Bazy-Danych-Projekt-2023/2024

Systemy Baz Danych 2023/2024 – projekt systemu bazodanowego dla firmy oferującej kursy i szkolenia

Autorzy:

Piotr Śmiałek
Robert Zuziak
Hubert Tułacz

Prowadzący

dr inż. Robert Marcjan

Funkcje użytkowników

Użytkownik anonimowy (gość):

- Przeglądanie dostępnych webinariów.
- Przeglądanie dostępnych kursów.
- Przeglądanie dostępnych studiów.
- Przeglądanie dostępnych informacji o wykładowcach.
- Przeglądanie dostępnych terminów i miejsc spotkań stacjonarnych.
- Rejestracja na darmowe webinaria.
- Przeglądanie nagrań webinariów dostępnych publicznie.
- Możliwość założenia konta

Użytkownik zarejestrowany:

- Logowanie do systemu.
- Przeglądanie dostępnych webinariów.
- Przeglądanie kursów.
- Przeglądanie studiów.
- Przeglądanie informacji o wykładowcach.
- Rejestracja na płatne webinaria.
- Zapisywanie się na kursy (wybór terminów i formy zajęć).
- Zapisywanie się na studia (wybór specjalizacji).
- Przeglądanie własnych zapisów i historii uczestnictwa.
- Przeglądanie informacji o płatnościach.
- Odrabianie nieobecności na zajęciach (jeśli to możliwe).

Wykładowca/Nauczyciel:

Zarządzanie Kursami/Spotkaniami:

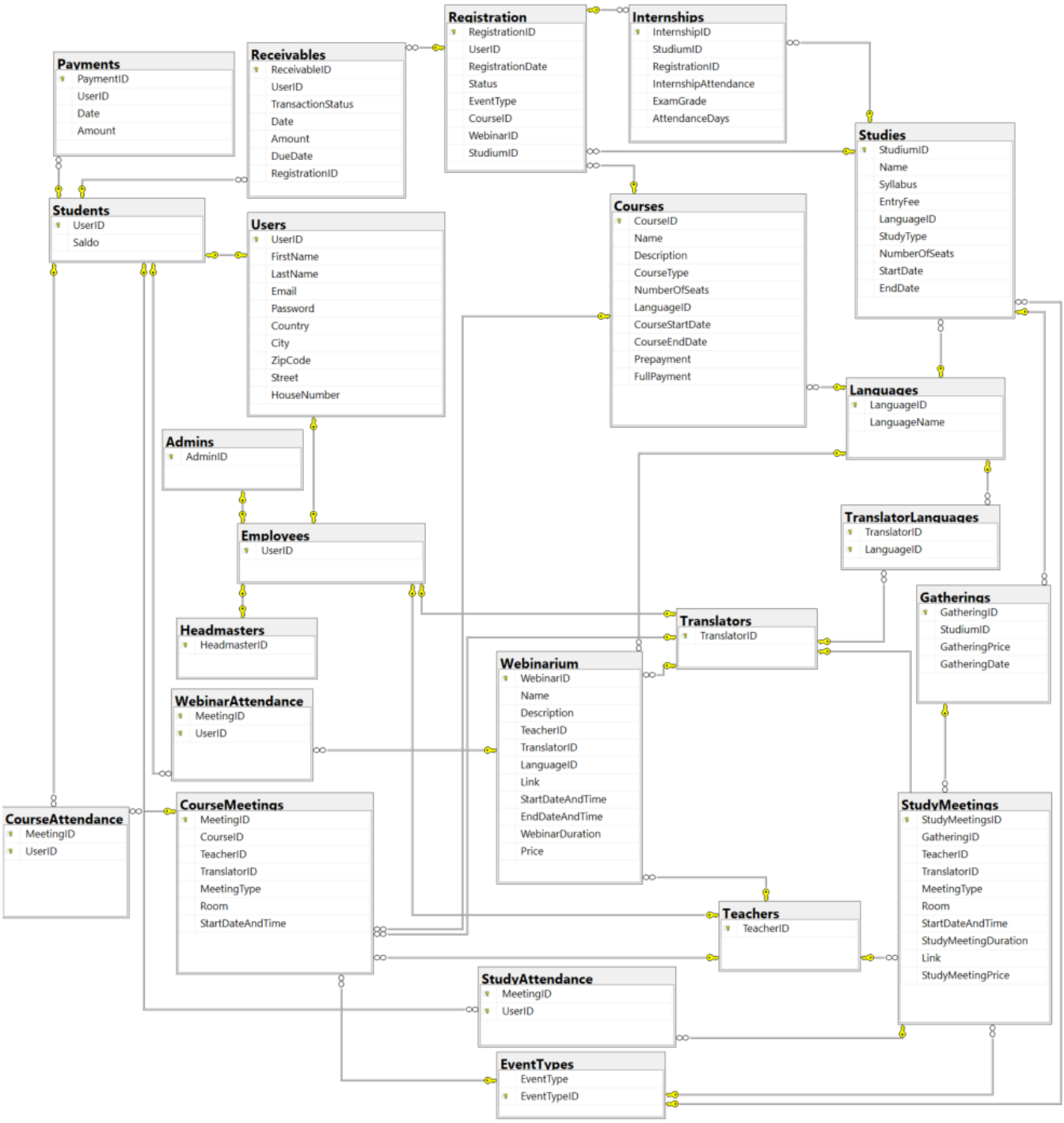
- Dodawanie nowych kursów, webinarów i studiów do systemu.
- Zarządzanie terminami i miejscami spotkań stacjonarnych.
- Aktualizacja informacji o programach nauczania (sylabusach).
- Przypisywanie uczestników do kursów i studiów.

Zarządzanie Ocenami i Frekwencją:

- Wprowadzanie ocen dla uczestników kursów.
- Zaznaczanie obecności na spotkaniach stacjonarnych i online.
- Generowanie raportów dotyczących frekwencji i ocen.

Administrator systemu:

- Dodawanie, edytowanie i usuwanie webinariów.
- Dodawanie, edytowanie i usuwanie kursów.
- Dodawanie, edytowanie i usuwanie studiów.
- Zarządzanie listą wykładowców.
- Zarządzanie terminami i miejscami spotkań stacjonarnych.
- Zarządzanie użytkownikami (edycja danych, blokowanie, usuwanie).
- Przeglądanie raportów finansowych.
- Generowanie listy "dłużników".
- Generowanie raportu dotyczącego liczby zapisanych osób na przyszłe wydarzenia.
- Generowanie raportu dotyczącego frekwencji na zakończonych wydarzeniach.
- Generowanie listy obecności dla każdego szkolenia.
- Generowanie raportu bilokacji.
- Zarządzanie rolami i uprawnieniami użytkowników.
- Dodawanie nowych użytkowników.
- Edycja treści i opisów kursów, studiów, webinariów.



Tabele:

Admins - zawiera id użytkowników, którzy są administratorami systemu

- AdminID (PK)- Identyfikator użytkownika (administratora)

```
CREATE TABLE [dbo].[Admins](
    [AdminID] [int] NOT NULL,

    CONSTRAINT [PK_Admins] PRIMARY KEY CLUSTERED ([AdminID] ASC)
    WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
    ON [PRIMARY])
    ON [PRIMARY]
```

```
    ALTER TABLE [dbo].[Admins]
    WITH CHECK ADD  CONSTRAINT [FK_Admins_Employees] FOREIGN KEY([AdminID])
    REFERENCES [dbo].[Employees] ([UserID])


    ALTER TABLE [dbo].[Admins] CHECK CONSTRAINT [FK_Admins_Employees]
```

CourseAttendance - Zawiera informacje o obecności studenta na zajęciach kursu

- MeetingID - identyfikator kursu
- UserID - identyfikator studenta

```
CREATE TABLE [dbo].[CourseAttendance](
    [MeetingID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    CONSTRAINT [PK_CourseAttendance] PRIMARY KEY CLUSTERED ([MeetingID] DESC,
    [UserID] DESC)

    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]

    ALTER TABLE [dbo].[CourseAttendance]
    WITH CHECK
    ADD  CONSTRAINT [FK_CourseAttendance_CourseMeetings]
    FOREIGN KEY([MeetingID])
    REFERENCES [dbo].[CourseMeetings] ([MeetingID])

    ALTER TABLE [dbo].[CourseAttendance]
    CHECK CONSTRAINT [FK_CourseAttendance_CourseMeetings]

    ALTER TABLE [dbo].[CourseAttendance]
    WITH CHECK
    ADD CONSTRAINT [FK_CourseAttendance_Students] FOREIGN KEY([UserID])
    REFERENCES [dbo].[Students] ([UserID])

    ALTER TABLE [dbo].[CourseAttendance] CHECK CONSTRAINT [FK_CourseAttendance_Students]
```

CourseMeetings - zawiera wszystkie spotkania w ramach jednego kursu

- MeetingID - klucz główny identyfikator każdego spotkania
- CourseID - identyfikator kursu do którego należy spotkanie
- TeacherID - identyfikator nauczyciela prowadzącego spotkanie
- TranslatorID - ID tłumacza
- MeetingType - czy spotkanie jest stacjonarne/zdalne/zdalnie asynchronicznie(1/2/3)
- Room - sala w jakiej odbywa się spotkanie
- StartDateAndTime - data i godzina odbycia się zajęć

```
CREATE TABLE [dbo].[CourseMeetings](
    [MeetingID] [int] NOT NULL,
    [CourseID] [int] NOT NULL,
    [TeacherID] [int] NOT NULL,
    [TranslatorID] [int] NULL,
    [MeetingType] [int] NULL,
    [Room] [varchar](50) NULL,
    [StartDateAndTime] [datetime] NOT NULL,
    CONSTRAINT [PK_CourseMeetings] PRIMARY KEY CLUSTERED ([MeetingID] ASC)

    WITH (PAD_INDEX = OFF,
```

```
        STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]) ON [PRIMARY]

    ALTER TABLE [dbo].[CourseMeetings]
    WITH CHECK
    ADD CONSTRAINT [FK_CourseMeetings_Courses] FOREIGN KEY([CourseID])
    REFERENCES [dbo].[Courses] ([CourseID])

    ALTER TABLE [dbo].[CourseMeetings]
    CHECK CONSTRAINT [FK_CourseMeetings_Courses]

    ALTER TABLE [dbo].[CourseMeetings]
    WITH CHECK
    ADD  CONSTRAINT [FK_CourseMeetings_EventTypes] FOREIGN KEY([MeetingType])
    REFERENCES [dbo].[EventTypes] ([EventTypeID])

    ALTER TABLE [dbo].[CourseMeetings] CHECK CONSTRAINT [FK_CourseMeetings_EventTypes]

    ALTER TABLE [dbo].[CourseMeetings]
    WITH CHECK ADD  CONSTRAINT [FK_CourseMeetings_Teachers] FOREIGN KEY([TeacherID])
    REFERENCES [dbo].[Teachers] ([TeacherID])

    ALTER TABLE [dbo].[CourseMeetings] CHECK CONSTRAINT [FK_CourseMeetings_Teachers]

    ALTER TABLE [dbo].[CourseMeetings]
    WITH CHECK ADD  CONSTRAINT [FK_CourseMeetings_Translators] FOREIGN KEY([TranslatorID])
    REFERENCES [dbo].[Translators] ([TranslatorID])

    ALTER TABLE [dbo].[CourseMeetings] CHECK CONSTRAINT [FK_CourseMeetings_Translators]
```

Courses - zawiera kursy oraz informacje o nich

- CoursesID (Pk) - klucz główny kursu
- Name - nazwa
- Description - tekstowy opis kursu
- CourseType - 1/2/3 (używane w słowniku EventTypes)
- NumberOfAvaliableSeats - liczba wolnych miejsc na kurs
- LanguageID - id języka w jakim jest prowadzony kurs
- CourseStartDate - data rozpoczęcia kursu
- CourseEndDate - data zakończenia kursu
- Prepayment - zaliczka przy zapisie
- Full Payment - dopłata całości kwoty (z wyłączeniem zaliczki)

```
CREATE TABLE [dbo].[Courses](
    [CourseID] [int] NOT NULL,
    [Name] [varchar](50) NOT NULL,
    [Description] [text] NOT NULL,
    [Prepayment] [money] NOT NULL,
    [FullPayment] [money] NOT NULL,
    [CourseStartDate] [datetime] NOT NULL,
    [Type] [varchar](50) NOT NULL,
    [Language] [varchar](50) NULL,
    [CourseType] [int] NULL,
    [CourseEndDate] [datetime] NULL,
    CONSTRAINT [PK_Courses] PRIMARY KEY CLUSTERED ([CourseID] ASC)

    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON,
```

```
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
    ) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

    ALTER TABLE [dbo].[Courses]
    WITH CHECK ADD  CONSTRAINT [FK_Courses_EventTypes] FOREIGN KEY([CourseType])
    REFERENCES [dbo].[EventTypes] ([EventTypeID])

    ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [FK_Courses_EventTypes]

    ALTER TABLE [dbo].[Courses]
    WITH CHECK ADD  CONSTRAINT [CK_Courses]
        CHECK  (([Type]='hybrid' OR [Type]='on-line' OR [Type]='stationary'))

    ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [CK_Courses]
```

Employees -zawiera id użytkowników, którzy są pracownikami

- UserID (Pk) - identyfikator w tabeli Users

```
CREATE TABLE [dbo].[Employees](
    [UserID] [int] NOT NULL
    ) ON [PRIMARY]

    ALTER TABLE [dbo].[Employees]
    ADD  CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED ([UserID] ASC)
    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Employees]
    WITH CHECK ADD  CONSTRAINT [FK_Employees_Users] FOREIGN KEY([UserID])
    REFERENCES [dbo].[Users] ([UserID])
    ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Users]
    GO
```

EventTypes- słownik mapujący EventType na EventTypeID stationary - 1 hybrid - 2 online - 3

```
CREATE TABLE [dbo].[EventTypes](
    [EventType] [varchar](50) NOT NULL,
    [EventTypeID] [int] NOT NULL,
    CONSTRAINT [PK_EventTypes] PRIMARY KEY CLUSTERED ([EventTypeID] ASC)
    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

Gatherings - zawiera informacje o zlotach dla danych studiów

- GatheringID - identyfikator zlotu
- StudiumID - ID studium do, którego jest przypisany
- GatheringPrice - cena zlotu
- GatheringDate - data zlotu

```
CREATE TABLE [dbo].[Gatherings](
    [GatheringID] [int] NOT NULL,
    [StudiumID] [int] NULL,
    [GatheringPrice] [money] NULL,
    [GatheringDate] [date] NULL
) ON [PRIMARY]

    ALTER TABLE [dbo].[Gatherings]
    ADD CONSTRAINT [PK_Gatherings] PRIMARY KEY CLUSTERED
    (
        [GatheringID] ASC
    )
    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Gatherings]
    WITH CHECK ADD  CONSTRAINT [FK_Gatherings_Studies]
     FOREIGN KEY([StudiumID])
    REFERENCES [dbo].[Studies] ([StudiumID])

    ALTER TABLE [dbo].[Gatherings] CHECK CONSTRAINT [FK_Gatherings_Studies]
```

Headmasters - zawiera id użytkowników, którzy są dyrektorami

- HeadmasterID (PK)- Identyfikator użytkownika (dyrektora)

```
CREATE TABLE [dbo].[Headmasters](
    [HeadmasterID] [int] NOT NULL,

    CONSTRAINT [PK_Headmasters] PRIMARY KEY CLUSTERED ([HeadmasterID] ASC)
    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    IGNORE_DUP_KEY = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON,
    OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
    ) ON [PRIMARY]
    GO

    ALTER TABLE [dbo].[Headmasters]
    WITH CHECK
    ADD  CONSTRAINT [FK_Headmasters_Employees] FOREIGN KEY([HeadmasterID])
    REFERENCES [dbo].[Employees] ([UserID])

    ALTER TABLE [dbo].[Headmasters] CHECK CONSTRAINT [FK_Headmasters_Employees]
```

Internships - zawiera informacje o praktykach studentów

- InternshipID - klucz główny identyfikatora praktyki
- StudiumID - identyfikator studiów do których częścią jest dana praktyka
- RegistrationID - identyfikator rejestracji na dane praktyki
- InternshipAttendance - procent na ilu zajęciach był obecny praktykant
- ExamGrade - ocena z egzaminu
- AttendanceDays - liczba dni na których student był obecy (od 0 do 14)

```
CREATE TABLE [dbo].[Internships](
    [InternshipID] [int] NOT NULL,
    [StudiumID] [int] NOT NULL,
    [RegistrationID] [int] NOT NULL,
    [InternshipAttendance] [float] NULL,
    [ExamGrade] [float] NULL,
    [AttendanceDays] [int] NULL) ON [PRIMARY]

    ALTER TABLE [dbo].[Internships]
    ADD CONSTRAINT [PK_Internships] PRIMARY KEY CLUSTERED ([InternshipID] ASC)

    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Internships]
    WITH CHECK
    ADD CONSTRAINT [FK_Internships_Registration]
    FOREIGN KEY([RegistrationID])
    REFERENCES [dbo].[Registration] ([RegistrationID])

    ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Internships_Registration]

    ALTER TABLE [dbo].[Internships]  WITH CHECK
    ADD  CONSTRAINT [FK_Internships_Studies] FOREIGN KEY([StudiumID])
    REFERENCES [dbo].[Studies] ([StudiumID])

    ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Internships_Studies]

    ALTER TABLE [dbo].[Internships]
    WITH CHECK ADD CONSTRAINT [CK_Internships]
    CHECK (([ExamGrade]>=(2) AND [ExamGrade]<=(5)))

    ALTER TABLE [dbo].[Internships]
    CHECK CONSTRAINT [CK_Internships]

    ALTER TABLE [dbo].[Internships]
    WITH CHECK
    ADD  CONSTRAINT [CK_Internships_1]
    CHECK  (([AttendanceDays]>=(0) AND [AttendanceDays]<=(14)))

    ALTER TABLE [dbo].[Internships]
    CHECK CONSTRAINT [CK_Internships_1]

    ALTER TABLE [dbo].[Internships]
    WITH CHECK ADD  CONSTRAINT [CK_Internships_2]
    CHECK (([InternshipAttendance]>=(0) AND [InternshipAttendance]<=(100)))

    ALTER TABLE [dbo].[Internships]
    CHECK CONSTRAINT [CK_Internships_2]

    --Trigger jest wyjasniony w dalszej czesci
    CREATE TRIGGER [dbo].[InternshipAttendanceUpdate]
    ON [dbo].[Internships]
    AFTER UPDATE
    AS
    BEGIN
    -- sprawdzenei czy kolumna AttendanceDays została zmieniona
        IF UPDATE(AttendanceDays)
        BEGIN
            UPDATE I
            SET I.InternshipAttendance = ROUND(CONVERT(float, I.AttendanceDays) / 14,2)
```

```
            FROM [dbo].[Internships] AS I
            INNER JOIN inserted AS ins ON I.InternshipID = ins.InternshipID;
        END
    END;
    ALTER TABLE [dbo].[Internships] ENABLE TRIGGER [InternshipAttendanceUpdate]
```

Languages - tabela słownikowa zawierająca języki, w których moga odbywać sie szkolenia

- LanguageID - id języka
- LanguageName - język

```
CREATE TABLE [dbo].[Languages](
    [LanguageID] [int] NOT NULL,
    [LanguageName] [varchar](50) NOT NULL
    ) ON [PRIMARY]
    ALTER TABLE [dbo].[Languages]
     ADD  CONSTRAINT [PK_Languages] PRIMARY KEY CLUSTERED
    (
        [LanguageID] ASC
    )WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON)
        ON [PRIMARY]
```

Payments - tabela zawierająca wszystkie wpłaty

- PaymentID (PK) - identyfikator płatności
- UserID (FK) - identyfikator użytkownika który dokonał płatności
- Date - data dokonania płatności
- Amount - wartość płatności

```
CREATE TABLE [dbo].[Payments](
    [PaymentID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NULL,
    [Date] [date] NULL,
    [Amount] [money] NULL) ON [PRIMARY]

    ALTER TABLE [dbo].[Payments] ADD PRIMARY KEY CLUSTERED ([PaymentID] ASC)

    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Payments]
    WITH CHECK ADD FOREIGN KEY([UserID])
    REFERENCES [dbo].[Students] ([UserID])

    --Trigger wyjasniony dalej
    CREATE TRIGGER [dbo].[UpdateSaldoAfterPayment]
    ON [dbo].[Payments]
    AFTER INSERT
    AS
    BEGIN
        SET NOCOUNT ON;
```

```
        -- Aktualizuj saldo dla każdego nowego wpisu w tabeli Payments
        UPDATE s
        SET s.Saldo = s.Saldo + i.Amount
        FROM dbo.Students s
        INNER JOIN inserted i ON s.UserID = i.UserID;
    END;
    ALTER TABLE [dbo].[Payments] ENABLE TRIGGER [UpdateSaldoAfterPayment]
```

Receivables - tabela zawierająca wszystkie należności

- ReceivableID (PK) - identyfikator należności
- UserID (FK) - identyfikator studenta do którego przypisana jest należność
- TransactionStatus - informacja czy należność została uregulowana
- Date - data powstania należności
- Amount - wartość należności
- DueDate - data do której należność ma być uregulowana
- RegistrationID(FK) - identyfikator rejestracji z której pochodzi należność

```
CREATE TABLE [dbo].[Receivables](
    [ReceivableID] [int] NOT NULL,
    [UserID] [int] NULL,
    [TransactionStatus] [nchar](10) NULL,
    [Date] [date] NULL,
    [Amount] [money] NULL,
    [DueDate] [date] NULL,
    [RegistrationID] [int] NULL) ON [PRIMARY]

    ALTER TABLE [dbo].[Receivables]
    ADD PRIMARY KEY CLUSTERED ([ReceivableID] ASC)
    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Receivables]
    WITH CHECK ADD  CONSTRAINT [FK__Receivabl__Regis__37FA4C37] FOREIGN KEY([RegistrationID])
    REFERENCES [dbo].[Registration] ([RegistrationID])

    ALTER TABLE [dbo].[Receivables]
    CHECK CONSTRAINT [FK__Receivabl__Regis__37FA4C37]

    ALTER TABLE [dbo].[Receivables]
    WITH CHECK ADD FOREIGN KEY([UserID])
    REFERENCES [dbo].[Students] ([UserID])

    ALTER TABLE [dbo].[Receivables]
    WITH CHECK ADD  CONSTRAINT [CK_Receivables]
    CHECK  (([TransactionStatus]='paid' OR [TransactionStatus]='pending'))
    ALTER TABLE [dbo].[Receivables] CHECK CONSTRAINT [CK_Receivables]
```

Registration - tabela zawiera rejestracje dla każdego studenta. Student może mieć wiele rejestracji (może uczęszczać na wiele eventów)

- RegistrationID (PK) - identyfikator rejestracji
- UserID (FK)- Identyfikator użytkownika
- RegistrationDate - data zarejestrowania na dane wydarzenie
- Status - completed/in_progress/failed
- EventType - 1/2/3, połączone z słownikiem EventTypes(stationary/hybrid/online)
- CourseID

- WebinarID
- StudiumID - dokładnie jedno z powyższych nie jest nullem

```
CREATE TABLE [dbo].[Registration](
    [RegistrationID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [RegistrationDate] [date] NOT NULL,
    [Status] [varchar](50) NOT NULL,
    [EventType] [text] NOT NULL,
    [CourseID] [int] NULL,
    [WebinarID] [int] NULL,
    [StudiumID] [int] NULL) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

ALTER TABLE [dbo].[Registration]
    ADD  CONSTRAINT [PK_Registration] PRIMARY KEY CLUSTERED ([RegistrationID] ASC)

WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

ALTER TABLE [dbo].[Registration]
WITH CHECK
    ADD CONSTRAINT [FK_Registration_Courses1] FOREIGN KEY([CourseID])
    REFERENCES [dbo].[Courses] ([CourseID])

ALTER TABLE [dbo].[Registration] CHECK CONSTRAINT [FK_Registration_Courses1]

ALTER TABLE [dbo].[Registration]
    WITH CHECK ADD  CONSTRAINT [FK_Registration_Studies] FOREIGN KEY([StudiumID])
    REFERENCES [dbo].[Studies] ([StudiumID])

ALTER TABLE [dbo].[Registration]
CHECK CONSTRAINT [FK_Registration_Studies]

ALTER TABLE [dbo].[Registration]
    WITH CHECK
    ADD  CONSTRAINT [CK_Registration]
    CHECK (([Status]='completed' OR [Status]='in_progress' OR [Status]='failed'))

ALTER TABLE [dbo].[Registration] CHECK CONSTRAINT [CK_Registration]
```

Students - tabela zawiera studentów z tabeli Users

- UserID - Identyfikator w tabeli Users
- Saldo - ilość pieniedzy jakie student ma na swoim indywidualnym koncie

```
CREATE TABLE [dbo].[Students](
    [UserID] [int] NOT NULL,
    [Saldo] [money] NULL,
    CONSTRAINT [PK_Students] PRIMARY KEY CLUSTERED ([UserID] ASC)

    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
    ) ON [PRIMARY]

    ALTER TABLE [dbo].[Students]
```

```
    WITH CHECK ADD  CONSTRAINT [FK_Students_Users] FOREIGN KEY([UserID])
    REFERENCES [dbo].[Users] ([UserID])

    ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [FK_Students_Users]
```

Studies - zawiera informacje dotyczące kierunków studiów

- StudiumID - klucz główny identyfikatora studiów
- Name - nazwa kierunku
- Syllabus - opis studiów
- EntryFee - wpisowe na studia
- LanguageID - połączone ze słownikiem Languages, język w którym odbywają się studia
- StudyType - 1/2/3 (używane w słowniku EventTypes)
- NumberOfAvaliableSeats - liczba wolnych miejsc na dany kierunek
- StartDate - data rozpoczęcia studiów
- EndDate - data zakończenia studiów

```
CREATE TABLE [dbo].[Studies](
    [StudiumID] [int] NOT NULL,
    [Name] [varchar](50) NOT NULL,
    [Syllabus] [text] NOT NULL,
    [EntryFee] [money] NULL,
    [LanguageID] [int] NULL,
    [StudyType] [int] NULL,
    [NumberOfSeats] [int] NULL,
    [StartDate] [datetime] NULL,
    [EndDate] [datetime] NULL
    ) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

    ALTER TABLE [dbo].[Studies]
    ADD  CONSTRAINT [PK_Studies] PRIMARY KEY CLUSTERED
    (
        [StudiumID] ASC
    )
    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Studies]
    WITH CHECK ADD  CONSTRAINT [FK_Studies_EventTypes]
     FOREIGN KEY([StudyType])
    REFERENCES [dbo].[EventTypes] ([EventTypeID])

    ALTER TABLE [dbo].[Studies]
    CHECK CONSTRAINT [FK_Studies_EventTypes]

    ALTER TABLE [dbo].[Studies]
    WITH CHECK ADD  CONSTRAINT [FK_Studies_Languages] FOREIGN KEY([LanguageID])
    REFERENCES [dbo].[Languages] ([LanguageID])

    ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT [FK_Studies_Languages]
```

StudyAttendance - Zawiera informacje o obecności studenta na zajęciach na studiach

- MeetingID - identyfikator webinaru
- UserID - identyfikator studenta

```
CREATE TABLE [dbo].[StudyAttendance](
    [MeetingID] [int] NOT NULL,
    [UserID] [int] NOT NULL
    ) ON [PRIMARY]

    ALTER TABLE [dbo].[StudyAttendance]
    ADD  CONSTRAINT [PK_StudyAttendance] PRIMARY KEY CLUSTERED
    (
        [MeetingID] ASC,
        [UserID] ASC
    )
    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[StudyAttendance]
    WITH CHECK ADD  CONSTRAINT [FK_StudyAttendance_Students] FOREIGN KEY([UserID])
    REFERENCES [dbo].[Students] ([UserID])

    ALTER TABLE [dbo].[StudyAttendance]
    CHECK CONSTRAINT [FK_StudyAttendance_Students]

    ALTER TABLE [dbo].[StudyAttendance]
    WITH CHECK ADD  CONSTRAINT [FK_StudyAttendance_StudyMeetings]
    FOREIGN KEY([MeetingID])
    REFERENCES [dbo].[StudyMeetings] ([StudyMeetingsID])

    ALTER TABLE [dbo].[StudyAttendance]
    CHECK CONSTRAINT [FK_StudyAttendance_StudyMeetings]
```

StudyMeetings - Zawiera informacje o zajęciach w ramach jednego kierunku

- StudyMeetingsID - Identyfikator spotkania
- GatheringID (FK) - identyfikator zjazdu
- TeacherID (FK) - identyfikator nauczyciela prowadzącego zajęcia
- TranslatorID (FK) - identyfikator tłumacza
- MeetingType - 1/2/3, połączone ze słownikiem EventTypes
- Room - numer sali w której odbywają się zajęcia
- StartDateAndTime - data i godzina o której odbędą się zajęcia
- StudyMeetingDuration - czas trwania spotkania
- Link - link do zewnętrznego komunikatora (jeżeli stacjonarne to NULL)
- StudyMeetingPrice - koszt zjazdu

```
CREATE TABLE [dbo].[StudyMeetings](
    [StudyMeetingsID] [int] NOT NULL,
    [GatheringID] [int] NULL,
    [TeacherID] [int] NULL,
    [TranslatorID] [int] NULL,
    [MeetingType] [int] NULL,
    [Room] [varchar](50) NULL,
    [StartDateAndTime] [datetime] NULL,
    [StudyMeetingDuration] [time](7) NULL,
    [Link] [text] NULL,
    [StudyMeetingPrice] [money] NULL
    ) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
    GO
    ALTER TABLE [dbo].[StudyMeetings]
     ADD  CONSTRAINT [PK_StudyMeetings] PRIMARY KEY CLUSTERED
    (
```

```
        [StudyMeetingsID] ASC
    )
    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[StudyMeetings]
    WITH CHECK ADD  CONSTRAINT [FK_StudyMeetings_EventTypes]
     FOREIGN KEY([MeetingType])
    REFERENCES [dbo].[EventTypes] ([EventTypeID])

    ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [FK_StudyMeetings_EventTypes]

    ALTER TABLE [dbo].[StudyMeetings]
    WITH CHECK
    ADD CONSTRAINT [FK_StudyMeetings_Gatherings]
    FOREIGN KEY([GatheringID])
    REFERENCES [dbo].[Gatherings] ([GatheringID])

    ALTER TABLE [dbo].[StudyMeetings]
    CHECK CONSTRAINT [FK_StudyMeetings_Gatherings]

    ALTER TABLE [dbo].[StudyMeetings]
    WITH CHECK ADD  CONSTRAINT [FK_StudyMeetings_Teachers] FOREIGN KEY([TeacherID])
    REFERENCES [dbo].[Teachers] ([TeacherID])

    ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [FK_StudyMeetings_Teachers]

    ALTER TABLE [dbo].[StudyMeetings]
    WITH CHECK ADD  CONSTRAINT [FK_StudyMeetings_Translators]
    FOREIGN KEY([TranslatorID])
    REFERENCES [dbo].[Translators] ([TranslatorID])

    ALTER TABLE [dbo].[StudyMeetings]
    CHECK CONSTRAINT [FK_StudyMeetings_Translators]
```

Teachers - tabela zawiera ID użytkowników, którzy są nauczycielami

```
CREATE TABLE [dbo].[Teachers](
    [TeacherID] [int] NOT NULL) ON [PRIMARY]
    ALTER TABLE [dbo].[Teachers]
    ADD  CONSTRAINT [PK_Teachers] PRIMARY KEY CLUSTERED ([TeacherID] ASC)
    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Teachers]
    WITH CHECK ADD  CONSTRAINT [FK_Teachers_Employees]
    FOREIGN KEY([TeacherID])
    REFERENCES [dbo].[Employees] ([UserID])

    ALTER TABLE [dbo].[Teachers] CHECK CONSTRAINT [FK_Teachers_Employees]
```

TranslatorLanguages - tabela pośrednia, łączy tłamacza z językiem

- TranslatorID - id tłumacza

- LanguageID - id języka

```
CREATE TABLE [dbo].[TranslatorLanguages](
    [TranslatorID] [int] NOT NULL,
    [LanguageID] [int] NOT NULL) ON [PRIMARY]

    ALTER TABLE [dbo].[TranslatorLanguages] ADD  CONSTRAINT [PK_TranslatorLanguages] PRIMARY KEY
CLUSTERED (
    [TranslatorID] ASC,
    [LanguageID] ASC)

    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF, SORT_IN_TEMPDB = OFF, IGNORE_DUP_KEY = OFF, ONLINE = OFF,
ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[TranslatorLanguages]  WITH CHECK ADD  CONSTRAINT
[FK_TranslatorLanguages_Languages] FOREIGN KEY([LanguageID])
    REFERENCES [dbo].[Languages] ([LanguageID])

    ALTER TABLE [dbo].[TranslatorLanguages] CHECK CONSTRAINT [FK_TranslatorLanguages_Languages]

    ALTER TABLE [dbo].[TranslatorLanguages]  WITH CHECK ADD  CONSTRAINT
[FK_TranslatorLanguages_Translators] FOREIGN KEY([TranslatorID])
    REFERENCES [dbo].[Translators] ([TranslatorID])

    ALTER TABLE [dbo].[TranslatorLanguages] CHECK CONSTRAINT [FK_TranslatorLanguages_Translators]
```

Translators - identyfikatory użytkowników którzy są tłumaczami

```
CREATE TABLE [dbo].[Translators](
    [TranslatorID] [int] NOT NULL
    ) ON [PRIMARY]

    ALTER TABLE [dbo].[Translators]
    ADD  CONSTRAINT [PK_Translators] PRIMARY KEY CLUSTERED([TranslatorID] ASC)

    WITH (PAD_INDEX = OFF,
    STATISTICS_NORECOMPUTE = OFF,
    SORT_IN_TEMPDB = OFF,
    IGNORE_DUP_KEY = OFF,
    ONLINE = OFF,
    ALLOW_ROW_LOCKS = ON,
    ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Translators]
    WITH CHECK
    ADD  CONSTRAINT
    [FK_Translators_Employees] FOREIGN KEY([TranslatorID])
    REFERENCES [dbo].[Employees] ([UserID])

    ALTER TABLE [dbo].[Translators] CHECK CONSTRAINT [FK_Translators_Employees]
```

Users - tabela zawiera wszystkich użytkowników w systemie (studentów i pracowników)

- UserID (PK)- Identyfikator użytkownika
- Firstname - imię użytkownika
- LastName - nazwisko użytkownika
- Email - email użytkownika
- Password - hasło użytkownika
- Country
- City
- ZipCode

- Street
- HouseNumber

```
CREATE TABLE [dbo].[Users](
    [UserID] [int] IDENTITY(1,1) NOT NULL,
    [FirstName] [varchar](50) NOT NULL,
    [LastName] [varchar](50) NOT NULL,
    [Email] [varchar](50) NOT NULL,
    [Password] [varchar](50) NOT NULL,
    [Country] [varchar](50) NOT NULL,
    [City] [varchar](50) NOT NULL,
    [ZipCode] [varchar](50) NOT NULL,
    [Street] [varchar](50) NOT NULL,
    [HouseNumber] [varchar](50) NOT NULL)
    ON [PRIMARY]

    ALTER TABLE [dbo].[Users]
    ADD  CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED ([UserID] ASC)
    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[Users]
    WITH CHECK
    ADD  CONSTRAINT [CK_Users] CHECK  (([Email] like '%%@%.%'))
    ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CK_Users]
```

WebinarAttendance - tabela zawierająca informacje o użytkownikach obecnych na webinarze

- MeetingID (FK) - identyfikator webinaru
- UserID (FK) - identyfikator uzytkownika

```
CREATE TABLE [dbo].[WebinarAttendance](
    [MeetingID] [int] NOT NULL,
    [UserID] [int] NOT NULL) ON [PRIMARY]

    ALTER TABLE [dbo].[WebinarAttendance] ADD  CONSTRAINT [PK_WebinarAttendance] PRIMARY KEY CLUSTERED
    (
    [MeetingID] ASC,
    [UserID] ASC
    )
    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        SORT_IN_TEMPDB = OFF,
        IGNORE_DUP_KEY = OFF,
        ONLINE = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]

    ALTER TABLE [dbo].[WebinarAttendance]
    WITH CHECK ADD  CONSTRAINT [FK_WebinarAttendance_Students]
     FOREIGN KEY([UserID])
    REFERENCES [dbo].[Students] ([UserID])

    ALTER TABLE [dbo].[WebinarAttendance]
     CHECK CONSTRAINT [FK_WebinarAttendance_Students]

    ALTER TABLE [dbo].[WebinarAttendance]
    WITH CHECK ADD  CONSTRAINT [FK_WebinarAttendance_Webinarium]
     FOREIGN KEY([MeetingID])
```

```
    REFERENCES [dbo].[Webinarium] ([WebinarID])

    ALTER TABLE [dbo].[WebinarAttendance]
    CHECK CONSTRAINT [FK_WebinarAttendance_Webinarium]
```

Webinarium - tabela zawierająca wszystkie webinary w systemie

- WebinarID (PK) - identyfikator webinaru
- Name - nazwa webinaru
- Descryption - opis webinaru
- TeacherID (FK) - identyfikator nauczyciela prowadzącego zajęcia
- TranslatorID (FK) - identyfikator tłumacza
- LanguageID - id języka webinarium
- Link (to external website) - link do webinaru
- StartDateAndTime - czas rozpoczęcia webinarium
- EndDateAndTime - czas zakończenia wevinarium
- WebinarDuration - czas trwania webinaru
- Price - cena webinaru (jeśli free to 0)

```
CREATE TABLE [dbo].[Webinarium](
    [WebinarID] [int] NOT NULL,
    [Name] [char](30) NOT NULL,
    [TeacherID] [int] NULL,
    [TranslatorID] [int] NULL,
    [Description] [text] NULL,
    [Price] [money] NOT NULL,
    [Link] [text] NULL,
    [StartDateAndTime] [datetime] NULL,
    [WebinarDuration] [time](7) NULL,
    [Language] [nchar](30) NULL,
 CONSTRAINT [PK_Webinarium] PRIMARY KEY CLUSTERED
(
    [WebinarID] ASC
)
    WITH (PAD_INDEX = OFF,
        STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY = OFF,
        ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS = ON,
        OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY])
         ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

    ALTER TABLE [dbo].[Webinarium]
    WITH CHECK ADD  CONSTRAINT [FK_Webinarium_Employees1] FOREIGN KEY([TranslatorID])
    REFERENCES [dbo].[Employees] ([UserID])

    ALTER TABLE [dbo].[Webinarium] CHECK CONSTRAINT [FK_Webinarium_Employees1]

    ALTER TABLE [dbo].[Webinarium]
    WITH CHECK ADD  CONSTRAINT [FK_Webinarium_Employees2] FOREIGN KEY([TeacherID])
    REFERENCES [dbo].[Employees] ([UserID])

    ALTER TABLE [dbo].[Webinarium] CHECK CONSTRAINT [FK_Webinarium_Employees2]
```

Widoki:

TeachersInfo Wyświetla informacje o nauczycielach w systemie.

	FirstName	LastName	Email
1	Emma	Davis	emma.davis@email.com
2	Michael	Williams	michael.williams@email.com
3	Michael	Williams	michael.williams@email.com
4	Mary	Johnson	mary.johnson@example.com
5	Emma	Davis	emma.davis@example.com
6	Helga	Schmidt	helga.schmidt@email.com

```
CREATE VIEW [dbo].[TeachersInfo]
AS
SELECT dbo.Users.FirstName, dbo.Users.LastName, dbo.Users.Email
FROM dbo.Teachers
INNER JOIN
    dbo.Employees
    ON dbo.Teachers.TeacherID = dbo.Employees.UserID
    INNER JOIN dbo.Users
    ON dbo.Employees.UserID = dbo.Users.UserID
```

StudentsInfo Wyświetla informacje o studentach w systemie.

	UserID	FirstN…	LastName	Email	Country	City	ZipCode	Street	HouseNumber	Saldo
1	1	John …	Doe …	john.doe@email…	USA …	New York …	10001 …	Main Street …	123	212,00
2	2	Alice …	Smith …	alice.smith@em…	UK …	London …	SW1A 1AA …	High Street …	456	400,00

```
CREATE VIEW [dbo].[StudentsInfo]
AS
SELECT dbo.Users.UserID, dbo.Users.FirstName, dbo.Users.LastName,
dbo.Users.Email, dbo.Users.Country, dbo.Users.City,
dbo.Users.ZipCode, dbo.Users.Street, dbo.Users.HouseNumber,
dbo.Students.Saldo
FROM  dbo.Students
INNER JOIN bo.Users
ON dbo.Students.UserID = dbo.Users.UserID
```

StudentsNotEnrolled Wyświetla informacje o studentach, którzy nie zarejestrowali się na żaden kurs/webinarium/studia.

	UserID	FirstName	LastName
1	3	Bob	Johnson

```
CREATE VIEW [dbo].[StudentsNotEnrolled]
AS
SELECT dbo.Users.UserID, dbo.Users.FirstName, dbo.Users.LastName
FROM   dbo.Users INNER JOIN
             dbo.Students ON dbo.Users.UserID = dbo.Students.UserID LEFT OUTER JOIN
             dbo.Registration ON dbo.Students.UserID = dbo.Registration.UserID
WHERE (dbo.Registration.UserID IS NULL)
```

WebinarsInfo Wyświetla informacje o dostępnych webinarach.

	Name	Description	Languag…	StartDateAndTime	WebinarDur…	Price	Teacher FirstName	Teacher LastName
1	Introduction to SQL …	Learn the basic…	English	2023-01-20 15:00:00.000	02:30:00	29,99	Emma	Davis
2	Web Development Basics …	Explore fundame…	English	2023-02-05 18:30:00.000	01:45:00	39,99	Emma	Davis
3	Data Science Essentials…	Essential conce…	Polish	2023-03-15 12:00:00.000	03:15:00	49,99	Michael	Williams

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

```
SELECT dbo.Webinarium.Name, dbo.Webinarium.Description,
dbo.Languages.LanguageName, dbo.Webinarium.StartDateAndTime,
dbo.Webinarium.WebinarDuration, dbo.Webinarium.Price,
dbo.Users.FirstName AS 'Teacher FirstName',
dbo.Users.LastName AS 'Teacher LastName'
FROM dbo.Webinarium
INNER JOIN
dbo.Languages
 dbo.Webinarium.LanguageID = dbo.Languages.LanguageID
INNER JOIN dbo.Teachers
ON dbo.Webinarium.TeacherID = dbo.Teachers.TeacherID
INNER JOIN dbo.Employees ON dbo.Teachers.TeacherID = dbo.Employees.UserID INNER JOIN
dbo.Users ON dbo.Employees.UserID = dbo.Users.UserID
```

CoursesInfo Wyświetla informacje o dostępnych kursach.

	Name	Description	CourseType	NumberOfSeats	Prepayment	FullPayment	CourseStartDate	CourseEndDate	LanguageNa…
1	Mathematics 101	Introduction to…	1	50	100,00	500,00	2023-01-15 00:00:00.0…	2023-03-15 00:00:00…	English
2	Programming Fundamental…	Fundamental con…	2	70	150,00	600,00	2023-02-01 00:00:00.0…	2023-08-15 00:00:00…	German
3	Graphic Design Basics	Introduction to…	3	80	120,00	550,00	2023-03-10 00:00:00.0…	2023-10-15 00:00:00…	Polish

```
CREATE VIEW [dbo].[CoursesInfo]
AS
SELECT dbo.Courses.Name, dbo.Courses.Description,
dbo.Courses.CourseType, dbo.Courses.NumberOfSeats, dbo.Courses.Prepayment, dbo.Courses.FullPayment,
dbo.Courses.CourseStartDate, dbo.Courses.CourseEndDate,
dbo.Languages.LanguageName
FROM dbo.Courses
INNER JOIN dbo.Languages
ON dbo.Courses.LanguageID = dbo.Languages.LanguageID
```

StudiesInfo Wyświetla informacje o dostępnych kierunkach studiów. Pokazuje nazwe, syllabus, liczbe miejsc, wpisowe, język w jakich są prowadzone.

	Name	Syllabus	EntryFee	LanguageName	NumberOfSe…	StartDate	EndDate
1	Computer Science	Advanced progra…	1500,00	English	47	2023-01-21 12:…	2023-08-21 12:00:00.0…
2	Business Administration…	Finance, market…	2000,00	English	40	2024-01-21 12:…	2024-10-21 12:00:00.0…
3	Medicine	Anatomy, physio…	2500,00	Polish	60	2023-05-15 12:…	2023-07-21 12:00:00.0…

```
CREATE VIEW [dbo].[StudiesInfo]
AS
SELECT dbo.Studies.Name, dbo.Studies.Syllabus, dbo.Studies.EntryFee, dbo.Languages.LanguageName,
dbo.Studies.NumberOfSeats, dbo.Studies.StartDate, dbo.Studies.EndDate
FROM dbo.Studies
INNER JOIN dbo.Languages
ON dbo.Studies.LanguageID = dbo.Languages.LanguageID
```

AttendancePerCourseMeeting Wyświetla statystyki frekwencji na dany course meeting. Pokazuje nazwe kursu, id kursu, id meetingu, frekwencja(jako procent), liczba obecnych, liczba zarejestrowanych.

	Name	CourseID	MeetingID	Attendance	NumberAttended	NumberRegistered	CourseStartDate	CourseEndDate
1	Mathematics 101	1	1	2	2	1	2023-01-15 00:00:00.000	2023-03-15 00:00:00.000
2	Programming Fundamentals	2	2	1	3	3	2023-02-01 00:00:00.000	2023-08-15 00:00:00.000
3	Programming Fundamentals	2	4	0,3333333333333333	1	3	2023-02-01 00:00:00.000	2023-08-15 00:00:00.000

```
CREATE VIEW [dbo].[AttendancePerCourseMeeting]
AS
SELECT dbo.Courses.Name, CAPM.CourseID, CAPM.MeetingID, ISNULL(CASE WHEN CNRU.NumberRegistered = 0 THEN
NULL ELSE CAST(CAPM.NumberAttended AS FLOAT) / CNRU.NumberRegistered END, 0) AS Attendance,
CAPM.NumberAttended, CNRU.NumberRegistered, CNRU.CourseStartDate, CNRU.CourseEndDate
FROM (SELECT dbo.CourseMeetings.CourseID, dbo.CourseMeetings.MeetingID,
```

```
COUNT(dbo.CourseAttendance.UserID) AS NumberAttended
FROM dbo.CourseMeetings
INNER JOIN dbo.CourseAttendance
ON dbo.CourseMeetings.MeetingID = dbo.CourseAttendance.MeetingID
GROUP BY dbo.CourseMeetings.CourseID, dbo.CourseMeetings.MeetingID) AS CAPM INNER JOIN
dbo.CourseNumberRegisteredUsers AS CNRU
ON CAPM.CourseID = CNRU.CourseID
INNER JOIN dbo.Courses
ON CAPM.CourseID = dbo.Courses.CourseID
```

AttendancePerEndedCourse Wyświetla statystyki frekwencji na dany zakończony kurs. Pokazuje id kursu, nazwę kursu, średnią frekwencję.

	CourseID	Name	AverageAttendan…	CourseStartDa…	CourseEndDate	CourseType
1	1	Mathematic…	2	2023-01-15 00:00:…	2023-03-15 00:00…	1
2	2	Programmin…	0,6666666666666666	2023-02-01 00:00:…	2023-08-15 00:00…	2
3	3	Graphic De…	0	2023-03-10 00:00:…	2023-10-15 00:00…	3

```
CREATE VIEW [dbo].[AttendancePerEndedCourse]
AS
    SELECT dbo.Courses.CourseID, dbo.Courses.Name, AVG(dbo.AttendancePerCourseMeeting.Attendance) AS
AverageAttendance, dbo.Courses.CourseStartDate, dbo.Courses.CourseEndDate, dbo.Courses.CourseType
    FROM   dbo.AttendancePerCourseMeeting INNER JOIN
                dbo.Courses ON dbo.AttendancePerCourseMeeting.CourseID = dbo.Courses.CourseID
    WHERE (dbo.Courses.CourseEndDate < GETDATE())
    GROUP BY dbo.Courses.CourseID, dbo.Courses.Name, dbo.Courses.CourseStartDate,
dbo.Courses.CourseEndDate, dbo.Courses.CourseType
```

RegisteredStudentsInfo Zwraca inforamacje o użytkownich zarejestowanych na dane szkolenie

	UserID	LastName	FirstName	Email	Country	City	ZipCode	Street	HouseNumber	CourseID	WebinarID	StudiumID	Registrati…
1	1	Doe	John	john.doe@email…	USA	New York	10001	Main Street	123	NULL	1	NULL	2023-01-20
2	1	Doe	John	john.doe@email…	USA	New York	10001	Main Street	123	2	NULL	NULL	2023-01-20
3	2	Smith	Alice	alice.smith@em…	UK	London	SW1A 1AA	High Street	456	2	NULL	NULL	2023-01-20
4	4	Davis	Emma	emma.davis@ema…	Australia	Sydney	2000	George Street	101	2	NULL	NULL	2023-01-20
5	1	Doe	John	john.doe@email…	USA	New York	10001	Main Street	123	NULL	NULL	1	2023-01-20

```
CREATE VIEW [dbo].[RegisteredStudentsInfo]
AS
SELECT dbo.Users.UserID, dbo.Users.LastName, dbo.Users.FirstName,
dbo.Users.Email, dbo.Users.Country, dbo.Users.City, dbo.Users.ZipCode,
dbo.Users.Street, dbo.Users.HouseNumber, dbo.Registration.CourseID,
dbo.Registration.WebinarID, dbo.Registration.StudiumID,
dbo.Registration.RegistrationDate
FROM  dbo.Registration INNER JOIN
dbo.Students ON dbo.Registration.UserID = dbo.Students.UserID
INNER JOIN dbo.Users ON dbo.Students.UserID = dbo.Users.UserID
```

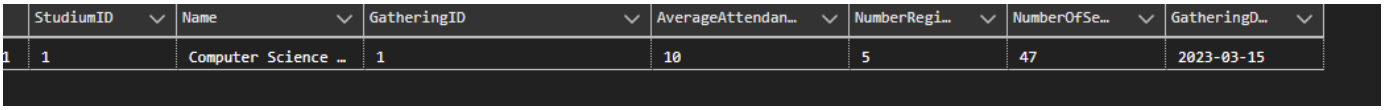
AttendancePerEndedWebinar Wyświetla statystyki frekwencji na dany zakończony webinar. Pokazuje id webinaru, nazwe, frekwencję.

	WebinarID	Name	Attendance	NumberRegiste…	StartDateAndT…	EndDateAndTime
1	1	Introducti…	2	1	2023-01-20 15:00…	2023-01-20 17:30:00…

```
CREATE VIEW [dbo].[AttendancePerEndedWebinar]
AS
    SELECT Webinarium_1.WebinarID, Webinarium_1.Name, CASE WHEN NumberRegistered = 0 THEN NULL ELSE
CAST(WAPM.NumberAttended AS FLOAT) / NumberRegistered END AS Attendance, Subquery.NumberRegistered,
Webinarium_1.StartDateAndTime,
    Webinarium_1.EndDateAndTime
    FROM   (SELECT dbo.Webinarium.WebinarID, dbo.Webinarium.Name, COUNT(dbo.WebinarAttendance.UserID) AS
NumberAttended
    FROM dbo.Webinarium INNER JOIN
    dbo.WebinarAttendance ON dbo.Webinarium.WebinarID = dbo.WebinarAttendance.MeetingID
    GROUP BY dbo.Webinarium.WebinarID, dbo.Webinarium.Name) AS WAPM
```

```
    INNER JOIN dbo.Webinarium AS Webinarium_1
    ON WAPM.WebinarID = Webinarium_1.WebinarID LEFT OUTER JOIN (SELECT WebinarID, COUNT(UserID) AS
NumberRegistered
    FROM dbo.Registration
    GROUP BY WebinarID) AS Subquery ON Webinarium_1.WebinarID = Subquery.WebinarID
    WHERE (Webinarium_1.EndDateAndTime < GETDATE())
```

AttendancePerEndedGathering Wyświetla statystyki frekwencji na dany zakończony zlot.



	StudiumID	Name	GatheringID	AverageAttendan…	NumberRegi…	NumberOfSe…	GatheringD…
1	1	Computer Science …	1	10	5	47	2023-03-15

```
CREATE VIEW [dbo].[AttendancePerEndedGathering]
AS
SELECT dbo.Gatherings.StudiumID, dbo.StudiesNumberRegisteredUsers.Name,
dbo.Gatherings.GatheringID,
CASE WHEN ISNULL(dbo.StudiesNumberRegisteredUsers.NumberRegistered, 0) = 0
THEN NULL ELSE CAST(SUM(SMAP.NumberAttended) AS FLOAT)
             / COUNT(SMAP.StudyMeetingsID) * dbo.StudiesNumberRegisteredUsers.NumberRegistered
             ND AS AverageAttendance,
             dbo.StudiesNumberRegisteredUsers.NumberRegistered,
             dbo.StudiesNumberRegisteredUsers.NumberOfSeats,
              dbo.Gatherings.GatheringDate
              FROM   (SELECT dbo.StudyMeetings.StudyMeetingsID, dbo.StudyMeetings.GatheringID,
COUNT(dbo.StudyAttendance.UserID) AS NumberAttended
            FROM    dbo.StudyMeetings INNER JOIN
            dbo.StudyAttendance
            ON dbo.StudyMeetings.StudyMeetingsID = dbo.StudyAttendance.MeetingID
            GROUP BY dbo.StudyMeetings.GatheringID, dbo.StudyMeetings.StudyMeetingsID) AS SMAP
            INNER JOIN dbo.Gatherings ON dbo.Gatherings.GatheringID = SMAP.GatheringID
            INNER JOIN dbo.StudiesNumberRegisteredUsers ON dbo.Gatherings.StudiumID =
dbo.StudiesNumberRegisteredUsers.StudiumID
WHERE (dbo.Gatherings.GatheringDate < GETDATE())
GROUP BY dbo.Gatherings.StudiumID, dbo.StudiesNumberRegisteredUsers.Name,
dbo.Gatherings.GatheringID, dbo.StudiesNumberRegisteredUsers.NumberRegistered,
 dbo.StudiesNumberRegisteredUsers.NumberOfSeats, dbo.Gatherings.GatheringDate,
dbo.StudiesNumberRegisteredUsers.NumberRegistered
```

CourseNumberRegisteredUsers Wyświetla liczbe osob, które ą zarejestrowane na podany kurs

	CourseID	Name	NumberRegistered	NumberOfSeats	CourseStar…	CourseEndD…
1	1	Mathematics 101 …	1	50	2023-01-15 00:…	2023-03-15 00:…
2	2	Programming Funda…	3	70	2023-02-01 00:…	2023-08-15 00:…
3	3	Graphic Design Ba…	0	80	2023-03-10 00:…	2023-10-15 00:…

```
CREATE VIEW [dbo].[CourseNumberRegisteredUsers]
AS
SELECT dbo.Courses.CourseID, dbo.Courses.Name,
COUNT(dbo.Registration.RegistrationID) AS NumberRegistered,
 dbo.Courses.NumberOfSeats, dbo.Courses.CourseStartDate,
 dbo.Courses.CourseEndDate
FROM   dbo.Registration RIGHT OUTER JOIN
            dbo.Courses
            ON dbo.Registration.CourseID = dbo.Courses.CourseID
GROUP BY dbo.Courses.CourseID, dbo.Courses.Name,
dbo.Courses.NumberOfSeats, dbo.Courses.CourseStartDate,
 dbo.Courses.CourseEndDate
```

CoursesInfo Wyświetla informacje o kursach

	Name	Descr…	CourseType	Number…	Prepayme…	FullPayment	CourseStar…	CourseEndD…	LanguageNa…
1	Mathematics 101 …	Introduct…	1	50	100,00	500,00	2023-01-15 00:…	2023-03-15 00:…	English
2	Programming Fund…	Fundament…	2	70	150,00	600,00	2023-02-01 00:…	2023-08-15 00:…	German
3	Graphic Design B…	Introduct…	3	80	120,00	550,00	2023-03-10 00:…	2023-10-15 00:…	Polish

```
CREATE VIEW [dbo].[CoursesInfo]
AS
SELECT dbo.Courses.Name, dbo.Courses.Description, dbo.Courses.CourseType, dbo.Courses.NumberOfSeats,
dbo.Courses.Prepayment, dbo.Courses.FullPayment, dbo.Courses.CourseStartDate, dbo.Courses.CourseEndDate,
dbo.Languages.LanguageName
FROM dbo.Courses
INNER JOIN dbo.Languages
ON dbo.Courses.LanguageID = dbo.Languages.LanguageID
```

FutureCourses Wyświetla przyszłe kursy

	CourseID	Name	CourseStartDate
1	4	Web Desing	2025-01-20 15:00:00…

```
SELECT        CourseID, Name, CourseStartDate
FROM            dbo.Courses
WHERE        (CourseStartDate > GETDATE())
```

FutureStudies Wyświetla przyszłe studia

```
SELECT        StudiumID, Name, StartDate
FROM            dbo.Studies
WHERE        (StartDate > GETDATE())
```

FutureWebinars Wyświetla webinary, które odbęda sie w przyszłosści.

```
SELECT        WebinarID, Name, StartDateAndTime
FROM            dbo.Webinarium
WHERE       (StartDateAndTime > GETDATE())
```

FutuEvents Wyświetla informacje o przyszłych wydarzeniach. Wywołuje funkcje, która jest podana dalej.

```
SELECT        t, CourseID, Name, CourseStartDate
FROM            dbo.GetFutureEvents() AS GetFutureEvents_1
```

DebtorsList Zwraca liste dłużników

	FirstName	LastName	UserID
	Alice …	Smith	2
	Helga …	Schmidt	20

```
CREATE VIEW [dbo].[DebtorsList]
AS
SELECT DISTINCT dbo.Users.FirstName, dbo.Users.LastName, dbo.Receivables.UserID
FROM dbo.Receivables INNER JOIN
dbo.Students ON dbo.Receivables.UserID = dbo.Students.UserID INNER JOIN
dbo.Users ON dbo.Students.UserID = dbo.Users.UserID
WHERE (dbo.Receivables.TransactionStatus = 'pending') AND (dbo.Receivables.DueDate < GETDATE())
```

FinancialReportCourses Wyświetla raport finansowy kursów

	CourseID	Name	Total Income
1	1	Mathematics 101 …	0,00
2	2	Programming Fundamentals …	0,00
3	3	Graphic Design Basics …	0,00
4	4	Web Desing	0,00

```
CREATE VIEW [dbo].[FinancialRaportCourses]
AS
SELECT cs.CourseID, cs.Name, SUM(ISNULL(rec.Amount, 0)) AS 'Total Income'
FROM dbo.Courses AS cs LEFT OUTER JOIN
dbo.Registration AS reg ON cs.CourseID = reg.CourseID LEFT OUTER JOIN
dbo.Receivables AS rec ON rec.RegistrationID = reg.RegistrationID AND rec.TransactionStatus = 'paid'
GROUP BY cs.CourseID, cs.Name
```

FinancialReportStudies Wyświetla raport finasowy studiów

	StudiumID	Name	Total Income
1	1	Computer Science …	100,00
2	2	Business Administration …	0,00
3	3	Medicine …	0,00

```
SELECT s.StudiumID, s.Name, SUM(ISNULL(rec.Amount, 0)) AS 'Total Income'
FROM dbo.Studies AS s LEFT OUTER JOIN
dbo.Registration AS reg ON s.StudiumID = reg.StudiumID LEFT OUTER JOIN
dbo.Receivables AS rec ON rec.RegistrationID = reg.RegistrationID
AND rec.TransactionStatus = 'paid'
GROUP BY s.StudiumID, s.Name
```

FiancialReportWebinars Zwraca raport finansowy webinarów

	WebinarID	Name	Total Income
1	1	Introduction to SQL	0,00
2	2	Web Development Basics	0,00
3	3	Data Science Essentials	0,00

```
CREATE VIEW [dbo].[FinancialRaportWebinars]
AS
SELECT wb.WebinarID, wb.Name, SUM(ISNULL(rec.Amount, 0)) AS 'Total Income'
FROM dbo.Webinarium AS wb LEFT OUTER JOIN
dbo.Registration AS reg
ON wb.WebinarID = reg.WebinarID LEFT OUTER JOIN
dbo.Receivables AS rec
ON rec.RegistrationID = reg.RegistrationID AND rec.TransactionStatus = 'paid'
GROUP BY wb.WebinarID, wb.Name
```

FutureEventsRegistrationNumber Zwraca liczbe osób zapisanych na przyszłe wydarzenia

	C/S/W	CourseID	Name	RegisteredNumber	EventType
1	Webinar	2	Web Development Basics	4	online
2	Webinar	3	Data Science Essentials	1	online
3	Course	4	Web Desing	1	stationary
4	Webinar	4	Marketing	1	online
5	Webinar	5	test	0	online

```
CREATE VIEW [dbo].[FutureEventsRegistrationNumber]
    AS
    SELECT GFE.[C/S/W], GFE.CourseID, GFE.Name, COUNT(dbo.Registration.RegistrationID) AS
RegisteredNumber, dbo.EventTypes.EventType
    FROM dbo.GetFutureEvents() AS GFE
    LEFT OUTER JOIN
    dbo.Registration
    ON GFE.CourseID = dbo.Registration.WebinarID OR GFE.CourseID = dbo.Registration.StudiumID OR
GFE.CourseID = dbo.Registration.CourseID
    LEFT OUTER JOIN dbo.EventTypes
    ON GFE.CourseType = dbo.EventTypes.EventTypeID
    GROUP BY GFE.CourseID, GFE.[C/S/W], GFE.Name, dbo.EventTypes.EventType
```

GatheringsForEachStudies Wyświetla zloty dla każdych studiów

StudiumID	GatheringID	GatheringPrice	GatheringDate
1	1	100,00	2023-03-15
2	2	75,50	2023-04-20
3	3	120,00	2023-05-10
1	4	100,00	2023-04-01

```
CREATE VIEW [dbo].[GatheringsForEachStudies]
AS
SELECT TOP (100) PERCENT dbo.Studies.StudiumID, dbo.Gatherings.GatheringID,
dbo.Gatherings.GatheringPrice, dbo.Gatherings.GatheringDate
FROM dbo.Studies
INNER JOIN dbo.Gatherings
ON dbo.Studies.StudiumID = dbo.Gatherings.StudiumID
```

Graduates Zwraca tabele w której znajdują sie osoby które ukończyły jakies szkolenie z pozytywnym wynikiem

	UserID	LastName	FirstName	Email	Country	City	ZipCode	Street	HouseNumber	CourseID	WebinarID	StudiumID
1	1	Doe	John	john.doe@email…	USA …	New York …	10001 …	Main Street …	123	NULL	1	NULL

```
CREATE VIEW [dbo].[Graduates]
AS
SELECTdbo.Users.UserID, dbo.Users.LastName, dbo.Users.FirstName,
dbo.Users.Email, dbo.Users.Country, dbo.Users.City,
dbo.Users.ZipCode, dbo.Users.Street, dbo.Users.HouseNumber,
dbo.Registration.CourseID, dbo.Registration.WebinarID, dbo.Registration StudiumID,
dbo.Registration.RegistrationDate
FROM dbo.Registration
INNER JOIN dbo.Students
ON dbo.Registration.UserID = dbo.Students.UserID
INNER JOIN dbo.Users ON dbo.Students.UserID = dbo.Users.UserID
WHERE (dbo.Registration.Status = 'completed')
```

Procedury:

AddCourseAttendance Procedura dodaje obecność do listy obecności na podanym spotkaniu w obrębie kursu

```
CREATE PROCEDURE [dbo].[AddCourseAttendance]
    @MeetingID int,
    @UserID int
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (SELECT 1 FROM dbo.CourseMeetings as cm
        WHERE cm.MeetingID = @MeetingID
        AND (cm.CourseID IN (SELECT reg.CourseID FROM dbo.Registration as reg WHERE @UserID =
reg.UserID))
        )
    BEGIN
        INSERT INTO dbo.CourseAttendance (MeetingID, UserID)
        VALUES (@MeetingID, @UserID);
    END
    ELSE
    BEGIN
        RAISERROR('UserID is not assigned to the specified CourseID.',-1,-1)
    END
END
GO
```

AddPaymentUser Dodaje płatność do tabeli payments.

```
CREATE PROCEDURE [dbo].[AddPaymentUser]
    @UserID INT,
    @Amount MONEY
AS
BEGIN
    SET NOCOUNT ON;
    INSERT INTO [dbo].[Payments] ([UserID],[Date],[Amount])
    VALUES (@UserID,GETDATE(),@Amount);
END
GO
```

AddRegistrationAdmin Procedura rejestracji na wybrany kurs. Używana przez admina.

```
CREATE PROCEDURE [dbo].[AddRegistrationAdmin]
    @EventType TEXT,
    @UserID INT,
    @RegistrationDate DATE,
    @Attendance FLOAT,
    @Status VARCHAR(50),
    @CourseID INT = NULL,
    @WebinarID INT = NULL,
    @StudiumID INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO [dbo].[Registration] (
        [EventType],
        [UserID],
        [RegistrationDate],
        [Attendance],
        [Status],
        [CourseID],
        [WebinarID],
        [StudiumID]
    )
```

```
    VALUES (
        @EventType,
        @UserID,
        @RegistrationDate,
        @Attendance,
        @Status,
        @CourseID,
        @WebinarID,
        @StudiumID
    );
END
GO
```

AddRegistrationUser Rejestracja użytkownika na event.

```
CREATE PROCEDURE [dbo].[AddRegistrationUser]
    @EventType VARCHAR(MAX),
    @UserID INT,
    @CourseID INT = NULL,
    @WebinarID INT = NULL,
    @StudiumID INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN
        IF @EventType = 'Course'
            BEGIN
                IF (SELECT TOP 1 NumberOfSeats FROM Courses WHERE CourseID = @CourseID) >= 1
                    BEGIN
                        INSERT INTO [dbo].[Registration] (
                            [EventType],
                            [RegistrationID],
                            [UserID],
                            [RegistrationDate],
                            [Status],
                            [CourseID],
                            [WebinarID],
                            [StudiumID]
                        )
                        VALUES (
                            'Course',
                            (SELECT MAX(RegistrationID) +1 FROM Registration),
                            @UserID,
                            GETDATE(),
                            'in_progress',
                            @CourseID,
                            @WebinarID,
                            @StudiumID
                        );
                        UPDATE dbo.Courses
                        SET NumberOfSeats = CASE WHEN NumberOfSeats > 0 THEN NumberOfSeats - 1 ELSE 0 END
                        WHERE Courses.CourseID = @CourseID;
                    END
                ELSE
                    BEGIN
                        -- Obsługa błędu - brak miejsc
                        RAISERROR ('Brak miejsc', -1,-1)
                    END
            END
        ELSE IF LTRIM(RTRIM(@EventType))= 'Studium'
            BEGIN
                IF (SELECT TOP 1 NumberOfSeats FROM Studies WHERE StudiumID = @StudiumID) >= 1
                    BEGIN
                        INSERT INTO [dbo].[Registration] (
```

```
                            [EventType],
                            [RegistrationID],
                            [UserID],
                            [RegistrationDate],
                            [Status],
                            [CourseID],
                            [WebinarID],
                            [StudiumID]
                        )
                        VALUES (
                            'Studium',
                            (SELECT MAX(RegistrationID) +1 FROM Registration),
                            @UserID,
                            GETDATE(),
                            'in_progress',
                            @CourseID,
                            @WebinarID,
                            @StudiumID
                        );
                        UPDATE dbo.Studies
                        SET NumberOfSeats = CASE WHEN NumberOfSeats > 0 THEN NumberOfSeats - 1 ELSE 0 END
                        WHERE Studies.StudiumID = @StudiumID;
                    END
                ELSE
                    BEGIN
                        -- Obsługa błędu - brak miejsc
                        RAISERROR ('Brak miejsc', -1,-1)
                    END
            END
        ELSE IF @EventType = 'Webinar'
            BEGIN
                -- Brak limitu miejsc dla webinaru, więc nie ma sprawdzania
                INSERT INTO [dbo].[Registration] (
                    [EventType],
                    [RegistrationID],
                    [UserID],
                    [RegistrationDate],
                    [Status],
                    [CourseID],
                    [WebinarID],
                    [StudiumID]
                )
                VALUES (
                    'Webinar',
                    (SELECT MAX(RegistrationID) + 1 FROM Registration),
                    @UserID,
                    GETDATE(),
                    'in_progress',
                    @CourseID,
                    @WebinarID,
                    @StudiumID
                );
            END
        ELSE
            BEGIN
                -- Obsługa błędu - nieobsługiwany rodzaj wydarzenia
                    RAISERROR ('Nieobslugiwane dane', -1,-1)
            END
    END
END
GO
```

AddStudyAttendance Procedura dodaje użytkownia do listy obecności danego spotkania na danych studiach.

```
CREATE PROCEDURE [dbo].[AddStudyAttendance]
    @MeetingID int,
    @UserID int
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @StudiumID int;

    -- Pobierz StudiumID z CTE i przypisz do zmiennej
    ;WITH currentStudyID AS (
        SELECT TOP 1 StudiumID
        FROM Gatherings ga
        JOIN StudyMeetings smt
        ON ga.GatheringID = smt.GatheringID
        WHERE smt.StudyMeetingsID = @MeetingID
    )
    SELECT @StudiumID = StudiumID FROM currentStudyID;

    -- Sprawdź, czy UserID jest zapisany na studia, na któym odbywają się te zajecia
    IF EXISTS (SELECT 1 FROM dbo.StudyMeetings as sm
        WHERE sm.StudyMeetingsID = @MeetingID AND (@StudiumID  IN (SELECT reg.StudiumID FROM
dbo.Registration as reg WHERE @UserID = reg.UserID)))
        BEGIN
            -- UserID jest przypisany do studiów, więc dodaj rekord do tabeli StudyAttendance
            INSERT INTO dbo.CourseAttendance (MeetingID, UserID)
            VALUES (@MeetingID, @UserID);
        END
    ELSE
        BEGIN
            RAISERROR('UserID is not assigned to the specified StudyID.',-1,-1)
        END
END
GO
```

AddUser Procedura dodaje użytkownika do systemu.

```
CREATE PROCEDURE [dbo].[AddUser]
    @FirstName nchar(20),
    @LastName nchar(20),
    @Email nchar(30),
    @Password nchar(30),
    @Country nchar(20),
    @City nchar(20),
    @ZipCode nchar(20),
    @Street nchar(30),
    @HouseNumber nchar(10)
AS
BEGIN
    INSERT INTO [dbo].[Users] (
        [FirstName],
        [LastName],
        [Email],
        [Password],
        [Country],
        [City],
        [ZipCode],
        [Street],
        [HouseNumber]
    )
    VALUES (
        @FirstName,
        @LastName,
        @Email,
```

```
        @Password,
        @Country,
        @City,
        @ZipCode,
        @Street,
        @HouseNumber
    )
END
GO
```

AddWebinarAttendance Procedura dodaje użytkownika do listy obecności na danym webinarze.

```
CREATE PROCEDURE [dbo].[AddWebinarAttendance]
    @MeetingID int,
    @UserID int
AS
BEGIN
    SET NOCOUNT ON;
    -- Sprawdź,czy istnieje taki webinar i czy użytkownik jest do niego przypisany
    IF EXISTS (SELECT 1 FROM dbo.Webinarium as wb
        WHERE @MeetingID = wb.WebinarID AND
            (@MeetingID IN (SELECT reg.WebinarID FROM dbo.Registration as reg WHERE @UserID =
reg.UserID))
    )
        BEGIN
            -- Jesli warunki się zgadzają wstaw rekord
            INSERT INTO dbo.CourseAttendance (MeetingID, UserID)
            VALUES (@MeetingID, @UserID);
        END
    ELSE
        BEGIN
            -- Jeżeli warunki się nie zgadzają wyświetl wiadomośc
            RAISERROR('MeetingID or UserID invalid',-1,-1)
        END
END
GO
```

CreateCourse Procedura tworzy kurs.

```
CREATE PROCEDURE [dbo].[CreateCourse]
    @CourseName varchar(50),
    @CourseDescription text,
    @Prepayment money,
    @FullPayment money,
    @CourseStartDate datetime,
    @CourseType int,
    @Language varchar(50),
    @CourseEndDate datetime,
    @NumberOfSeats int
AS
BEGIN
    INSERT INTO [dbo].[Courses] (
        [Name], [Description], [Prepayment], [FullPayment],
        [CourseStartDate], [Language], [CourseType],
        [CourseEndDate], [NumberOfSeats]
    )
    VALUES (
        @CourseName, @CourseDescription, @Prepayment, @FullPayment,
        @CourseStartDate, @Language,@CourseType,
        @CourseEndDate, @NumberOfSeats
    )
    SELECT SCOPE_IDENTITY() AS NewCourseID
```

```
END
GO
```

CreateInternship Procedura tworzy internship, wypełniając InternshipAttendance, ExamGrade, AttendanceDays domyślnymi danymi.

```
CREATE PROCEDURE [dbo].[CreateInternship]
    @StudiumID INT,
    @RegistrationID INT
AS
BEGIN
    INSERT INTO [dbo].[Internships]
           ([InternshipID]
           ,[StudiumID]
           ,[RegistrationID]
           ,[InternshipAttendance]
           ,[ExamGrade]
           ,[AttendanceDays])
     VALUES
           ((SELECT (MAX(InternshipID) + 1) FROM Internships)
           ,@StudiumID
           ,@RegistrationID
           ,0
           ,2
           ,0);
END;
GO
```

PayForEvent Procedura służy do opłaty należności.

```
CREATE PROCEDURE [dbo].[PayForEvent]
    @UserID INT,
    @ReceivableID INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ToPay MONEY;
    DECLARE @UserBalance MONEY;

    -- Pobierz informacje o uzytkowniku i szkoleniu
    SELECT TOP 1
        @ToPay = R.Amount,
        @UserBalance = S.Saldo
    FROM
        dbo.Receivables R
    INNER JOIN
        dbo.Students S ON R.UserID = S.UserID
    WHERE
        R.UserID = @UserID AND S.UserID = @UserID
        AND R.TransactionStatus = 'pending' AND R.ReceivableID = @ReceivableID
    ORDER BY
        R.DueDate;

    -- Sprawdź, czy użytkownik ma wystarczająco środków na koncie
    IF @UserBalance >= @ToPay
        BEGIN
            -- Oznacz transakcję jako opłaconą
            UPDATE dbo.Receivables
            SET TransactionStatus = 'paid'
            WHERE ReceivableID = @ReceivableID;

            -- Odejmij kwotę kursu od salda użytkownika
            UPDATE dbo.Students
```

```
            SET Saldo = Saldo - @ToPay
            WHERE UserID = @UserID;

            PRINT 'Płatność zrealizowana pomyślnie.';
        END
    ELSE
        BEGIN TRY
            THROW 50004, 'Platność sie nie powiodła', 1;
        END TRY
        BEGIN CATCH
            PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        END CATCH
END;
GO
```

PaymentDeferral Pozwala dyrektorowki szkoły odroczyć daną należność.

```
CREATE PROCEDURE [dbo].[PaymentDeferral]
    @ReceivableID INT,
    @NewDueDate DATE
AS
BEGIN
    SET NOCOUNT ON;
    -- odracza date zaplaty za szkolenia
    --moze byc wykorzystana przez dyrektora szkoly
    UPDATE dbo.Receivables
    SET DueDate = @NewDueDate
    WHERE ReceivableID = @ReceivableID;
    IF @@ROWCOUNT = 1
        BEGIN
            PRINT 'DueDate dla ReceivableID ' + CAST(@ReceivableID AS NVARCHAR(10)) + ' została
zaktualizowana.';
        END
    ELSE
        BEGIN
            PRINT 'Nie znaleziono zadłużenia o ReceivableID ' + CAST(@ReceivableID AS NVARCHAR(10)) +
'.';
        END
END;
GO
```

ScheduleMeetingForCourse Pozwala na zaplanowanie meetingu w ramach kursu.

```
CREATE PROCEDURE [dbo].[ScheduleMeetingForCourse]
    @CourseID int,
    @Room varchar(50),
    @StartDateAndTime datetime,
    @MeetingType int,
    @TeacherID int,
    @TranslatorID int
AS
BEGIN
    -- sprawdza czy tłumacz mówi w języku kurs
    IF EXISTS (SELECT 1
        FROM [dbo].[TranslatorLanguages] TL
            WHERE TL.[TranslatorID] = @TranslatorID
            AND TL.[LanguageID] = (SELECT [LanguageID] FROM [dbo].[Courses] WHERE [CourseID] = @CourseID)
    )
        BEGIN
            --sprawdza czy dla danego kursu nie pokrywa się dane spotkanie czasowo z innym spotkaniem
tego samego kursu
            --spotkania trwają 90 min
            IF NOT EXISTS (SELECT 1
```

```
                FROM [dbo].[CourseMeetings] WHERE @CourseID = CourseID
                AND @StartDateAndTime < DATEADD(MINUTE, 90, StartDateAndTime)
                AND DATEADD(MINUTE, 90, @StartDateAndTime) > StartDateAndTime)

                BEGIN
                    INSERT INTO [dbo].[CourseMeetings] (
                        [CourseID], [TeacherID], [Room], [StartDateAndTime],
                        [MeetingType], [TranslatorID]
                    )
                    VALUES (
                        @CourseID, @TeacherID, @Room, @StartDateAndTime,
                        @MeetingType, @TranslatorID
                    )
                    SELECT SCOPE_IDENTITY() AS NewMeetingID
                END
            ELSE
            BEGIN
                SELECT -1 AS NewMeetingID
            END
        END
    ELSE
        BEGIN
            SELECT -2 AS NewMeetingID
        END
END
GO
```

UpdateSingleRegistration Aktualizuje status danego rekordu

```
CREATE PROCEDURE [dbo].[UpdateSingleRegistration]
    @RegistrationID int
AS
BEGIN
    SET NOCOUNT ON;

    --procedura aktualizuje Status danego rekordu w tabeli Registration
    -- completed - uzytkjownik zaliczyl szkolenie
    -- failed - uzytkownik nie zaliczyl szkolenia
    -- in_progress uzytkownik jest w trakcie realizacji

    IF (SELECT TOP 1 CourseID FROM Registration WHERE RegistrationID = @RegistrationID) IS NOT NULL
        BEGIN
            DECLARE @userID INT ;
            DECLARE @courseID INT;
            DECLARE @attendanceFreq FLOAT;

            SET @userID = (SELECT TOP 1 UserID FROM Registration WHERE RegistrationID = @RegistrationID)
            SET @courseID = (SELECT TOP 1 CourseID FROM Registration WHERE RegistrationID =
@RegistrationID)
            SET @attendanceFreq = CONVERT(FLOAT, dbo.ParticipatedMeetingsNumberCourse(@courseID,
@userID)) * 100 / CONVERT(FLOAT, dbo.NumberOfMeetingsInCourse(@courseID))

            IF @attendanceFreq < 0.80 AND (SELECT CourseEndDate FROM Courses WHERE CourseID = @courseID)
< GETDATE()
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'failed'
                    WHERE [RegistrationID] = @RegistrationID;
                END
            ELSE IF @attendanceFreq >= 0.80 AND (SELECT CourseEndDate FROM Courses WHERE CourseID =
@courseID) < GETDATE()
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'completed'
                    WHERE [RegistrationID] = @RegistrationID;
```

```
                END
            ELSE
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'in_progress'
                    WHERE [RegistrationID] = @RegistrationID;
                END
        END

    ELSE IF (SELECT TOP 1 WebinarID FROM Registration WHERE RegistrationID = @RegistrationID) IS NOT NULL
        BEGIN

            DECLARE @webID INT;
            DECLARE @userWebID INT;
            SET @userWebID = (SELECT TOP 1 UserID FROM Registration WHERE RegistrationID =
@RegistrationID)
            SET @webID = (SELECT TOP 1 WebinarID FROM Registration WHERE RegistrationID =
@RegistrationID)


            IF (SELECT TOP 1 AttendanceStatus FROM [dbo].[AttendanceWebinarListFunction] (@webID) WHERE
UserID = @userWebID) = 'PRESENT' AND (SELECT TOP 1 EndDateAndTime FROM Webinarium WHERE WebinarID =
@webID) < GETDATE()
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'completed'
                    WHERE [RegistrationID] = @RegistrationID;
                END

            ELSE IF (SELECT TOP 1 AttendanceStatus FROM [dbo].[AttendanceWebinarListFunction] (@webID)
WHERE UserID = @userWebID) = 'ABSENT' AND (SELECT TOP 1 EndDateAndTime FROM Webinarium WHERE WebinarID =
@webID) < GETDATE()
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'failed'
                    WHERE [RegistrationID] = @RegistrationID;
                END
            ELSE
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'in_progress'
                    WHERE [RegistrationID] = @RegistrationID;
                END
    END

    ELSE IF (SELECT TOP 1 StudiumID FROM Registration WHERE RegistrationID = @RegistrationID) IS NOT NULL
        BEGIN
            DECLARE @studyID INT;
            DECLARE @attendanceStudyFreq INT;
            DECLARE @userStudyID INT;

            SET @userStudyID = (SELECT TOP 1 UserID FROM Registration WHERE RegistrationID =
@RegistrationID)
            SET @studyID = (SELECT TOP 1 StudiumID FROM Registration WHERE RegistrationID =
@RegistrationID)
            SET @attendanceStudyFreq = CONVERT(FLOAT, dbo.ParticipatedMeetingsNumberStudies(@studyID,
@userStudyID)) * 100 / CONVERT(FLOAT, dbo.NumberOfMeetingsInStudies(@studyID))

            IF (((SELECT TOP 2 AVG(ExamGrade) FROM Internships WHERE RegistrationID = @RegistrationID) <
3) OR ((SELECT TOP 2 AVG(InternshipAttendance) FROM Internships WHERE RegistrationID = @RegistrationID) <
100))
                AND (GETDATE() > (SELECT TOP 1  EndDate FROM Studies WHERE StudiumID = @studyID))
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'failed'
                    WHERE [RegistrationID] = @RegistrationID;
                END
```

```
            ELSE IF (((SELECT TOP 2 AVG(ExamGrade) FROM Internships WHERE RegistrationID =
@RegistrationID) >= 3) AND ((SELECT AVG(InternshipAttendance) FROM Internships WHERE RegistrationID =
@RegistrationID) = 100))
                AND (GETDATE() > (SELECT TOP 1  EndDate FROM Studies WHERE StudiumID = @studyID))
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'completed'
                    WHERE [RegistrationID] = @RegistrationID;
                END

            ELSE
                BEGIN
                    UPDATE [dbo].[Registration]
                    SET [Status] = 'in_progress'
                    WHERE [RegistrationID] = @RegistrationID;
                END
        END
END
GO
```

UpdateAllRegistrations Aktualizuje statusy wszystkich rekordów w tablicy Registrations

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

    --Procedura aktualizuje wyszstkie rekordy w tablicy Registrations
    DECLARE @RegistrationID INT;
    SET @RegistrationID = (SELECT MIN(RegistrationID) FROM [dbo].[Registration]);

    -- Pętla przetwarzająca wszystkie rekordy
    WHILE @RegistrationID IS NOT NULL
    BEGIN
        -- Wywołanie procedury UpdateSingleRegistration dla aktualnego RegistrationID
        EXEC [dbo].[UpdateSingleRegistration] @RegistrationID;

        -- Pobranie kolejnego RegistrationID
        SET @RegistrationID = (SELECT MIN(RegistrationID) FROM [dbo].[Registration] WHERE RegistrationID
> @RegistrationID);
    END
END
GO
```

UpdateStudyDates Procedura aktualizuje daty rozpoczęcia i zakończenia studiów.

```
CREATE PROCEDURE [dbo].[UpdateStudyDates]
    @StudiumID INT,
    @NewStartDate DATETIME,
    @NewEndDate DATETIME
AS
BEGIN
    IF EXISTS (SELECT 1 FROM [dbo].[Studies] WHERE [StudiumID] = @StudiumID)
        BEGIN
            -- Aktualizuj pola StartDate i EndDate
            UPDATE [dbo].[Studies]
            SET [StartDate] = @NewStartDate,
                [EndDate] = @NewEndDate
            WHERE [StudiumID] = @StudiumID;

            PRINT 'Pola StartDate i EndDate zaktualizowane poprawnie.';
        END
```

```
    ELSE
        BEGIN
            RAISERROR('StudiumID nie istnieje. Aktualizacja nieudana.',-1,-1)
        END
END;
GO
```

UpdateStudyName Procedura aktualizuje nazwę studiów.

```
CREATE PROCEDURE [dbo].[UpdateStudyName]
    @StudiumID INT,
    @NewName VARCHAR(50)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM [dbo].[Studies] WHERE [StudiumID] = @StudiumID)
        BEGIN
            UPDATE [dbo].[Studies]
            SET [Name] = @NewName
            WHERE [StudiumID] = @StudiumID;

            PRINT 'Pole Name zaktualizowane poprawnie.';
        END
    ELSE
        BEGIN
            RAISERROR('StudiumID nie istnieje. Aktualizacja nieudana.',-1,-1)
        END
END;
GO
```

UpdateStudySyllabus Procedura aktualizuje plan studiów.

```
CREATE PROCEDURE [dbo].[UpdateStudySyllabus]
    @StudiumID INT,
    @NewSyllabus TEXT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM [dbo].[Studies] WHERE [StudiumID] = @StudiumID)
        BEGIN
            UPDATE [dbo].[Studies]
            SET [Syllabus] = @NewSyllabus
            WHERE [StudiumID] = @StudiumID;

            PRINT 'Pole Syllabus zaktualizowane poprawnie.';
        END
    ELSE
        BEGIN
            RAISERROR('StudiumID nie istnieje. Aktualizacja nieudana.',-1,-1)
        END
END;
GO
```

UpdateWebinarDescription Procedura aktualizuje opis webinaru.

```
CREATE PROCEDURE [dbo].[UpdateWebinarDescription]
    @WebinarID INT,
    @NewDescription TEXT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM [dbo].[Webinarium] WHERE [WebinarID] = @WebinarID)
        BEGIN
            UPDATE [dbo].[Webinarium]
```

```
            SET [Description] = @NewDescription
            WHERE [WebinarID] = @WebinarID;

            PRINT 'Pole Description zaktualizowane poprawnie.';
        END
    ELSE
        BEGIN
            PRINT 'WebinarID nie istnieje. Aktualizacja nieudana.';
        END
END;
GO
```

UpdateWebinarName Procedura aktualizuje nazwę webinaru.

```
CREATE PROCEDURE [dbo].[UpdateWebinarName]
    @WebinarID INT,
    @NewName VARCHAR(50)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM [dbo].[Webinarium] WHERE [WebinarID] = @WebinarID)
        BEGIN
            -- Aktualizuj pole Name
            UPDATE [dbo].[Webinarium]
            SET [Name] = @NewName
            WHERE [WebinarID] = @WebinarID;

            PRINT 'Pole Name zaktualizowane poprawnie.';
        END
    ELSE
        BEGIN
            PRINT 'WebinarID nie istnieje. Aktualizacja nieudana.';
        END
END;
GO
```

UpdateWebinarTimes Aktualizuje czas rozpoczęcia i zakończenia webinaru.

```
CREATE PROCEDURE [dbo].[UpdateWebinarTimes]
    @WebinarID INT,
    @NewStartTime DATETIME,
    @NewEndTime DATETIME
AS
BEGIN
    IF EXISTS (SELECT 1 FROM [dbo].[Webinarium] WHERE [WebinarID] = @WebinarID)
    BEGIN
        UPDATE [dbo].[Webinarium]
        SET [StartDateAndTime] = @NewStartTime,
            [EndDateAndTime] = @NewEndTime
        WHERE [WebinarID] = @WebinarID;

        PRINT 'Pola StartDateAndTime i EndDateAndTime zaktualizowane poprawnie.';
    END
    ELSE
    BEGIN
        PRINT 'WebinarID nie istnieje. Aktualizacja nieudana.';
    END
END;
GO
```

Triggery

NewRegistration Po dodaniu nowej rejestracji automatycznie zostaną dodane należności w tabeli Receivables z podziałem na podpłatności (zaliczka, opłaty za zjazdy itd)ad

```
CREATE TRIGGER [dbo].[NewRegistration]
on [dbo].[Registration]
after INSERT
AS
BEGIN
    DECLARE @StudiumID INT;
    DECLARE @CourseID INT;
    DECLARE @WebinarID INT;
    DECLARE @StudentID INT;
    DECLARE @Date DATE;
    DECLARE @Amount1 MONEY;
    DECLARE @DueDate1 DATE;
    DECLARE @RegID INT;
    -- przypisanie wartości
    DECLARE @RowCount INT;
    SELECT @RowCount = COUNT(*) FROM inserted
    IF @RowCount <> 1
    BEGIN
        THROW 50001, 'Wstawiono niewłaściwą liczbę wierszy.', 1;
    END
    SELECT @StudiumID = StudiumID, @CourseID = CourseID,
    @WebinarID = WebinarID,
    @StudentID = UserID,
    @Date = RegistrationDate,
    @RegID = RegistrationID FROM inserted;
    -- w zależności od typu wydarzenia różne kroki
    IF @StudiumID IS NOT NULL
    BEGIN
        DECLARE @id1 INT;
        select @Amount1 = EntryFee, @DueDate1 = StartDate from Studies
        where StudiumID = @StudiumID
        select @id1 = ISNULL(max(ReceivableID) + 1,1) from Receivables
        INSERT INTO Receivables (ReceivableID, UserID, TransactionStatus,
        Date, Amount, DueDate,RegistrationID)
        -- dodaje czesne
        VALUES (@id1, @StudentID, 'pending', @Date,
         @Amount1, @DueDate1,@RegID);
        INSERT INTO Receivables (ReceivableID, UserID, TransactionStatus,
        Date, Amount, DueDate,RegistrationID)
        select (select max(ReceivableID) from Receivables) + ROW_NUMBER()
        OVER (ORDER BY GatheringID), @StudentID, 'pending', @Date,
        GatheringPrice, DATEADD(DAY,-3,GatheringDate), @RegID from Gatherings g
        where g.StudiumID = @StudiumID

        EXEC CreateInternship @StudiumID = @StudiumID, @RegistrationID = @RegID
        EXEC CreateInternship @StudiumID = @StudiumID, @RegistrationID = @RegID
        -- znajduje i dodaje wszystkie zjazdy
    END
    ELSE IF @CourseID IS NOT NULL
    BEGIN
        DECLARE @id2 INT;
        select @Amount1 = Prepayment, @DueDate1 = GETDATE() from Courses
        where CourseID = @CourseID
        select @id2 = ISNULL(max(ReceivableID) + 1,1) from Receivables
        INSERT INTO Receivables (ReceivableID, UserID, TransactionStatus,
         Date, Amount, DueDate,RegistrationID)
        VALUES (@id2, @StudentID, 'pending', @Date, @Amount1, @DueDate1,@RegID);
        -- dodaje należność za zaliczke
        DECLARE @Amount2 MONEY;
        DECLARE @DueDate2 DATE;
        select @Amount2 = FullPayment - Prepayment,
        @DueDate2 = DATEADD(DAY,-3,CourseStartDate)
        from Courses
```

```
        where CourseID = @CourseID
        INSERT INTO Receivables (ReceivableID, UserID, TransactionStatus,
        Date, Amount, DueDate,RegistrationID)
        VALUES (@id2 + 1, @StudentID, 'pending', @Date, @Amount2, @DueDate2,@RegID);
        -- dodaje reszte należności (bez zaliczki)
    END
    ELSE
    BEGIN
        DECLARE @id3 INT;
        select @id3 = ISNULL(max(ReceivableID) + 1,1) from Receivables
        select @Amount1 = Price, @DueDate1 = Cast(StartDateAndTime as Date) from Webinarium
        where WebinarID = @WebinarID
        INSERT INTO Receivables (ReceivableID, UserID, TransactionStatus, Date, Amount,
DueDate,RegistrationID)
        VALUES (@id3, @StudentID, 'pending', @Date, @Amount1, @DueDate2,@RegID);
        --dodaje należność za webinar
    END
END
GO
ALTER TABLE [dbo].[Registration] ENABLE TRIGGER [NewRegistration]
GO
```

UpdateSaldoAfterPayment - uaktualnia saldo studenta po dokonaniu płatności

```
CREATE TRIGGER [dbo].[UpdateSaldoAfterPayment]
ON [dbo].[Payments]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE s
    SET s.Saldo = s.Saldo + i.Amount
    FROM dbo.Students s
    INNER JOIN inserted i ON s.UserID = i.UserID;
END;
GO
ALTER TABLE [dbo].[Payments] ENABLE TRIGGER [UpdateSaldoAfterPayment]
GO
```

IntershipAttendanceUpdate - aktualizuje procent obecności

```
CREATE TRIGGER [dbo].[InternshipAttendanceUpdate]
ON [dbo].[Internships]
AFTER UPDATE
AS
BEGIN
    -- Sprawdź, czy kolumna AttendanceDays została zmieniona
    IF UPDATE(AttendanceDays)
    BEGIN
        -- Aktualizuj kolumnę InternshipAttendance
        UPDATE I
        SET I.InternshipAttendance = ROUND(CONVERT(float, I.AttendanceDays) / 14,2)
        FROM [dbo].[Internships] AS I
        INNER JOIN inserted AS ins ON I.InternshipID = ins.InternshipID;
    END
END;
```

Funkcje:

AttendanceCourseInPercenatge - jako argument id kursu. Funkcja zwraca tabelę, która zawiera użytkowników zapisanych na kurs i ich procentowy udział w zajęciach.

```
CREATE FUNCTION [dbo].[AttendanceCourseInPercentage](@CourseID int)
RETURNS TABLE
AS
RETURN
(
    WITH t as (SELECT u.UserID, u.FirstName, u.LastName
        FROM Registration reg
        JOIN Students s ON s.UserID = reg.UserID
        JOIN Users u ON s.UserID = u.UserID
        WHERE @CourseID = reg.CourseID)
    SELECT t.UserID as 'UserID', t.FirstName as 'First Name', t.LastName as 'Last Name',
        ROUND(CONVERT(FLOAT, dbo.ParticipatedMeetingsNumberCourse(@CourseID, t.UserID)) * 100 /
CONVERT(FLOAT, dbo.NumberOfMeetingsInCourse(@CourseID)), 2) as 'Attendance Percentage',
        dbo.ParticipatedMeetingsNumberCourse(@CourseID, t.UserID) as 'Presences',
dbo.NumberOfMeetingsInCourse(@CourseID) as 'MeetingsNo'
    FROM t
)
```

AttendanceCourseListFunction - jako argument przyjmuje id kursu. Zwraca listę meetingów dla danego kursu, do każdego meetingu jest przypisany zapisany na kurs użytkownik. Obok jest informacja czy rzeczywiście był czy też nie.

```
CREATE FUNCTION [dbo].[AttendanceCourseListFunction](@CourseID int)
RETURNS TABLE AS
RETURN
(
    WITH t as (SELECT u.UserID, u.FirstName, u.LastName
        FROM Registration reg
        JOIN Students s ON s.UserID = reg.UserID
        JOIN Users uON s.UserID = u.UserID
        WHERE @CourseID = reg.CourseID)
    SELECT t.UserID as 'UserID', t.FirstName as 'First Name', t.LastName as 'Last Name', cm.MeetingID as
'MeetingID',cm.StartDateAndTime as 'Date',
        CASE
            WHEN ca.UserID IS NOT NULL THEN 'PRESENT'
            ELSE 'ABSENT'
        END AS AttendanceStatus
    FROM t
    LEFT JOIN CourseMeetings cmON cm.CourseID = @CourseID
    LEFT JOIN CourseAttendance caON cm.MeetingID = ca.MeetingID AND t.UserID = ca.UserID
)
GO
```

AttendanceStudiesInPercentage - jako argument przyjmuje id studiów. Funkcja zwraca tabelę, która zawiera użytkowników zapisanych na studia i ich procentowy udział w zajęciach.

```
CREATE FUNCTION [dbo].[AttendanceStudiesInPercentage](@StudiumID int)
RETURNS TABLE
AS
RETURN
(

    WITH t as (SELECT u.UserID, u.FirstName, u.LastName
        FROM Registration reg
        JOIN Students s ON s.UserID = reg.UserID
        JOIN Users u ON s.UserID = u.UserID
        WHERE @StudiumID = reg.StudiumID)
    SELECT t.UserID as 'UserID', t.FirstName as 'First Name', t.LastName as 'Last Name',
        ROUND(CONVERT(FLOAT, dbo.ParticipatedMeetingsNumberStudies(@StudiumID, t.UserID)) * 100 /
```

```
CONVERT(FLOAT, dbo.NumberOfMeetingsInStudies(@StudiumID)), 2) as 'Attendance Percentage',
        dbo.ParticipatedMeetingsNumberCourse(@StudiumID,t.UserID) as 'Presences',
dbo.NumberOfMeetingsInCourse(@StudiumID) as 'MeetingsNo'
    FROM t
)
GO
```

AttendanceStudyListFunction -- jako argument przyjmuje id studiów. Zwraca imię i nazwisko każdego studenta, zapisanego na dany zlot w ramach studiów oraz datę tego zlotu, informację czy użytkownik był obecny.

```
CREATE FUNCTION [dbo].[AttendanceStudyListFunction](@StudiumID int)
RETURNS TABLE
AS
RETURN
(
    WITH t as (SELECT u.UserID, u.FirstName, u.LastName
        FROM Registration reg
        JOIN Students s ON s.UserID = reg.UserID
        JOIN Users uON s.UserID = u.UserID
        WHERE @StudiumID = reg.StudiumID)
    SELECT t.UserID as 'UserID', t.FirstName as 'First Name', t.LastName as 'Last Name', ga.GatheringID
as 'GatheringID',ga.GatheringDate as 'Date',
        CASE
            WHEN sa.UserID IS NOT NULL THEN 'PRESENT'
            ELSE 'ABSENT'
        END AS AttendanceStatus
    FROM t
        LEFT JOIN Gatherings ga ON ga.StudiumID = @StudiumID
        LEFT JOIN StudyMeetings sm ON sm.GatheringID = ga.GatheringID
        LEFT JOIN StudyAttendance sa ON sm.StudyMeetingsID = sa.MeetingID AND sa.UserID = t.UserID
)
GO
```

AttendanceWebinarListFunction - jako argument przyjmuje id webinaru. Zwraca użytkowników zarejestrowanych na dany webinar oraz informację czy byli oni obecni.

```
CREATE FUNCTION [dbo].[AttendanceWebinarListFunction](@WebinarID int)
RETURNS TABLE
AS
RETURN
(
    WITH t as (SELECT u.UserID, u.FirstName, u.LastName
        FROM Registration reg
        JOIN Students s ON s.UserID = reg.UserID
        JOIN Users u ON s.UserID = u.UserID
        WHERE @WebinarID = reg.WebinarID)
    SELECT t.UserID as 'UserID', t.FirstName as 'First Name', t.LastName as 'Last Name', wb.WebinarID as
'WebinarID',wb.StartDateAndTime as 'Date',
        CASE
            WHEN wba.UserID IS NOT NULL THEN 'PRESENT'
            ELSE 'ABSENT'
        END AS AttendanceStatus
    FROM t
        LEFT JOIN Webinarium wb ON wb.WebinarID = @WebinarID
        LEFT JOIN WebinarAttendance wba ON wba.MeetingID = @WebinarID AND t.UserID = wba.UserID
)
GO
```

CourseAttendancePerMeeting - jako argument przyjmuje id kursu. Zwraca id meetingów, liczbę obenych studentów, typ meetingu i date rozpoczęcia meetingu dla danego kursu.

```
CREATE FUNCTION [dbo].[CourseAttendancePerMeeting](@CourseID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT CourseMeetings.CourseID, CourseMeetings.MeetingID, COUNT(CourseAttendance.UserID) AS
NumberAttended,
    CourseMeetings.MeetingType,  CourseMeetings.StartDateAndTime
        FROM CourseMeetings
        JOIN CourseAttendance ON CourseMeetings.MeetingID = CourseAttendance.MeetingID
        WHERE CourseMeetings.CourseID = @CourseID
        GROUP BY CourseMeetings.CourseID, CourseMeetings.MeetingID,CourseMeetings.MeetingType,
CourseMeetings.StartDateAndTime
)
GO
```

GetBilocationReport Zwraca użytkowników, którym nakładają się czasowo dwa szkolenia.

```
CREATE FUNCTION [dbo].[GetBilocationReport]()
RETURNS TABLE
AS
RETURN
(
    WITH t as (SELECT reg.UserID, cs.CourseID, cs.CourseStartDate, cs.CourseEndDate, wb.WebinarID,
wb.StartDateAndTime, wb.EndDateAndTime,
    std.StudiumID, std.StartDate, std.EndDate
        FROM Registration reg
        JOIN Webinarium as wb ON reg.WebinarID = wb.WebinarID AND GETDATE() < wb.StartDateAndTime
        JOIN Courses cs ON reg.CourseID = cs.CourseID AND GETDATE() < cs.CourseStartDate
        JOIN Studies std on reg.StudiumID = std.StudiumID AND GETDATE() < std.StartDate)
    SELECT t.UserID
    FROM t
    JOIN (SELECT reg.UserID, cs.CourseID, cs.CourseStartDate, cs.CourseEndDate, wb.WebinarID,
wb.StartDateAndTime, wb.EndDateAndTime,
    std.StudiumID, std.StartDate, std.EndDate
        FROM Registration reg
        JOIN Webinarium as wb ON reg.WebinarID = wb.WebinarID AND GETDATE() < wb.StartDateAndTime
        JOIN Courses cs ON reg.CourseID = cs.CourseID AND GETDATE() < cs.CourseStartDate
        JOIN Studies std on reg.StudiumID = std.StudiumID AND GETDATE() < std.StartDate) as s
        ON t.UserID = s.UserID AND NOT (t.CourseID=s.CourseID) AND NOT (t.WebinarID = s.WebinarID) AND
NOT (t.StudiumID = s.StudiumID)
        WHERE (t.CourseStartDate >= s.CourseStartDate AND t.CourseEndDate <= s.CourseEndDate)
        OR (t.CourseStartDate >= s.CourseStartDate AND t.CourseEndDate >=s.CourseEndDate AND
t.CourseStartDate <= s.CourseEndDate)
        OR (t.CourseStartDate <= s.CourseStartDate AND t.CourseEndDate <= s.CourseEndDate AND
t.CourseEndDate >= s.CourseStartDate)
        OR (t.StartDateAndTime >= s.StartDateAndTime AND t.EndDateAndTime <= s.EndDateAndTime)
        OR (t.StartDateAndTime >= s.StartDateAndTime AND t.EndDateAndTime >=s.EndDateAndTime AND
t.StartDateAndTime <= s.EndDateAndTime)
        OR (t.StartDateAndTime <= s.StartDateAndTime AND t.EndDateAndTime <= s.EndDateAndTime AND
t.EndDateAndTime >= s.StartDateAndTime)
        OR (t.StartDate >= s.StartDate AND t.EndDate <= s.EndDate) OR (t.StartDate >= s.StartDate AND
t.EndDate >=s.EndDate AND t.StartDate <= s.EndDate)
        OR (t.StartDate <= s.StartDate AND t.EndDate <= s.EndDate AND t.EndDate >= s.StartDate)
)
GO
```

GetFutureEvents Funckaj zwraca przszłe wydarzenia

```
CREATE FUNCTION [dbo].[GetFutureEvents]()
RETURNS TABLE
AS
```

```
RETURN
(
    (SELECT 'Course' as [C/S/W], cs.CourseID, cs.Name, cs.CourseType, cs.CourseStartDate
        FROM Courses as cs WHERE cs.CourseID > GETDATE())
    UNION
    (SELECT 'Webinar' as [C/S/W], wb.WebinarID, wb.Name, 3, wb.StartDateAndTime
        FROM Webinarium as wb WHERE wb.StartDateAndTime > GETDATE())
    UNION
    (SELECT 'Studium' as [C/S/W], st.StudiumID, st.Name, st.StudyType, st.StartDate
        FROM Studies as stWHERE st.StartDate > GETDATE())
)
GO
```

GetFutureEventsParticipants Zwraca tablice uzytkownikow ktorzy sa zapisani na jakies przyszle wydarzenia

```
CREATE FUNCTION [dbo].[GetFutureEventsParticipants]()
RETURNS TABLE
AS
RETURN
(
    SELECT reg.UserID, reg.CourseID, reg.WebinarID, reg.StudiumID FROM Registration reg
    JOIN Webinarium as wb ON reg.WebinarID = wb.WebinarID AND GETDATE() < wb.StartDateAndTime
    JOIN Courses cs ON reg.CourseID = cs.CourseID AND GETDATE() < cs.CourseStartDate
    JOIN Studies std ON reg.StudiumID = std.StudiumID AND GETDATE() < std.StartDate
)
GO
```

GetPastEvents Funkcja zwraca zakończone eventy

```
CREATE FUNCTION [dbo].[GetPastEvents]()
RETURNS TABLE
AS
RETURN
(
    (SELECT 'Course' as [C/S/W], cs.CourseID, cs.Name, cs.CourseType, cs.CourseStartDate as 'StartDate'
        FROM Courses as cs WHERE cs.CourseEndDate < GETDATE())
    UNION
    (SELECT 'Webinar' as [C/S/W], wb.WebinarID, wb.Name, 3, wb.StartDateAndTime as 'StartDate'
        FROM Webinarium as wb WHERE wb.EndDateAndTime < GETDATE())
    UNION
    (SELECT 'Studium' as [C/S/W], st.StudiumID, st.Name, st.StudyType, st.StartDate as 'StartDate'
        FROM Studies as st WHERE st.EndDate < GETDATE())
)
GO
```

RegisteredUsersEvens - jako argument przyjmuje id użytkownika Funkcja zwraca eventy, na które dany użytkownik jest zapisany.

```
CREATE FUNCTION [dbo].[RegisteredUsersEvents](
    @UserID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT CourseID, WebinarID, StudiumID
        FROM Registration reg
        WHERE @UserID = reg.UserID
)
GO
```

StudyMeetingsAttendancePerMeeting - jako argument przyjmuje id gatheringu. Zwraca id meetingów, liczbę obenych studentów, typ meetingu i date rozpoczęcia meetingu dla danego gatheringu.

```
CREATE FUNCTION [dbo].[StudyMeetingsAttendancePerMeeting](@GatheringID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        StudyMeetings.StudyMeetingsID, StudyMeetings.GatheringID, COUNT(StudyAttendance.UserID) AS
NumberAttended,
            StudyMeetings.MeetingType, StudyMeetings.StartDateAndTime
        FROM StudyMeetings
        JOIN StudyAttendance ON StudyMeetings.StudyMeetingsID = StudyAttendance.MeetingID
        WHERE StudyMeetings.GatheringID = @GatheringID
    GROUP BY
        StudyMeetings.GatheringID,
        StudyMeetings.StudyMeetingsID,
        StudyMeetings.MeetingType,
        StudyMeetings.StartDateAndTime
)
GO
```

WebinariumAttendancePerMeeting - jako argument przyjmuje id webinaru. Zwraca id webinaru, liczbę obenych studentów i date rozpoczęcia meetingu.

```
CREATE FUNCTION [dbo].[WebinariumAttendancePerMeeting](@WebinarID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT Webinarium.WebinarID, Webinarium.Name, COUNT(WebinarAttendance.UserID) AS NumberAttended,
Webinarium.StartDateAndTime
        FROM Webinarium
        JOIN WebinarAttendance ON Webinarium.WebinarID = WebinarAttendance.MeetingID
        WHERE Webinarium.WebinarID = @WebinarID
        GROUP BY Webinarium.WebinarID, Webinarium.Name, Webinarium.StartDateAndTime
)
GO
```

NumberOfMeetingsInCourse - jako argument przyjmuje id kursu. Zwraca liczbę meetingów dla danego kursu.

```
CREATE FUNCTION [dbo].[NumberOfMeetingsInCourse] (@CourseID INT)
RETURNS INT
AS
BEGIN
    DECLARE @Result INT
    SET @Result = (SELECT COUNT(*) FROM CourseMeetings WHERE CourseID = @CourseID)
    RETURN @Result
END
GO
```

NumberOfMeetingsInStudies - jako id przyjmuje id studiów. Zwraca liczbę meetingów wśród wszystkich gatheringów podczas danych studiów.

```
CREATE FUNCTION [dbo].[NumberOfMeetingsInStudies](@StudiumID INT)
RETURNS INT
AS
BEGIN
```

```
    DECLARE @Result INT
    SET @Result = (SELECT COUNT(*) from StudyMeetings sm
                   JOIN Gatherings g
                   ON sm.GatheringID = g.GatheringID AND @StudiumID = g.StudiumID )
    RETURN @Result
END
GO
```

ParticipatedMeetingsNumberCourse - jako argumenty przyjmuje id kursu, id użytkownika. Zwraca ilość meetingów dla danego kursu, na których był dany użytkownik.

```
CREATE FUNCTION [dbo].[ParticipatedMeetingsNumberCourse](@CourseID INT, @UserID INT)
RETURNS INT
AS
BEGIN
    DECLARE @Result INT
    SET @Result = (SELECT COUNT(*) FROM CourseAttendance as ca
                   JOIN CourseMeetings cm ON cm.MeetingID = ca.MeetingID
                   AND cm.CourseID = @CourseID
                   WHERE ca.UserID = @UserID)
    RETURN @Result
END
GO
```

ParticipatedMeetingsNumberStudies - jako argumenty przyjmuje id studiów, id użytkownika. Zwraca ilość meetingów dla danych studiów, na których był dany użytkownik.

```
CREATE FUNCTION [dbo].[ParticipatedMeetingsNumberStudies](@StudiumID INT,@UserID INT)
RETURNS INT
AS
BEGIN
    DECLARE @Result INT
    SET @Result = (SELECT COUNT(*) FROM StudyAttendance as sa
                   JOIN StudyMeetings sm ON sm.StudyMeetingsID = sa.MeetingID
                   JOIN Gatherings ga
                   ON sm.GatheringID = ga.GatheringID AND ga.StudiumID = @StudiumID
                   WHERE sa.UserID = @UserID)
    RETURN @Result
END
GO
```

Uprawnienia użytkowników:

Administrator

```
CREATE ROLE administrator
GRANT ALL PRIVILEGES ON u_pismiale.dbo to administrator
```

Headmaster

```
CREATE ROLE Headmaster
GRANT SELECT ON AttendancePerCourseMeeting TO Headmaster
GRANT SELECT ON AttendancePerEndedCourse TO Headmaster
GRANT SELECT ON AttendancePerEndedGathering TO Headmaster
GRANT SELECT ON AttendancePerEndedWebinar TO Headmaster
GRANT SELECT ON BilocationReport TO Headmaster
GRANT SELECT ON CourseNumberRegistratedUsers TO Headmaster
GRANT SELECT ON CoursesInfo TO Headmaster
```

```
GRANT SELECT ON DebtorsInfo TO Headmaster
GRANT SELECT ON FinancialReportCourses TO Headmaster
GRANT SELECT ON FinancialReportStudies TO Headmaster
GRANT SELECT ON FinancialReportWebinars TO Headmaster
GRANT SELECT ON FutureCourses TO Headmaster
GRANT SELECT ON FutureWebinars TO Headmaster
GRANT SELECT ON GatheringsForEachStudents TO Headmaster
GRANT SELECT ON Graduates TO Headmaster
GRANT SELECT ON RegisteredStudentsInfo TO Headmaster
GRANT SELECT ON StudentsInfo TO Headmaster
GRANT SELECT ON StudiesNumberRegisteredUsers TO Headmaster
GRANT SELECT ON TeachersInfo TO Headmaster
GRANT SELECT ON WebinarsInfo TO Headmaster

GRANT EXECUTE ON AddUser TO Headmaster
GRANT EXECUTE ON PaymentDefferal TO Headmaster
GRANT EXECUTE ON ScheduleMeetingForCourse TO Headmaster
GRANT EXECUTE ON UpdateAllRegistrations TO Headmaster
GRANT EXECUTE ON UpdateSingleRegistrations TO Headmaster
GRANT EXECUTE ON UpdateStudyDate TO Headmaster
GRANT EXECUTE ON UpdateStudyName TO Headmaster
GRANT EXECUTE ON UpdateStudySyllabus TO Headmaster
GRANT EXECUTE ON UpdateWebinarDescryption TO Headmaster
GRANT EXECUTE ON UpdateWebinarName TO Headmaster
GRANT EXECUTE ON UpdateWebinarTimes TO Headmaster
```

Teacher

```
CREATE ROLE Teacher
GRANT EXECUTE ON AddCourseAttendance to Teacher
GRANT EXECUTE ON AddStudyAttendance to Teacher
GRANT EXECUTE ON AddWebinarAttendance to Teacher
GRANT EXECUTE ON CreateCourse to Teacher
GRANT EXECUTE ON StudentsInfo to Teacher
GRANT EXECUTE ON UpdateStudyDates to Teacher
GRANT EXECUTE ON UpdateStudyName to Teacher
GRANT EXECUTE ON UpdateStudySyllabus to Teacher
GRANT EXECUTE ON UpdateWebinarDescription to Teacher
GRANT EXECUTE ON UpdateWebinarName to Teacher
GRANT EXECUTE ON UpdateWebinarTimes to Teacher
GRANT EXECUTE ON AttendanceCourseListFunction to Teacher
GRANT EXECUTE ON AttendanceStudiesInPercentage to Teacher
GRANT EXECUTE ON AttendanceStudyListFunction to Teacher
GRANT EXECUTE ON GetFutureParticipants to Teacher
GRANT EXECUTE ON CourseAttendancePerMeeting to Teacher
```

Translator

```
CREATE ROLE [Translator]
GRANT EXECUTE ON UpdateWebinarTimes to Translator
GRANT SELECT ON AttendanceCourseListFunction to Translator
GRANT SELECT ON AttendanceStudyListFunction to Translator
GRANT SELECT ON GetFutureEvents to Translator
GRANT SELECT ON WebinarsInfo to Translator
GRANT SELECT ON StudiesInfo to Translator
GRANT SELECT ON CoursesInfo to Translator
GRANT SELECT ON GatheringsForEachStudies to Translator
```

Student

```
CREATE ROLE [Student]
GRANT EXECUTE ON AddRegistrationUser to Student
GRANT EXECUTE ON PayForEvent to Student
GRANT SELECT ON GetFutureEvents to Student
GRANT SELECT ON WebinarsInfo to Student
GRANT SELECT ON StudiesInfo to Student
GRANT SELECT ON CoursesInfo to Student
GRANT SELECT ON GatheringsForEachStudies to Student
```