_new_email] CHECK (([nowy_adres_email] like '%@%'))
GO
ALTER TABLE [dbo].[UczestnicyZmiany] CHECK CONSTRAINT [valid_new_email]
GO
ALTER TABLE [dbo].[UczestnicyZmiany] WITH CHECK ADD CONSTRAINT
[valid_old_email] CHECK (([stary_adres_email] like '%@%'))
GO
ALTER TABLE [dbo].[UczestnicyZmiany] CHECK CONSTRAINT [valid_old_email]
GO
```

## Webinary

Informacja na temat organizowanych webinarów.

- id\_produktu (int) – klucz główny
- id\_prowadzacego (int) – osoba prowadząca webinar

```
CREATE TABLE [dbo].[Webinary](
    [id_produktu] [int] NOT NULL,
    [id_prowadzacego] [int] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Webinary] ADD CONSTRAINT [PK_Webinary] PRIMARY KEY
CLUSTERED
(
    [id_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Webinary] WITH CHECK ADD CONSTRAINT
[FK_Webinary_Produkty] FOREIGN KEY([id_produktu])
REFERENCES [dbo].[Produkty] ([id_produktu])
GO
ALTER TABLE [dbo].[Webinary] CHECK CONSTRAINT [FK_Webinary_Produkty]
GO
ALTER TABLE [dbo].[Webinary] WITH CHECK ADD CONSTRAINT
[FK_Webinary_Prowadzacy] FOREIGN KEY([id_prowadzacego])
REFERENCES [dbo].[Prowadzacy] ([id_prowadzacego])
GO
ALTER TABLE [dbo].[Webinary] CHECK CONSTRAINT [FK_Webinary_Prowadzacy]
GO
```

## Zajecia

Istotne informacje na temat zajęć odbywających się w ramach produktów.

- id\_zajec (int) – klucz główny
- id\_produktu (int) – produkt, w ramach których odbywają się zajęcia
- id\_miejsca (int) – miejsce, w którym odbywają się zajęcia
- data\_godz\_start (datetime) – data i godzina rozpoczęcia
- data\_godz\_koniec (datetime) – data i godzina zakończenia

Warunki integralności:

- data\_godz\_start < data\_godz\_koniec

```
CREATE TABLE [dbo].[Zajecia](
    [id_zajec] [int] NOT NULL,
    [id_produktu] [int] NOT NULL,
    [id_miejsca] [int] NOT NULL,
    [data_godz_start] [datetime] NOT NULL,
    [data_godz_koniec] [datetime] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Zajecia] ADD CONSTRAINT [PK_Zajecia] PRIMARY KEY
CLUSTERED
(
    [id_zajec] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Zajecia] WITH CHECK ADD CONSTRAINT
[FK_Zajecia_Miejsca] FOREIGN KEY([id_miejsca])
REFERENCES [dbo].[Miejsca] ([id_miejsca])
GO
ALTER TABLE [dbo].[Zajecia] CHECK CONSTRAINT [FK_Zajecia_Miejsca]
GO
ALTER TABLE [dbo].[Zajecia] WITH CHECK ADD CONSTRAINT
[FK_Zajecia_Produkty] FOREIGN KEY([id_produktu])
REFERENCES [dbo].[Produkty] ([id_produktu])
GO
ALTER TABLE [dbo].[Zajecia] CHECK CONSTRAINT [FK_Zajecia_Produkty]
GO
ALTER TABLE [dbo].[Zajecia] WITH CHECK ADD CONSTRAINT [Valid_Date]
CHECK (([data_godz_koniec]>[data_godz_start]))
GO
ALTER TABLE [dbo].[Zajecia] CHECK CONSTRAINT [Valid_Date]
GO
```



## Zapisy

Przypisanie uczestników do zajęć z zaznaczeniem, kiedy dokonano zapisu oraz czy opłata została uiszczona.

- id\_uczestnika (int) – klucz główny 1
- id\_produktu (int) – klucz główny 2
- data\_zapisu (datetime) – data dokonania zapisu na produkt przez uczestnika
- czy\_zaplacono (bit) – 1 jeśli dokonano opłaty, 0 jeśli nie dokonano opłaty

```
CREATE TABLE [dbo].[Zapisy](
    [id_uczestnika] [int] NOT NULL,
    [id_produktu] [int] NOT NULL,
    [data_zapisu] [datetime] NOT NULL,
    [czy_zaplacono] [bit] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Zapisy] ADD CONSTRAINT [PK_Zapisy] PRIMARY KEY
CLUSTERED
(
    [id_uczestnika] ASC,
    [id_produktu] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Zapisy] WITH CHECK ADD CONSTRAINT
[FK_Zapisy_Produkty] FOREIGN KEY([id_produktu])
REFERENCES [dbo].[Produkty] ([id_produktu])
GO
ALTER TABLE [dbo].[Zapisy] CHECK CONSTRAINT [FK_Zapisy_Produkty]
GO
ALTER TABLE [dbo].[Zapisy] WITH CHECK ADD CONSTRAINT
[FK_Zapisy_Uczestnicy] FOREIGN KEY([id_uczestnika])
REFERENCES [dbo].[Uczestnicy] ([id_uczestnika])
GO
ALTER TABLE [dbo].[Zapisy] CHECK CONSTRAINT [FK_Zapisy_Uczestnicy]
GO
```

## ZapisyPojedynczeZajecia

Tabela odpowiadająca za przypisanie do zajęć uczestników, którzy nie są zapisani na kurs lub studia obejmujące te zajęcia.

- id\_uczestnika (int) – klucz główny 1
- id\_zajec (int) – klucz główny 2
- data\_zapisu (datetime) – data dokonania zapisu na zajęcia przez uczestnika
- czy\_zaplacono (bit) – 1 jeśli dokonano opłaty, 0 jeśli nie dokonano opłaty

```
CREATE TABLE [dbo].[ZapisyPojedynczeZajecia](
    [id_uczestnika] [int] NOT NULL,
    [id_zajec] [int] NOT NULL,
    [data_zapisu] [datetime] NOT NULL,
    [czy_zaplacono] [bit] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[ZapisyPojedynczeZajecia] ADD CONSTRAINT
[PK_ZapisyPojedynczeZajecia] PRIMARY KEY CLUSTERED
(
    [id_uczestnika] ASC,
    [id_zajec] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB =
OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
GO
ALTER TABLE [dbo].[ZapisyPojedynczeZajecia] WITH CHECK ADD CONSTRAINT
[FK_ZapisyPojedynczeZajecia_Uczestnicy] FOREIGN KEY([id_uczestnika])
REFERENCES [dbo].[Uczestnicy] ([id_uczestnika])
GO
ALTER TABLE [dbo].[ZapisyPojedynczeZajecia] CHECK CONSTRAINT
[FK_ZapisyPojedynczeZajecia_Uczestnicy]
GO
ALTER TABLE [dbo].[ZapisyPojedynczeZajecia] WITH CHECK ADD CONSTRAINT
[FK_ZapisyPojedynczeZajecia_Zajecia] FOREIGN KEY([id_zajec])
REFERENCES [dbo].[Zajecia] ([id_zajec])
GO
ALTER TABLE [dbo].[ZapisyPojedynczeZajecia] CHECK CONSTRAINT
[FK_ZapisyPojedynczeZajecia_Zajecia]
GO
```

# Widoki

## ListaDluznikow

Wyświetlanie listy dłużników (osób, które zapisały się na dany produkt, a nie zapłaciły za niego w terminie krótszym niż 3 dni do jego rozpoczęcia).

```
CREATE VIEW [dbo].[ListaDluznikow] AS

SELECT Uczestnicy.id_uczestnika, Uczestnicy.imie, Uczestnicy.nazwisko
FROM Uczestnicy
WHERE Uczestnicy.id_uczestnika IN (
    SELECT DISTINCT Uczestnicy.id_uczestnika
    FROM Uczestnicy
    INNER JOIN Zapisy
    ON Uczestnicy.id_uczestnika = Zapisy.id_uczestnika
    INNER JOIN Produkty
    ON Zapisy.id_produktu = Produkty.id_produktu
    WHERE Zapisy.czy_zaplacono = 0
    AND DATEADD(day, -3, Produkty.data_start) < GETDATE()
)

GO
```

## ListaUczestnikowNaZajeciach

Wyświetlanie listy osób, które uczestniczą w danych zajęciach.

```
CREATE VIEW [dbo].[ListaUczestnikowNaZajeciach] AS

SELECT DISTINCT Zajecia.id_zajec, Produkty.nazwa,
Uczestnicy.id_uczestnika, Uczestnicy.imie, Uczestnicy.nazwisko,
Frekwencja.czy_obecny
FROM Zajecia
INNER JOIN Produkty
ON Zajecia.id_produktu = Produkty.id_produktu
INNER JOIN Frekwencja
ON Zajecia.id_zajec = Frekwencja.id_zajec
INNER JOIN Uczestnicy
ON Frekwencja.id_uczestnika = Uczestnicy.id_uczestnika
GROUP BY Zajecia.id_zajec, Produkty.nazwa, Uczestnicy.id_uczestnika,
Uczestnicy.imie, Uczestnicy.nazwisko, Frekwencja.czy_obecny

GO
```

## ListaUczestnikowZapisanychNaPrzyszleWydarzenia

Lista uczestników zapisanych na wydarzenia, które jeszcze się nie odbyły.

```
CREATE VIEW [dbo].[ListaUczestnikowZapisanychNaPrzyszleWydarzenia] AS

SELECT DISTINCT Uczestnicy.id_uczestnika, Uczestnicy.imie,
Uczestnicy.nazwisko, Zajecia.id_zajec, Zajecia.data_godz_start,
Produkty.nazwa
FROM Zajecia
INNER JOIN Produkty
ON Zajecia.id_produktu = Produkty.id_produktu
INNER JOIN Zapisy
ON Produkty.id_produktu = Zapisy.id_produktu
LEFT OUTER JOIN ZapisyPojedynczeZajecia ON Zajecia.id_zajec =
ZapisyPojedynczeZajecia.id_zajec
INNER JOIN Uczestnicy
ON Zapisy.id_uczestnika = Uczestnicy.id_uczestnika OR
ZapisyPojedynczeZajecia.id_uczestnika = Uczestnicy.id_uczestnika
WHERE GETDATE() < Zajecia.data_godz_start AND Uczestnicy.czy_aktywny = 1

GROUP BY Uczestnicy.id_uczestnika, Uczestnicy.imie, Uczestnicy.nazwisko,
Zajecia.id_zajec, Zajecia.data_godz_start, Produkty.nazwa

GO
```

## PrzychodyZProduktu

Łączny przychód z poszczególnych produktów.

```
CREATE VIEW [dbo].[PrzychodyZProduktu] AS

SELECT Produkty.id_produktu, Produkty.nazwa, SUM(Platnosci.wartosc) AS
laczny_przychod
FROM Produkty
INNER JOIN Zapisy
ON Produkty.id_produktu = Zapisy.id_produktu
INNER JOIN Uczestnicy
ON Zapisy.id_uczestnika = Uczestnicy.id_uczestnika
INNER JOIN Platnosci
ON Uczestnicy.id_uczestnika = Platnosci.id_uczestnika AND
Produkty.id_produktu = Platnosci.id_produktu
GROUP BY Produkty.id_produktu, Produkty.nazwa

GO
```

## PrzychodyZTypuProduktu

Łączny przychód z poszczególnych typów produktów (webinary/kursy/studia).

```
CREATE VIEW [dbo].[PrzychodyZTypuProduktu] AS

SELECT Produkty.typ, SUM(Platnosci.wartosc) AS laczny_przychod_z_typu
FROM Produkty
INNER JOIN Zapisy
ON Produkty.id_produktu = Zapisy.id_produktu
INNER JOIN Uczestnicy
ON Zapisy.id_uczestnika = Uczestnicy.id_uczestnika
INNER JOIN Platnosci
ON Uczestnicy.id_uczestnika = Platnosci.id_uczestnika AND
Produkty.id_produktu = Platnosci.id_produktu
GROUP BY Produkty.typ

GO
```

## RaportFrekwencji

Raport dotyczący frekwencji na zajęciach (wynik procentowy).

```
CREATE VIEW [dbo].[RaportFrekwencji] AS

WITH Wszyscy AS
(
    SELECT Zajecia.id_zajec AS w_id, COUNT(*) AS liczba_zapisanych
    FROM Zajecia
    INNER JOIN Frekwencja
    ON Zajecia.id_zajec = Frekwencja.id_zajec
    GROUP BY Zajecia.id_zajec
),

Obecni AS
(
    SELECT Zajecia.id_zajec AS o_id, COUNT(*) AS liczba_obecných
    FROM Zajecia
    INNER JOIN Frekwencja
    ON Zajecia.id_zajec = Frekwencja.id_zajec
    WHERE Frekwencja.czy_obecny = 1
    GROUP BY Zajecia.id_zajec
),

Nieobecni AS
(
    SELECT Zajecia.id_zajec AS n_id, COUNT(*) AS liczba_nieobecných
    FROM Zajecia
    INNER JOIN Frekwencja
    ON Zajecia.id_zajec = Frekwencja.id_zajec
    WHERE Frekwencja.czy_obecny = 0
    GROUP BY Zajecia.id_zajec
)

SELECT Zajecia.id_zajec, Zajecia.data_godz_start,
Zajecia.data_godz_koniec, Wszyscy.liczba_zapisanych AS
wszyscy_uczestnicy, Obecni.liczba_obecných AS obecni_uczestnicy,
Nieobecni.liczba_nieobecných AS nieobecni_uczestnicy,
ROUND((CAST(Obecni.liczba_obecných AS float) /
CAST(Wszyscy.liczba_zapisanych AS float))*100, 2 ) AS procent_obecných
FROM Zajecia
INNER JOIN Wszyscy
ON Zajecia.id_zajec = Wszyscy.w_id
INNER JOIN Obecni
ON Zajecia.id_zajec = Obecni.o_id
```



```
INNER JOIN Nieobecni  
ON Zajecia.id_zajec = Nieobecni.n_id
```

```
GO
```

## WyswietlSylabus

Możliwość sprawdzenia sylabusu dla danego kierunku studiów.

```
CREATE VIEW [dbo].[WyswietlSylabus] AS  
  
SELECT Studia.id_produktu, Produkty.nazwa, sylabus  
FROM Studia  
INNER JOIN Produkty ON Produkty.id_produktu = Studia.id_produktu  
  
GO
```

## RaportBilokacji

Wyświetla studentów zapisanych na zajęcia kolidujące ze sobą czasowo.

```
CREATE VIEW RaportBilokacji AS

WITH Pary1 AS
(
    SELECT Uczestnicy.id_uczestnika AS Ucz, Z1.id_zajec AS Zaj,
    Z1.data_godz_start AS Ds, Z1.data_godz_koniec AS Dk
    FROM Uczestnicy
    INNER JOIN Zapisy
    ON Uczestnicy.id_uczestnika = Zapisy.id_uczestnika
    INNER JOIN Produkty
    ON Zapisy.id_produkту = Produkty.id_produkту
    INNER JOIN Zajecia AS Z1
    ON Produkty.id_produkту = Z1.id_produkту
)

SELECT DISTINCT P1.Ucz AS Uczestnik, P1.Zaj AS Zaj1, P1.Ds AS Zaj1Start,
P1.Dk AS Zaj1Koniec, P2.Zaj AS Zaj2, P2.Ds AS Zaj2Start, P2.Dk AS
Zaj2Koniec
FROM Pary1 AS P1
INNER JOIN Pary1 AS P2
ON P1.Ucz = P2.Ucz AND P1.Zaj < P2.Zaj AND (
    (P2.Ds >= P1.Ds AND P2.Ds <= P1.Dk)
    OR
    (P1.Ds >= P2.Ds AND P1.Ds <= P2.Dk)
)

GO
;
```

## ObecnoscStudentow

Wyświetla, jak często (0.00 – 1.00) dany student był obecny na zajęciach odbywających się w ramach studiów. Przydatne do określania, czy student spełnił wymóg minimalnej obecności (np. uczęszczał na 80% zajęć).

```
CREATE VIEW ObecnoscStudentow AS

SELECT Studia.id_produktu, Uczestnicy.id_uczestnika, Uczestnicy.imie,
Uczestnicy.nazwisko, ROUND(AVG(CAST(Frekwencja.czy_obecny AS float)),2)
AS SredniaObecnosc
FROM Studia
INNER JOIN Produkty
ON Studia.id_produktu = Produkty.id_produktu
INNER JOIN Zajecia
ON Produkty.id_produktu = Zajecia.id_produktu
INNER JOIN Frekwencja
ON Zajecia.id_zajec = Frekwencja.id_zajec
INNER JOIN Uczestnicy
ON Frekwencja.id_uczestnika = Uczestnicy.id_uczestnika
GROUP BY Studia.id_produktu, Uczestnicy.id_uczestnika, Uczestnicy.imie,
Uczestnicy.nazwisko

GO
```

# Procedury

## Add\_Webinar

Dodaje Webinar do bazy przyjmując jako zmienne id\_prowadzacego, nazwe webinaru, datę i czas startu, datę i czas zakończenia, cenę oraz nazwę języka

```
CREATE PROCEDURE Add_Webinar @id_prowadzacego int,
                             @nazwa varchar(50),
                             @data_start datetime,
                             @data_koniec datetime,
                             @cena money,
                             @nazwa_jezyka varchar(50)

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Produkty
            WHERE nazwa = @nazwa
        )
        BEGIN
            ;
            THROW 52000, 'Taki Webinar juz istnieje', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM Języki
            WHERE nazwa = @nazwa_jezyka
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego języka', 1
        END
        DECLARE @id_jezyka INT
        SELECT TOP 1 @id_jezyka = id_jezyka
        FROM Języki
        WHERE nazwa = @nazwa_jezyka
        DECLARE @id_produktu INT
        SELECT @id_produktu = ISNULL(MAX(id_produktu), 0) + 1
        FROM Produkty
        INSERT INTO Webinary(id_produktu, id_prowadzacego)
```

```
VALUES (@id_produktu, @id_prowadzacego);
INSERT INTO Produkty(id_produktu, typ, nazwa, data_start,
data_koniec, cena, id_jezyka )
VALUES (@id_produktu, 'Webinar', @nazwa, @data_start,
@data_koniec, @cena, @id_jezyka);
END TRY
BEGIN CATCH
    DECLARE @msg varchar(255)
        = 'Bład dodadnia Webinaru: ' + ERROR_MESSAGE();
    THROW 52000, @msg, 1;
END CATCH
END
GO
```

## AddCourse

Dodaje kurs do bazy, przyjmując jako zmienne id prowadzącego, nazwę kursu, datę rozpoczęcia, datę zakończenia, cenę, wymaganą zaliczkę i nazwę języka.

```
CREATE PROCEDURE AddCourse @id_prowadzacego int,
                           @typ_modulu varchar(50),
                           @nazwa varchar(50),
                           @data_start datetime,
                           @data_koniec datetime,
                           @cena money,
                           @wymagana_zaliczka money,
                           @nazwa_jezyka varchar(50)

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Produkty
            WHERE nazwa = @nazwa
        )
        BEGIN
            ;
            THROW 52000, 'Taki produkt juz istnieje', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM Jezyki
            WHERE nazwa = @nazwa_jezyka
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego jezyka', 1
        END
        IF (@wymagana_zaliczka > @cena)
        BEGIN
            ;
            THROW 52000, 'Wymagana zaliczka nie moze byc wieksza od
ceny', 1
        END
        DECLARE @id_jezyka INT
```

```
SELECT TOP 1 @id_jezyka = id_jezyka
FROM Jezyki
WHERE nazwa = @nazwa_jezyka
DECLARE @id_produktu INT
SELECT @id_produktu = ISNULL(MAX(id_produktu), 0) + 1
FROM Produkty
INSERT INTO Kursy(id_produktu, typ_modulu, id_prowadzacego,
wymagana_zaliczka)
VALUES (@id_produktu, @typ_modulu, @id_prowadzacego,
@wymagana_zaliczka);
INSERT INTO Produkty(id_produktu, typ, nazwa, data_start,
data_koniec, cena, id_jezyka )
VALUES (@id_produktu, 'Kurs', @nazwa, @data_start,
@data_koniec, @cena, @id_jezyka);
END TRY
BEGIN CATCH
DECLARE @msg varchar(255)
      = 'Bład dodadnia kursu: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1;
END CATCH
END
GO
```



## AddStudies

Dodaje studia do bazy, przyjmując jako argumenty id prowadzącego, nazwę, datę rozpoczęcia, datę zakończenia, cenę, nazwę języka, sylabus i wymaganą zaliczkę.

```
CREATE PROCEDURE AddStudies @id_prowadzacego int,
                           @nazwa varchar(50),
                           @data_start datetime,
                           @data_koniec datetime,
                           @cena money,
                           @nazwa_jezyka varchar(50),
                           @sylabus varchar(50),
                           @wymagana_zaliczka int

AS
BEGIN
    -- SET NOCOUNT ON added to prevent extra result sets from
    -- interfering with SELECT statements.
    SET NOCOUNT ON;
    BEGIN TRY
        IF @wymagana_zaliczka > @cena
        BEGIN
            ;
            THROW 52000, 'Wymagana zaliczka jest większa od ceny', 1
        END
        IF EXISTS(
            SELECT *
            FROM Produkty
            WHERE nazwa = @nazwa
        )
        BEGIN
            ;
            THROW 52000, 'Takie Studia już istnieją', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM Języki
            WHERE nazwa = @nazwa_jezyka
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego języka', 1
        END
        DECLARE @id_jezyka INT
        SELECT TOP 1 @id_jezyka = id_jezyka
```

```
FROM Jezyki
WHERE nazwa = @nazwa_jezyka
DECLARE @id_produktu INT
SELECT @id_produktu = ISNULL(MAX(id_produktu), 0) + 1
FROM Produkty
INSERT INTO Studia(id_produktu, wymagana_zaliczka, sylabus)
VALUES (@id_produktu, @wymagana_zaliczka, @sylabus);
INSERT INTO Produkty(id_produktu, typ, nazwa, data_start,
data_koniec, cena, id_jezyka )
VALUES (@id_produktu, 'Studia', @nazwa, @data_start,
@data_koniec, @cena, @id_jezyka);
END TRY
BEGIN CATCH
DECLARE @msg varchar(255)
      ='Bład dodadnia Studiów: ' + ERROR_MESSAGE();
      THROW 52000, @msg, 1;
END CATCH
END
GO
```

## AddParticipant

Dodaje Uczestnika do bazy przyjmując jako argumenty: imię i nazwisko, adres pocztowy, adres email oraz datę urodzenia

```
CREATE PROCEDURE AddParticipant @imie varchar(50),
                                @nazwisko varchar(50),
                                @adres_pocztowy
                                varchar(50),
                                @adres_email varchar(50),
                                @data_ur datetime

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Uczestnicy
            WHERE adres_email = @adres_email
        )
        BEGIN
            ;
            THROW 52000, 'Taki adres email jest juz zajety',
1
        END
        DECLARE @id_uczestnika INT
        SELECT @id_uczestnika = ISNULL(MAX(id_uczestnika), 0) + 1
        FROM Uczestnicy
        INSERT INTO Uczestnicy(id_uczestnika, imie, nazwisko,
adres_pocztowy, adres_email, data_ur, czy_aktywny)
        VALUES (@id_uczestnika, @imie, @nazwisko, @adres_pocztowy,
@adres_email, @data_ur, 1);
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
        = 'Bład dodadnia Uczestnika: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## AddInstructor

```
CREATE PROCEDURE AddInstructor @imie varchar(50),
                                @nazwisko varchar(50),
                                @id_jezyka int,
                                @adres_email varchar(50)

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF EXISTS(
            SELECT *
            FROM Prowadzacy
            WHERE adres_email = @adres_email
        )
            BEGIN
                ;
                THROW 52000, 'Taki adres email jest juz zajety', 1
            END
        DECLARE @id_prowadzacego INT
        SELECT @id_prowadzacego = ISNULL(MAX(id_prowadzacego), 0) + 1
        FROM Prowadzacy
        INSERT INTO Prowadzacy(id_prowadzacego, imie, nazwisko,
id_jezyka, adres_email)
        VALUES (@id_prowadzacego, @imie, @nazwisko, @id_jezyka,
@adres_email);
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
        ='Bład dodadnia Prowadzacego: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## RemoveWebinar

Usuwa webinar o podanym ID produktu.

```
CREATE PROCEDURE RemoveWebinar @id_produkту int

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Webinar
            WHERE id_produkту = @id_produkту
        )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego webinaru      ', 1
            END
        DELETE FROM Webinar
            WHERE id_produkту = @id_produkту
        DELETE FROM Produkty
            WHERE id_produkту = @id_produkту
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
            = 'Bład usuwania webinaru: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## RemoveCourse

Usuwa kurs o podanym ID produktu.

```
CREATE PROCEDURE RemoveCourse @id_produkту int
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Kursy
            WHERE id_produkту = @id_produkту
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego kursu ', 1
        END
        DELETE FROM Kursy
        WHERE id_produkту = @id_produkту
        DELETE FROM Produkty
        WHERE id_produkту = @id_produkту
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
            = 'Bład usuwania kursu: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## RemoveStudies

Usuwa studia o podanym ID produktu.

```
CREATE PROCEDURE RemoveStudies @id_produktu int

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Studia
            WHERE id_produktu = @id_produktu
        )
            BEGIN
                ;
                THROW 52000, 'Nie ma takich studiow ', 1
            END
        DELETE FROM Studia
            WHERE id_produktu = @id_produktu
        DELETE FROM Produkty
            WHERE id_produktu = @id_produktu
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
            = 'Bład usuwania studiow: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## RemoveParticipant

Usuwa uczestnika o podanym ID.

```
CREATE PROCEDURE RemoveParticipant @id_uczestnika int

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Uczestnicy
            WHERE id_uczestnika = @id_uczestnika
        )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego Uczestnika ', 1
            END
        DELETE FROM Uczestnicy
        WHERE id_uczestnika = @id_uczestnika
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
            = 'Bład usuwania Uczestnika: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```



## RemoveInstructor

Usuwa prowadzacego o podanym ID prowadzacego.

```
CREATE PROCEDURE RemoveInstructor @id_prowadzacego int
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Prowadzacy
            WHERE id_prowadzacego = @id_prowadzacego
        )
            BEGIN
                ;
                THROW 52000, 'Nie ma takiego Uczestnika ', 1
            END
        DELETE FROM Prowadzacy
        WHERE id_prowadzacego = @id_prowadzacego
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
        = 'Bład usuwania Prowadzacego: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## AddPayment

Dodaje do bazy opłaconą płatność przyjmując jako argumenty id uczestnika, wartość opłaty, id produktu oraz datę dokonania opłaty

```
CREATE PROCEDURE AddPayment @id_uczestnika int,
                           @wartosc money,
                           @id_produkту int,
                           @data_dokonania_oplaty money

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Uczestnicy
            WHERE id_uczestnika = @id_uczestnika
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego Uczestnika ', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM Produkty
            WHERE id_produkту = @id_produkту
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego Produkту ', 1
        END
        DECLARE @id_platnosci INT
        SELECT @id_platnosci = ISNULL(MAX(id_platnosci), 0) + 1
        FROM Platnosci
        INSERT INTO Platnosci(id_platnosci, id_uczestnika, wartosc,
id_produkту, data_dokonania_oplaty)
        VALUES (@id_platnosci, @id_uczestnika, @wartosc, @id_produkту,
@data_dokonania_oplaty);
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
        = 'Bład dodawania opłaty: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## ModifyParticipant

Modyfikuje dane danego Uczestnika i zapisuje zmiany do tabeli UczestnicyZmiany. Przyjmuje jako zmienne: id uczestnika, imię i nazwisko, adres pocztowy, adres email oraz jego status.

```
CREATE PROCEDURE ModifyParticipant @id_uczestnika int,
                                   @imie varchar(50),
                                   @nazwisko varchar(50),
                                   @adres_pocztowy
                                   varchar(50),
                                   @adres_email
                                   varchar(50),
                                   @czy_aktywny bit
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Uczestnicy
            WHERE id_uczestnika = @id_uczestnika
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego Uczestnika ', 1
        END
        DECLARE @stare_imie varchar(50)
        SELECT @stare_imie = imie
        FROM Uczestnicy
        WHERE id_uczestnika = @id_uczestnika

        DECLARE @stare_nazwisko varchar(50)
        SELECT @stare_nazwisko = nazwisko
        FROM Uczestnicy
        WHERE id_uczestnika = @id_uczestnika

        DECLARE @stary_adres_pocztowy varchar(50)
        SELECT @stary_adres_pocztowy = adres_pocztowy
        FROM Uczestnicy
        WHERE id_uczestnika = @id_uczestnika

        DECLARE @stary_adres_email varchar(50)
        SELECT @stary_adres_email = adres_email
```

```
FROM Uczestnicy
WHERE id_uczestnika = @id_uczestnika

    DECLARE @stary_status_czy_aktywny varchar(50)
    SELECT @stary_status_czy_aktywny = czy_aktywny
    FROM Uczestnicy
    WHERE id_uczestnika = @id_uczestnika

    DECLARE @id_zmiany INT
    SELECT @id_zmiany = ISNULL(MAX(id_zmiany), 0) + 1
    FROM UczestnicyZmiany

    UPDATE Uczestnicy
        SET imie = @imie, nazwisko = @nazwisko, adres_pocztowy
        = @adres_pocztowy, adres_email = @adres_email, czy_aktywny =
        @czy_aktywny
        WHERE id_uczestnika = @id_uczestnika

    INSERT INTO UczestnicyZmiany(id_zmiany, id_uczestnika,
data_zmiany, stare_imie, stare_nazwisko, stary_adres_pocztowy,
stary_adres_email, stary_status_czy_aktywny, nowe_imie, nowe_nazwisko,
nowy_adres_pocztowy, nowy_adres_email, nowy_status_czy_aktywny)
        VALUES (@id_zmiany, @id_uczestnika, GETDATE(), @stare_imie,
@stare_imie, @stary_adres_pocztowy, @stary_adres_email,
@stary_status_czy_aktywny, @imie, @nazwisko, @adres_pocztowy,
@adres_email, @czy_aktywny);

END TRY
BEGIN CATCH
    DECLARE @msg varchar(255)
        = 'Bład modyfikacji danych Uczestnika: ' +
ERROR_MESSAGE();
    THROW 52000, @msg, 1;
END CATCH
END
GO
```

## ModifyProductPrice

Ustawia cenę produktu o podanym ID na podaną wartość.

```
CREATE PROCEDURE ModifyProductPrice @id_produktu int,
                                     @cena money
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Produkty
            WHERE id_produktu = @id_produktu
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego Produktu', 1
        END

        UPDATE Produkty
        SET cena = @cena
        WHERE id_produktu = @id_produktu
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
            = 'Bład modyfikacji ceny Produktu: ' +
ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## SModifyOutsiderPrice

Ustawia cenę za uczestnictwo w zajęciach o podanym ID dla uczestników z zewnątrz. Nowa cena jest równa wartości podanej w parametrze.

```
CREATE PROCEDURE ModifyOutsiderPrice @id_zajec int,
                                     @cena money

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM CenaDlaOsobyZewn
            WHERE id_zajec = @id_zajec
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego Zajecia', 1
        END

        UPDATE CenaDlaOsobyZewn
        SET cena = @cena
        WHERE id_zajec = @id_zajec
    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
            = 'Bład modyfikacji ceny Zajec dla osob z zewnatrz: '
        + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END
GO
```

## TakeAttendanceOfAStudent

Wpisuje obecność/nieobecność na zajęciach danego studenta. Jako zmienne przyjmuje id uczestnika, id zajęć i informacje czy był obecny lub też nie

```
CREATE PROCEDURE TakeAttendanceOfAStudent @id_uczestnika int,
                                           @id_zajec money,
                                           @czy_obecny bit

AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        IF NOT EXISTS(
            SELECT *
            FROM Uczestnicy
            WHERE id_uczestnika = @id_uczestnika
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takiego Uczestnika', 1
        END
        IF NOT EXISTS(
            SELECT *
            FROM Zajecia
            WHERE id_zajec = @id_zajec
        )
        BEGIN
            ;
            THROW 52000, 'Nie ma takich Zajec', 1
        END

        DECLARE @id_wpisu INT
        SELECT @id_wpisu = ISNULL(MAX(id_wpisu), 0) + 1
        FROM Frekwencja

        INSERT INTO Frekwencja(id_wpisu, id_uczestnika, id_zajec,
czy_obecny)
VALUES (@id_wpisu, @id_uczestnika, @id_zajec, @czy_obecny);

    END TRY
    BEGIN CATCH
        DECLARE @msg varchar(255)
        = 'Bład sprawdzania obecności: ' + ERROR_MESSAGE();
```

```
        THROW 52000, @msg, 1;  
    END CATCH  
END  
GO
```



# Funkcje

## GetClassByID

Wyświetlanie zajęć po ID

```
CREATE FUNCTION [dbo].[GetClassByID] (@id_zajec INT)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM Zajecia
    WHERE id_zajec = @id_zajec
);
GO
```

## GetInstructorByID

Wyświetlanie prowadzącego po ID.

```
CREATE FUNCTION [dbo].[GetInstructorByID] (@id_prowadzacego INT)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM Prowadzacy
    WHERE id_prowadzacego = @id_prowadzacego
);
GO
```

## GetParticipantByID

Wyświetlanie uczestnika po ID.

```
CREATE FUNCTION [dbo].[GetParticipantByID] (@id_uczestnika INT)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM Uczestnicy
    WHERE id_uczestnika = @id_uczestnika
);
GO
```

## GetPaymentByID

Wyświetlanie płatności po ID płatności.

```
CREATE FUNCTION [dbo].[GetPaymentByID] (@id_platnosci INT)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM Platnosci
    WHERE id_platnosci = @id_platnosci
);
GO
```

## GetPaymentsBetweenDates

Wyświetlanie płatności dokonanych w podanym przedziale czasowym.

```
CREATE FUNCTION [dbo].[GetPaymentsBetweenDates] (@data_start DATETIME,  
@data_koniec DATETIME)  
RETURNS TABLE  
AS  
RETURN  
(  
    SELECT *  
    FROM Platnosci  
    WHERE Platnosci.data_dokonania_oplaty BETWEEN @data_start AND  
@data_koniec  
)  
;  
GO
```

## GetProductByID

Wyświetlanie produktu po ID.

```
CREATE FUNCTION [dbo].[GetProductByID] (@id_produktu INT)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM Produkty
    WHERE id_produktu = @id_produktu
);
GO
```

## GetProductsAboveXPrice

Wyświetlanie produktów o cenie wyższej lub równej niż podana.

```
CREATE FUNCTION [dbo].[GetProductsAboveXPrice] (@cena money)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM Produkty
    WHERE cena >= @cena
);
GO
```

## GetProductsBetweenDates

Wyświetlanie produktów, które odbywają się w danym przedziale czasowym.

```
CREATE FUNCTION [dbo].[GetProductsBetweenDates] (@data_start DATETIME,  
@data_koniec DATETIME)  
RETURNS TABLE  
AS  
RETURN  
(  
    SELECT *  
    FROM Produkty  
    WHERE data_start BETWEEN @data_start AND @data_koniec  
    AND data_koniec BETWEEN @data_start AND @data_koniec  
);  
GO
```



## GetRegistrationsByDate

Wyświetlanie zapisów, które miały miejsce w podanym dniu.

```
CREATE FUNCTION [dbo].[GetRegistrationsByDate] (@data_zapisu DATETIME)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM Zapisy
    WHERE data_zapisu = @data_zapisu
);
GO
```

## GetLanguageForClass

Wyświetlanie języka, w którym odbywają się dane zajęcia.

```
CREATE FUNCTION [dbo].[GetLanguageForClass](@id_zajec INT)
RETURNS VARCHAR(50)
AS
BEGIN
    DECLARE @nazwa_jezyka VARCHAR(50);

    SELECT @nazwa_jezyka = Jezyki.nazwa
    FROM Zajecia
    INNER JOIN Produkty
    ON Zajecia.id_produktu = Produkty.id_produktu
    INNER JOIN Jezyki
    ON Produkty.id_jezyka = Jezyki.id_jezyka
    WHERE Zajecia.id_zajec = @id_zajec;

    RETURN @nazwa_jezyka;
END;
GO
```

## GetLocationForClass

Wyświetlanie miejsca, w którym odbywają się dane zajęcia.

```
CREATE FUNCTION [dbo].[GetLocationForClass](@id_zajec INT)
RETURNS VARCHAR(50)
AS
BEGIN
    DECLARE @nazwa_miejsca VARCHAR(50);

    SELECT @nazwa_miejsca = Miejsca.nazwa_miejsca
    FROM Zajecia
    INNER JOIN Miejsca
    ON Zajecia.id_miejsca = Miejsca.id_miejsca
    WHERE Zajecia.id_zajec = @id_zajec;

    RETURN @nazwa_miejsca;
END;
GO
```

## GetProductPriceByID

Wyświetlanie ceny produktu o podanym ID.

```
CREATE FUNCTION [dbo].[GetProductPriceByID] (@id_produktu INT)
RETURNS money
AS
BEGIN
    DECLARE @cena money;
    SELECT @cena = cena
    FROM Produkty
    WHERE id_produktu = @id_produktu;
    RETURN @cena;
END;
GO
```

## GetSlotsLimit

Wyświetlanie limitu miejsc na dane kursy lub studia.

```
CREATE FUNCTION [dbo].[GetSlotsLimit] (@id_produktu INT)
RETURNS INT
AS
BEGIN
    DECLARE @limit_miejsc INT;
    SELECT @limit_miejsc = limit_miejsc
    FROM Produkty
    INNER JOIN LimityMiejsc ON LimityMiejsc.id_produktu =
Produkty.id_produktu
    WHERE Produkty.id_produktu = @id_produktu;
    RETURN @limit_miejsc;
END;
GO
```

# Triggery

## RemoveWebinarLinkAfter30Days

Trigger odpowiedzialny za usunięcie linków do webinarów, które odbyły się wcześniej niż 30 dni przed bieżącą datą.

```
CREATE TRIGGER [dbo].[RemoveWebinarLinkAfter30Days]
ON [dbo].[Linki]
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;
    DELETE FROM Linki
    WHERE id_zajec in (
        SELECT Linki.id_zajec
        FROM Linki
        INNER JOIN Zajecia ON Zajecia.id_zajec = Linki.id_zajec
        INNER JOIN Produkty ON Produkty.id_produktu =
Zajecia.id_produktu
        WHERE DATEADD(day, -30, Produkty.data_koniec) < GETDATE()
    )

END
GO
ALTER TABLE [dbo].[Linki] ENABLE TRIGGER [RemoveWebinarLinkAfter30Days]
GO
```

# Indeksy

Lista pól, na które ustawione są indeksy

- Produkty.id\_produktu
- Webinary.id\_produktu
- Kursy.id\_produktu
- Studia.id\_produktu
- Prowadzacy.id\_prowadzacego
- LimityMiejsc.id\_produktu
- Jezyki.id\_jezyka
- Tlumacze.id\_tlumacza
- Tlumaczenia.id\_zajec
- Zajecia.id\_zajec
- Miejsca.id\_miejsca
- CenaDlaOsobyZewn.id\_zajec
- LimityMiejscNaZajecia.id\_zajec
- Frekwencja.id\_wpisu
- Odrobienia.id\_wpisu
- Linki.id\_zajec
- ZapisyPojedynczeZajecia.id\_uczestnika, ZapisyPojedynczeZajecia.id\_zajec
- Uczestnicy.id\_uczestnika
- Zapisy.id\_uczestnika, Zapisy.id\_zajec
- Platnosci.id\_platnosci
- Praktyki.id\_praktyki
- PraktykiFrekwencja.id\_praktyki, PraktykiFrekwencja.nr\_dnia

# Uprawnienia

## Dyrektor Szkoły

Ma dostęp do wszystkich funkcji systemu bazodanowego.

Przykładowe czynności:

- Zarządzanie wszystkimi danymi w systemie.
- Tworzenie, edycja i usuwanie kursów, webinarów, studiów.
- Generowanie raportów finansowych.
- Zarządzanie danymi użytkowników
- Odroczenie opłat.

## Wykładowca

- Przypisanie do kursów, webinarów, studiów.
- Wprowadzanie ocen, obecności.
- Generowanie list obecności.

## Uczestnik

- Zakup dostępu do webinarów.
- Zapisy na kursy i studia.
- Przegląd własnych danych.