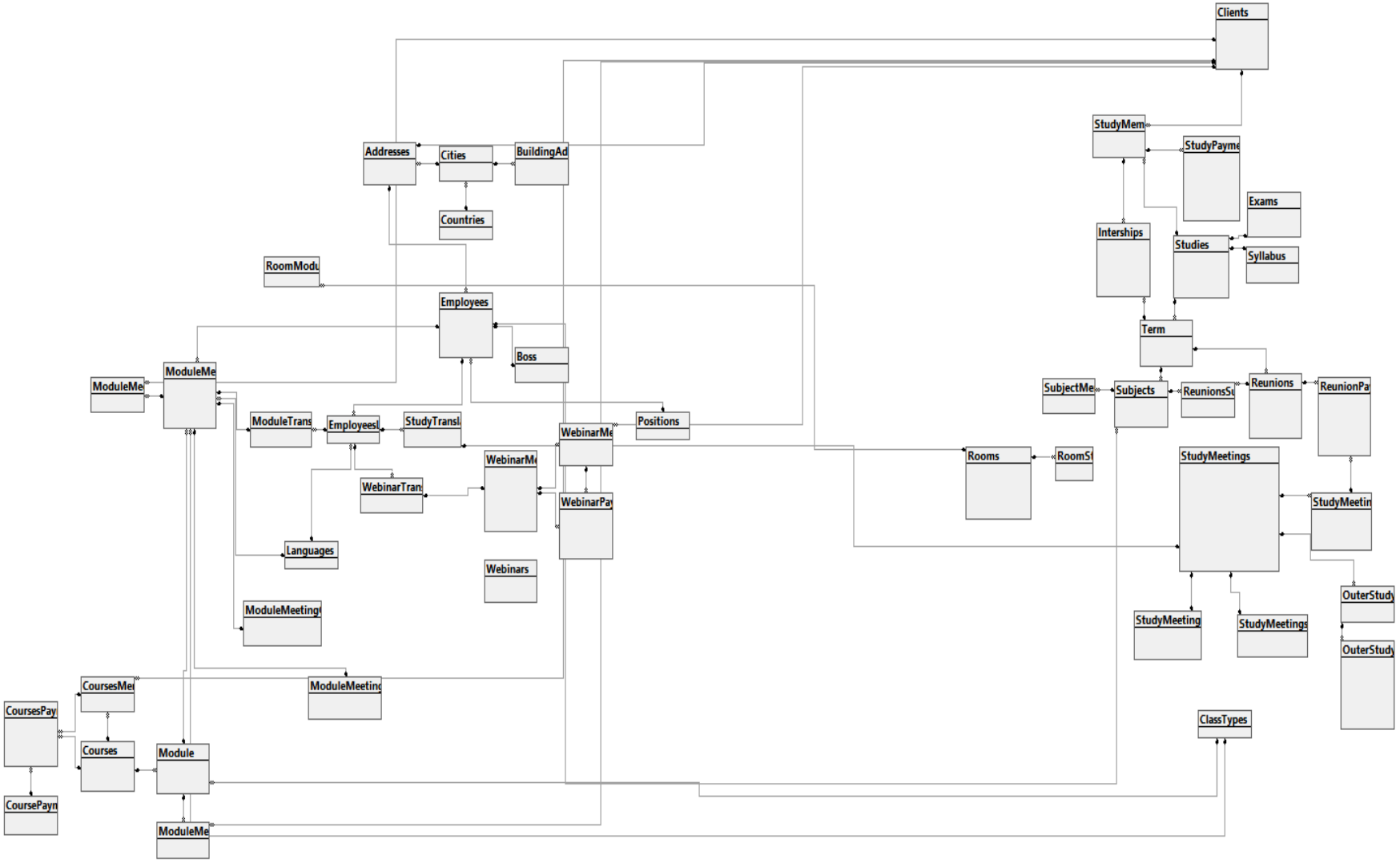
Projekt i implementacja systemu bazodanowego

Monika Etrych, Judyta Bąkowska, Anastasiya Yahorava

Spis treści

Aktorzy **3**
Funkcje systemu **3**
Tabele: **5**
WIDOKI: **36**
PROCEDURY: **61**
TRIGGERY: **75**
UPRAWNIENIA: **80**
INDEKSY: **86**


Clients
StudyMem
StudyPayme
Addresses
Cities
BuildingAd
Exams
Countries
Interships
Studies
Syllabus
RoomModu
Employees
Term
Boss
ModuleMe
ModuleMe
SubjectMe
Subjects
ReunionsSu
Reunions
ReunionPa
ModuleTrans
EmployeesL
StudyTransl
WebinarMe
Positions
Rooms
RoomSt
StudyMeetings
WebinarM
WebinarTrans
WebinarPa
StudyMeetin
Languages
Webinars
OuterStudy
ModuleMeeting
StudyMeeting
StudyMeetings
OuterStudy
CoursesMe
ModuleMeetinc
CoursesPay
ClassTypes
Courses
Module
CoursePaym
ModuleMe

Aktorzy

1. Administrator
2. Klient
3. Dyrektor
4. Wykładowca
5. Tłumacz
6. Sekretarka
7. Księgowa

Funkcje systemu

Administrator
Ma uprawnienia do:
- zarządzania systemem
- wszystkich funkcji systemu
- zarządzania kontami użytkownika

Klient bez założonego konta
Ma uprawnienia do:
- wyświetlania linków do syllabusa

Klient
Ma uprawnienia do:
- informacji o kolidujących ze sobą zajęciach na które jest zapisany
- informacji o zajęciach na które jest zapisany
- informacji o swoich płatnościach za zajęcia na które jest zapisany
- informacji o zaliczeniu zajęć
- wyświetlania linków do syllabusa
- zapisania się na wybrane zajęcia

Dyrektor:
Ma uprawnienia do:
- dodania uczestnika do kursu, webinaru i studiów
- dodanie nowego webinaru, studiów, kursu
- zmiany statusu zdania studiów/kursu ? X
- wysłania certyfikatu
- przedłużenia terminu zapłaty
- nadania dostępu do kursu/ webinaru/studiów ? X
- informacji o obecności na kursach/studiach / webinarach
- informacji o studentach i zajęciach na które są zapisani i ich zaliczeniach
- informacji o pracownikach

- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami
- informacji o zaliczeniu praktyk i zdania egzaminu przez studentów

Wykładowca:
Ma uprawnienia do:
- zmiany statusu zdania studiów/kursu? X
- informacji o obecności na kursach/studiach/webinarach
- informacji o zajęciach które prowadzi
- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami

Tłumacz:
Ma uprawnienia do:
- informacji o zajęciach które tłumaczy

Sekretarka:
Ma uprawnienia do:
- wysłania certyfikatu
- nadania dostępu do kursu/ webinaru/studiów ? X
- rezerwacji sali
- informacji o obecności na kursach/studiach
- informacji o studentach i zajęciach na które są zapisani
- informacji o kursach, modułach, webinarach
- informacji o pracownikach
- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami
- informacji o zaliczeniu praktyk i zdania egzaminu przez studentów
- zarezerwowania sali ?

Księgowa
Ma uprawnienia do:
- podsumowania zarobków za kursy/webinary/kursy
- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami

Tabele:

Tabela “Address” (Reprezentuje adresy w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **AddressID** - unikalny identyfikator adresu
  - KLUCZ OBCY: **CountryID** - identyfikator państwa
  - KLUCZ OBCY: **CityID** - identyfikator miasta
  - **Address** - adres (ulica + numer domu)
  - **PostCode** - kod pocztowy
- Warunki integralności:
  - Kod pocztowy (**PostCode**) jest postaci ‘XX-XXX’, gdzie X jest liczbą [0-9]

```
CREATE TABLE [dbo].[Addresses] (
[AddressID] INT IDENTITY (1, 1) NOT NULL,
[CityID] INT NOT NULL,
[Address] VARCHAR (50) NOT NULL,
[PostCode] NCHAR (6) NOT NULL,
CONSTRAINT [PK_Address] PRIMARY KEY CLUSTERED ([AddressID] ASC),
CONSTRAINT [Addresses_PostCodeValid] CHECK ([PostCode] like
'[0-9][0-9]-[0-9][0-9][0-9]'),
CONSTRAINT [FK_Addresses_Cities] FOREIGN KEY ([CityID]) REFERENCES [dbo].[Cities]
([CityID])
);
```

Tabela “Boss” (Reprezentuje dyrektorów w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **BossID** - unikalny identyfikator dyrektor
  - **StartDate** - data objęcia stanowiska dyrektora
  - **EndDate** - data utraty stanowiska dyrektora
- Warunki integralności:
  - **‘StartDate’** musi być wcześniejsze od **‘EndDate’** lub **‘EndDate’** jest Nullem

```
CREATE TABLE [dbo].[Boss] (
[EmployeeID] INT NOT NULL,
[StartDate] DATE NOT NULL,
[EndDate] DATE NULL,
CONSTRAINT [PK_Boss] PRIMARY KEY CLUSTERED ([EmployeeID] ASC),
CONSTRAINT [Boss_DatesValid] CHECK ([EndDate]>[StartDate] OR [EndDate] IS NULL),
CONSTRAINT [FK_Boss_Employees] FOREIGN KEY ([EmployeeID]) REFERENCES
[dbo].[Employees] ([EmployeeID])
);
```

Tabela “BuildingAddresses” (Reprezentuje adresy budynków zajęć w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **BuildingAddressID** - unikalny identyfikator budynku
  - KLUCZ OBCY: **CountryID** - identyfikator państwa
  - KLUCZ OBCY: **CityID** - identyfikator miasta
  - **Address** - adres
  - **PostCode** - kod pocztowy
- Warunki integralności:
  - Kod pocztowy (**PostCode**) jest postaci ‘XX-XXX’, gdzie X jest liczbą [0-9]

```
CREATE TABLE [dbo].[BuildingAddresses] (
[BuildingAddressID] INT IDENTITY (1, 1) NOT NULL,
[CityID] INT NOT NULL,
[Address] VARCHAR (50) NOT NULL,
[PostCode] NCHAR (6) NOT NULL,
CONSTRAINT [PK_BuildingAddresses] PRIMARY KEY CLUSTERED ([BuildingAddressID] ASC),
CONSTRAINT [BuildingAddresses_PostCodeValid] CHECK ([PostCode] like
'[0-9][0-9]-[0-9][0-9][0-9]'),
CONSTRAINT [FK_BuildingAddresses_Cities] FOREIGN KEY ([CityID]) REFERENCES
[dbo].[Cities] ([CityID])
);
```

Tabela “Cities” (Reprezentuje miasta w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **CityID** - unikalny identyfikator miasta
  - **Name** - nazwa miasta
  - KLUCZ OBCY: **CountryID** - identyfikator kraju

```
CREATE TABLE [dbo].[Cities] (
[CityID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
[CountryID] INT NOT NULL,
CONSTRAINT [PK_Cities] PRIMARY KEY CLUSTERED ([CityID] ASC),
CONSTRAINT [Cities_NameValid] CHECK (NOT [Name] like '%[^0-9]%'),
CONSTRAINT [FK_Cities_Countries] FOREIGN KEY ([CountryID]) REFERENCES
[dbo].[Countries] ([CountryID])
);
```

Tabela “ClassTypes” (Reprezentuje typ zajęć w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **ClassTypeID** - unikalny identyfikator typu zajęć
  - **Name** - typ zajęć
- Warunki integralności:
  - **‘Name’** może mieć wartość: ‘Hybrid’, ‘Online Asynchronic’, ‘Online Synchronic’, ‘Offline’

```
CREATE TABLE [dbo].[ClassTypes] (
[ClassTypeID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
CONSTRAINT [PK_ClassTypes] PRIMARY KEY CLUSTERED ([ClassTypeID] ASC),
CONSTRAINT [CK_ClassTypes_1] CHECK ([Name]='Hybrid' OR [Name]='Online Asynchronic'
OR [Name]='Online Synchronic' OR [Name]='Offline')
);
```

Tabela “Clients” (Reprezentuje klientów, uczniów w bazie danych) :

- Pola:
  - KLUCZ GŁÓWNY: **ClientID** - unikalny identyfikator klienta
  - **FirstName** - imię
  - **LastName** - nazwisko
  - KLUCZ OBCY: **AddressID** - identyfikator adresu powiązany z tabelą “Address”
  - **Phone** - numer telefonu
  - **Email** - adres email
  - **RegularCustomer** - informacja czy jest stałym klientem
- Warunki integralności:
  - **‘Email’** musi posiadać znak ‘@’ i ‘.’
  - **‘Email’** musi być unikalny
  - **‘Phone’** musi się składać z samych cyfr i mieć długość równą 9
  - **‘Phone’** musi być unikalny

```
CREATE TABLE [dbo].[Clients] (
[ClientID] INT IDENTITY (1, 1) NOT NULL,
[FirstName] VARCHAR (50) NOT NULL,
[LastName] VARCHAR (50) NOT NULL,
[Phone] NCHAR (9) NOT NULL,
[Email] VARCHAR (100) NOT NULL,
[AddressID] INT NOT NULL,
[RegularCustomer] BIT NOT NULL,
CONSTRAINT [PK_Clients_1] PRIMARY KEY CLUSTERED ([ClientID] ASC),
CONSTRAINT [Clients_EmailValid] CHECK ([Email] like ('%@%'+'.')+'%'),
```

```
CONSTRAINT [Clients_PhoneValid] CHECK (isnumeric([Phone])=(1) AND len([Phone])=(9)),
CONSTRAINT [FK_Clients_Addresses] FOREIGN KEY ([AddressID]) REFERENCES
[dbo].[Addresses] ([AddressID])
);


GO
CREATE UNIQUE NONCLUSTERED INDEX [Clients_PhoneUnique]
ON [dbo].[Clients]([Phone] ASC);

GO
CREATE UNIQUE NONCLUSTERED INDEX [Clients_EmailUnique]
ON [dbo].[Clients]([Email] ASC);
```

Tabela “Countries” (Reprezentuje państwa w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **CountryID** - unikalny identyfikator państwa
  - **Name** - nazwa państwa

```
CREATE TABLE [dbo].[Countries] (
[CountryID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
CONSTRAINT [PK_Countries] PRIMARY KEY CLUSTERED ([CountryID] ASC)
);
```

Tabela “CoursePaymentTypes” (Reprezentuje kwotę do zapłaty za kurs w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **CoursePaymentTypeID** - unikalny identyfikator typu płatności za kurs
  - **Name** - typ płatności
- Warunki integralności:
  - **‘Name’** może mieć wartość: ‘total’, ‘advance’, ‘remaining’

```
CREATE TABLE [dbo].[CoursePaymentTypes] (
[CoursePaymentTypeID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
CONSTRAINT [PK_CoursePaymentTypes] PRIMARY KEY CLUSTERED ([CoursePaymentTypeID]
ASC),
CONSTRAINT [CoursePaymentTypes_NameValid] CHECK ([Name]='total' OR [Name]='advance'
OR [Name]='remaining')
);
```

Tabela “Courses” (Reprezentuje informacje o kursach w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **CourseID** - unikalny identyfikator kursu
  - **Name** - nazwa kursu
  - **Price** - cena za kurs
  - **AdvancePaymentPrice** - procent zaliczki
- Warunki integralności:
  - **‘Price’** musi być większa od zera
  - **‘AdvancePaymentPrice’** musi być większe bądź równe zero i musi być mniejsze od ‘Price’

```
CREATE TABLE [dbo].[Courses] (
[CourseID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
[Description] VARCHAR (100) NOT NULL,
[Price] DECIMAL (16, 2) NOT NULL,
[AdvancePaymentPrice] DECIMAL (16, 2) NOT NULL,
CONSTRAINT [PK_Course] PRIMARY KEY CLUSTERED ([CourseID] ASC),
CONSTRAINT [Courses_AdvancePaymentPriceValid] CHECK ([AdvancePaymentPrice]>=(0) AND
[AdvancePaymentPrice]<[Price]),
CONSTRAINT [Courses_PriceValid] CHECK ([Price]>(0))
);
```

Tabela “CoursesMembers” (Reprezentuje członków kursu w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **ClientID** - unikalny identyfikator klienta
  - KLUCZ OBCY: **CourseID** - identyfikator kursu
  - **Access** - informacja czy klient ma dostęp do kursu (czy zapłacił)

```
CREATE TABLE [dbo].[CoursesMembers] (
[ClientID] INT NOT NULL,
[CourseID] INT NOT NULL,
[Access] BIT NOT NULL,
CONSTRAINT [PK_CompositePrimaryKey_CoursesMembers] PRIMARY KEY CLUSTERED ([ClientID]
ASC, [CourseID] ASC),
CONSTRAINT [FK_CoursesMembers_Clients] FOREIGN KEY ([ClientID]) REFERENCES
[dbo].[Clients] ([ClientID]),
CONSTRAINT [FK_CoursesMembers_Courses] FOREIGN KEY ([CourseID]) REFERENCES
[dbo].[Courses] ([CourseID])
);
```

Tabela “CoursesPayments” (Reprezentuje płatności za kursy w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **CoursesPaymentsID** - unikalny identyfikator płatności za kurs
  - KLUCZ OBCY: **CourseID** - identyfikator kursu
  - KLUCZ OBCY: **ClientID** - identyfikator klienta
  - **PayDate** - data płatności za kurs
  - KLUCZ OBCY: **PaymentsID** - identyfikator płatności za wszystkie produkty
  - **Link** - link do płatności
  - KLUCZ OBCY: **CoursePaymentTypeID** - identyfikator typu płatności
- Warunki integralności:
  - **‘PayDate’** nie może być wcześniejsze od dzisiejszej daty

```
CREATE TABLE [dbo].[CoursesPayments] (
[CoursesPaymentsID] INT IDENTITY (1, 1) NOT NULL,
[CourseID] INT NOT NULL,
[ClientID] INT NOT NULL,
[PayDate] DATETIME NOT NULL,
[Link] VARCHAR (50) NOT NULL,
[CoursePaymentTypeID] INT NOT NULL,
```

```
[Success] BIT CONSTRAINT [DF_CoursesPayments_Success] DEFAULT ((0)) NOT NULL,
CONSTRAINT [PK_CoursesPayments_1] PRIMARY KEY CLUSTERED ([CoursesPaymentsID] ASC),
CONSTRAINT [FK_CoursesPayments_CoursePaymentTypes] FOREIGN KEY
([CoursePaymentTypeID]) REFERENCES [dbo].[CoursePaymentTypes]
([CoursePaymentTypeID]),
CONSTRAINT [FK_CoursesPayments_Courses] FOREIGN KEY ([CourseID]) REFERENCES
[dbo].[Courses] ([CourseID]),
CONSTRAINT [FK_CoursesPayments_CoursesMembers] FOREIGN KEY ([ClientID], [CourseID])
REFERENCES [dbo].[CoursesMembers] ([ClientID], [CourseID])
);
```

Tabela "Employees" (Reprezentuje pracowników szkoły w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **EmployeeID** - unikalny identyfikator pracownika
  - **FirstName** - imię
  - **LastName** - nazwisko
  - KLUCZ OBCY: **AddressID** - identyfikator adresu
  - **Phone** - numer telefonu
  - **Email** - adres email
  - KLUCZ OBCY: **PositionID** - identyfikator stanowiska pracownika
- Warunki integralności:
  - **'Email'** musi być unikalny
  - **'Email'** musi posiadać znak '@' i '.'
  - **'Phone'** musi się składać z samych cyfr i mieć długość równą 9
  - **'Phone'** musi być unikalny

```
CREATE TABLE [dbo].[Employees] (
[EmployeeID] INT IDENTITY (1, 1) NOT NULL,
[FirstName] VARCHAR (50) NOT NULL,
[LastName] VARCHAR (50) NOT NULL,
[Phone] NCHAR (9) NOT NULL,
[Email] VARCHAR (100) NOT NULL,
[AddressID] INT NOT NULL,
[PositionID] INT NOT NULL,
CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED ([EmployeeID] ASC),
```

```
CONSTRAINT [Employees_EmailValid] CHECK ([Email] like ('%@%'+'.')+'%'),
CONSTRAINT [Employees_PhoneValid] CHECK (isnumeric([Phone])=(1)),
CONSTRAINT [FK_Employees_Address] FOREIGN KEY ([AddressID]) REFERENCES
[dbo].[Addresses] ([AddressID]),
CONSTRAINT [FK_Employees_Positions1] FOREIGN KEY ([PositionID]) REFERENCES
[dbo].[Positions] ([PositionID]),
CONSTRAINT [IX_Employees] UNIQUE NONCLUSTERED ([Phone] ASC)
);


GO
CREATE UNIQUE NONCLUSTERED INDEX [Employees_PhoneUnique]
ON [dbo].[Employees]([Phone] ASC);


GO
CREATE UNIQUE NONCLUSTERED INDEX [Employees_EmailUnique]
ON [dbo].[Employees]([Email] ASC);
```

Tabela “EmployeesLanguages” (Tabela pośrednia między Employees i Languages)

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **EmployeeID** - identyfikator pracownika
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **LanguageID** - identyfikator języka

```
CREATE TABLE [dbo].[EmployeesLanguages] (
[LanguageID] INT NOT NULL,
[EmployeeID] INT NOT NULL,
CONSTRAINT [PK_CompositePrimaryKey1] PRIMARY KEY CLUSTERED ([LanguageID] ASC,
[EmployeeID] ASC),
CONSTRAINT [FK_EmployeesLanguages_Employees1] FOREIGN KEY ([EmployeeID]) REFERENCES
[dbo].[Employees] ([EmployeeID]),
CONSTRAINT [FK_EmployeesLanguages_Languages1] FOREIGN KEY ([LanguageID]) REFERENCES
[dbo].[Languages] ([LanguageID])
);
```

Tabela “Exams” (Reprezentuje informacje o egzaminach):

- Pola:
  - KLUCZ GŁÓWNY: **ExamID** - unikalny identyfikator egzaminu
  - KLUCZ OBCY: **StudiesID** - identyfikator studiów
  - KLUCZ OBCY: **ClientID** - identyfikator klienta
  - KLUCZ OBCY: **LecturerID** - identyfikator wykładowcy
  - **ExamDate** - data egzaminu
  - **PassExam** - informacja czy egzamin został zdany czy nie
  - **Grade** - ocena

```
CREATE TABLE [dbo].[Exams] (
[ExamID] INT IDENTITY (1, 1) NOT NULL,
[StudiesID] INT NOT NULL,
[LecturerID] INT NOT NULL,
[ExamDate] DATE NOT NULL,
[Grade] DECIMAL (1, 1) NULL,
CONSTRAINT [PK_Exams] PRIMARY KEY CLUSTERED ([ExamID] ASC),
CONSTRAINT [Exams_ExamDateValid] CHECK ([ExamDate]>=getdate()),
CONSTRAINT [FK_Exams_Studies] FOREIGN KEY ([StudiesID]) REFERENCES [dbo].[Studies]
([StudiesID])
);
```

Tabela “Interships” (Przetrzymuje informacje o zaliczeniu praktyk w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **IntershipID** - unikalny identyfikator praktyk
  - KLUCZ OBCY: **ClientID** - identyfikator klienta
  - **CompanyName** - nazwa firmy w której klient odbywał praktyki
  - **Pass** - informacja czy praktyki zostały zaliczone czy niezaliczone
  - **StartDate** - data rozpoczęcia praktyk
  - **EndDate** - data zakończenia praktyk
  - KLUCZ OBCY: **TermID** - identyfikator semestru
  - KLUCZ OBCY: **StudiesID** - identyfikator studiów
- Warunki integralności:
  - **‘StartDate’** musi być wcześniejsza od **‘EndDate’**

```
CREATE TABLE [dbo].[Interships] (
[IntershipID] INT IDENTITY (1, 1) NOT NULL,
[ClientID] INT NOT NULL,
[CompanyName] VARCHAR (50) NOT NULL,
[Pass] BIT CONSTRAINT [DEFAULT_Interships_Pass] DEFAULT ((0)) NOT NULL,
[StartDate] DATETIME NOT NULL,
[EndDate] DATETIME NOT NULL,
[TermID] INT NOT NULL,
[StudiesID] INT NOT NULL,
CONSTRAINT [PK_Interships] PRIMARY KEY CLUSTERED ([IntershipID] ASC),
CONSTRAINT [Interships_DatesValid] CHECK ([EndDate]>[StartDate]),
CONSTRAINT [FK_Interships_StudyMembers] FOREIGN KEY ([ClientID], [StudiesID])
REFERENCES [dbo].[StudyMembers] ([ClientID], [StudiesID]),
CONSTRAINT [FK_Interships_Term] FOREIGN KEY ([TermID]) REFERENCES [dbo].[Term]
([TermID])
);
```

Tabela “Languages” (Reprezentuje języki w jakich są zajęcia w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **LanguageID** - unikalny identyfikator języka
  - **Name** - nazwa języka

```
CREATE TABLE [dbo].[Languages] (
[LanguageID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
CONSTRAINT [PK_Language] PRIMARY KEY CLUSTERED ([LanguageID] ASC)
);
```

Tabela “Module” (Reprezentuje informacje o modułach kursów w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **ModuleID** - unikalny identyfikator modułu
  - KLUCZ OBCY: **CourseID** - identyfikator kursu
  - **StartDate** - data rozpoczęcia modułu
  - **EndDate** - data zakończenia modułu
  - KLUCZ OBCY: **ClassTypeID** - identyfikator typu zajęć
- Warunki integralności:
  - **‘StartDate’** musi być wcześniejsza od **‘EndDate’**

```
CREATE TABLE [dbo].[Module] (
[ModuleID] INT IDENTITY (1, 1) NOT NULL,
[CourseID] INT NOT NULL,
[StartDate] DATETIME NOT NULL,
[EndDate] DATETIME NOT NULL,
[ClassTypeID] INT NOT NULL,
CONSTRAINT [PK_Module] PRIMARY KEY CLUSTERED ([ModuleID] ASC),
CONSTRAINT [Module_DatesValid] CHECK ([StartDate]<[EndDate]),
CONSTRAINT [FK_Module_ClassTypes] FOREIGN KEY ([ClassTypeID]) REFERENCES
[dbo].[ClassTypes] ([ClassTypeID]),
CONSTRAINT [FK_Module_Courses] FOREIGN KEY ([CourseID]) REFERENCES [dbo].[Courses]
([CourseID])
);
```

Tabela “ModuleMeeting” (Reprezentuje spotkanie w module w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **ModuleMeetingID** - unikalny identyfikator spotkania w module
  - KLUCZ OBCY: **ModuleID** - identyfikator modułu
  - KLUCZ OBCY: **LecturerID** - identyfikator wykładowcy
  - **StartTime** - czas rozpoczęcia
  - **EndTime** - czas zakończenia
  - KLUCZ OBCY: **TranslatorID** - identyfikator tłumacza
  - KLUCZ OBCY: **ClassTypeID** - identyfikator typu zajęć
  - KLUCZ OBCY: **OriginalLanguageID** - identyfikator języka w którym orginalnie są zajęcia
- Warunki integralności:
  - **‘StartDate’** musi być wcześniejsza niż **‘EndDate’**
  - **ClassTypeID** musi być 1, 2 lub 3 (dane spotkanie nie może być hybrydowe)

```
CREATE TABLE [dbo].[ModuleMeeting] (
[ModuleMeetingID] INT IDENTITY (1, 1) NOT NULL,
[ModuleID] INT NOT NULL,
[LecturerID] INT NOT NULL,
[StartTime] DATETIME NOT NULL,
[EndTime] DATETIME NOT NULL,
[ClassTypeID] INT NOT NULL,
[OriginalLanguageID] INT NOT NULL,
CONSTRAINT [PK_ModuleClasses] PRIMARY KEY CLUSTERED ([ModuleMeetingID] ASC),
CONSTRAINT [CK_ModuleMeeting] CHECK ([ClassTypeID]=(1) OR [ClassTypeID]=(2) OR
[ClassTypeID]=(3) OR [ClassTypeID]=(4)),
CONSTRAINT [ModuleMeeting_DatesValid] CHECK ([StartTime]<[EndTime]),
CONSTRAINT [FK_ModuleClasses_Module] FOREIGN KEY ([ModuleID]) REFERENCES
[dbo].[Module] ([ModuleID]),
CONSTRAINT [FK_ModuleMeeting_ClassTypes] FOREIGN KEY ([ClassTypeID]) REFERENCES
[dbo].[ClassTypes] ([ClassTypeID]),
CONSTRAINT [FK_ModuleMeeting_Employees] FOREIGN KEY ([LecturerID]) REFERENCES
[dbo].[Employees] ([EmployeeID]),
CONSTRAINT [FK_ModuleMeeting_Languages] FOREIGN KEY ([OriginalLanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
);
```

Tabela “ModuleMeetingMembers” (Reprezentuje klientów zapisanych na dane spotkanie w module w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ClientID** - unikalny identyfikator klienta
  - **Attendence** - informacja o obecności na spotkaniu
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ModuleMeetingID** - identyfikator spotkania w module

```
CREATE TABLE [dbo].[ModuleMeetingMembers] (
[ClientID] INT NOT NULL,
[Attendence] BIT NOT NULL,
[ModuleMeetingID] INT NOT NULL,
CONSTRAINT [PK_ModuleMeetingMembers] PRIMARY KEY CLUSTERED ([ClientID] ASC,
[ModuleMeetingID] ASC),
CONSTRAINT [FK_ModuleMeetingMembers_Clients] FOREIGN KEY ([ClientID]) REFERENCES
[dbo].[Clients] ([ClientID]),
CONSTRAINT [FK_ModuleMeetingMembers_ModuleMeeting] FOREIGN KEY ([ModuleMeetingID])
REFERENCES [dbo].[ModuleMeeting] ([ModuleMeetingID])
);
```

Tabela “ModuleMeetingOnlineSynchronic”

- Pola:
  - KLUCZ GŁÓWNY: **ModuleMeetingOnlineSynchronicID**
  - **ClassLink** - link do zajęć
  - KLUCZ OBCY: **ModuleMeetingID** - identyfikator spotkania w module

```
CREATE TABLE [dbo].[ModuleMeetingOnlineSynchronic] (
[ModuleMeetingOnlineSynchronicID] INT IDENTITY (1, 1) NOT NULL,
[ClassLink] VARCHAR (MAX) NOT NULL,
[ModuleMeetingID] INT NOT NULL,
CONSTRAINT [PK_ModuleMeetingOnlineSynchronic_1] PRIMARY KEY CLUSTERED
([ModuleMeetingOnlineSynchronicID] ASC),
CONSTRAINT [FK_ModuleMeetingOnlineSynchronic_ModuleMeeting] FOREIGN KEY
([ModuleMeetingID]) REFERENCES [dbo].[ModuleMeeting] ([ModuleMeetingID]),
CONSTRAINT [IX_ModuleMeetingOnlineSynchronic] UNIQUE NONCLUSTERED ([ModuleMeetingID]
ASC)
);
```

Tabela “ModuleMeetingsOnlineAsynchronic”

- Pola:
  - KLUCZ GŁÓWNY: **ModuleMeetingOnlineAsynchronicID**
  - **VideoLink** - link do nagrania zajęć
  - **LinkTermination** - data wygaśniecia linku do zajęć
  - KLUCZ OBCY: **ModuleMeetingID** - identyfikator spotkania w module
- Warunki integralności:
  - **‘LinkTermination’** musi mieć dłuższy termin niż dzisiejsza data

```
CREATE TABLE [dbo].[ModuleMeetingsOnlineAsynchronic] (
[ModuleMeetingOnlineAsynchronicID] INT IDENTITY (1, 1) NOT NULL,
[VideoLink] VARCHAR (MAX) NOT NULL,
[LinkTermination] VARCHAR (MAX) NOT NULL,
[ModuleMeetingID] INT NOT NULL,
CONSTRAINT [PK_ModuleMeetingsOnlineAsynchronic] PRIMARY KEY CLUSTERED
([ModuleMeetingOnlineAsynchronicID] ASC),
CONSTRAINT [CK_ModuleMeetingsOnlineAsynchronic_LinkTerminationValid] CHECK
([LinkTermination]>getdate()),
CONSTRAINT [FK_ModuleMeetingsOnlineAsynchronic_ModuleMeeting] FOREIGN KEY
([ModuleMeetingID]) REFERENCES [dbo].[ModuleMeeting] ([ModuleMeetingID]),
CONSTRAINT [IX_ModuleMeetingsOnlineAsynchronic] UNIQUE NONCLUSTERED
([ModuleMeetingID] ASC)
);
```

Tabela “ModuleMembers” (Reprezentuje klientów zapisanych na dany moduł w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ClientID** - unikalny identyfikator klienta
  - **PassModule** - informacja, czy klient zdał moduł
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ModuleID** - identyfikator modułu

```
CREATE TABLE [dbo].[ModuleMembers] (
[ClientID] INT NOT NULL,
[PassModule] BIT NOT NULL,
[ModuleID] INT NOT NULL,
CONSTRAINT [PK_CompositePrimaryKey] PRIMARY KEY CLUSTERED ([ClientID] ASC,
[ModuleID] ASC),
CONSTRAINT [FK_ModuleMembers_Clients] FOREIGN KEY ([ClientID]) REFERENCES
[dbo].[Clients] ([ClientID]),
CONSTRAINT [FK_ModuleMembers_Module] FOREIGN KEY ([ModuleID]) REFERENCES
[dbo].[Module] ([ModuleID])
);
```

Tabela “ModuleTranslatingLanguages” (Reprezentuje tłumacza i tłumaczony język w module):

- Pola:
  - KLUCZ GŁÓWNY: **ModuleMeetingID** - unikalny identyfikator spotkanie modułu
  - KLUCZ OBCY: **TranslatorID** - identyfikator tłumacza
  - KLUCZ OBCY: **LanguageID** - identyfikator języka

```
CREATE TABLE [dbo].[ModuleTranslatingLanguages] (
[ModuleMeetingID] INT NOT NULL,
[TranslatorID] INT NOT NULL,
[LanguageID] INT NOT NULL,
CONSTRAINT [PK_ModuleTranslatingLanguages] PRIMARY KEY CLUSTERED ([ModuleMeetingID]
ASC),
CONSTRAINT [FK_ModuleTranslatingLanguages_EmployeesLanguages] FOREIGN KEY
([LanguageID], [TranslatorID]) REFERENCES [dbo].[EmployeesLanguages] ([LanguageID],
[EmployeeID]),
CONSTRAINT [FK_ModuleTranslatingLanguages_ModuleMeeting] FOREIGN KEY
([ModuleMeetingID]) REFERENCES [dbo].[ModuleMeeting] ([ModuleMeetingID])
);
```

Tabela “OuterStudyMembers” (Reprezentuje klientów zewnętrznych w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ClientID** - unikalny identyfikator klienta
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **StudyMeetingID** - identyfikator spotkania na studiach
  - **Access** - informacja o dostępie do zajęć

```
CREATE TABLE [dbo].[OuterStudyMembers] (
[ClientID] INT NOT NULL,
[StudyMeetingID] INT NOT NULL,
[Access] BIT CONSTRAINT [DF_OuterStudyMembers_Access] DEFAULT ((0)) NOT NULL,
CONSTRAINT [PK_CompositePrimaryKey_OuterStudyMembers_1] PRIMARY KEY CLUSTERED
([ClientID] ASC, [StudyMeetingID] ASC),
CONSTRAINT [FK_OuterStudyMembers_StudyMeetings] FOREIGN KEY ([StudyMeetingID])
REFERENCES [dbo].[StudyMeetings] ([StudyMeetingID])
);
```

Tabela “OuterStudyPayments” (Reprezentuje informacje o płatnościach klienta zewnętrznego):

- Pola:
  - KLUCZ GŁÓWNY: **OuterStudyPaymentID** - unikalny identyfikator płatności klienta zewnętrznego
  - KLUCZ OBCY: **StudyMeetingID** - identyfikator spotkania na studiach
  - KLUCZ OBCY: **ClientID** - identyfikator klienta
  - **PayDate** - data do kiedy trzeba zapłacić
  - **Success** - informacja czy zapłacone
  - **Link** - link do płatności
  - **PaymentDate** - data zapłacenia
- Warunki integralności:
  - **‘PayDate’** musi być późniejsza od dzisiejszej daty

```
CREATE TABLE [dbo].[OuterStudyPayments] (
[OuterStudyPaymentID] INT IDENTITY (1, 1) NOT NULL,
[ClientID] INT NOT NULL,
[PayDate] DATE NOT NULL,
[StudyMeetingID] INT NOT NULL,
[Success] BIT CONSTRAINT [DEFAULT_OuterStudyPayments_Paid] DEFAULT ((0)) NOT NULL,
[Link] VARCHAR (100) NOT NULL,
```

```
[PaymentDate] DATETIME NULL,
CONSTRAINT [PK_OuterStudyPayments] PRIMARY KEY CLUSTERED ([OuterStudyPaymentID]
ASC),
CONSTRAINT [CK_OuterStudyPayments_PayDateValid] CHECK ([PayDate]>getdate()),
CONSTRAINT [FK_OuterStudyPayments_OuterStudyMembers] FOREIGN KEY ([ClientID],
[StudyMeetingID]) REFERENCES [dbo].[OuterStudyMembers] ([ClientID],
[StudyMeetingID])
);
```

Tabela “Positions” (Reprezentuje stanowiska pracowników w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **PositionID** - unikalny identyfikator stanowiska
  - **Name** - nazwa stanowiska
- Warunki integralności:
  - **‘Name’** może mieć wartość: ‘Boss’, ‘Lecturer’, ‘Translator’

```
CREATE TABLE [dbo].[Positions] (
[PositionID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
CONSTRAINT [PK_Positions] PRIMARY KEY CLUSTERED ([PositionID] ASC),
CONSTRAINT [Positions_NameValid] CHECK ([Name]='Boss' OR [Name]='Lecturer' OR
[Name]='Translator')
);
```

Tabela “ReunionPayments” (Reprezentuje płatności za zjazd):

- Pola:
  - KLUCZ GŁÓWNY: **ReunionPaymentID** - unikalny identyfikator płatności za zjazd
  - KLUCZ OBCY: **ClientID** - identyfikator klienta
  - **PayDate** - data do kiedy trzeba zapłacić
  - KLUCZ OBCY: **StudiesID** - identyfikator studiów
  - KLUCZ OBCY: **ReunionID** - identyfikator zjazdu
  - KLUCZ OBCY: **StudyMeetingID** - identyfikator spotkania na studiach
  - Success - informacja czy płatność się udała
  - Link - link do płatności
  - PaymentDate - data kiedy zapłacono
- Warunki integralności:
  - **‘PayDate’** musi być poźniejsza niż dzisiejsza data

```
CREATE TABLE [dbo].[ReunionPayments] (
[ReunionPaymentID] INT IDENTITY (1, 1) NOT NULL,
[ClientID] INT NOT NULL,
[PayDate] DATE NOT NULL,
[StudiesID] INT NOT NULL,
[ReunionID] INT NOT NULL,
[StudyMeetingID] INT NOT NULL,
[Success] BIT CONSTRAINT [DF_ReunionPayments_Success] DEFAULT ((0)) NOT NULL,
[Link] VARCHAR (100) NOT NULL,
[PaymentDate] DATETIME NULL,
CONSTRAINT [PK_ReunionPayments] PRIMARY KEY CLUSTERED ([ReunionPaymentID] ASC),
CONSTRAINT [FK_ReunionPayments_Reunions] FOREIGN KEY ([ReunionID]) REFERENCES
[dbo].[Reunions] ([ReunionID]),
CONSTRAINT [FK_ReunionPayments_StudyMeetingMembers] FOREIGN KEY ([ClientID],
[StudyMeetingID]) REFERENCES [dbo].[StudyMeetingMembers] ([ClientID],
[StudiesMeetingID])
);
```

Tabela “Reunions” (Reprezentuje zjazdy w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **ReunionID** - unikalny identyfikator zjazdu
  - **StartDate** - data rozpoczęcia zjazdu
  - **EndDate** - data zakończenia zjazdu
  - KLUCZ OBCY: **TermID** - identyfikator semestru
  - **Price** - cena za zjazd
- Warunki integralności:
  - **‘StartDate’** musi być wcześniejsza niż **‘EndDate’**
  - **‘Price’** musi mieć wartość większą od 0

```
CREATE TABLE [dbo].[Reunions] (
[ReunionID] INT IDENTITY (1, 1) NOT NULL,
[StartDate] DATE NOT NULL,
[EndDate] DATE NOT NULL,
[TermID] INT NOT NULL,
[Price] DECIMAL (16, 2) NOT NULL,
CONSTRAINT [PK_Reunions] PRIMARY KEY CLUSTERED ([ReunionID] ASC),
CONSTRAINT [Reunions_DatesValid] CHECK ([StartDate]<[EndDate]),
CONSTRAINT [Reunions_PriceValid] CHECK ([Price]>(0)),
CONSTRAINT [FK_Reunions_Term] FOREIGN KEY ([TermID]) REFERENCES [dbo].[Term]
([TermID])
);
```

Tabela “ReunionsSubjects” (Reprezentuje przedmioty na danym zjeździe w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **ReunionsSubjectsID** - unikalny identyfikator przedmiotu na zjeździe
  - KLUCZ OBCY: **ReunionID** - identyfikator zjazdu
  - KLUCZ OBCY: **SubjectID** - identyfikator przedmiotu

```
CREATE TABLE [dbo].[ReunionsSubjects] (
[ReunionsSubjectsID] INT IDENTITY (1, 1) NOT NULL,
[ReunionID] INT NOT NULL,
[SubjectID] INT NOT NULL,
CONSTRAINT [PK_ReunionsSubjects] PRIMARY KEY CLUSTERED ([ReunionsSubjectsID] ASC),
CONSTRAINT [FK_ReunionsSubjects_Reunions] FOREIGN KEY ([ReunionID]) REFERENCES
[dbo].[Reunions] ([ReunionID]),
CONSTRAINT [FK_ReunionsSubjects_Subjects] FOREIGN KEY ([SubjectID]) REFERENCES
[dbo].[Subjects] ([SubjectID])
);
```

Tabela “RoomModuleMeetings” (łączy sale z module meetings):

- Pola:
  - KLUCZ GŁÓWNY: **ModuleMeetingID** - unikalny identyfikator spotkania w module
  - KLUCZ OBCY: **RoomID** - identyfikator sali

```
CREATE TABLE [dbo].[RoomModuleMeetings] (
[RoomID] INT NOT NULL,
[ModuleMeetingID] INT NOT NULL,
CONSTRAINT [PK_RoomModuleMeetings] PRIMARY KEY CLUSTERED ([ModuleMeetingID] ASC),
CONSTRAINT [FK_RoomModuleMeetings_Rooms] FOREIGN KEY ([RoomID]) REFERENCES
[dbo].[Rooms] ([RoomID])
);
```

Tabela “Rooms” (Reprezentuje sale zajęciowe w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **RoomID** - unikalny ID sali;
  - **PersonLimit** - limit osób w sali
  - **Floor** - piętro na którym znajduje się sala
  - **Room** - sala
  - KLUCZ OBCY: **BuildingAddressID** - identyfikator adresu budynku
- Warunki integralności:
  - **‘Room’** musi mieć wartość większą od 0
  - **‘PersonLimit’** musi mieć wartość większą od 0

```
CREATE TABLE [dbo].[Rooms] (
[RoomID] INT IDENTITY (1, 1) NOT NULL,
[PersonLimit] INT NOT NULL,
[Floor] INT CONSTRAINT [DEFAULT_Rooms_Floor] DEFAULT ((0)) NULL,
[Room] NCHAR (10) NOT NULL,
[BuildingAddressID] INT NOT NULL,
CONSTRAINT [PK_Rooms] PRIMARY KEY CLUSTERED ([RoomID] ASC),
CONSTRAINT [Rooms_PersonLimitValid] CHECK ([PersonLimit]>(0)),
CONSTRAINT [Rooms_RoomValid] CHECK ([Room]>(0))
);
```

Tabela “RoomStudyMeetings” (Łączy room z study meetings):

- Pola:
  - KLUCZ OBCY: **RoomID -** identyfikator sali
  - KLUCZ GŁÓWNY: **StudyMeetingID -** unikalny identyfikator spotkania na studiach

```
CREATE TABLE [dbo].[RoomStudyMeetings] (
[RoomID] INT NOT NULL,
[StudyMeetingID] INT NOT NULL,
CONSTRAINT [PK_RoomStudyMeetings] PRIMARY KEY CLUSTERED ([StudyMeetingID] ASC),
CONSTRAINT [FK_RoomStudyMeetings_Rooms] FOREIGN KEY ([RoomID]) REFERENCES
[dbo].[Rooms] ([RoomID])
);
```

Tabela “Studies” (Reprezentuje studia w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **StudiesID** - unikalny identyfikator studiów
  - **PersonLimit** - limit osób na studiach
  - **Name** - nazwa studiów
  - **Description** - opis studiów
  - **EntryFee** - kwota wpisowego na studia
- Warunki Integralności
  - **‘PersonLimit’** musi mieć wartość większą od 0
  - **‘EntryFee’** musi mieć wartość większą bądź równą 0

```
CREATE TABLE [dbo].[Studies] (
[StudiesID] INT IDENTITY (1, 1) NOT NULL,
[PersonLimit] INT NOT NULL,
[Name] VARCHAR (50) NOT NULL,
[Description] VARCHAR (150) NOT NULL,
[EntryFee] DECIMAL (16, 2) NOT NULL,
CONSTRAINT [PK_Lecturers] PRIMARY KEY CLUSTERED ([StudiesID] ASC),
CONSTRAINT [Studies_EntryFeeValid] CHECK ([EntryFee]>=(0)),
CONSTRAINT [Studies_PersonLimitValid] CHECK ([PersonLimit]>(0))
);
```

Tabela “StudyMeetingMembers” (Reprezentuje informacje o osobach zapisanych na spotkanie na studiach w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ClientID** - unikalny identyfikator klienta
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **StudiesMeetingID** - identyfikator spotkania na studiach
  - **Attendence** - informacja o obecności
  - **ReunionAccess** - informacja o dostępie do zjazdu

```
CREATE TABLE [dbo].[StudyMeetingMembers] (
[ClientID] INT NOT NULL,
[StudiesMeetingID] INT NOT NULL,
[Attendence] BIT CONSTRAINT [DF_StudyMeetingMembers_Attendence] DEFAULT ((0)) NOT
NULL,
[ReunionAccess] BIT CONSTRAINT [DF_StudyMeetingMembers_ReunionAccess] DEFAULT ((0))
NOT NULL,
CONSTRAINT [PK_CompositePrimaryKey_StudyMeetingMembers] PRIMARY KEY CLUSTERED
([ClientID] ASC, [StudiesMeetingID] ASC),
CONSTRAINT [FK_StudyMeetingMembers_StudyMeeting] FOREIGN KEY ([StudiesMeetingID])
REFERENCES [dbo].[StudyMeetings] ([StudyMeetingID])
);
```

Tabela “StudyMeetings”(Reprezentuje informacje o spotkaniu na studiach w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **StudyMeetingID** - unikalny identyfikator spotkania na studiach
  - **Date** - data zajęć
  - **StartTime** - czas rozpoczęcia zajęć
  - **EndTime** - czas zakończenia zajęć
  - KLUCZ OBCY: **SubjectID** - identyfikator przedmiotu
  - KLUCZ OBCY: **LecturerID** - identyfikator wykładowcy
  - KLUCZ OBCY: **ClassTypeID** - identyfikator typu spotkania
  - KLUCZ OBCY: **OriginalLanguageID** - orginalny język zajęć
  - KLUCZ OBCY: **ReunionID** - identyfikator zjazdu
  - **PriceForOuterStudent** - cena dla zewnętrznego klienta
  - **OuterStudentLimit** - limit studentów zewnętrznych na dane spotkanie
  - **StudentLimit** - limit studentów
- Warunki integralności:
  - **‘PriceForOuterStudent’** musi mieć wartość większą bądź równą 0
  - **‘OuterStudentLimit’** musi mieć wartość większą bądź równą 0
  - **‘StartTime’** musi być wcześniejsze niż **‘EndTime’**
  - **‘StudentLimit’** musi mieć wartość większą od 0

```
CREATE TABLE [dbo].[StudyMeetings] (
[StudyMeetingID] INT IDENTITY (1, 1) NOT NULL,
[Date] DATE NOT NULL,
[StartTime] TIME (7) NOT NULL,
[EndTime] TIME (7) NOT NULL,
[SubjectID] INT NOT NULL,
[LecturerID] INT NOT NULL,
[ClassTypeID] INT NOT NULL,
[OriginalLanguageID] INT NOT NULL,
[ReunionID] INT NOT NULL,
[PriceForOuterStudent] DECIMAL (16, 2) NOT NULL,
[OuterStudentLimit] INT NOT NULL,
[StudentLimit] INT NULL,
CONSTRAINT [PK_StudyMeeting] PRIMARY KEY CLUSTERED ([StudyMeetingID] ASC),
CONSTRAINT [CK_StudyMeetings_StudentLimitValid] CHECK ([StudentLimit]>(0)),
CONSTRAINT [StudyMeetings_DateValid] CHECK ([Date]>=getdate()),
CONSTRAINT [StudyMeetings_OuterStudentLimitValid] CHECK ([OuterStudentLimit]>=(0)),
CONSTRAINT [StudyMeetings_PriceForOuterStudentValid] CHECK
([PriceForOuterStudent]>=(0)),
CONSTRAINT [StudyMeetings_TimesValid] CHECK ([EndTime]>[StartTime])
);
```

Tabela “StudyMeetingsOnlineAsynchronic” (Reprezentuje zajęcia asynchroniczne na studiach w bazie danych) :

- Pola:
  - KLUCZ GŁÓWNY: **AsynchronicID** - unikalny identyfikator zajęć asynchronicznych
  - **VideoLink** - link do zajęć asynchronicznych
  - **LinkTermination** - data wygaśnięcia linku do zajęć
  - KLUCZ OBCY: **StudyMeetingID** - identyfikator spotkania na studiach
- Warunki integralności:
  - **‘LinkTermination’** musi mieć dłuższy termin niż dzisiejsza data

```
CREATE TABLE [dbo].[StudyMeetingsOnlineAsynchronic] (
[StudyMeetingOnlineAsynchronicID] INT IDENTITY (1, 1) NOT NULL,
[VideoLink] VARCHAR (MAX) NOT NULL,
[LinkTermination] DATETIME NOT NULL,
[StudyMeetingID] INT NOT NULL,
CONSTRAINT [PK_Asynchronic] PRIMARY KEY CLUSTERED ([StudyMeetingOnlineAsynchronicID]
ASC),
CONSTRAINT [OnlineAsynchronic_LinkValid] CHECK ([LinkTermination]>getdate()),
CONSTRAINT [FK_OnlineAsynchronic_StudyMeeting] FOREIGN KEY ([StudyMeetingID])
REFERENCES [dbo].[StudyMeetings] ([StudyMeetingID]),
CONSTRAINT [IX_StudyMeetingsOnlineAsynchronic] UNIQUE NONCLUSTERED ([StudyMeetingID]
ASC)
);
```

Tabela “StudyMeetingsOnlineSynchronic” (Reprezentuje zajęcia synchroniczne na studiach w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **OnlineSynchronicID** - unikalny identyfikator zajęć online
  - **ClassLink** - link do zajęć
  - KLUCZ OBCY: **StudyMeetingID** - identyfikator spotkania na studiach

```
CREATE TABLE [dbo].[StudyMeetingsOnlineSynchronic] (
[StudyMeetingOnlineSynchronicID] INT IDENTITY (1, 1) NOT NULL,
[ClassLink] VARCHAR (MAX) NOT NULL,
[StudyMeetingID] INT NOT NULL,
CONSTRAINT [PK_Synchronic] PRIMARY KEY CLUSTERED ([StudyMeetingOnlineSynchronicID]
ASC),
CONSTRAINT [FK_OnlineSynchronic_StudyMeeting] FOREIGN KEY ([StudyMeetingID])
REFERENCES [dbo].[StudyMeetings] ([StudyMeetingID]),
CONSTRAINT [IX_StudyMeetingsOnlineSynchronic] UNIQUE NONCLUSTERED ([StudyMeetingID]
ASC));
```

Tabela “StudyMembers” (Reprezentuje klientów studiujących w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ClientID** - unikalny identyfikator klienta
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **StudiesID** - identyfikator studiów
  - **PassExam** - informacja czy egzamin zdany
  - **ExamGrade** - ocena z egzaminu
- Warunki integralności:
  - **ExamGrade** może mieć wartość 3.0,3.5,4.0,4.5,5.0, chyba że egzamin nie został zdany to **ExamGrade** może być nullem

```
CREATE TABLE [dbo].[StudyMembers] (
[ClientID] INT NOT NULL,
[StudiesID] INT NOT NULL,
[PassExam] BIT CONSTRAINT [DF_StudyMembers_PassExam] DEFAULT ((0)) NOT NULL,
[ExamGrade] DECIMAL (2, 1) NULL,
CONSTRAINT [PK_CompositePrimaryKey2] PRIMARY KEY CLUSTERED ([ClientID] ASC,
[StudiesID] ASC),
CONSTRAINT [CK_ExamGrade] CHECK ([PassExam]=(1) AND ([ExamGrade]=(5.0) OR
[ExamGrade]=(4.5) OR [ExamGrade]=(4.0) OR [ExamGrade]=(3.5) OR [ExamGrade]=(3.0)) OR
[PassExam]=(0) AND [ExamGrade] IS NULL),
CONSTRAINT [FK_StudyMembers_Clients] FOREIGN KEY ([ClientID]) REFERENCES
[dbo].[Clients] ([ClientID]),
CONSTRAINT [FK_StudyMembers_Studies] FOREIGN KEY ([StudiesID]) REFERENCES
[dbo].[Studies] ([StudiesID])
);
```

Tabela “StudyPayments” (Reprezentuje płatności za studia w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **StudyPaymentID** - unikalny identyfikator płatności za studia
  - KLUCZ OBCY: **ClientID** - identyfikator klienta
  - **PayDate** - data do kiedy zapłacić
  - KLUCZ OBCY: **StudyID** - identyfikator studiów
  - **Link** - link do płatności
  - **Success** - informacja czy zapłacone
  - **PaymentDate** - data kiedy zostało zapłacone
- Warunki integralności:
  - **‘PayDate’** musi być późniejsza od daty dzisiejszej

```
CREATE TABLE [dbo].[StudyPayments] (
[StudyPaymentID] INT IDENTITY (1, 1) NOT NULL,
[ClientID] INT NOT NULL,
[PayDate] DATETIME NOT NULL,
[StudyID] INT NOT NULL,
[Link] VARCHAR (100) NOT NULL,
[Success] BIT CONSTRAINT [DF_StudyPayments_Success] DEFAULT ((0)) NOT NULL,
[PaymentDate] DATETIME NULL,
CONSTRAINT [PK_StudyPayments] PRIMARY KEY CLUSTERED ([StudyPaymentID] ASC),
CONSTRAINT [FK_StudyPayments_StudyMembers] FOREIGN KEY ([ClientID], [StudyID])
REFERENCES [dbo].[StudyMembers] ([ClientID], [StudiesID]));
```

Tabela “StudyTranslatingLanguages” (Reprezentuje tłumacza i tłumaczony język na studiach):

- Pola:
  - KLUCZ GŁÓWNY: **StudyMeetingID** - unikalny identyfikator spotkania na studiach
  - KLUCZ OBCY: **TranslatorID** - identyfikator tłumacza
  - KLUCZ OBCY: **LanguageID** - identyfikator języka

```
CREATE TABLE [dbo].[StudyTranslatingLanguages] (
[StudyMeetingID] INT NOT NULL,
[TranslatorID] INT NOT NULL,
[LanguageID] INT NOT NULL,
CONSTRAINT [PK_StudyTranslatingLanguages] PRIMARY KEY CLUSTERED ([StudyMeetingID]
ASC),
CONSTRAINT [FK_StudyTranslatingLanguages_EmployeesLanguages] FOREIGN KEY
([LanguageID], [TranslatorID]) REFERENCES [dbo].[EmployeesLanguages] ([LanguageID],
[EmployeeID]),
CONSTRAINT [FK_StudyTranslatingLanguages_StudyMeetings] FOREIGN KEY
([StudyMeetingID]) REFERENCES [dbo].[StudyMeetings] ([StudyMeetingID])
);
```

Tabela “SubjectMembers” (Reprezentuje studentów uczeszczających na dany przedmiot w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ClientID** - unikalny identyfikator klienta
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **SubjectID** - identyfikator przedmiotu
  - **Pass** - informacja czy przedmiot został zaliczony
  - **Grade** - ocena

```
CREATE TABLE [dbo].[SubjectMembers] (
[ClientID] INT CONSTRAINT [DF_SubjectMembers_ClientID] DEFAULT ((0)) NOT NULL,
[SubjectID] INT NOT NULL,
[Pass] BIT CONSTRAINT [DF_SubjectMembers_Pass] DEFAULT ((0)) NOT NULL,
[Grade] DECIMAL (2, 1) NULL,
CONSTRAINT [PK_CompositePrimaryKey_SubjectMembers] PRIMARY KEY CLUSTERED ([ClientID]
ASC, [SubjectID] ASC),
CONSTRAINT [FK_SubjectMembers_Subjects] FOREIGN KEY ([SubjectID]) REFERENCES
[dbo].[Subjects] ([SubjectID])
);
```

Tabela “Subjects” (Reprezentuje informacje o przedmiotach na studiach w bazie danych) :

- Pola:
  - KLUCZ GŁÓWNY: **SubjectID** - unikalny identyfikator przedmiotu
  - **Name** - nazwa przedmiotu
  - KLUCZ OBCY: **TermID** - identyfikator semestru
  - KLUCZ OBCY: **LecturerID** - identyfikator wykładowcy

```
CREATE TABLE [dbo].[Subjects] (
[SubjectID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
[TermID] INT NOT NULL,
[LecturerID] INT NOT NULL,
CONSTRAINT [PK_Subjects_1] PRIMARY KEY CLUSTERED ([SubjectID] ASC),
CONSTRAINT [FK_Subjects_Lecturers1] FOREIGN KEY ([LecturerID]) REFERENCES
[dbo].[Employees] ([EmployeeID]),
CONSTRAINT [FK_Subjects_Term] FOREIGN KEY ([TermID]) REFERENCES [dbo].[Term]
([TermID])
);
```

Tabela “Syllabus” (Reprezentuje syllabusy przedmiotów w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **SyllabusID** - unikalny identyfikator sylabusu
  - KLUCZ OBCY: **StudiesID** - identyfikator studiów
  - **Link** - link do sylabusa

```
CREATE TABLE [dbo].[Syllabus] (
[StudiesID] INT NOT NULL,
[SyllabusID] INT IDENTITY (1, 1) NOT NULL,
[Link] VARCHAR (MAX) NOT NULL,
CONSTRAINT [PK_Syllabus] PRIMARY KEY CLUSTERED ([SyllabusID] ASC),
CONSTRAINT [FK_Syllabus_Studies1] FOREIGN KEY ([StudiesID]) REFERENCES
[dbo].[Studies] ([StudiesID])
);
```

Tablica “Term” (Reprezentuje semestry studiów w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **TermID** - unikalny identyfikator semestru
  - KLUCZ OBCY: **StudiesID** - identyfikator studiów
  - **TermNumber** - cena za semestr

```
CREATE TABLE [dbo].[Term] (
[TermID] INT IDENTITY (1, 1) NOT NULL,
[StudiesID] INT NOT NULL,
[TermNumber] INT NULL,
CONSTRAINT [PK_Term] PRIMARY KEY CLUSTERED ([TermID] ASC),
CONSTRAINT [CK_Term_1] CHECK ([TermNumber]=(1) OR [TermNumber]=(2) OR
[TermNumber]=(3) OR [TermNumber]=(4) OR [TermNumber]=(5) OR [TermNumber]=(6)),
CONSTRAINT [FK_Term_Studies] FOREIGN KEY ([StudiesID]) REFERENCES [dbo].[Studies]
([StudiesID])
);
```

Tabela “WebinarMeetings” (Reprezentuje spotkanie webinaru w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **WebinarMeetingID** - unikalny identyfikator spotkania webinaru
  - **ClassLink** - link do zajęć
  - **VideoLink** - link do nagrania z zajęć
  - **LinkTermination** - data wygaśnięcia linku do nagrania
  - KLUCZ OBCY: **OriginalLanguageID** - identyfikator orginalnego języka zajęć
  - KLUCZ OBCY: **LecturerID** - identyfikator wykładowcy
  - **StartDate** - data rozpoczęcia spotkania webinaru
  - **EndDate** - data zakończenia spotkania webinaru
  - KLUCZ OBCY: **WebinarID** - identyfikator webinaru
- Warunki integralności:
  - **‘StartDate’** musi być wcześniejsza od ‘EndDate’
  - **‘LinkTermination’** musi być poźniejsze niż dzisiejsza data

```
CREATE TABLE [dbo].[WebinarMeetings] (
[WebinarMeetingID] INT IDENTITY (1, 1) NOT NULL,
[ClassLink] VARCHAR (MAX) NOT NULL,
[VideoLink] VARCHAR (MAX) NULL,
[LinkTermination] DATETIME NOT NULL,
[OriginalLanguageID] INT NOT NULL,
[LecturerID] INT NOT NULL,
[StartDate] DATETIME NOT NULL,
[EndDate] DATETIME NOT NULL,
[WebinarID] INT NOT NULL,
CONSTRAINT [PK_WebinarMeetings] PRIMARY KEY CLUSTERED ([WebinarMeetingID] ASC),
CONSTRAINT [WebinarMeetings_DatesValid] CHECK ([EndDate]>[StartDate]),
CONSTRAINT [WebinarMeetings_LinkTermValid] CHECK ([LinkTermination]>=getdate())
);
```

Tabela “WebinarMembers” (Reprezentuje osoby zapisane na webinar w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **WebinarMeetingID** - unikalny identyfikator spotkania webinaru
  - KLUCZ GŁÓWNY, KLUCZ OBCY: **ClientID** - identyfikator klienta
  - **Access** - informacja czy klient posiada dostęp do webinaru
  - **LinkTermination** - termin do kiedy dostęp do webinaru

```
CREATE TABLE [dbo].[WebinarMembers] (
[WebinarMeetingID] INT NOT NULL,
[ClientID] INT NOT NULL,
[Access] BIT CONSTRAINT [DF_WebinarMembers_Access] DEFAULT ((0)) NOT NULL,
[LinkTermination] DATETIME NULL,
CONSTRAINT [PK_CompositePrimaryKey_WebinarMembers] PRIMARY KEY CLUSTERED ([ClientID]
ASC, [WebinarMeetingID] ASC),
CONSTRAINT [FK_WebinarMembers_Clients] FOREIGN KEY ([ClientID]) REFERENCES
[dbo].[Clients] ([ClientID]),
CONSTRAINT [FK_WebinarMembers_Webinars] FOREIGN KEY ([WebinarMeetingID]) REFERENCES
[dbo].[WebinarMeetings] ([WebinarMeetingID])
);
```

Tabela “WebinarPayments” (Reprezentuje płatności za webinar w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **WebinarPaymentsID** - unikalny identyfikator płatności za webinar
  - KLUCZ OBCY: **WebinarMeetingID** - identyfikator spotkania webinaru
  - KLUCZ OBCY: **ClientID** - identyfikator klienta
  - **PayDate** - data do kiedy trzeba zapłacić
  - **Link** - link do płatności
  - **Success** - informacja czy płatność się udała
  - **PayedDate** - data kiedy zapłacono
- Warunki integralności:
  - **‘PayDate’** musi być późniejsza niż dzisiejsza data

```
CREATE TABLE [dbo].[WebinarPayments] (
[WebinarPaymentsID] INT IDENTITY (1, 1) NOT NULL,
[WebinarMeetingID] INT NOT NULL,
[ClientID] INT NOT NULL,
[PayDate] DATETIME NULL,
[Link] VARCHAR (MAX) NOT NULL,
[Success] BIT CONSTRAINT [DF_WebinarPayments_Success] DEFAULT ((0)) NOT NULL,
[PayedDate] DATETIME NULL,
CONSTRAINT [PK_WebinarPayments_1] PRIMARY KEY CLUSTERED ([WebinarPaymentsID] ASC),
CONSTRAINT [WebinarPayments_PayDateValid] CHECK ([PayDate]>=getdate()),
CONSTRAINT [FK_WebinarPayments_WebinarMembers] FOREIGN KEY ([ClientID],
[WebinarMeetingID]) REFERENCES [dbo].[WebinarMembers] ([ClientID],
[WebinarMeetingID]),
CONSTRAINT [FK_WebinarPayments_Webinars] FOREIGN KEY ([WebinarMeetingID]) REFERENCES
[dbo].[WebinarMeetings] ([WebinarMeetingID])
);
```

Tabela “Webinars” (Reprezentuje webinary w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **WebinarID** - unikalny identyfikator webinaru
  - **Name** - nazwa webinaru
  - **Description** - opis webinaru
  - **Price** - cena
- Warunki integralności:
  - **‘Price’** musi mieć wartość większą bądź równą 0

```
CREATE TABLE [dbo].[Webinars] (
[WebinarID] INT IDENTITY (1, 1) NOT NULL,
[Name] VARCHAR (50) NOT NULL,
[Description] VARCHAR (150) NOT NULL,
[Price] DECIMAL (16, 2) NOT NULL,
CONSTRAINT [PK_Webinars_1] PRIMARY KEY CLUSTERED ([WebinarID] ASC),
CONSTRAINT [Webinars_PriceValid] CHECK ([Price]>=(0))
);
```

Tabela “WebinarTranslatingLanguages” (Reprezentuje tłumaczy i języki w webinarze w bazie danych):

- Pola:
  - KLUCZ GŁÓWNY: **WebinarMeetingID** - unikalny identyfikator spotkania webinaru
  - KLUCZ OBCY: **TranslatorID** - identyfikator tłumacza
  - KLUCZ OBCY: **LanguageID** - identyfikator języka

```
CREATE TABLE [dbo].[WebinarTranslatingLanguages] (
[WebinarMeetingID] INT NOT NULL,
[TranslatorID] INT NOT NULL,
[LanguageID] INT NOT NULL,
CONSTRAINT [PK_WebinarTranslatingLanguages] PRIMARY KEY CLUSTERED
([WebinarMeetingID] ASC),
CONSTRAINT [FK_WebinarTranslatingLanguages_EmployeesLanguages] FOREIGN KEY
([LanguageID], [TranslatorID]) REFERENCES [dbo].[EmployeesLanguages] ([LanguageID],
[EmployeeID]),
CONSTRAINT [FK_WebinarTranslatingLanguages_WebinarMeetings] FOREIGN KEY
([WebinarMeetingID]) REFERENCES [dbo].[WebinarMeetings] ([WebinarMeetingID])
);
```

WIDOKI:

AttendanceOnCourses [Judyta]

Wyświetla informacje o obecności na kursach

```
CREATE VIEW [dbo].[AttendanceOnCourses] AS
SELECT
CC.CourseID,
CC.Name AS CourseName,
MM.ModuleMeetingID,
MM.StartTime,
C.FirstName,
C.LastName,
CASE
WHEN MMM.Attendence = 1 THEN 'Present'
ELSE 'Absent'
END AS AttendanceStatus
FROM
ModuleMeetingMembers MMM
JOIN
ModuleMeeting MM ON MMM.ModuleMeetingID = MM.ModuleMeetingID
JOIN
Clients C ON MMM.ClientID = C.ClientID
Join
[Module] M On M.ModuleID = MM.ModuleID
Join
Courses CC on CC.CourseID = M.CourseID
```

AttendanceOnStudies [Judyta]

Wyświetla informacje o obecności na zajęciach na studiach

```
CREATE VIEW [dbo].[AttendanceOnStudies] AS
SELECT
S.SubjectID,
S.Name as SubjectName,
SM.StudyMeetingID,
SM.Date AS MeetingDate,
C.FirstName,
C.LastName,
CASE
WHEN SMM.Attendence = 1 THEN 'Present'
ELSE 'Absent'
END AS AttendanceStatus
FROM
StudyMeetingMembers SMM
JOIN
StudyMeetings SM ON SMM.StudiesMeetingID = SM.StudyMeetingID
JOIN
Clients C ON SMM.ClientID = C.ClientID
join
Subjects S on S.SubjectID = SM.SubjectID;
```

AttendanceOnWebinars [Judyta]

Wyświetla informacje o ludziach zapisanych na webinar (z dostępem na webinar)

```
CREATE VIEW [dbo].[AttendanceOnWebinars] AS
SELECT
W.WebinarID,
W.Name as WebinarName,
WBM.WebinarMeetingID,
WBM.StartDate AS MeetingDate,
C.FirstName,
C.LastName,
CASE
WHEN WM.Access = 1 THEN 'Access'
ELSE 'WithoutAccess'
END AS Status
FROM
WebinarMembers WM
JOIN
WebinarMeetings WBM ON WM.WebinarMeetingID = WBM.WebinarMeetingID
JOIN
Clients C ON WM.ClientID = C.ClientID
JOIN
Webinars W on W.WebinarID = WBM.WebinarID
```

ClientOverlappingMeetings [Judyta]

Wyświetla informacje o czasowo kolidujących ze sobą zajęciach na które klient jest zapisany

```
CREATE VIEW [dbo].[ClientOverlappingMeetings] AS
WITH ClientMeetings AS (
SELECT
'Study' AS MeetingType,
SMM.ClientID,
SM.StudyMeetingID AS MeetingID,
CAST(SM.Date AS DATETIME) + CAST(SM.StartTime AS DATETIME) AS StartDateTime,
CAST(SM.Date AS DATETIME) + CAST(SM.EndTime AS DATETIME) AS EndDateTime
FROM
StudyMeetingMembers SMM
JOIN
StudyMeetings SM ON SMM.StudiesMeetingID = SM.StudyMeetingID

UNION ALL

SELECT
'Webinar' AS MeetingType,
WM.ClientID,
WM.WebinarMeetingID AS MeetingID,
WBM.StartDate AS StartDateTime,
WBM.EndDate AS EndDateTime
FROM
WebinarMembers WM
JOIN
WebinarMeetings WBM ON WM.WebinarMeetingID = WBM.WebinarMeetingID

UNION ALL

SELECT
'Module' AS MeetingType,
MMM.ClientID,
MMM.ModuleMeetingID AS MeetingID,
MM.StartTime AS StartDateTime,
MM.EndTime AS EndDateTime
FROM
ModuleMeetingMembers MMM
JOIN
ModuleMeeting MM ON MMM.ModuleMeetingID = MM.ModuleMeetingID
```

```
)
SELECT
CM1.ClientID AS ClientID1,
CM1.MeetingType AS MeetingType1,
CM1.MeetingID AS MeetingID1,
CM1.StartDateTime AS StartDateTime1,
CM1.EndDateTime AS EndDateTime1,
CM2.MeetingType AS MeetingType2,
CM2.MeetingID AS MeetingID2,
CM2.StartDateTime AS StartDateTime2,
CM2.EndDateTime AS EndDateTime2
FROM
ClientMeetings CM1
INNER JOIN
ClientMeetings CM2 ON CM1.ClientID = CM2.ClientID
AND CM1.MeetingID < CM2.MeetingID
AND CM1.StartDateTime < CM2.EndDateTime
AND CM1.EndDateTime > CM2.StartDateTime;
```

ClientsAddress [Judyta]

Wyświetla informacje o adresie klienta

```
CREATE VIEW [dbo].[ClientsAddress] AS
SELECT
C.FirstName + ' ' + C.LastName AS 'NameAndLastname',
C.Phone,
C.Email,
CONCAT(A.Address, ', ', Ci.Name, ', ', Co.Name) AS 'FullAddress'
FROM Clients C
JOIN Addresses A ON C.AddressID = A.AddressID
JOIN Cities Ci ON A.CityID = Ci.CityID
JOIN Countries Co ON Ci.CountryID = Co.CountryID;
```

ClientsClasses [Judyta]

Wyświetla informacje o zajęciach na które jest zapisany klient

```
CREATE VIEW [dbo].[ClientsClasses] AS
SELECT
C.FirstName + ' ' + C.LastName AS 'FirstnameAndLastname',
ISNULL(STRING_AGG(S.Name, ', '), '') AS 'Studies',
ISNULL(STRING_AGG(W.Name, ', '), '') AS 'Webinars',
ISNULL(STRING_AGG(CO.Name, ', '), '') AS 'Courses'
FROM Clients C
LEFT JOIN StudyMembers SM ON C.ClientID = SM.ClientID
LEFT JOIN Studies S ON SM.StudiesID = S.StudiesID
LEFT JOIN WebinarMembers WM ON C.ClientID = WM.ClientID
LEFT JOIN Webinars W ON WM.WebinarMeetingID = W.WebinarID
LEFT JOIN CoursesMembers CM ON C.ClientID = CM.ClientID
LEFT JOIN Courses CO ON CM.CourseID = CO.CourseID
GROUP BY C.FirstName, C.LastName;
```

ClientsCoursePayments [Judyta]

Wyświetla informacje o płatnościach klientów zapisanych na kursy

```
CREATE VIEW [dbo].[ClientsCoursePayments] AS
SELECT
C.ClientID,
C.FirstName,
C.LastName,
CC.Name,
CASE
WHEN CP.CoursePaymentTypeID = 1 THEN 'Advance'
WHEN CP.CoursePaymentTypeID = 2 THEN 'TotalPrice'
When CP.CoursePaymentTypeID = 3 THEN 'RemainingAmount'
END AS PaymentType,
COALESCE(SUM(CASE
WHEN CP.CoursePaymentTypeID = 1 THEN CC.AdvancePaymentPrice
WHEN CP.CoursePaymentTypeID = 2 THEN CC.Price
When CP.CoursePaymentTypeID = 3 THEN (CC.Price - CC.AdvancePaymentPrice)
END), 0) AS TotalPaymentAmount
FROM Clients C
JOIN CoursesPayments CP ON C.ClientID = CP.ClientID
JOIN Courses CC ON CP.CourseID = CC.CourseID
GROUP by c.ClientID,C.FirstName,C.LastName,CC.Name,CP.CoursePaymentTypeID,
CC.AdvancePaymentPrice,CC.Price
```

ClientsWebinarPayments [Judyta]

Wyświetla informacje o płatnościach klientów zapisanych na kurs

```
CREATE VIEW [dbo].[ClientsWebinarsPayments] AS
SELECT
C.ClientID,
C.FirstName,
C.LastName,
ISNULL(SUM(ISNULL(W.Price, 0)), 0) AS TotalWebinarsPayment
FROM Clients C
```

```
LEFT JOIN WebinarMembers WM ON C.ClientID = WM.ClientID
LEFT JOIN Webinars W ON WM.WebinarMeetingID = W.WebinarID
GROUP BY C.ClientID, C.FirstName, C.LastName;
```

ClientsStudiesTotalPayments [Judyta]

Wyświetla podsumowanie płatności klientów dotyczących studiów

```
CREATE VIEW [dbo].[ClientsStudiesTotalPayments] AS
SELECT
C.ClientID,
C.FirstName,
C.LastName,
ISNULL(SUM(CASE
WHEN OSM.Access = 1 AND sms.PriceForOuterStudent IS NOT NULL THEN
sms.PriceForOuterStudent
ELSE 0
END), 0) AS TotalOuterStudentMeetingPayment,
ISNULL(SUM(R.Price), 0) AS TotalReunionPayment,
ISNULL(SUM(CASE
WHEN SP.PaymentTypeID = 1 THEN T.Price
WHEN SP.PaymentTypeID = 2 THEN S.EntryFee
ELSE 0
END), 0) AS TotalStudyPayment,
ISNULL(SUM(CASE
WHEN SP.PaymentTypeID = 1 THEN T.Price
WHEN SP.PaymentTypeID = 2 THEN S.EntryFee
ELSE 0
END), 0) +
ISNULL(SUM(CASE
WHEN OSM.Access = 1 AND sms.PriceForOuterStudent IS NOT NULL THEN
sms.PriceForOuterStudent
ELSE 0
END), 0) +
ISNULL(SUM(R.Price), 0) AS TotalPayment
FROM Clients C
LEFT JOIN StudyMeetingMembers SMM ON C.ClientID = SMM.ClientID
LEFT JOIN OuterStudyMembers OSM ON C.ClientID = OSM.ClientID AND
SMM.StudiesMeetingID = OSM.StudyMeetingID
LEFT JOIN OuterStudyPayments OSP ON OSM.ClientID = OSP.ClientID AND
OSM.StudyMeetingID = OSP.StudyMeetingID
LEFT JOIN ReunionPayments RP ON C.ClientID = RP.ClientID
LEFT JOIN Reunions R ON RP.ReunionID = R.ReunionID
LEFT JOIN StudyPayments SP ON C.ClientID = SP.ClientID
LEFT JOIN StudyMembers SM ON SP.ClientID = SM.ClientID AND SP.StudyID = SM.StudiesID
LEFT JOIN Studies S ON SM.StudiesID = S.StudiesID
LEFT JOIN Term T ON S.StudiesID = T.StudiesID
Left join StudyMeetings SMs on sms.StudyMeetingID = osp.StudyMeetingID
GROUP BY C.ClientID, C.FirstName, C.LastName;
```

ClientsTotalPayments [Judyta]

Wyświetla podsumowanie płatności za webinary, kursy i studia dla klientów

```
CREATE VIEW [dbo].[ClientsTotalPayments] AS
SELECT
C.ClientID,
C.FirstName,
C.LastName,
ISNULL(WP.TotalWebinarsPayment, 0) AS TotalWebinarsPayment,
ISNULL(SUM(ISNULL(CP.TotalPaymentAmount, 0)), 0) AS TotalCoursesPayment,
ISNULL(SUM(ISNULL(STP.TotalPayment, 0)), 0) AS TotalStudiesPayment,
ISNULL(WP.TotalWebinarsPayment, 0) + ISNULL(SUM(ISNULL(CP.TotalPaymentAmount, 0)),
0) + ISNULL(SUM(ISNULL(STP.TotalPayment, 0)), 0) AS TotalOverallPayment
FROM Clients C
LEFT JOIN (
SELECT
ClientID,
ISNULL(SUM(ISNULL(cwp.TotalWebinarsPayment, 0)), 0) AS TotalWebinarsPayment
FROM ClientsWebinarsPayments cwp
GROUP BY ClientID
) WP ON C.ClientID = WP.ClientID
LEFT JOIN (
SELECT
CP.ClientID,
COALESCE(SUM(CASE
WHEN CP.CoursePaymentTypeID = 1 THEN CC.AdvancePaymentPrice
WHEN CP.CoursePaymentTypeID = 2 THEN CC.Price
ELSE (CC.Price - CC.AdvancePaymentPrice)
END), 0) AS TotalPaymentAmount
FROM CoursesPayments CP
LEFT JOIN Courses CC ON CP.CourseID = CC.CourseID
GROUP BY CP.ClientID
) CP ON C.ClientID = CP.ClientID
LEFT JOIN (
SELECT
C.ClientID,
ISNULL(SUM(CASE
WHEN OSM.Access = 1 AND sms.PriceForOuterStudent IS NOT NULL THEN
sms.PriceForOuterStudent
```

```
ELSE 0
END), 0) AS TotalOuterStudentMeetingPayment,
ISNULL(SUM(R.Price), 0) AS TotalReunionPayment,
ISNULL(SUM(CASE
WHEN SP.PaymentTypeID = 1 THEN T.Price
WHEN SP.PaymentTypeID = 2 THEN S.EntryFee
ELSE 0
END), 0) AS TotalPayment
FROM Clients C
LEFT JOIN StudyMeetingMembers SMM ON C.ClientID = SMM.ClientID
LEFT JOIN OuterStudyMembers OSM ON C.ClientID = OSM.ClientID AND
SMM.StudiesMeetingID = OSM.StudyMeetingID
LEFT JOIN OuterStudyPayments OSP ON OSM.ClientID = OSP.ClientID AND
OSM.StudyMeetingID = OSP.StudyMeetingID
LEFT JOIN ReunionPayments RP ON C.ClientID = RP.ClientID
LEFT JOIN Reunions R ON RP.ReunionID = R.ReunionID
LEFT JOIN StudyPayments SP ON C.ClientID = SP.ClientID
LEFT JOIN StudyMembers SM ON SP.ClientID = SM.ClientID AND SP.StudyID = SM.StudiesID
LEFT JOIN Studies S ON SM.StudiesID = S.StudiesID
LEFT JOIN Term T ON S.StudiesID = T.StudiesID
Left join StudyMeetings SMs on sms.StudyMeetingID = osp.StudyMeetingID
GROUP BY C.ClientID
) STP ON C.ClientID = STP.ClientID
GROUP BY C.ClientID, C.FirstName, C.LastName, WP.TotalWebinarsPayment;
```

CourseProgress [Judyta]

Wyświetla informacje o zaliczeniu kursu przez studenta

```
CREATE VIEW [dbo].[CourseProgress] AS
SELECT
C.FirstName,
C.LastName,
CO.Name AS CourseName,
CASE
WHEN
CAST(COUNT(CASE WHEN MM.Attendence = 1 THEN 1 END) * 100.0 /
NULLIF(COUNT(MM.ClientID), 0) AS DECIMAL(5, 2)) >= 80.0
THEN
'Passed'
ELSE
'Failed'
END AS CourseStatus,
CASE
WHEN COUNT(MM.ClientID) > 0 THEN
CAST(COUNT(CASE WHEN MM.Attendence = 1 THEN 1 END) * 100.0 /
NULLIF(COUNT(MM.ClientID), 0) AS DECIMAL(5, 2))
ELSE
0.0 -- To handle division by zero scenario
END AS AttendancePercentage
FROM
Clients C
JOIN
CoursesMembers CM ON CM.ClientID = C.ClientID
JOIN
Courses CO ON CO.CourseID = CM.CourseID
LEFT JOIN
ModuleMeetingMembers MM ON C.ClientID = MM.ClientID
LEFT JOIN
ModuleMeeting M ON MM.ModuleMeetingID = M.ModuleMeetingID
GROUP BY
C.FirstName, C.LastName, CO.Name
```

CoursesAndModules [Judyta]

Wyświetla nazwy kursów i moduły do nich przypisane

```
CREATE VIEW [dbo].[CoursesAndModules] AS
SELECT
C.Name AS 'Course',
COALESCE(STRING_AGG('Module: ' + CONVERT(VARCHAR, M.StartDate, 120) + ' to ' +
CONVERT(VARCHAR, M.EndDate, 120), ', '), '') AS 'Modules'
FROM Courses C
LEFT JOIN Module M ON C.CourseID = M.CourseID
GROUP BY C.CourseID, C.Name;
```

CoursesMembersView [Judyta]

Wyświetla informacje o klientach zapisanych na kurs

```
CREATE VIEW [dbo].[CoursesMembersView] AS
SELECT
CO.Name AS 'Course',
COALESCE(STRING_AGG(C.FirstName + ' ' + C.LastName, ', '), '') AS 'Members'
FROM Courses CO
LEFT JOIN CoursesMembers CM ON CO.CourseID = CM.CourseID
LEFT JOIN Clients C ON CM.ClientID = C.ClientID
GROUP BY CO.CourseID, CO.Name;
```

CoursesRevenue [Judyta]

Wyświetla informacje o zarobkach ze sprzedanych kursów

```
CREATE VIEW [dbo].[CoursesRevenue] AS
SELECT
C.CourseID,
C.Name AS CourseName,
ISNULL(CM.EnrolledCount, 0) AS EnrolledStudents,
ISNULL(C.Price, 0) AS CoursePrice,
ROUND(ISNULL(TR.TotalCourseRevenue, 0), 2) AS TotalCourseRevenue
FROM
Courses C
LEFT JOIN (
SELECT CourseID, COUNT(DISTINCT ClientID) AS EnrolledCount
FROM CoursesMembers
GROUP BY CourseID
) AS CM ON C.CourseID = CM.CourseID
LEFT JOIN (
SELECT CM.CourseID, SUM(ISNULL(C.AdvancePaymentPrice, 0)) AS TotalCourseRevenue
FROM Courses C
LEFT JOIN CoursesMembers CM ON C.CourseID = CM.CourseID
GROUP BY CM.CourseID
) AS TR ON C.CourseID = TR.CourseID;
```

EmployeeAddress [Judyta]

Wyświetla informacje o pełnym adresie pracowników

```
CREATE VIEW [dbo].[EmployeeAddress] AS
SELECT
E.FirstName + ' ' + E.LastName AS 'FirstnameAndLastname',
E.Phone,
E.Email,
CONCAT(A.Address, ', ', Ci.Name, ', ', Co.Name) AS 'FullAddress'
FROM Employees E
JOIN Addresses A ON E.AddressID = A.AddressID
JOIN Cities Ci ON A.CityID = Ci.CityID
JOIN Countries Co ON Ci.CountryID = Co.CountryID;
```

FieldOfStudy_Syllabus [Judyta]

Wyświetla informacje o linkach do sylabusa dla kierunków studiów

```
CREATE VIEW [dbo].[FieldOfStudy_Syllabus] AS
SELECT
S.Name AS 'FieldOfStudy',
SY.Link AS 'SyllabusLink'
FROM Studies S
LEFT JOIN Syllabus SY ON S.StudiesID = SY.StudiesID;
```

FieldsOfStudy_Students [Judyta]

Wyświetla informacje o kierunkach studiów i studentów studiujących na tych kierunkach

```
CREATE VIEW [dbo].[FielsOfStudy_Students] AS
SELECT S.Name AS 'FieldOfStudy',
STRING_AGG(C.FirstName + ' ' + C.LastName, ', ') AS 'Students'
FROM Studies S
INNER JOIN StudyMembers SM ON S.StudiesID = SM.StudiesID
INNER JOIN Clients C ON SM.ClientID = C.ClientID
GROUP BY S.Name;
```

FutureEventRegistrations [Judyta]

Wyświetla informacje o przyszłych wydarzeniach z informacją czy online czy offline i z informacją o zapisanych klientach na dane wydarzenie

```
CREATE VIEW [dbo].[FutureEventRegistrations] AS
SELECT
'StudyMeeting' AS EventType,
SM.Date AS EventDate,
CASE
WHEN SM.ClassTypeID = 2 THEN 'Online Asynchronous'
WHEN SM.ClassTypeID = 3 THEN 'Online Synchronous'
WHEN SM.ClassTypeID = 4 THEN 'Offline'
ELSE 'Unknown'
END AS EventFormat,
STRING_AGG(C.FirstName + ' ' + C.LastName, ', ') AS RegisteredParticipants
FROM StudyMeetings SM
LEFT JOIN StudyMeetingMembers SMM ON SM.StudyMeetingID = SMM.StudiesMeetingID
LEFT JOIN Clients C ON SMM.ClientID = C.ClientID
WHERE SM.Date >= GETDATE()
GROUP BY SM.StudyMeetingID, SM.Date, SM.StartTime, SM.EndTime, SM.ClassTypeID

UNION ALL

SELECT
'ModuleMeeting' AS EventType,
CONVERT(DATE, MM.StartTime) AS EventDate,
CASE
WHEN MM.ClassTypeID = 2 THEN 'Online Asynchronous'
WHEN MM.ClassTypeID = 3 THEN 'Online Synchronous'
WHEN MM.ClassTypeID = 4 THEN 'Offline'
ELSE 'Unknown'
END AS EventFormat,
STRING_AGG(C.FirstName + ' ' + C.LastName, ', ') AS RegisteredParticipants
FROM ModuleMeeting MM
LEFT JOIN ModuleMeetingMembers MMM ON MM.ModuleMeetingID = MMM.ModuleMeetingID
LEFT JOIN Clients C ON MMM.ClientID = C.ClientID
WHERE CONVERT(DATE, MM.StartTime) >= GETDATE()
GROUP BY MM.ModuleMeetingID, CONVERT(DATE, MM.StartTime), CONVERT(TIME,
MM.StartTime), CONVERT(TIME, MM.EndTime), MM.ClassTypeID
```

```
UNION ALL

SELECT
'Webinar' AS EventType,
WM.StartDate AS EventDate,
'Online' AS EventFormat,
STRING_AGG(C.FirstName + ' ' + C.LastName, ', ') AS RegisteredParticipants
FROM WebinarMeetings WM
LEFT JOIN WebinarMembers WMm ON WM.WebinarMeetingID = WMm.WebinarMeetingID
LEFT JOIN Clients C ON WMm.ClientID = C.ClientID
WHERE WM.StartDate >= GETDATE()
GROUP BY WM.WebinarMeetingID, WM.StartDate, CONVERT(TIME, WM.StartDate),
CONVERT(TIME, WM.EndDate)
```

OverallFinancialSummary [Judyta]

Wyświetla podsumowanie zarobków za kursy, webinary i studia

```
CREATE VIEW [dbo].[OverallFinancialSummary] AS
SELECT
'Courses' AS ClassType,
SUM(CR.TotalCourseRevenue) AS TotalRevenue
FROM
CoursesRevenue CR

UNION ALL

SELECT
'Studies' AS ClassType,
SUM(SR.TotalOverallRevenue) AS TotalRevenue
FROM
StudiesRevenue SR

UNION ALL

SELECT
'Webinars' AS ClassType,
SUM(WFS.TotalRevenue) AS TotalRevenue
FROM
WebinarFinancialSummary WFS;
```

OverlappingMeetingsView [Judyta]

Wyświetla informacje o pokrywających się ze sobą zajęciach

```
CREATE VIEW [dbo].[OverlappingMeetingsView] AS
WITH OverlappingMeetings AS (
SELECT
'Webinar' AS MeetingType,
CONVERT(DATETIME, WM.StartDate) AS StartDateTime,
CONVERT(DATETIME, WM.EndDate) AS EndDateTime,
WM.WebinarMeetingID AS MeetingID
FROM
WebinarMeetings WM
UNION ALL
SELECT
'Study' AS MeetingType,
CAST(SM.Date AS DATETIME) + CAST(SM.StartTime AS DATETIME) AS StartDateTime,
CAST(SM.Date AS DATETIME) + CAST(SM.EndTime AS DATETIME) AS EndDateTime,
SM.StudyMeetingID AS MeetingID
FROM
StudyMeetings SM
UNION ALL
SELECT
'Module' AS MeetingType,
MM.StartTime AS StartDateTime,
MM.EndTime AS EndDateTime,
MM.ModuleMeetingID AS MeetingID
FROM
ModuleMeeting MM
)
SELECT
OM1.MeetingType AS MeetingType1,
OM1.MeetingID AS MeetingID1,
OM1.StartDateTime AS StartDateTime1,
OM1.EndDateTime AS EndDateTime1,
OM2.MeetingType AS MeetingType2,
OM2.MeetingID AS MeetingID2,
OM2.StartDateTime AS StartDateTime2,
OM2.EndDateTime AS EndDateTime2
FROM
OverlappingMeetings OM1
INNER JOIN
OverlappingMeetings OM2 ON OM1.MeetingID < OM2.MeetingID
AND OM1.StartDateTime < OM2.EndDateTime
AND OM1.EndDateTime > OM2.StartDateTime;
```

StudiesAndTermsAndSubjects [Judyta]

Wyświetla informacje o studiach, semestrach na studiach, oraz przedmiotach na tych semestrach

```
CREATE VIEW [dbo].[StudiesAndTermsAndSubjects] AS
SELECT
S.Name AS 'FieldOfStudy',
T.Price AS 'PriceForSemester',
STRING_AGG(Sub.Name, ', ') AS 'Subjects'
FROM Studies S
INNER JOIN Term T ON S.StudiesID = T.StudiesID
INNER JOIN Subjects Sub ON T.TermID = Sub.TermID
GROUP BY S.Name, T.TermID, T.Price;
```

StudiesRevenue [Judyta]

Wyświetla informacje o przychodach ze studiów

```
CREATE VIEW [dbo].[StudiesRevenue] AS
SELECT DISTINCT
Studies.StudiesID,
Studies.Name AS StudyName,
EnrolledStudents.EnrolledCount AS EnrolledStudents,
Round(ISNULL(TotalRevenue.TotalStudentRevenue, 0),2) AS TotalStudentRevenue,
Round(ISNULL(RevenueFromOuter.RevenueFromOuterStudents, 0),2) AS
RevenueFromOuterStudents,
Round(ISNULL(TotalRevenue.TotalStudentRevenue, 0) +
ISNULL(RevenueFromOuter.RevenueFromOuterStudents, 0),2) AS TotalOverallRevenue
FROM
Studies
LEFT JOIN
Term t ON Studies.StudiesID = T.StudiesID
LEFT JOIN (
SELECT StudiesID, COUNT(ClientID) AS EnrolledCount
FROM StudyMembers
GROUP BY StudiesID
) AS EnrolledStudents ON Studies.StudiesID = EnrolledStudents.StudiesID
LEFT JOIN (
SELECT SubjectID, SUM(ISNULL(PriceForOuterStudent, 0)) AS RevenueFromOuterStudents
FROM StudyMeetings
JOIN OuterStudyMembers ON StudyMeetings.StudyMeetingID =
OuterStudyMembers.StudyMeetingID
GROUP BY SubjectID
) AS RevenueFromOuter ON Studies.StudiesID = RevenueFromOuter.SubjectID
LEFT JOIN (
SELECT StudyMembers.StudiesID, SUM(ISNULL(Studies.EntryFee, 0) + ISNULL(Term.Price,
0)) AS TotalStudentRevenue
FROM Studies
LEFT JOIN Term ON Studies.StudiesID = Term.StudiesID
LEFT JOIN StudyMembers ON Studies.StudiesID = StudyMembers.StudiesID
GROUP BY StudyMembers.StudiesID
) AS TotalRevenue ON Studies.StudiesID = TotalRevenue.StudiesID;
```

WebinarFinancialSummary [Judyta]

Wyświetla informacje o przychodach z webinarów

```
CREATE VIEW [dbo].[WebinarFinancialSummary] AS
SELECT
W.Name AS WebinarName,
COALESCE(COUNT(WB.ClientID), 0) AS ParticipantsCount,
SUM(W.Price * COALESCE(WB.QuantityPurchased, 0)) AS TotalRevenue
FROM
Webinars W
INNER JOIN
WebinarMeetings WM ON W.WebinarID = WM.WebinarID
LEFT JOIN
(
SELECT
WM.WebinarMeetingID,
WM.ClientID,
COUNT(WM.ClientID) AS QuantityPurchased
FROM
WebinarMembers WM
GROUP BY
WM.WebinarMeetingID, WM.ClientID
) WB ON WM.WebinarMeetingID = WB.WebinarMeetingID
GROUP BY
W.Name;
GO
```

WebinarMeetingView [Judyta]

Wyświetla informacje o spotkaniach w webinarze, wraz z jego nazwą

```
CREATE VIEW [dbo].[WebinarMeetingView] AS
SELECT
W.Name AS 'Webinar',
WM.ClassLink,
WM.VideoLink,
WM.LinkTermination,
WM.StartDate AS 'StartDate',
WM.EndDate AS 'EndDate'
FROM Webinars W
INNER JOIN WebinarMeetings WM ON W.WebinarID = WM.WebinarID;
```

WebinarsMembersView [Judyta]

Wyświetla informacje o osobach zapisanych na webinar

```
CREATE VIEW [dbo].[WebinarsMembersView] AS
SELECT
W.Name AS 'Webinar',
COALESCE(STRING_AGG(C.FirstName + ' ' + C.LastName, ', '), '') AS 'Members'
FROM Webinars W
LEFT JOIN WebinarMembers WM ON W.WebinarID = WM.WebinarMeetingID
LEFT JOIN Clients C ON WM.ClientID = C.ClientID
GROUP BY W.Name;
```

StudentsInterships[Judyta]

Pokazuje studentów i praktyki

```
CREATE VIEW [dbo].[StudentsInterships] AS
SELECT
C.ClientID,
C.FirstName,
C.LastName,
I.IntershipID,
T.TermNumber,
I.StartDate,
I.EndDate,
I.Pass
from Clients C
join Interships I on I.ClientID = C.ClientID
join Term T on I.TermID = T.TermID
```

UnavailableRooms[Anastazja]

Pokazuje zajęte sale

```
CREATE VIEW [dbo].[UnavailableRooms]
AS
SELECT dbo.RoomStudyMeetings.RoomID, dbo.Rooms.Room, dbo.Rooms.Floor,
dbo.Rooms.PersonLimit, 'StudyMeeting' AS MeetingType,
dbo.StudyMeetings.StudyMeetingID AS ID, dbo.StudyMeetings.StartTime as StartTime
```

```
FROM dbo.Rooms
INNER JOIN
dbo.RoomStudyMeetings ON dbo.Rooms.RoomID = dbo.RoomStudyMeetings.RoomID
INNER JOIN
dbo.StudyMeetings ON dbo.RoomStudyMeetings.StudyMeetingID =
dbo.StudyMeetings.StudyMeetingID

UNION

SELECT dbo.Rooms.RoomID, dbo.Rooms.Room, dbo.Rooms.Floor, dbo.Rooms.PersonLimit,
'ModuleMeeting' AS MeetingType, dbo.ModuleMeeting.ModuleMeetingID AS ID,
dbo.ModuleMeeting.StartTime as StartTime
FROM dbo.Rooms INNER JOIN
dbo.RoomModuleMeetings ON dbo.Rooms.RoomID = dbo.RoomModuleMeetings.RoomID
INNER JOIN
dbo.ModuleMeeting ON dbo.RoomModuleMeetings.ModuleMeetingID =
dbo.ModuleMeeting.ModuleMeetingID
```

SubjectProgressView [Monika]

Wyświetla informacje o osobach zapisanych dany przedmiot oraz procent obecności

```
CREATE VIEW [dbo].[SubjectProgressView] AS
select C.ClientID,
S.StudiesID,
C.FirstName,
C.LastName,
S.Name StudyName,
       CASE
       WHEN
              CAST(COUNT(CASE WHEN SMM.Attendence = 1 THEN 1 END) * 100.0 /
NULLIF(COUNT(SMM.ClientID), 0) AS DECIMAL(5, 2)) >= 80.0
       THEN
              'Passed'
       ELSE
              'Failed'
    END AS CourseStatus,
    CASE
        WHEN COUNT(SMM.ClientID) > 0 THEN
```

```
            CAST(COUNT(CASE WHEN SMM.Attendence = 1 THEN 1 END) * 100.0 /
NULLIF(COUNT(SMM.ClientID), 0) AS DECIMAL(5, 2))
        ELSE
            0.0 -- To handle division by zero scenario
    END AS AttendancePercentage

FROM Clients C
JOIN StudyMembers SM ON SM.ClientID = C.ClientID
JOIN Studies S ON S.StudiesID = SM.StudiesID
LEFT JOIN StudyMeetingMembers SMM ON C.ClientID = SMM.ClientID

GROUP BY
      C.ClientID, S.StudiesID, FirstName, C.LastName, S.Name
```

StudyCertificatesView [Monika]

Wyświetla informacje o osobach, które ukończyły studia (zdały wszystkie przedmioty i egzamin końcowy) + info o semestrach i przedmiotach

```
create view [dbo].[StudyCertificates] as
select
sm.ClientID,
c.FirstName,
c.LastName,
s.Name as StudyName,
t.TermNumber,
su.Name as Subject

from Clients c
join StudyMembers sm on sm.ClientID = c.ClientID
join Studies s on s.StudiesID = sm.StudiesID
join Term t on s.StudiesID = t.StudiesID
join Subjects su on su.TermID = t.TermID
join SubjectMembers subm on subm.ClientID = sm.ClientID
join Interships i on i.ClientID = sm.ClientID

where sm.PassExam = 1 and subm.Pass = 1 and i.Pass = 1
```

ModuleAndModuleMeetingView[Monika]

Wyświetla informacje modułach i spotkaniach

```
CREATE VIEW [dbo].[ModulesAndModuleMeetingsView]
AS
SELECT Module.ModuleID, Module.StartDate, Module.EndDate,
ModuleMeeting.ModuleMeetingID, ModuleMeeting.LecturerID,
```

```
    ModuleMeeting.StartTime, ModuleMeeting.EndTime, ModuleMeeting.ClassTypeID,
ModuleMeeting.OriginalLanguageID
FROM Module
LEFT JOIN ModuleMeeting ON Module.ModuleID = ModuleMeeting.ModuleID
```

WebinarAndWebinarMeetingsView[Monika]

Wyświetla informacje Webinarach i ich spotkaniach

```
CREATE or alter VIEW [dbo].[WebinarsAndWebinarMeetings]
AS
SELECT dbo.Webinars.WebinarID, dbo.Webinars.Name,
dbo.WebinarMeetings.WebinarMeetingID,
dbo.WebinarMeetings.StartDate, dbo.WebinarMeetings.EndDate
FROM dbo.WebinarMeetings right JOIN
dbo.Webinars ON dbo.WebinarMeetings.WebinarID = dbo.Webinars.WebinarID
```

PROCEDURY:

addCourseAndModule[Monika]

Dodaje kurs i moduły

```
ALTER   procedure [dbo].[addCourseAndModule]
@Name varchar(50),
@Description varchar(100),
@Price decimal(16,2),
@AdvancePaymentPrice decimal(16,2),
@json VARCHAR(MAX)
as
    IF ISJSON(@json) > 0
    begin
        --add course
        insert into Courses
        values(@Name, @Description, @Price, @AdvancePaymentPrice)
        print 'course added'

        --select id of this course
        DECLARE @CourseID INT;
        SET @CourseID = SCOPE_IDENTITY();

        --create temp table modules
        DECLARE @modules TABLE (CourseID INT, StartDate DATETIME, EndDate datetime,
ClassTypeID int);
        INSERT INTO @modules (CourseID, StartDate, EndDate, ClassTypeID)
        SELECT @CourseID,
        json_value(value, '$.StartDate'),
        json_value(value, '$.EndDate'),
        json_value(value, '$.ClassTypeID')
        FROM OPENJSON(@json, '$.modules');

        -- insert temp table to Module table
        INSERT INTO Module (CourseID, StartDate, EndDate, ClassTypeID)
        SELECT @CourseID, StartDate, EndDate, ClassTypeID
        FROM @modules;

        select * from module
    end
    else
    begin
        print 'invalid jason'
    end
```

addCourseMemberProcedure [Monika]

Dodaje uczestnika kursu(wszystkie moduły) i tworzy płatności do zapłacenia

```
ALTER procedure [dbo].[addCourseMemberProcedure]
@ClientID int,
@CourseID int
as
declare @CourseStartDate datetime

INSERT INTO CoursesMembers(ClientID, CourseID, Access)
VALUES (@ClientID, @CourseID, 0);

INSERT INTO ModuleMembers(ClientID, ModuleID, PassModule) -- dodaj do wszystkich
moduły
select distinct @ClientID, ModuleID, 0
from Module m
join Courses c on @CourseID = m.CourseID

INSERT INTO ModuleMeetingMembers(ClientID, ModuleMeetingID, Attendence)  -- do
wszystkich meetingów
select distinct @ClientID, ModuleMeetingID, 0
from ModuleMeeting mm
join Module m on m.ModuleID = mm.ModuleID
join Courses c on @CourseID = m.CourseID

INSERT INTO CoursesPayments(CourseID, ClientID, PayDate, Link, CoursePaymentTypeID)
VALUES (@CourseID, @ClientID, GETDATE(), 'www.paylink.com/advance/'+ CAST(@ClientID
AS VARCHAR(50)), 1);

-- get first meeting date
select @CourseStartDate = min(StartDate)
from Module
where CourseID = @CourseID

INSERT INTO CoursesPayments(CourseID, ClientID, PayDate, Link, CoursePaymentTypeID)
select distinct @CourseID, @ClientID, @CourseStartDate,
'www.paylink.com/remaining/'+ CAST(@ClientID AS VARCHAR(50)), 2
from Module
```

addModuleMeeting [Judyta]

Dodaje spotkanie w ramach modułu (kurs)

```
ALTER   procedure [dbo].[addModuleMeeting]
@ModuleID int,
@LecturerID int,
```

```
@StartTime datetime,
@EndTime datetime,
@Price decimal(16, 2),
@ClassTypeID int,
@OriginalLanguageID int
as

insert into ModuleMeeting(ModuleID, LecturerID, StartTime, EndTime, ClassTypeID,
OriginalLanguageID)
values(@ModuleID, @LecturerID, @StartTime, @EndTime, @ClassTypeID,
@OriginalLanguageID)
```

addOuterStudentMemberProcedure [Monika]

Dodaje studenta z zewnątrz na dane spotkanie i płatność

```
ALTER   procedure [dbo].[addOuterStudentMemberProcedure]
@ClientID int,
@StudyMeetingID int
as

insert Into OuterStudyMembers(ClientID, StudyMeetingID)
values(@ClientID, @StudyMeetingID)

insert into OuterStudyPayments(ClientID, PayDate, StudyMeetingID, Link)
select distinct @ClientID, DATEADD(day, -3, Date), @StudyMeetingID,
'www.paylink.com/remaining/'+ CAST(@ClientID AS VARCHAR(50))
from StudyMeetings sm
where @StudyMeetingID = StudyMeetingID
```

addReunionAndStudyMeeting[Monika]

Dodaje zjazd i spotkania dla studiów

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE   procedure [dbo].[addReunionAndStudyMeeting]
@StartDate datetime,
@EndDate datetime,
@TermID int,
@Price decimal(16, 2),
```

```
@json varchar(max)
as
DECLARE @ReunionID INT;

--add reunion
insert into Reunions(StartDate, EndDate, TermID, Price)
OUTPUT INSERTED.ReunionID
values(@StartDate, @EndDate, @TermID, @Price)


-- Pobierasz identyfikator reunion
SELECT @ReunionID AS ReunionID;


--add study meetings
INSERT INTO StudyMeetings (Date, StartTime, EndTime, SubjectID,
LecturerID, ClassTypeID, OriginalLanguageID,
    PriceForOuterStudent, OuterStudentLimit, StudentLimit, ReunionID)
SELECT
    json_value(value, '$.Date'),
    json_value(value, '$.StartTime'),
    json_value(value, '$.EndTime'),
    json_value(value, '$.SubjectID'),
    json_value(value, '$.LecturerID'),
    json_value(value, '$.ClassTypeID'),
    json_value(value, '$.OriginalLanguageID'),
    json_value(value, '$.PriceForOuterStudent'),
    json_value(value, '$.OuterStudentLimit'),
    json_value(value, '$.StudentLimit'),
    @ReunionID
FROM OPENJSON(@json, '$.meetings');
```

addStudyAndTermsAndSubjects[Monika]

Dodaje studia, semestry i przedmioty

```
CREATE   procedure [dbo].[addStudyAndTermsAndSubjects]
@PersonLimit int,
@Name varchar(50),
@Description varchar(50),
@EntryFee decimal(16,2),
@json varchar(MAX)
AS
```

```
IF ISJSON(@json) > 0
begin
    --add study
    INSERT INTO Studies
    VALUES(@PersonLimit, @Name, @Description, @EntryFee);
    PRINT 'study added';

    --get study id
    DECLARE @StudyID INT;
    SET @StudyID = SCOPE_IDENTITY();

    --create temp table terms
    DECLARE @terms TABLE (StudyID INT, TermNumber INT);
    INSERT INTO @terms (StudyID, TermNumber)
    SELECT @StudyID, json_value(value, '$.TermNumber')
    FROM OPENJSON(@json, '$.terms');

    -- insert terp table to Term table
    INSERT INTO Term (StudiesID, TermNumber)
    SELECT StudyID, TermNumber
    FROM @terms;


    DECLARE @TermNumber INT;
    DECLARE cur CURSOR FOR
    SELECT TermNumber
    FROM @terms;

    OPEN cur;
    FETCH NEXT FROM cur INTO @TermNumber;
    WHILE @@FETCH_STATUS = 0
    BEGIN

        --get term id
        DECLARE @TermID INT;
        SELECT @TermID = TermID
        FROM Term
        WHERE StudiesID = @StudyID AND TermNumber = @TermNumber;

        --insert subjects
        INSERT INTO Subjects(Name, TermID, LecturerID)
        SELECT
            Name,
```

```
                @TermID,
                LecturerID
            FROM OPENJSON(@json, '$.terms') WITH (
                subjects NVARCHAR(MAX) '$.subjects' AS JSON,
                TermNumber INT '$.TermNumber'
            ) AS termsData
            CROSS APPLY OPENJSON(termsData.subjects) WITH (
                Name varchar(50) '$.Name',
                LecturerID INT '$.LecturerID'
            ) AS data
            WHERE termsData.TermNumber = @TermNumber;

        FETCH NEXT FROM cur INTO @TermNumber;
        END;
        CLOSE cur;
        DEALLOCATE cur;


        select * from Term
        select * from Subjects
    end
    else
    begin
        print 'invalid jason'
    end
GO
```

addStudyMemberProcedure [Monika]

Dodaje studenta do studiów, zapisuje na wszystkie spotkania wszystkich przedmiotów w ramach studiów i dodaje płatności do zapłacenia

```
ALTER   procedure [dbo].[addStudyMemberProcedure]
@ClientID int,
@StudiesID int
as
declare @ExamID int,
@StudyMeetingID int

select @ExamID = ExamID
from Exams
where StudiesID = @StudiesID

INSERT INTO StudyMembers(ClientID, StudiesID)
```

```
VALUES (@ClientID, @StudiesID);

INSERT INTO SubjectMembers(ClientID, SubjectID)
select distinct @ClientID, SubjectID
from Studies s
join Term t on t.StudiesID = s.StudiesID
join Subjects su on t.TermID = su.TermID

-- get studyMeetingID
select distinct @StudyMeetingID = StudyMeetingID
from Studies s
join Term t on t.StudiesID = s.StudiesID
join Subjects su on t.TermID = su.TermID
join StudyMeetings sm on sm.SubjectID = su.SubjectID

INSERT INTO StudyMeetingMembers(ClientID, StudiesMeetingID)
select distinct @ClientID, @StudyMeetingID
from Studies s

insert into StudyPayments(ClientID, PayDate, StudyID, Link)
values( @ClientID, GETDATE(), @StudiesID, CONCAT('https:/link_to_pay/entryfee/',
CONVERT(varchar(50), @ClientID),'/',CONVERT(varchar(50), @StudiesID)))

insert into ReunionPayments(ClientID, PayDate, StudiesID, ReunionID, StudyMeetingID,
Link)
select distinct @ClientID, DATEADD(DAY, -3, StartDate), @StudiesID, ReunionID,
@StudyMeetingID,
CONCAT('https:/link_to_pay/reunion/', CONVERT(varchar(50), @ClientID),
'/',CONVERT(varchar(50), ReunionID))
from Reunions
```

addWebinarAndWebinarMeeting

Dodaje webinar i spotkania webinarowe

```
CREATE   procedure [dbo].[addWebinarAndWebinarMeeting]
@Name varchar (50),
@Description varchar(50),
@Price decimal(16,2),
@json varchar(MAX)
as
if ISJSON(@json) > 0
begin
    --add webinar
    INSERT INTO Webinars (Name, Description, Price)
```

```
    VALUES (@Name, @Description, @Price);

    --select id of this webinar
    DECLARE @WebinarID INT;
    SET @WebinarID = SCOPE_IDENTITY();

    --add webinar meetings
    INSERT INTO WebinarMeetings (ClassLink, VideoLink, LinkTermination,
OriginalLanguageID,
    LecturerID, StartDate, EndDate, WebinarID)
    SELECT
    json_value(value, '$.ClassLink'),
    json_value(value, '$.VideoLink'),
    json_value(value, '$.LinkTermination'),
    json_value(value, '$.OriginalLanguageID'),
    json_value(value, '$.LecturerID'),
    json_value(value, '$.StartDate'),
    json_value(value, '$.EndDate'),
    @WebinarID
    FROM OPENJSON(@json, '$.meetings');

    select * from Webinars
    select * from WebinarMeetings
end
else
begin
print 'json is not valid'
end
```

addWebinarMemberProcedure [Monika]

Dodaje klienta do webinaru i dodaje płatność

```
ALTER   PROCEDURE [dbo].[addWebinarMemberProcedure]
@WebinarMeetingID int,
@ClientID int
AS
declare @WebinarID int,
@Price int,
@MeetingDate datetime

--add new member
INSERT INTO WebinarMembers(WebinarMeetingID, ClientID)
VALUES (@WebinarMeetingID, @ClientID);
```

```
--select webinarID and meeting date
select @WebinarID = WebinarID, @MeetingDate = StartDate
from WebinarMeetings
where WebinarMeetingID = @WebinarMeetingID

-- select webinar price
select @Price = Price from Webinars
where WebinarID = @WebinarID

-- add new payment or give access
if @Price = 0
    begin
    UPDATE WebinarMembers
    SET Access = 1
    WHERE ClientID = @ClientID and WebinarMeetingID = @WebinarMeetingID
    end
else
    begin
    INSERT INTO WebinarPayments(WebinarMeetingID, ClientID, PayDate, Link)
    VALUES (@WebinarMeetingID, @ClientID, @MeetingDate, 'https://pay_link.com/' +
CAST(@ClientID AS VARCHAR(50)));
    end
```

changeLectureModuleMeeting[Anastazja]

zmienia prowadzącego w spotkaniu kursie spotkaniu

```
    CREATE PROCEDURE [dbo].[ChangeLecturerModuleMeeting]
    @ModuleMeetingID INT,
    @NewLecturerID INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @StartTime datetime, @EndTime datetime;


    SELECT @StartTime = StartTime, @EndTime = EndTime
    FROM ModuleMeeting
    WHERE ModuleMeetingID = @ModuleMeetingID;


    IF EXISTS (
        SELECT 1
        FROM StudyMeetings
```

```
        WHERE LecturerID = @NewLecturerID
              AND CONVERT(datetime, StartTime) <= @EndTime
              AND CONVERT(datetime,EndTime) >= @StartTime
    )
    BEGIN
        RAISERROR('Prowadzący prowadzi zajęcia na studiach w
określonym czasie', 16, 1);
        RETURN;
    END;


    IF EXISTS (
        SELECT 1
        FROM WebinarMeetings
        WHERE LecturerID = @NewLecturerID
              AND StartDate < @EndTime
              AND EndDate > @StartTime
    )
    BEGIN
        RAISERROR('Prowadzący prowadzi webinar w określonym
czasie', 16, 1);
        RETURN;
    END;

    IF EXISTS (
    SELECT 1
    FROM ModuleMeeting
    WHERE LecturerID = @NewLecturerID
          AND StartTime < @EndTime
          AND EndTime > @StartTime
)
BEGIN
    RAISERROR('Prowadzący prowadzi moduł kursu w określonym
czasie', 16, 1);
    RETURN;
END;

    UPDATE ModuleMeeting
    SET LecturerID = @NewLecturerID
    WHERE ModuleMeetingID = @ModuleMeetingID;

    PRINT 'Zmieniono prowadzącego w ModuleMeeting.';
END;
```

changeLecturerStudyMeeting[Anastazja]

zmienia prowadzącego na spotkaniu na studiach

```
CREATE PROCEDURE [dbo].[ChangeLecturerStudyMeeting]
    @StudyMeetingID INT,
    @NewLecturerID INT
AS
BEGIN


    DECLARE @StartTime datetime, @EndTime datetime;


    SELECT @StartTime = StartTime, @EndTime = EndTime
    FROM StudyMeetings
    WHERE StudyMeetingID = @StudyMeetingID;


    IF EXISTS (
        SELECT 1
        FROM StudyMeetings
        WHERE LecturerID = @NewLecturerID
              AND CONVERT(datetime, StartTime) <= @EndTime
              AND CONVERT(datetime,EndTime) >= @StartTime
    )
    BEGIN
        RAISERROR('Prowadzący prowadzi zajęcia na studiach w określonym
czasie', 16, 1);
        RETURN;
    END;


    IF EXISTS (
        SELECT 1
        FROM WebinarMeetings
        WHERE LecturerID = @NewLecturerID
              AND StartDate < @EndTime
              AND EndDate > @StartTime
    )
    BEGIN
        RAISERROR('Prowadzący prowadzi webinar w określonym czasie', 16,
1);
        RETURN;
```

```
    END;

    IF EXISTS (
    SELECT 1
    FROM ModuleMeeting
    WHERE LecturerID = @NewLecturerID
          AND StartTime < @EndTime
          AND EndTime > @StartTime
)
BEGIN
    RAISERROR('Prowadzący prowadzi moduł kursu w określonym czasie', 16,
1);
    RETURN;
END;

    UPDATE StudyMeetings
    SET LecturerID = @NewLecturerID
    WHERE StudyMeetingID = @StudyMeetingID;

    PRINT 'Zmieniono prowadzącego w WebinarMeeting.';
END;
```

changeLecturerWebinar[Anastazja]

zmienia prowadzącego na webinarze

```
CREATE PROCEDURE [dbo].[ChangeLecturerWebinarMeeting]
    @WebinarMeetingID INT,
    @NewLecturerID INT
AS
BEGIN

    DECLARE @StartTime datetime, @EndTime datetime;


    SELECT @StartTime = StartDate, @EndTime = EndDate
    FROM WebinarMeetings
    WHERE WebinarMeetingID = @WebinarMeetingID;


    IF EXISTS (
        SELECT 1
        FROM StudyMeetings
        WHERE LecturerID = @NewLecturerID
```

```
            AND CONVERT(datetime, StartTime) <= @EndTime
            AND CONVERT(datetime,EndTime) >= @StartTime
    )
    BEGIN
        RAISERROR('Prowadzący prowadzi zajęcia na studiach w określonym
czasie', 16, 1);
        RETURN;
    END;


    IF EXISTS (
        SELECT 1
        FROM WebinarMeetings
        WHERE LecturerID = @NewLecturerID
            AND StartDate < @EndTime
            AND EndDate > @StartTime
    )
    BEGIN
        RAISERROR('Prowadzący prowadzi webinar w określonym czasie', 16,
1);
        RETURN;
    END;

    IF EXISTS (
    SELECT 1
    FROM ModuleMeeting
    WHERE LecturerID = @NewLecturerID
        AND StartTime < @EndTime
        AND EndTime > @StartTime
)
BEGIN
    RAISERROR('Prowadzący prowadzi moduł kursu w określonym czasie', 16,
1);
    RETURN;
END;

    UPDATE WebinarMeetings
    SET LecturerID = @NewLecturerID
    WHERE WebinarMeetingID = @WebinarMeetingID;

    PRINT 'Zmieniono prowadzącego w WebinarMeeting.';
END;
```

payEntryFeeForStudyProcedure [Monika]

Zmienia status płatności na zapłacony

```
ALTER   procedure [dbo].[payEntryForStudyProcedure]
@ClientID int,
@StudyID int
as
update StudyPayments
set Success = 1, PaymentDate = GETDATE()
where ClientID = @ClientID and StudyID = @StudyID
```

payForCourseProcedure [Judyta]

Zmienia status płatności na zapłacony

```
ALTER   procedure [dbo].[payForCourseProcedure]
@CoursesPaymentsID int
as

-- change payment status
update CoursesPayments
set Success = 1
where CoursesPaymentsID = @CoursesPaymentsID
```

payForOuterStudentPaymentProcedure [Monika]

Zmienia status płatności na zapłacony, dodanie dostępu studentowi do zajęć

```
ALTER   procedure [dbo].[payForOuterStudetnPaymentProcedure]
@ClientID int,
@StudyMeetingID int
as

update OuterStudyPayments
set Success = 1, PaymentDate = GETDATE()
where ClientID = @ClientID and StudyMeetingID = @StudyMeetingID

update OuterStudyMembers
set Access = 1
where ClientID = @ClientID and @StudyMeetingID = @StudyMeetingID
```

payForStudyReunionPaymentProcedure [Monika]

Zmienia status płatności na zapłacony, dodanie dostępu do zjazdu

```
ALTER  procedure [dbo].[payForStudyReunionPaymentProcedure]
@ClientID int,
@ReunionID int
```

```
as
declare @StudyMeetingID int,
@StudyID int,
@Success int

select @StudyID = StudiesID
from ReunionPayments
where ClientID = @ClientID and ReunionID = @ReunionID

update ReunionPayments
set Success = 1, PaymentDate = GETDATE()
where ClientID = @ClientID and ReunionID = @ReunionID

select @StudyMeetingID = StudyMeetingID
from StudyMeetings
where ReunionID = @ReunionID

select Success
from StudyPayments
where ClientID = @ClientID and StudyID = @StudyID

if @Success = 1
    begin
    update StudyMeetingMembers
    set ReunionAccess = 1
    where ClientID = @ClientID and StudiesMeetingID = @StudyMeetingID
end
```

payForWebinarProcedure [Monika]

Zmienia status płatności na zapłacony, dodaje dostęp, ustawia czas dostępu do nagrania (w zależności od daty zapłacenia przed lub po wydarzeniu)

```
ALTER PROCEDURE [dbo].[payForWebinarProcedure]
   @WebinarPaymentsID INT
AS
   DECLARE @ClientID int,
   @WebinarMeetingID int,
   @WebinarDate datetime,
   @PayedDate datetime

   --get ClientID and WebinarPaymentsID
   SELECT @ClientID = ClientID, @WebinarMeetingID = WebinarMeetingID
   FROM WebinarPayments
   WHERE WebinarPaymentsID = @WebinarPaymentsID;
```

```
-- change payment status
update WebinarPayments
set Success = 1
where @WebinarPaymentsID = WebinarPaymentsID

--select payed date
select @PayedDate=PayedDate
from WebinarPayments

--select webinar date
select @WebinarDate = StartDate
from WebinarMeetings

--set termination date
if @PayedDate < @WebinarDate
    begin
    update WebinarMembers
    set LinkTermination = DATEADD(DAY, 30, @WebinarDate)
    where WebinarMeetingID = @WebinarMeetingID and ClientID = @ClientID
    end
else
begin
update WebinarMembers
set LinkTermination = DATEADD(Day, 30, @PayedDate)
where WebinarMeetingID = @WebinarMeetingID and ClientID = @ClientID
end

    -- update access
 exec updateWebinarAccessStatusUpProcedure
 @WebinarMeetingID = @WebinarMeetingID,
 @ClientID = @ClientID
```

ReserveRoom [Anastazja]

???

```
CREATE PROCEDURE [dbo].[ReserveRoom]
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;


        WITH cte_rooms AS (
            SELECT RoomID, rn = ROW_NUMBER() OVER (ORDER BY RoomID)
            FROM dbo.GetAvailableRooms()
```

```
        ),
        cte_meetings AS (
            SELECT StudyMeetingID, mn = ROW_NUMBER() OVER (ORDER BY
StudyMeetingID)
            FROM dbo.StudyMeetings
            WHERE ClassTypeID = 4
        )
        INSERT INTO dbo.RoomStudyMeetings (RoomID, StudyMeetingID)
        SELECT r.RoomID, m.StudyMeetingID
        FROM cte_rooms r
        INNER JOIN cte_meetings m ON r.rn = m.mn
        WHERE NOT EXISTS (
            SELECT 1
            FROM dbo.RoomStudyMeetings rm
            WHERE rm.StudyMeetingID = m.StudyMeetingID
        );

        PRINT CAST(@@ROWCOUNT AS NVARCHAR(10)) + ' rows affected.';

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END;
```

ReserveRoomModuleMeeting[Anastazja]

???

```
CREATE PROCEDURE [dbo].[ReserveRoomModuleMeeting]
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;


        WITH cte_rooms AS (
            SELECT RoomID, rn = ROW_NUMBER() OVER (ORDER BY RoomID)
            FROM dbo.GetAvailableRooms()
        ),
        cte_meetings AS (
```

```
            SELECT ModuleMeetingID, mn = ROW_NUMBER() OVER (ORDER BY
ModuleMeetingID)
            FROM dbo.ModuleMeeting
            WHERE ClassTypeID = 4
        )
        INSERT INTO dbo.RoomModuleMeetings (RoomID, ModuleMeetingID)
        SELECT r.RoomID, m.ModuleMeetingID
        FROM cte_rooms r
        INNER JOIN cte_meetings m ON r.rn = m.mn
        WHERE NOT EXISTS (
            SELECT 1
            FROM dbo.RoomModuleMeetings rm
            WHERE rm.ModuleMeetingID = m.ModuleMeetingID
        );

        PRINT CAST(@@ROWCOUNT AS NVARCHAR(10)) + ' rows affected.';

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
        THROW;
    END CATCH
END;
```

SendCertificateNotificationProcedure [Anastazja]

Sprawdza ile osób ukończyło studia i wysyła do nie maile z zatwierdzeniem wysyłania certyfikatu

```
ALTER PROCEDURE [dbo].[SendCertificateNotification]
AS
BEGIN


       PRINT 'Email Subject: Notification: Certificate Delivery';

       DECLARE @RecipientEmail NVARCHAR(MAX);
       DECLARE @EducationName NVARCHAR(MAX);
       DECLARE @ClientAddress NVARCHAR(MAX);
```

```
	DECLARE CertificatesCursor CURSOR FOR
	SELECT
		Email,
		Name,
		ClientAddress
	FROM
		[dbo].GraduatedStudents;

	OPEN CertificatesCursor;

	FETCH NEXT FROM CertificatesCursor INTO @RecipientEmail, @EducationName,
@ClientAddress;

	WHILE @@FETCH_STATUS = 0
	BEGIN
	PRINT 'Email Body: Drogi Kliencie, Informujemy, że już wysyłamy Twój
certyfikat ze studiów: "' + @EducationName + '" wyślemy certyfикat pod adresem:'  +
@ClientAddress  + 'Pozdrawiamy, ME JB AY';
	PRINT 'Recipient Email: ' + @RecipientEmail;

	FETCH NEXT FROM CertificatesCursor INTO @RecipientEmail, @EducationName,
@ClientAddress;
	END

	CLOSE CertificatesCursor;
	DEALLOCATE CertificatesCursor;
END;
```

sp_UpdateOuterStudentLimit[Anastazja]

zmienia limit osób z zewnątrz na spotkaniu na studiach

```
CREATE PROCEDURE [dbo].[sp_UpdateOuterStudentLimit]
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @StudyMeetingID INT, @StudentLimit INT, @PersonLimit INT,
@OuterStudentLimit INT;

    DECLARE cur CURSOR FOR
    SELECT sm.StudyMeetingID, sm.StudentLimit, ISNULL(Ss.PersonLimit, 0)
    FROM dbo.StudyMeetings sm
    INNER JOIN dbo.Subjects s ON sm.SubjectID = s.SubjectID
    INNER JOIN dbo.Term t on s.TermID = t.TermID
```

```
    INNER JOIN dbo.Studies Ss on t.StudiesID = Ss.StudiesID;

    OPEN cur;

    FETCH NEXT FROM cur INTO @StudyMeetingID, @StudentLimit,
@PersonLimit;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @OuterStudentLimit = @StudentLimit - @PersonLimit;

        UPDATE dbo.StudyMeetings
        SET OuterStudentLimit = @OuterStudentLimit
        WHERE StudyMeetingID = @StudyMeetingID;

        PRINT 'OuterStudentLimit updated for StudyMeetingID = ' +
CAST(@StudyMeetingID AS VARCHAR);

        FETCH NEXT FROM cur INTO @StudyMeetingID, @StudentLimit,
@PersonLimit;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
```

sp_UpdateOuterStudentLimit[Anastazja]

zmienia limit osób z zewnątrz na spotkaniu na studiach

```
CREATE PROCEDURE [dbo].[sp_UpdateOuterStudentLimit]
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @StudyMeetingID INT, @StudentLimit INT, @PersonLimit INT,
@OuterStudentLimit INT;

    DECLARE cur CURSOR FOR
    SELECT sm.StudyMeetingID, sm.StudentLimit, ISNULL(Ss.PersonLimit, 0)
    FROM dbo.StudyMeetings sm
    INNER JOIN dbo.Subjects s ON sm.SubjectID = s.SubjectID
    INNER JOIN dbo.Term t on s.TermID = t.TermID
    INNER JOIN dbo.Studies Ss on t.StudiesID = Ss.StudiesID;
```

```
    OPEN cur;

    FETCH NEXT FROM cur INTO @StudyMeetingID, @StudentLimit,
@PersonLimit;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        SET @OuterStudentLimit = @StudentLimit - @PersonLimit;

        UPDATE dbo.StudyMeetings
        SET OuterStudentLimit = @OuterStudentLimit
        WHERE StudyMeetingID = @StudyMeetingID;

        PRINT 'OuterStudentLimit updated for StudyMeetingID = ' +
CAST(@StudyMeetingID AS VARCHAR);

        FETCH NEXT FROM cur INTO @StudyMeetingID, @StudentLimit,
@PersonLimit;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
```

sp_UpdateStudyMembersGrades[Anastazja]

Ustawia ocenę na przedmiotach na studiach

```
CREATE PROCEDURE [dbo].[sp_UpdateStudyMembersGrades]
AS
BEGIN
    --SET NOCOUNT ON;

    DECLARE @ClientID INT;
    DECLARE @ExamGrade DECIMAL(2,1);
      Declare @Grade Decimal(2, 1);

    DECLARE cur CURSOR FOR
    SELECT ClientID
    FROM dbo.StudyMembers;

    OPEN cur;

    FETCH NEXT FROM cur INTO @ClientID;
```

```
    WHILE @@FETCH_STATUS = 0
    BEGIN

        SELECT TOP 1 @ExamGrade = Grade
        FROM @Grades
        ORDER BY NEWID();

        UPDATE dbo.StudyMembers
        SET ExamGrade = @ExamGrade
        WHERE ClientID = @ClientID AND PassExam = 1;

        PRINT 'Grade updated for MemberID = ' + TRY_CAST(@ClientID AS
VARCHAR);

        FETCH NEXT FROM cur INTO @ClientID;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
```

sp_updateSubjectMembersGrades[Anastazja]

ustawia oceny z przedmiotów

```
CREATE PROCEDURE [dbo].[sp_UpdateSubjectMembersGrades]
AS
BEGIN
    --SET NOCOUNT ON;

    DECLARE @ClientID INT;
    DECLARE @Grade DECIMAL(2,1);


    DECLARE @Grades TABLE (Grade DECIMAL(2,1));
    INSERT INTO @Grades VALUES (3.0), (3.5), (4.0), (4.5), (5.0);

    DECLARE cur CURSOR FOR
    SELECT ClientID
    FROM dbo.StudyMeetingMembers;

    OPEN cur;

    FETCH NEXT FROM cur INTO @ClientID;
```

```
    WHILE @@FETCH_STATUS = 0
    BEGIN

        SELECT TOP 1 @Grade = Grade
        FROM @Grades
        ORDER BY NEWID();

        UPDATE dbo.SubjectMembers
        SET Grade = @Grade
        WHERE ClientID = @ClientID AND Pass = 1;

        PRINT 'Grade updated for MemberID = ' + TRY_CAST(@ClientID AS
VARCHAR);

        FETCH NEXT FROM cur INTO @ClientID;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
```

updateCourseAccessStatusUpProcedure[Monika]

Ustawia dostęp do kursu, gdy zaliczka i reszta kwoty zostały zapłacone

```
ALTER procedure [dbo].[updateCourseAccessStatusUpProcedure]
@ClientID int,
@CourseID int
as
declare
@Payment1 bit,
@Payment2 bit

select @Payment1 = Success
from CoursesPayments
where CourseID = @CourseID and CoursePaymentTypeID = 1

select @Payment2 = Success
from CoursesPayments
where CourseID = @CourseID and CoursePaymentTypeID = 2
```

```
if @Payment1 = 1 and @Payment2 = 1
    UPDATE CoursesMembers
    SET Access = 1
    WHERE ClientID = @ClientID and CourseID = @CourseID
```

updateCoursePassYesProcedure[Monika]

Zmienia status zdania kursu na zdany

```
ALTER PROCEDURE [dbo].[updateCoursePassYesProcedure]
@ClientID INT
AS
BEGIN
       DECLARE @AttendancePerc DECIMAL(5, 2);

       SELECT @AttendancePerc = dbo.calculateModuleAttendance(@ClientID);
       IF (@AttendancePerc >= 80)
       BEGIN
       UPDATE ModuleMembers
       SET PassModule = 1
       WHERE ClientID = @ClientID;
       END
       ELSE
       BEGIN
    UPDATE ModuleMembers
       SET PassModule = 0
       WHERE ClientID = @ClientID;
```

updateCoursePayDateProcedure[Monika]

Przedłuża czas zapłaty za kurs, dodaje dostęp do kursu

```
ALTER   Procedure [dbo].[updateCoursePayDateProcedure]
@CoursesPaymentsID INT,
@NewPayDate DATETIME
as
DECLARE @OldPayDate DATETIME,
@ClientID int,
@RegularCustomer bit,
@CourseID int

-- get clientID
select @ClientID = ClientID from CoursesPayments
WHERE CoursesPaymentsID = @CoursesPaymentsID;
print @ClientID
```

```
-- check if client is regular
select @RegularCustomer = RegularCustomer from Clients
where ClientID = @ClientID

print @RegularCustomer
if @RegularCustomer = 1
        begin
        -- get old paydate, get coursemeetingid
        SELECT @OldPayDate = PayDate, @CourseID = CourseID
        FROM CoursesPayments
        WHERE @CoursesPaymentsID = CoursesPaymentsID;

        if @OldPayDate < @NewPayDate
               begin
               -- update paydate for longer
               UPDATE CoursesPayments
               SET PayDate = @NewPayDate
               WHERE CoursesPaymentsID = @CoursesPaymentsID;

               -- change access to up
               exec updateCourseAccessStatusUpProcedure
                      @CourseID = @CourseID,
                      @ClientID = @ClientID
               end
        else
        print 'Give later date'
        end
    else
        PRINT 'User status do not give access to prolongue pay date.';
```

updateStrudyEntryFeePayDateProcedure[Anastazja]

Przedłuża czas zapłaty za wpisowe na studia

```
create Procedure [dbo].[updateStrudyEntryFeePayDateProcedure]
@StudyPaymentsID INT,
@NewPayDate DATETIME
as
declare
@OldPayDate DATETIME,
@ClientID int,
@RegularCustomer bit,
@StudyID int

-- get clientID
select @ClientID = ClientID from StudyPayments
```

```
WHERE StudyPaymentID = @StudyPaymentsID;
print @ClientID


-- check if client is regular
select @RegularCustomer = RegularCustomer from Clients
where ClientID = @ClientID

if @RegularCustomer = 1
begin
-- get old paydate, get coursemeetingid
SELECT @OldPayDate = PayDate, @StudyID = StudyID
FROM StudyPayments
WHERE @StudyPaymentsID = StudyPaymentID;

if @OldPayDate < @NewPayDate
begin
-- update paydate for longer
UPDATE StudyPayments
SET PayDate = @NewPayDate
WHERE StudyPaymentID = @StudyPaymentsID;
end
else
print 'Give later date'
end
else
PRINT 'User status do not give access to prolongue pay date.';
```

UpdateStudyMeetingStudentLimit[Anastazja]

Zmienia limit osób na studiach

```
CREATE PROCEDURE [dbo].[UpdateStudyMeetingStudentLimit]
@MinStudentLimit INT,
@MaxStudentLimit INT
AS
BEGIN

DECLARE @RC INT;
DECLARE @CurrentStudyMeetingID INT;

DECLARE StudyMeetingCursor CURSOR FOR
SELECT StudyMeetingID
FROM dbo.StudyMeetings;
```

```
OPEN StudyMeetingCursor;
FETCH NEXT FROM StudyMeetingCursor INTO @CurrentStudyMeetingID;

WHILE @@FETCH_STATUS = 0
BEGIN
DECLARE @RandomStudentLimit INT;
SET @RandomStudentLimit = ABS(CHECKSUM(NEWID())) % (@MaxStudentLimit -
@MinStudentLimit + 1) + @MinStudentLimit;

UPDATE dbo.StudyMeetings
SET StudentLimit = @RandomStudentLimit
WHERE StudyMeetingID = @CurrentStudyMeetingID;

FETCH NEXT FROM StudyMeetingCursor INTO @CurrentStudyMeetingID;
END

CLOSE StudyMeetingCursor;
DEALLOCATE StudyMeetingCursor;

PRINT 'Zmieniono limit studenów na zajęciu';
END;
```

UpdateStudyMembersGrades[Judyta]

Ustawia oceny studentów

```
CREATE PROCEDURE [dbo].[UpdateStudyMembersGrades]
@ClientID INT,
@ExamGrade DECIMAL(2, 1)
AS
BEGIN
IF EXISTS (SELECT 1 FROM dbo.StudyMembers WHERE ClientID = @ClientID AND PassExam =
1)
BEGIN
UPDATE dbo.StudyMembers
SET ExamGrade = @ExamGrade
WHERE ClientID = @ClientID AND PassExam = 1;

PRINT 'Zmieniono ocenę dla MemberID: ' + TRY_CAST(@ClientID AS VARCHAR);
END
ELSE
BEGIN
PRINT 'Wybrana osoba nie zdała egzaminu: ' + TRY_CAST(@ClientID AS VARCHAR);
```

```
END
END;
```

updateStudyEntryFeePayDateProcedure[Monika]

Przedłuża czas zapłaty za wpisowe na studia

```
ALTER  Procedure [dbo].[updateStrudyEntryFeePayDateProcedure]
@StudyPaymentsID INT,
@NewPayDate DATETIME
as
declare
@OldPayDate DATETIME,
@ClientID int,
@RegularCustomer bit,
@StudyID int

-- get clientID
select @ClientID = ClientID from StudyPayments
WHERE StudyPaymentID = @StudyPaymentsID;
print @ClientID


-- check if client is regular
select @RegularCustomer = RegularCustomer from Clients
where ClientID = @ClientID

if @RegularCustomer = 1
        begin
        -- get old paydate, get coursemeetingid
        SELECT @OldPayDate = PayDate, @StudyID = StudyID
        FROM StudyPayments
        WHERE @StudyPaymentsID = StudyPaymentID;

        if @OldPayDate < @NewPayDate
               begin
               -- update paydate for longer
               UPDATE StudyPayments
               SET PayDate = @NewPayDate
               WHERE StudyPaymentID = @StudyPaymentsID;
               end
        else
        print 'Give later date'
        end
    else
```

```
PRINT 'User status do not give access to prolongue pay date.';
```

updateStudyPassYesProcedure[Monika]

Zmienia status zdania studiów na zdany

```
ALTER procedure [dbo].[updateStudyPassYesProcedure]
@StudyID int,
@ClientID int
as
declare @AttendancePercentage decimal (5,2)

--get attendence perc for subject
select AttendancePercentage
from SubjectProgressView
where ClientID = @ClientID and StudiesID = @StudyID

IF @AttendancePercentage >= 80
    update SubjectMembers
    set Pass = 1
    where ClientID = @ClientID
ELSE
    Print 'Client did not pass subject.'
```

UpdateSubjectMembersGrades[Judyta]

Ustawia oceny na przedmiotach

```
CREATE PROCEDURE [dbo].[UpdateSubjectMembersGrades]
@ClientID INT,
@Grade DECIMAL(2,1)
AS
BEGIN
IF EXISTS (SELECT 1 FROM dbo.SubjectMembers WHERE ClientID = @ClientID AND Pass = 1)
BEGIN

UPDATE dbo.SubjectMembers
SET Grade = @Grade
WHERE ClientID = @ClientID AND Pass = 1;

PRINT 'Zmieniono ocenę dla MemberID: ' + TRY_CAST(@ClientID AS VARCHAR);
END
ELSE
BEGIN
```

```
PRINT 'Wybrana osoba nie zaliczyła przedmiotu: ' + TRY_CAST(@ClientID AS VARCHAR);
END
END;
```

updateWebinarAccessStatusUpProcedure[Monika]

Zmienia status dostępu do webinaru

```
ALTER   procedure [dbo].[updateWebinarAccessStatusUpProcedure]
@WebinarMeetingID int,
@ClientID int
as
declare
@Success bit,
@Access bit,
@LinkTermination datetime

select @Success = Success
from WebinarPayments
where WebinarMeetingID = @WebinarMeetingID

select @LinkTermination = LinkTermination
from WebinarMembers
where WebinarMeetingID = @WebinarMeetingID and ClientID = @ClientID


IF (@Success = 1 and GETDATE() < @LinkTermination)
    UPDATE WebinarMembers
    SET Access = 1
    WHERE ClientID = @ClientID and WebinarMeetingID = @WebinarMeetingID
```

updateWebinarPayDateProcedure [Monika]

Przedłuża czas zapłaty za webinar, dodaje dostęp do webinaru

```
ALTER PROCEDURE [dbo].[updateWebinarPayDateProcedure]
       @WebinarPaymentsID INT,
       @NewPayDate DATETIME
AS
BEGIN
       DECLARE @OldPayDate DATETIME,
    @ClientID int,
    @RegularCustomer bit,
```

```
    @WebinarMeetingID int

    -- get clientID
    select @ClientID = ClientID from WebinarPayments
       WHERE WebinarPaymentsID = @WebinarPaymentsID;

    -- check if client is regular
    select @RegularCustomer = RegularCustomer from Clients
    where ClientID = @ClientID

    if @RegularCustomer = 1
        begin
        -- get old paydate, get webinarmeetingid
        SELECT @OldPayDate = PayDate, @WebinarMeetingID = WebinarMeetingID
        FROM WebinarPayments
        WHERE WebinarPaymentsID = @WebinarPaymentsID;

        if @OldPayDate < @NewPayDate
               begin
               -- update paydate for longer
               UPDATE WebinarPayments
               SET PayDate = @NewPayDate
               WHERE WebinarPaymentsID = @WebinarPaymentsID;

               -- change access to up
               UPDATE WebinarMembers
               SET Access = 1
               WHERE ClientID = @ClientID and WebinarMeetingID = @WebinarMeetingID
               end
        else
        print 'Give later date'
        end
    else
        PRINT 'User status do not give access to prolongue pay date.';

END;
```

FUNKCJE:

FunctionClientsTotalPayments[Judyta]

```
CREATE FUNCTION [dbo].[FunctionClientsTotalPayments](@ClientID INT)
RETURNS TABLE
AS
RETURN (
SELECT
ISNULL(Sum(CW.TotalWebinarsPayment), 0) AS AmountForWebinars,
ISNULL(SUM(CC.TotalPaymentAmount), 0) AS AmountForCourses,
ISNULL(SUM(CS.TotalAmountToPay), 0) AS AmountForStudies,
ISNULL(Sum(CW.TotalWebinarsPayment), 0) + ISNULL(SUM(CC.TotalPaymentAmount), 0) +
ISNULL(SUM(CS.TotalAmountToPay), 0) AS TotalPaymentAmount
FROM Clients C
LEFT JOIN ClientsWebinarsPayments CW ON CW.ClientID = C.ClientID
LEFT JOIN ClientsStudiesTotalPayments CS ON C.ClientID = CS.ClientID
LEFT JOIN ClientsCoursePayments CC ON CC.ClientID = C.ClientID
WHERE C.ClientID = @ClientID
GROUP BY C.ClientID);
```

GetAvailableRooms[Judyta]

Poszukiwanie dostępnych pokojów, w których nie prowadzą się zajęcia dla kursów lub studiów

```
CREATE FUNCTION [dbo].[GetAvailableRooms]()
RETURNS TABLE
AS
RETURN
(
SELECT r.RoomID
FROM dbo.Rooms r
WHERE NOT EXISTS (
SELECT r.RoomID
FROM dbo.RoomStudyMeetings rm
WHERE r.RoomID = rm.RoomID
) AND NOT EXISTS(
SELECT r.RoomID
FROM dbo.RoomModuleMeetings mm
```

```
WHERE r.RoomID = mm.RoomID)
);
```

calculateModuleAttendance[Monika]

```
CREATE FUNCTION [dbo].[calculateModuleAttendance](@ClientID int)
RETURNS decimal(5, 2)
AS
BEGIN
DECLARE @RowCount INT,
@AttendancePerc decimal(5,2),
@Attendance decimal(5,2)

SELECT @RowCount = COUNT(*)
FROM ModuleMeetingMembers
Where @ClientID = ClientID
select @Attendance = Attendence
from ModuleMeetingMembers
where @ClientID = ClientID

IF @RowCount > 0
begin
SET @AttendancePerc = CAST(SUM(CASE WHEN @Attendance = 1 THEN 1 ELSE 0 END) * 100.0
/ NULLIF(@RowCount, 0) AS DECIMAL(5, 2))
end
else
begin
SET @AttendancePerc = 0.0
end
RETURN @AttendancePerc
END;
```

fn_RoundToNearestGrade[Anastazja]

Zaokrąglamy ocenę podczas zmiany oceny na dyplomie

```
CREATE FUNCTION [dbo].[fn_RoundToNearestGrade](@Grade DECIMAL(2,1))
RETURNS DECIMAL(2,1)
AS
BEGIN
DECLARE @RoundedGrade DECIMAL(2,1);

DECLARE @Grades TABLE (Grade DECIMAL(2,1));
INSERT INTO @Grades VALUES (3.0), (3.5), (4.0), (4.5), (5.0);

DECLARE cur CURSOR FOR SELECT Grade FROM @Grades;
DECLARE @CurrentGrade DECIMAL(2,1);
DECLARE @Diff DECIMAL(2,1) = 5.0;

OPEN cur;
FETCH NEXT FROM cur INTO @CurrentGrade;

WHILE @@FETCH_STATUS = 0
BEGIN
IF ABS(@Grade - @CurrentGrade) < @Diff
BEGIN
SET @Diff = ABS(@Grade - @CurrentGrade);
SET @RoundedGrade = @CurrentGrade;
END
FETCH NEXT FROM cur INTO @CurrentGrade;
END

CLOSE cur;
DEALLOCATE cur;

RETURN @RoundedGrade;
END;
```

TRIGGERY:

UpdateRegularClientStatus[Judyta]

Jeżeli klient uczestniczy w minimalnie czterech zajęciach to uzyskuje status stałego klienta

```
CREATE TRIGGER [dbo].[UpdateRegularClientStatus]
ON [dbo].[Clients]
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
UPDATE c
SET RegularCustomer = CASE
WHEN ISNULL((SELECT COUNT(*) FROM ClientsClasses WHERE ClientID = c.ClientID AND
Webinars > 0), 0) +
ISNULL((SELECT COUNT(*) FROM ClientsClasses WHERE ClientID = c.ClientID AND Studies
> 0), 0) +
ISNULL((SELECT COUNT(*) FROM ClientsClasses WHERE ClientID = c.ClientID AND Courses
> 0), 0) > 4
THEN 1
ELSE 0
END
FROM Clients c;
END;
```

PreventDuplicateCourseMember[Anastazja]

Zakaz dodania osoby do kursu, w którym już bierze udział

```
CREATE TRIGGER [dbo].[PreventDuplicateCourseMember]
ON [dbo].[CoursesMembers]
INSTEAD OF INSERT
AS
BEGIN
IF EXISTS (
SELECT 1
FROM inserted i
WHERE EXISTS (
SELECT 1
FROM CoursesPayments cp
WHERE cp.ClientID = i.ClientID AND cp.CourseID = i.CourseID
) AND Access = 1
```

```
)
BEGIN
RAISERROR('Taki uczestnik już ma dostęp do kursu', 16, 1);
END
ELSE
BEGIN
INSERT INTO CoursesMembers (ClientID, CourseID, Access)
SELECT ClientID, CourseID, Access
FROM inserted;
END
END;
```

PreventDuplicateCoursePayment[Anastazja

Zakaz tej samej płatności dla osoby, która już zapłaciła

```
CREATE TRIGGER [dbo].[PreventDuplicateCoursePayment]
ON [dbo].[CoursesPayments]
INSTEAD OF UPDATE
AS
BEGIN
DECLARE @ClientID INT, @CourseID INT, @CoursePaymentTypeID INT, @Success BIT;

SELECT TOP 1 @ClientID = ClientID, @CourseID = CourseID, @CoursePaymentTypeID =
CoursePaymentTypeID, @Success = 1
FROM inserted;

IF EXISTS (
SELECT 1
FROM CoursesPayments cp
WHERE cp.ClientID = @ClientID
AND cp.CourseID = @CourseID
AND cp.CoursePaymentTypeID = @CoursePaymentTypeID
AND cp.Success = 1
)
BEGIN
RAISERROR('Ta płatność już została zrealizowana', 16, 1);
END
ELSE
BEGIN
UPDATE cp
SET
cp.Success = i.Success
FROM CoursesPayments cp
```

```
INNER JOIN inserted i ON cp.CoursesPaymentsID = i.CoursesPaymentsID;
END
END
```

tr_AfterInsert_ModuleMeeting[Monika]

Po wprowadzeniu modułu dopisuje translatora

```
CREATE TRIGGER [dbo].[tr_AfterInsert_ModuleMeeting]
ON [dbo].[ModuleMeeting]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;
DECLARE @ModuleMeetingID int,
@OriginalLanguageID int,
@StartTime datetime,
@EndTime datetime

--get last row
SELECT
@ModuleMeetingID = i.ModuleMeetingID,
@OriginalLanguageID = i.OriginalLanguageID
FROM INSERTED i;

IF @OriginalLanguageID > 1
begin

DECLARE @AvailableTranslatorID INT;
SELECT TOP 1 @AvailableTranslatorID = e.EmployeeID
FROM Employees e
WHERE NOT EXISTS (
SELECT 1
FROM TranslatorMeetings tm
WHERE e.EmployeeID = tm.EmployeeID
AND @StartTime BETWEEN tm.StartTime AND tm.EndTime
and @EndTime between tm.StartTime and tm.Endtime
);
--get free translator

insert into ModuleTranslatingLanguages(ModuleMeetingID, TranslatorID, LanguageID)
values(@ModuleMeetingID, @AvailableTranslatorID, @OriginalLanguageID)
```

```
end
END
```

tr_AfterInsert_ModuleMeetingMembers[Judyta]

Sprawdzenie czy wystarczy miejsca w sali przy dodaniu nowego uczestnika

```
CREATE TRIGGER [dbo].[tr_AfterInsert_ModuleMeetingMembers]
ON [dbo].[ModuleMeetingMembers]
AFTER INSERT
AS
BEGIN
IF EXISTS (
SELECT i.ModuleMeetingID
FROM inserted i
INNER JOIN ModuleMeeting m ON i.ModuleMeetingID = m.ModuleMeetingID
INNER JOIN Rooms r ON m.ModuleMeetingID = r.ModuleMeetingID
WHERE m.ClassTypeID = 4 and i.ModuleMeetingID = m.ModuleMeetingID
GROUP BY i.ModuleMeetingID, r.PersonLimit
HAVING SUM(i.ClientID) > r.PersonLimit
)
BEGIN
RAISERROR('Not enough seats is available', 16, 1);
ROLLBACK;
RETURN;
END
END;
```

tr_PreventDuplicateModuleMeetingrMember[Anastazja]

Zabronienie dodania osoby do webinaru, w którym już bierze udział

```
CREATE TRIGGER [dbo].[tr_PreventDuplicateModuleMeetingrMember]
ON [dbo].[ModuleMeetingMembers]
INSTEAD OF INSERT
AS
BEGIN
IF EXISTS (
SELECT 1
```

```
FROM inserted i
WHERE EXISTS (
SELECT 1
FROM ModuleMeetingMembers mmm
WHERE mmm.ClientID = i.ClientID AND mmm.ModuleMeetingID = i.ModuleMeetingID
)
)
BEGIN
RAISERROR('Taki uczestnik już istnieje na tym spotkaniu', 16, 1);
END
ELSE
BEGIN
INSERT INTO ModuleMeetingMembers (ClientID, moduleMeetingID)
SELECT ClientID,ModuleMeetingID
FROM inserted;
END
END;
```

updateCoursePassAfterModuleMeetingMemberChange[Anastazja]

Przy dodaniu nowej osoby/zmiany obecności danej osoby sprawdzamy, czy ona już może zaliczyć kurs

```
CREATE TRIGGER [dbo].[updateCoursePassAfterModuleMeetingMemberChange]
ON [dbo].[ModuleMeetingMembers]
AFTER UPDATE, INSERT
AS
BEGIN
SET NOCOUNT ON;
DECLARE @ClientID INT;

SELECT @ClientID = ClientID
FROM inserted;

EXEC [dbo].[updateCoursePassYesProcedure] @ClientID;
END
```

tr_PreventDuplicateModuleMember[Anastazja]

Zakaz dodania osoby do modułu, w którym już bierze udział

```
CREATE TRIGGER [dbo].[tr_PreventDuplicateModuleMember]
ON [dbo].[ModuleMembers]
INSTEAD OF INSERT
AS
BEGIN
IF EXISTS (
SELECT 1
FROM inserted i
WHERE EXISTS (
SELECT 1
FROM ModuleMembers mm
WHERE mm.ClientID = i.ClientID AND mm.ModuleID = i.ModuleID
)
)
BEGIN
RAISERROR('Taki uczestnik już bierze udział w tym module', 16, 1);
END
ELSE
BEGIN
INSERT INTO ModuleMembers (ClientID, ModuleID)
SELECT ClientID, ModuleID
FROM inserted;
END
END;
```

PreventDuplicateOuterPayment[Anastazja]

Zakaz ponownej opłaty dla studentów zewnętrznych

```
CREATE TRIGGER [dbo].[PreventDuplicateOuterPayment]
ON [dbo].[OuterStudyPayments]
INSTEAD OF UPDATE
AS
BEGIN
DECLARE @ClientID INT, @OuterStudyPaymentID INT, @Success BIT;

SELECT TOP 1 @ClientID = ClientID, @OuterStudyPaymentID = OuterStudyPaymentID,
@Success = 1
FROM inserted;
```

```
IF EXISTS (
SELECT 1
FROM OuterStudyPayments osp
WHERE osp.ClientID = @ClientID
AND osp.OuterStudyPaymentID = @OuterStudyPaymentID
AND osp.Success = 1
)
BEGIN
RAISERROR('Ta płatność już została zrealizowana', 16, 1);
END
ELSE
BEGIN
UPDATE osp
SET
osp.Success = i.Success
FROM OuterStudyPayments osp
INNER JOIN inserted i ON osp.OuterStudyPaymentID = i.OuterStudyPaymentID;
END
END
```

PreventDuplicateReunionPayment[Anastazja]

Zakaz ponownej opłaty za zjazd

```
CREATE TRIGGER [dbo].[PreventDuplicateReunionPayment]
ON [dbo].[ReunionPayments]
INSTEAD OF UPDATE
AS
BEGIN
DECLARE @ClientID INT, @ReunionID INT, @Success BIT;

SELECT TOP 1 @ClientID = ClientID, @ReunionID = ReunionID, @Success = 1
FROM inserted;

IF EXISTS (
SELECT 1
FROM ReunionPayments rp
WHERE rp.ClientID = @ClientID
AND rp.ReunionID = @ReunionID
AND rp.Success = 1
```

```
)
BEGIN
RAISERROR('Ta płatność już została zrealizowana', 16, 1);
END
ELSE
BEGIN
UPDATE rp
SET
rp.Success = i.Success
FROM ReunionPayments rp
INNER JOIN inserted i ON rp.ReunionPaymentID = i.ReunionPaymentID;
END
END
```

tr_AfterInsert_Rooms[Anastazja]

Zakaz dodania nowych spotkań jeżeli ten pokój już jest zarezerwowany

```
CREATE TRIGGER [dbo].[tr_AfterInsert_Rooms]
ON [dbo].[Rooms]
AFTER INSERT
AS
BEGIN
IF EXISTS (
SELECT 1
FROM dbo.UnavailableRooms ur
INNER JOIN inserted i ON ur.ID = i.RoomID
WHERE ur.MeetingType = 'StudyMeeting' OR ur.MeetingType = 'ModuleMeeting'
)
BEGIN
RAISERROR('Room is already booked for the specified meeting type.', 16, 1);
ROLLBACK;
RETURN;
END
END;
```

tr_CheckPersonLimit[Anastazja]

Zakaz dodania osób na studia, jeżeli przekracza to limit osób

```
CREATE TRIGGER [dbo].[tr_CheckPersonLimit]
ON [dbo].[Studies]
AFTER INSERT
AS
BEGIN
SET NOCOUNT ON;

IF (EXISTS (
SELECT 1
FROM inserted i
INNER JOIN dbo.Term t ON i.StudiesID = t.StudiesID
INNER JOIN dbo.Subjects s ON t.TermID = s.TermID
INNER JOIN dbo.StudyMeetings sm ON s.SubjectID = sm.SubjectID
WHERE i.PersonLimit <= (SELECT MIN(StudentLimit) FROM dbo.StudyMeetings WHERE
SubjectID = s.SubjectID)
)
)
BEGIN
RAISERROR('Liczba osób na studiach ma być mniejsza niż dozwolona liczba osób na
spotkaniu', 16, 1);
ROLLBACK;
RETURN;
END
END;
```

tr_PreventDuplicateStudyMeetingMember[Anastazja]

Zabranianie dodawanie osoby na spotkanie, jeżeli już bierze tam udział

```
CREATE TRIGGER [dbo].[tr_PreventDuplicateStudyMeetingMember]
ON [dbo].[StudyMeetingMembers]
INSTEAD OF INSERT
AS
BEGIN
IF EXISTS (
SELECT 1
FROM inserted i
```

```
WHERE EXISTS (
SELECT 1
FROM StudyMeetingMembers smm
WHERE smm.ClientID = i.ClientID AND smm.StudiesMeetingID = i.StudiesMeetingID
)
)
BEGIN
RAISERROR('Taki uczestnik już istnieje na tym spotkaniu', 16, 1);
END
ELSE
BEGIN
INSERT INTO StudyMeetingMembers (ClientID, StudiesMeetingID)
SELECT ClientID, StudiesMeetingID
FROM inserted;
END
END;
```

updateSubjectPassAfterStudyMeetingMemberChange[Judyta]

Sprawdzenie czy zaliczyła osoba przedmiot

```
CREATE TRIGGER [dbo].[updateSubjectPassAfterStudyMeetingMemberChange]
ON [dbo].[StudyMeetingMembers]
AFTER UPDATE, INSERT
AS
BEGIN
SET NOCOUNT ON;
DECLARE @ClientID INT;

SELECT @ClientID = ClientID
FROM inserted;

EXEC [dbo].[updateStudyPassYesProcedure] @ClientID;
END
```

tr_AfterInsert_StudyMeeting[Monika]

Po wprowadzeniu modułu dopisuje translatora

```
CREATE TRIGGER [dbo].[tr_AfterInsert_StudyMeeting]
ON [dbo].[StudyMeetings]
AFTER INSERT
AS
BEGIN
DECLARE @StudyMeetingID int,
@OriginalLanguageID int,
@StartTime datetime,
@EndTime datetime

--get last row
SELECT
@StudyMeetingID = i.StudyMeetingID,
@OriginalLanguageID = i.OriginalLanguageID
FROM INSERTED i;

IF @OriginalLanguageID > 1
begin

DECLARE @AvailableTranslatorID INT;
SELECT TOP 1 @AvailableTranslatorID = e.EmployeeID
FROM Employees e
WHERE NOT EXISTS (
SELECT 1
FROM TranslatorMeetings tm
WHERE e.EmployeeID = tm.EmployeeID
AND @StartTime BETWEEN CONVERT(datetime,tm.StartTime) AND tm.EndTime
and @EndTime between CONVERT(datetime,tm.StartTime) and tm.Endtime
);
--get free translator

insert into StudyTranslatingLanguages(StudyMeetingID, TranslatorID, LanguageID)
values(@StudyMeetingID, @AvailableTranslatorID, @OriginalLanguageID)

end
end;
```

tr_UpdateOuterStudentLimit[Anastazja]

Ustala ile studentów zewnętrznych może być na zajęciach

```
CREATE TRIGGER [dbo].[tr_UpdateOuterStudentLimit]
ON [dbo].[StudyMeetings]
AFTER INSERT, UPDATE
AS
BEGIN
SET NOCOUNT ON;

EXEC sp_UpdateOuterStudentLimit;
END;
```

tr_PreventDuplicateStudyMember[Anastazja]

Zabranianie dodania studenta, który już jest uczestnikiem

```
CREATE TRIGGER [dbo].[tr_PreventDuplicateStudyMember]
ON [dbo].[StudyMembers]
INSTEAD OF INSERT
AS
BEGIN
IF EXISTS (
SELECT 1
FROM inserted i
WHERE EXISTS (
SELECT 1
FROM StudyMembers sm
WHERE sm.ClientID = i.ClientID AND sm.StudiesID = i.StudiesID
)
)
BEGIN
RAISERROR('Taki uczestnik już istnieje na tych studiach', 16, 1);
END
ELSE
BEGIN
INSERT INTO StudyMembers (ClientID, StudiesID)
SELECT ClientID, StudiesID
FROM inserted;
END
END;
```

PreventDuplicateStudyPayment[Anastazja]

Zabranianie opłaty studiów jeżeli już zapłacono

```
CREATE TRIGGER [dbo].[PreventDuplicateStudyPayment]
ON [dbo].[StudyPayments]
INSTEAD OF UPDATE
AS
BEGIN
DECLARE @ClientID INT, @StudyID INT, @Success BIT;

SELECT TOP 1 @ClientID = ClientID, @StudyID = StudyID, @Success = 1
FROM inserted;

IF EXISTS (
SELECT 1
FROM StudyPayments sp
WHERE sp.ClientID = @ClientID
AND sp.StudyID = @StudyID
AND sp.Success = 1
)
BEGIN
RAISERROR('Ta płatność już została zrealizowana', 16, 1);
END
ELSE
BEGIN
UPDATE sp
SET
sp.Success = i.Success
FROM StudyPayments sp
INNER JOIN inserted i ON sp.StudyPaymentID = i.StudyPaymentID;
END
END
```

tr_PreventDuplicateWebinarMember[Anastazja]

Zabranianie dodania osoby, jeżeli już bierze udział w webinarze

```
CREATE TRIGGER [dbo].[tr_PreventDuplicateWebinarMember]
ON [dbo].[WebinarMembers]
INSTEAD OF INSERT
AS
BEGIN
IF EXISTS (
SELECT 1
FROM inserted i
WHERE EXISTS (
SELECT 1
FROM WebinarMembers wm
WHERE wm.ClientID = i.ClientID AND wm.WebinarMeetingID = i.WebinarMeetingID
)
)
BEGIN
RAISERROR('Taki uczestnik już istnieje na tym spotkaniu', 16, 1);
END
ELSE
BEGIN
INSERT INTO WebinarMembers (ClientID, WebinarMeetingID)
SELECT ClientID, WebinarMeetingID
FROM inserted;
END
END;
```

PreventDuplicateWebinarPayment[Anastazja]

Zabranianie opłaty webinaru jeżeli już zapłacono

```
CREATE TRIGGER [dbo].[PreventDuplicateWebinarPayment]
ON [dbo].[WebinarPayments]
INSTEAD OF UPDATE
AS
BEGIN
DECLARE @ClientID INT, @WebinarMeetingID INT, @Success BIT;

SELECT TOP 1 @ClientID = ClientID, @WebinarMeetingID = WebinarMeetingID, @Success =
1
FROM inserted;

IF EXISTS (
SELECT 1
FROM WebinarPayments wp
WHERE wp.ClientID = @ClientID
AND wp.WebinarMeetingID = @WebinarMeetingID
AND wp.Success = 1
)
BEGIN
RAISERROR('Ta płatność już została zrealizowana', 16, 1);
END
ELSE
BEGIN
UPDATE wp
SET
wp.Success = i.Success
FROM WebinarPayments wp
INNER JOIN inserted i ON wp.WebinarPaymentsID = i.WebinarPaymentsID;
END
END
```

trg_InsertStudyPass[Anastazja]

Sprawdza czy osoba zaliczyła przedmiot

```
CREATE TRIGGER [dbo].[trg_InsertStudyPass]
ON [dbo].[SubjectProgressView]
INSTEAD OF INSERT
AS
BEGIN
SET NOCOUNT ON;

DECLARE @ClientID INT;
DECLARE @StudyID INT;

SELECT @ClientID = ClientID, @StudyID = StudiesID
FROM inserted;

EXEC [dbo].[updateStudyPassYesProcedure] @StudyID, @ClientID;
END;
```

trg_UpdateStudyPass[Anastazja]

Zmiana statusu ukończenia studiów

```
CREATE TRIGGER [dbo].[trg_UpdateStudyPass]
ON [dbo].[SubjectProgressView]
INSTEAD OF UPDATE
AS
BEGIN
SET NOCOUNT ON;

DECLARE @ClientID INT;
DECLARE @StudyID INT;

SELECT @ClientID = ClientID, @StudyID = StudiesID
FROM inserted;

EXEC [dbo].[updateStudyPassYesProcedure] @StudyID, @ClientID;
END;
```

UPRAWNIENIA:

Klient bez założonego konta

Ma uprawnienia do:

- wyświetlania linków do syllabusa

```
create role client_without_account
```

– widoki

```
grant select on FieldOfStudy_Syllabus to client_without_account
```

Klient

Ma uprawnienia do:

- informacji o kolidujących ze sobą zajęciach na które jest zapisany
- informacji o zajęciach na które jest zapisany
- informacji o swoich płatnościach za zajęcia na które jest zapisany
- informacji o zaliczeniu zajęć
- wyświetlania linków do syllabusa
- zapisania się na wybrane zajęcia
- zapłacenia za zajęcia

```
create role client
```

– widoki

```
grant select on FieldOfStudy_Syllabus to client
grant select on ClientsClasses to client
grant select on CourseProgress to client
grant select on CombinedMeetingsInformation to client
grant select on CoursesAndModules to client
grant select on ModulesAndModuleMeetingsView to client
grant select on StudiesAndTermsAndSubjects to client
grant select on SubjectProgressView to client
grant select on WebinarMeetingView to client
grant select on WebinarsAndWebinarMeetingsView to client
```

– procedury

```
grant exec on payEntryForStudyProcedure to client
grant exec on payForCourseProcedure to client
grant exec on payForOuterStudetnPaymentProcedure to client
grant exec on payForStudyReunionPaymentProcedure to client
grant exec on payForWebinarProcedure to client
grant exec on addCourseMemberProcedure to client
grant exec on addStudyMemberProcedure to client
grant exec on addWebinarMemberProcedure to client
```

Wykładowca:

Ma uprawnienia do:

- zmiany statusu zdania studiów/kursu
- informacji o obecności na kursach/studiach/webinarach
- informacji o zajęciach które prowadzi
- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami

```
create role lecturer
```

— widoki

```
grant select on AttendanceOnCourses to lecturer
grant select on AttendanceOnStudies to lecturer
grant select on AttendanceOnWebinars to lecturer
grant select on CoursesAndModules to lecturer
grant select on ModulesAndModuleMeetingsView to lecturer
grant select on StudiesAndTermsAndSubjects to lecturer
grant select on FutureEventRegistrations to lecturer
grant select on WebinarMeetingView to lecturer
grant select on WebinarsAndWebinarMeetingsView to lecturer
```

— procedury

```
grant exec on sp_UpdateStudyMembersGrades to lecturer
grant exec on sp_UpdateSubjectMembersGrades to lecturer
```

Tłumacz:

Ma uprawnienia do:

- informacji o zajęciach które tłumaczy

```
create role translator
```

– widoki

```
grant select on TranslatorMeetings to translator
grant select on CoursesAndModules to translator
grant select on ModulesAndModuleMeetingsView to translator
grant select on StudiesAndTermsAndSubjects to translator
grant select on WebinarMeetingView to translator
grant select on WebinarsAndWebinarMeetingsView to translator
```

Księgowa

Ma uprawnienia do:

- podsumowania zarobków za kursy/webinary/kursy
- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami

```
create role accountant
```

– widoki

```
grant select on ClientsCoursePayments to accountant
grant select on ClientsStudiesTotalPayments to accountant
grant select on ClientsTotalPayments to accountant
grant select on ClientsWebinarsPayments to accountant
grant select on CoursesRevenue to accountant
grant select on FutureEventRegistrations to accountant
grant select on ListOfDebtors to accountant
grant select on OverallFinancialSummary to accountant
grant select on StudiesRevenue to accountant
grant select on WebinarFinancialSummary to accountant
```

– funkcje

```
grant select on FunctionClientsTotalPayments to accountant
```

Sekretarka:

Ma uprawnienia do:

- wysłania certyfikatu
- nadania dostępu do kursu/ webinaru/studiów ? X
- rezerwacji sali
- informacji o obecności na kursach/studiach
- informacji o studentach i zajęciach na które są zapisani
- informacji o kursach, modułach, webinarach
- informacji o pracownikach
- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami
- informacji o zaliczeniu praktyk i zdania egzaminu przez studentów
- zarezerwowania sali ?
- przydzielenia limitu osób na dane spotkanie ?

```
create role secretary
```

– widoki

```
grant select on AttendanceOnCourses to secretary
grant select on AttendanceOnStudies to secretary
grant select on AttendanceOnWebinars to secretary
grant select on CoursesAndModules to secretary
grant select on ModulesAndModuleMeetingsView to secretary
grant select on StudiesAndTermsAndSubjects to secretary
```

```
grant select on FutureEventRegistrations to secretary
grant select on WebinarMeetingView to secretary
grant select on WebinarsAndWebinarMeetingsView to secretary
grant select on ClientsAddress to secretary
grant select on ClientsClasses to secretary
grant select on CombinedMeetingsInformation to secretary
grant select on CourseProgress to secretary
grant select on EmployeeAddress to secretary
grant select on FielsOfStudy_Students to secretary
grant select on GraduatedStudents to secretary
grant select on StudentsInterships to secretary
grant select on StudyCertificates to secretary
grant select on SubjectProgressView to secretary
grant select on CoursesMembersView to secretary
grant select on WebinarsMembersView to secretary
grant select on ClientOverlappingMeetings to secretary
grant select on UnavailableRooms to secretary
```

– procedury

```
grant exec on addCourseAndModule to secretary
grant exec on addCourseMemberProcedure to secretary
grant exec on addModuleMeeting to secretary
grant exec on addOuterStudentMemberProcedure to secretary
grant exec on addReunionAndStudyMeeting to secretary
grant exec on addStudyAndTermsAndSubjects to secretary
grant exec on addStudyMemberProcedure to secretary
grant exec on addWebinarAndWebinarMeeting to secretary
grant exec on addWebinarMemberProcedure to secretary
grant exec on ChangeLecturerModuleMeeting to secretary
grant exec on ChangeLecturerStudyMeeting to secretary
grant exec on ChangeLecturerWebinarMeeting to secretary
grant exec on ReserveRoom to secretary
grant exec on SendCertificateNotification to secretary
grant exec on updateStrudyEntryFeePayDateProcedure to secretary
grant exec on UpdateModuleMeetingAttendenceProcedure to secretary
```

– funkcje

```
grant select on GetAvailableRooms to secretary
```

Dyrektor:

Ma uprawnienia do:

- dodania uczestnika do kursu, webinaru i studiów
- dodanie nowego webinaru, studiów, kursu
- wysłania certyfikatu
- przedłużenia terminu zapłaty
- nadania dostępu do kursu/ webinaru/studiów
- informacji o obecności na kursach/studiach / webinarach
- informacji o studentach i zajęciach na które są zapisani i ich zaliczeniach
- informacji o pracownikach
- informacji o przyszłych wydarzeniach wraz z zapisanymi klientami
- informacji o zaliczeniu praktyk i zdania egzaminu przez studentów

```
create role boss
```

– widoki

```
grant select on AttendanceOnCourses to boss
grant select on AttendanceOnStudies to boss
grant select on AttendanceOnWebinars to boss
grant select on CoursesAndModules to boss
grant select on ModulesAndModuleMeetingsView to boss
grant select on StudiesAndTermsAndSubjects to boss
grant select on FutureEventRegistrations to boss
grant select on WebinarMeetingView to boss
grant select on WebinarsAndWebinarMeetingsView to boss
grant select on ClientsAddress to boss
grant select on ClientsClasses to boss
grant select on CombinedMeetingsInformation to boss
grant select on CourseProgress to boss
grant select on EmployeeAddress to boss
grant select on FielsOfStudy_Students to boss
grant select on GraduatedStudents to boss
grant select on StudentsInterships to boss
grant select on StudyCertificates to boss
grant select on SubjectProgressView to boss
grant select on CoursesMembersView to boss
grant select on WebinarsMembersView to boss
grant select on ClientOverlappingMeetings to boss
grant select on UnavailableRooms to boss
```

– procedury

```
grant exec on addCourseAndModule to boss
grant exec on addCourseMemberProcedure to boss
grant exec on addModuleMeeting to boss
grant exec on addOuterStudentMemberProcedure to boss
grant exec on addReunionAndStudyMeeting to boss
```

```
grant exec on addStudyAndTermsAndSubjects to boss
grant exec on addStudyMemberProcedure to boss
grant exec on addWebinarAndWebinarMeeting to boss
grant exec on addWebinarMemberProcedure to boss
grant exec on ChangeLecturerModuleMeeting to boss
grant exec on ChangeLecturerStudyMeeting to boss
grant exec on ChangeLecturerWebinarMeeting to boss
grant exec on ReserveRoom to boss
grant exec on SendCertificateNotification to boss
grant exec on updateStrudyEntryFeePayDateProcedure to boss
grant exec on UpdateModuleMeetingAttendenceProcedure to boss
```

– funkcje

```
grant select on GetAvailableRooms to boss
grant select on FunctionClientsTotalPayments to boss
```

Administrator

Ma dostęp do wszystkiego w bazie

```
create role admin

grant all privileges to admin
```

INDEKSY:

Studies

Po nazwie studiów

```
Create NONCLUSTERED INDEX Studies_Names On Studies ([Name])
```

Courses

Po nazwie kursu

```
Create NONCLUSTERED INDEX Courses_Names On Courses ([Name])
```

Webinars

Po nazwie webinaru

```
Create NONCLUSTERED INDEX Webinars_Names On Webinars ([Name])
```

Subjects

Po nazwie przedmiotu

```
Create NONCLUSTERED INDEX Subjects_Names On Subjects ([Name])
```

BuildingAddresses

Po adresie budynku

```
Create NONCLUSTERED INDEX BuildingAddress_Address On
BuildingAddresses ([Address])
```

WebinarMeetings

Po ID webinaru

```
Create NONCLUSTERED INDEX WebinarMeetings_WebinarID On
WebinarMeetings ([WebinarID])
```

Po dacie rozpoczęcia

```
Create NONCLUSTERED INDEX WebinarMeetings_StartDate On
WebinarMeetings ([StartDate])
```

Po dacie zakończenia

```
Create NONCLUSTERED INDEX WebinarMeetings_EndDate On WebinarMeetings
([EndDate])
```

Reunions

Po ID semestru

```
Create NONCLUSTERED INDEX Reunions_TermID On Reunions ([TermID])
```

Po dacie rozpoczęcia

```
Create NONCLUSTERED INDEX Reunions_StartDate On Reunions
([StartDate])
```

Po dacie zakończenia

```
Create NONCLUSTERED INDEX Reunions_EndDate On Reunions ([EndDate])
```

Term

Po ID studiów

```
Create NONCLUSTERED INDEX Term_StudiesID On Term ([StudiesID])
```

Module

Po ID kursu

```
Create NONCLUSTERED INDEX Module_CourseID On Module ([CourseID])
```

Po dacie rozpoczęcia

```
Create NONCLUSTERED INDEX Module_StartDate On Module ([StartDate])
```

Po dacie zakończenia

```
Create NONCLUSTERED INDEX Module_EndDate On Module ([EndDate])
```

Room

Po ID adresu budynku

```
Create NONCLUSTERED INDEX Rooms_BuildingAddressID On Rooms
([BuildingAddressID])
```

Po limicie osób w sali

```
Create NONCLUSTERED INDEX Rooms_PersonLimit On Rooms ([PersonLimit])
```