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.

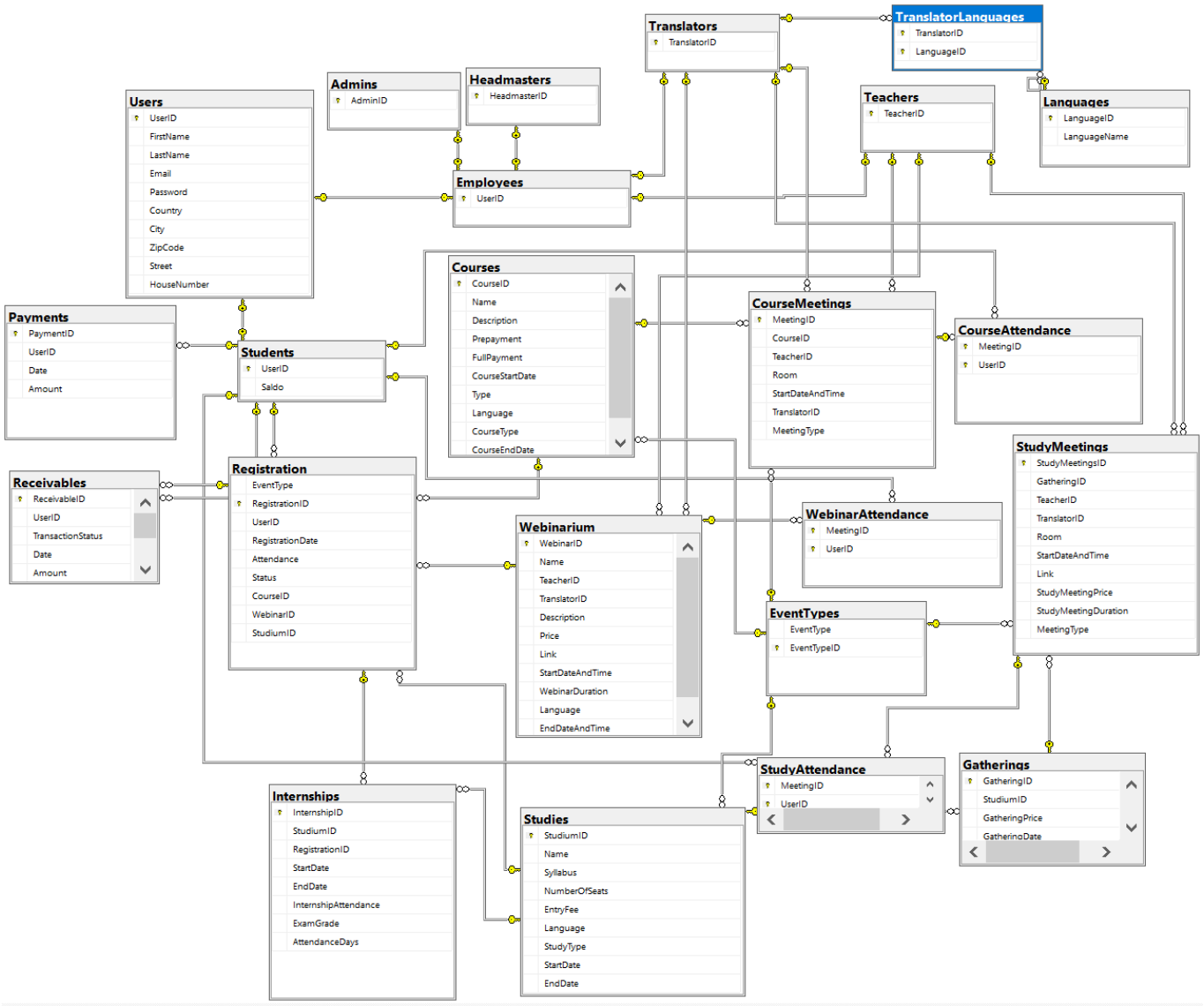
Zarządzanie Ocenami i Frekwencją:

- 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]
GO

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

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

Headmasters - zawiera id użytkowników, którzy są dyrektorami

- AdminID (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])
GO

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

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

- UserID (PK)- Identyfikator użytkownika
- Firstname - imię użytkownika
- LastName - nazwisko użytkownika
- Address - adres użytkownika
- Email - email użytkownika
- Password - hasło użytkownika

```
CREATE TABLE [dbo].[Users](
    [UserID] [int] IDENTITY(1,1) NOT NULL,
```

```
    [FirstName] [nchar](20) NOT NULL,
    [LastName] [nchar](20) NOT NULL,
    [Email] [nchar](30) NOT NULL,
    [Password] [nchar](30) NOT NULL,
    [Country] [nchar](20) NOT NULL,
    [City] [nchar](20) NOT NULL,
    [ZipCode] [nchar](20) NOT NULL,
    [Street] [nchar](30) NOT NULL,
    [HouseNumber] [nchar](10) NOT NULL,
 CONSTRAINT [PK_Users] 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]
GO

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

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CK_Users]
GO
```

Employees -zawiera id użytkowników, którzy są pracownikami

- UserID (Pk) - identyfikator w tabeli Users

```
CREATE TABLE [dbo].[Employees](
    [UserID] [int] NOT NULL,
 CONSTRAINT [PK_Employees] 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]
GO

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

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Users]
GO
```

Students - tabela zawiera tylko studentów z tabeli Users

- UserID - Identyfikator w tabeli Users

- Saldo - ilość pieniedzy jakie student ma na swoim infywidualnym koncie

```
CREATE TABLE [dbo].[Students](
    [UserID] [int] NOT 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]
GO

ALTER TABLE [dbo].[Students]  WITH CHECK ADD  CONSTRAINT [FK_Students_Users]
FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [FK_Students_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]
GO
```

Courses - zawiera kursy oraz informacje o nich

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

```
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]
GO

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

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

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

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

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

- RegistrationID (PK) - identyfikator rejestracji
- EventType - rodzaj wydarzenia na który zapisał się student
- EventID - identyfikator wydarzenia
- UserID (PK)- Identyfikator użytkownika
- RegistrationDate - data zarejestrowania na dane wydarzenie
- Attendance - procentowa wartość obecności na danym wydarzeniu
- Status - czy student zakończył, jest w trakcie lub nie ukończył wydarzenia

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE TABLE [dbo].[Registration](
    [EventType] [text] NOT NULL,
    [RegistrationID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [RegistrationDate] [date] NOT NULL,
    [Attendance] [float] NOT NULL,
    [Status] [varchar](50) NOT NULL,
    [CourseID] [int] NULL,
    [WebinarID] [int] NULL,
    [StudiumID] [int] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
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]
GO
ALTER TABLE [dbo].[Registration]  WITH CHECK ADD  CONSTRAINT
[FK_Registration_Courses1] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Courses] ([CourseID])
GO
ALTER TABLE [dbo].[Registration] CHECK CONSTRAINT [FK_Registration_Courses1]
GO
ALTER TABLE [dbo].[Registration]  WITH CHECK ADD  CONSTRAINT
[FK_Registration_Students] FOREIGN KEY([UserID])
REFERENCES [dbo].[Students] ([UserID])
GO
ALTER TABLE [dbo].[Registration] CHECK CONSTRAINT [FK_Registration_Students]
GO
ALTER TABLE [dbo].[Registration]  WITH CHECK ADD  CONSTRAINT
[FK_Registration_Studies1] FOREIGN KEY([StudiumID])
REFERENCES [dbo].[Studies] ([StudiumID])
GO
ALTER TABLE [dbo].[Registration] CHECK CONSTRAINT [FK_Registration_Studies1]
GO
ALTER TABLE [dbo].[Registration]  WITH CHECK ADD  CONSTRAINT
[FK_Registration_Webinarium1] FOREIGN KEY([WebinarID])
REFERENCES [dbo].[Webinarium] ([WebinarID])
GO
ALTER TABLE [dbo].[Registration] CHECK CONSTRAINT [FK_Registration_Webinarium1]
GO
ALTER TABLE [dbo].[Registration]  WITH CHECK ADD  CONSTRAINT [CK_Registration]
CHECK  (([Status]='completed' OR [Status]='in_progress' OR [Status]='failed'))
GO
ALTER TABLE [dbo].[Registration] CHECK CONSTRAINT [CK_Registration]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
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;
    SELECT @StudiumID = StudiumID, @CourseID = CourseID, @WebinarID = WebinarID,
@StudentID = UserID, @Date = RegistrationDate, @RegID = RegistrationID
    FROM inserted;
    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)
        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,
GatheringDate, @RegID from Gatherings g
        where g.StudiumID = @StudiumID
    END
    ELSE IF @CourseID IS NOT NULL
    BEGIN
        DECLARE @id2 INT;
        select @Amount1 = Prepayment, @DueDate1 = CourseStartDate 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);
        DECLARE @Amount2 MONEY;
        DECLARE @DueDate2 DATE;
        select @Amount2 = FullPayment - Prepayment, @DueDate2 = CourseEndDate from
Courses
        where CourseID = @CourseID
        INSERT INTO Receivables (ReceivableID, UserID, TransactionStatus, Date,
Amount, DueDate,RegistrationID)
        VALUES (@id2 + 1, @StudentID, 'pending', @Date, @Amount2,
@DueDate2,@RegID);
    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);
    END
END
GO
ALTER TABLE [dbo].[Registration] ENABLE TRIGGER [NewRegistration]
GO
```

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
- Room - sala w jakiej odbywa się spotkanie
- StartDateAndTime - data i godzina odbycia się zajęć
- Type - czy spotkanie jest stacjonarne/zdalne/zdalnie asynchronicznie
- Link - link do zewnętrznego komunikatora (jeżeli stacjonarnie to NULL)

```
CREATE TABLE [dbo].[CourseMeetings](
    [MeetingID] [int] NOT NULL,
    [CourseID] [int] NOT NULL,
    [TeacherID] [int] NOT NULL,
    [Room] [nchar](10) NULL,
    [StartDateAndTime] [datetime] NOT NULL,
    [Type] [nchar](10) 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]
GO

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

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

ALTER TABLE [dbo].[CourseMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_CourseMeetings_Employees] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[Employees] ([UserID])
GO
```

```
ALTER TABLE [dbo].[CourseMeetings] CHECK CONSTRAINT [FK_CourseMeetings_Employees]
GO

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

ALTER TABLE [dbo].[CourseMeetings] CHECK CONSTRAINT [CK_CourseMeetings]
GO
```

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
) ON [PRIMARY]
GO

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

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

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

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

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
- StartDate - data rozpoczęcia praktyk
- EndDate - data zakończenia praktyk
- AttendanceDays - liczba dni na których student był obecy (od 0 do 14)
- ExamGrade - ocena z egzaminu

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Internships](
    [InternshipID] [int] NOT NULL,
    [StudiumID] [int] NOT NULL,
    [RegistrationID] [int] NOT NULL,
    [StartDate] [date] NULL,
    [EndDate] [date] NULL,
    [InternshipAttendance] [float] NULL,
    [ExamGrade] [float] NULL,
    [AttendanceDays] [int] NULL
) ON [PRIMARY]
GO
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]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT
[FK_Internships_Registration] FOREIGN KEY([RegistrationID])
REFERENCES [dbo].[Registration] ([RegistrationID])
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Internships_Registration]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT
[FK_Internships_Studies] FOREIGN KEY([StudiumID])
REFERENCES [dbo].[Studies] ([StudiumID])
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Internships_Studies]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT [CK_Internships] CHECK
(([ExamGrade]>=(2) AND [ExamGrade]<=(5)))
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [CK_Internships]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT [CK_Internships_1]
CHECK  (([AttendanceDays]>=(0) AND [AttendanceDays]<=(14)))
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [CK_Internships_1]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT [CK_Internships_2]
CHECK  (([InternshipAttendance]>=(0) AND [InternshipAttendance]<=(100)))
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [CK_Internships_2]
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
```

```
GO
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;
GO
ALTER TABLE [dbo].[Internships] ENABLE TRIGGER [InternshipAttendanceUpdate]
GO
```

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

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
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]
GO
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]
GO
ALTER TABLE [dbo].[Receivables]  WITH CHECK ADD FOREIGN KEY([RegistrationID])
REFERENCES [dbo].[Registration] ([RegistrationID])
GO
ALTER TABLE [dbo].[Receivables]  WITH CHECK ADD FOREIGN KEY([UserID])
REFERENCES [dbo].[Students] ([UserID])
GO
```

Payments - tabela zawierająca wszystkie wpłaty

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

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Payments](
    [PaymentID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NULL,
    [Date] [date] NULL,
    [Amount] [money] NULL
) ON [PRIMARY]
GO
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]
GO
ALTER TABLE [dbo].[Payments]  WITH CHECK ADD FOREIGN KEY([UserID])
REFERENCES [dbo].[Students] ([UserID])
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
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;
GO
ALTER TABLE [dbo].[Payments] ENABLE TRIGGER [UpdateSaldoAfterPayment]
GO
```

Translators - identyfikatory użytkowników którzy są tłumaczami

```
CREATE TABLE [dbo].[Translators](
    [TranslatorID] [int] NOT NULL
) ON [PRIMARY]
GO
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]
GO
ALTER TABLE [dbo].[Translators]  WITH CHECK ADD  CONSTRAINT
[FK_Translators_Employees] FOREIGN KEY([TranslatorID])
REFERENCES [dbo].[Employees] ([UserID])
GO
ALTER TABLE [dbo].[Translators] CHECK CONSTRAINT [FK_Translators_Employees]
GO
```

Teachers - tabela zawiera ID użytkowników, którzy są nauczycielami

```
CREATE TABLE [dbo].[Teachers](
    [TeacherID] [int] NOT NULL,
 CONSTRAINT [PK_Teachers] PRIMARY KEY CLUSTERED
(
    [TeacherID] 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].[Teachers]  WITH CHECK ADD  CONSTRAINT [FK_Teachers_Employees]
FOREIGN KEY([TeacherID])
REFERENCES [dbo].[Employees] ([UserID])
GO

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

TranslatorLanguages - tabela zawierająca informacje o tym jaki który tłumacz tłumaczy jaki język

```
CREATE TABLE [dbo].[TranslatorLanguages](
    [TranslatorID] [int] NOT NULL,
    [LanguageID] [int] NOT NULL
) ON [PRIMARY]
GO
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]
GO
ALTER TABLE [dbo].[TranslatorLanguages]  WITH CHECK ADD  CONSTRAINT
[FK_TranslatorLanguages_Languages] FOREIGN KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
GO
ALTER TABLE [dbo].[TranslatorLanguages] CHECK CONSTRAINT
[FK_TranslatorLanguages_Languages]
GO
ALTER TABLE [dbo].[TranslatorLanguages]  WITH CHECK ADD  CONSTRAINT
[FK_TranslatorLanguages_Translators] FOREIGN KEY([TranslatorID])
REFERENCES [dbo].[Translators] ([TranslatorID])
GO
ALTER TABLE [dbo].[TranslatorLanguages] CHECK CONSTRAINT
[FK_TranslatorLanguages_Translators]
GO
```

Languages - tabela zawierająca języki które są tłumaczone

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Languages](
    [LanguageID] [int] NOT NULL,
    [LanguageName] [varchar](50) NOT NULL
) ON [PRIMARY]
GO
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]
```

```
GO
```

Webinarium - tabela zawierająca wszystkie webinary w systemie

- WebinarID (PK) - identyfikator webinaru
- TeacherID (FK) - identyfikator nauczyciela prowadzącego zajęcia
- TranslatorID (FK) - identyfikator tłumacza
- Name - nazwa webinaru
- Descryption - opis webinaru
- Price - cena webinaru (jeśli free to 0)
- Type (free or not) - darmowy czy płatny
- Link (to external website) - link do webinaru
- StartDateAndTime - dokładna data starru webinaru
- ExpirationDate - data wygaśnięcia webinaru
- WebinariumDuration - czas trwania webinaru

```
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]
GO

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

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

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

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

WebinarAttendance - tabela zawierająca informacje o użytkownikach obecnych na webinarze

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

```
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]
GO

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

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

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

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

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

- Room - numer sali w której odbywają się zajęcia
- StartDateAndTime - data i godzina o której odbędą się zajęcia
- Type - typ stacjonarne/zdalne/zdalne asynchroniczne
- Link - link do zewnętrznego komunikatora (jeżeli stacjonarne to NULL)
- StudyMeetingPrice - koszt zjazdu
- StudyMeetingDuration - czas trwania spotkania

```
CREATE TABLE [dbo].[StudyMeetings](
    [StudyMeetingsID] [int] NOT NULL,
    [GatheringID] [int] NULL,
    [TeacherID] [int] NULL,
    [TranslatorID] [int] NULL,
    [Room] [nchar](20) NULL,
    [StartDateAndTime] [datetime] NULL,
    [Type] [nchar](20) NULL,
    [Link] [text] NULL,
    [StudyMeetingPrice] [money] NULL,
    [StudyMeetingDuration] [time](7) NULL,
 CONSTRAINT [PK_StudyMeetings] PRIMARY KEY CLUSTERED
(
    [StudyMeetingsID] 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]
GO

ALTER TABLE [dbo].[StudyMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_StudyMeetings_Employees] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[Employees] ([UserID])
GO

ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [FK_StudyMeetings_Employees]
GO

ALTER TABLE [dbo].[StudyMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_StudyMeetings_Employees1] FOREIGN KEY([TranslatorID])
REFERENCES [dbo].[Employees] ([UserID])
GO

ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [FK_StudyMeetings_Employees1]
GO

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

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

ALTER TABLE [dbo].[StudyMeetings]  WITH CHECK ADD  CONSTRAINT [CK_StudyMeetings]
```

```
CHECK  (([Type] like 'stationary%' OR [Type] like 'on-line%' OR [Type] like
'hybrid%'))
GO

ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [CK_StudyMeetings]
GO
```

StudyAttendance - Zawiera informacje o obecności studenta na zajęciach na studiach

- MeetingID - identyfikator webinaru
- UserID - identyfikator studenta

```
CREATE TABLE [dbo].[StudyMeetings](
    [StudyMeetingsID] [int] NOT NULL,
    [GatheringID] [int] NULL,
    [TeacherID] [int] NULL,
    [TranslatorID] [int] NULL,
    [Room] [nchar](20) NULL,
    [StartDateAndTime] [datetime] NULL,
    [Type] [nchar](20) NULL,
    [Link] [text] NULL,
    [StudyMeetingPrice] [money] NULL,
    [StudyMeetingDuration] [time](7) NULL,
 CONSTRAINT [PK_StudyMeetings] PRIMARY KEY CLUSTERED
(
    [StudyMeetingsID] 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]
GO

ALTER TABLE [dbo].[StudyMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_StudyMeetings_Employees] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[Employees] ([UserID])
GO

ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [FK_StudyMeetings_Employees]
GO

ALTER TABLE [dbo].[StudyMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_StudyMeetings_Employees1] FOREIGN KEY([TranslatorID])
REFERENCES [dbo].[Employees] ([UserID])
GO

ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [FK_StudyMeetings_Employees1]
GO

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

```
GO

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

ALTER TABLE [dbo].[StudyMeetings]  WITH CHECK ADD  CONSTRAINT [CK_StudyMeetings]
CHECK  (([Type] like 'stationary%' OR [Type] like 'on-line%' OR [Type] like
'hybrid%'))
GO

ALTER TABLE [dbo].[StudyMeetings] CHECK CONSTRAINT [CK_StudyMeetings]
GO
```

Studies - zawiera informacje dotyczące kierunków studiów

- StudiumID - klucz główny identyfikatora studiów
- Name - nazwa kierunku
- Syllabus - opis studiów
- NumberOfSeats - liczba miejsc na dany kierunek
- EntryFee - wpisowe na studia
- Language - język w jakim prowadzone są studia
- StudyType - 1/2/3 (używane w słowniku EventTypes)
- StartDate - data rozpoczęcia studiów
- EndDate - data zakończenia studiów

```
USE [u_pismiale]
GO

/****** Object:  Table [dbo].[Studies]    Script Date: 14.01.2024 23:08:26 ******/
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[Studies](
    [StudiumID] [int] NOT NULL,
    [Name] [varchar](50) NOT NULL,
    [Syllabus] [text] NOT NULL,
    [NumberOfSeats] [int] NULL,
    [EntryFee] [money] NULL,
    [Language] [varchar](50) NULL,
    [StudyType] [int] NULL,
    [StartDate] [datetime] NULL,
    [EndDate] [datetime] NULL,
 CONSTRAINT [PK_Studies] PRIMARY KEY CLUSTERED
(
    [StudiumID] 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]
GO

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

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

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [CK_Studies] CHECK
(([StudyType]=(3) OR [StudyType]=(2) OR [StudyType]=(1)))
GO

ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT [CK_Studies]
GO
```

Widoki:

TeachersInfo Wyświetla informacje o nauczycielach w systemie.

```
CREATE VIEW [dbo].[TeachersInfo]
AS
SELECT dbo.Users.UserID, dbo.Users.FirstName, dbo.Users.LastName
FROM   dbo.Employees INNER JOIN
            dbo.Users ON dbo.Employees.UserID = dbo.Users.UserID
WHERE (dbo.Employees.Role LIKE 'Teacher')
```

StudentsInfo Wyświetla informacje o studentach w systemie.

```
CREATE VIEW [dbo].[StudentsInfo]
AS
SELECT        dbo.Students.UserID AS Expr1, dbo.Users.*
FROM            dbo.Students INNER JOIN
                         dbo.Users ON dbo.Students.UserID = dbo.Users.UserID
```

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

```
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. Pokazuje cenę, sylabus, datę, czas trwania i język w jakim będzie prowadzony webinar.

```
CREATE VIEW [dbo].[WebinarsInfo]
AS
SELECT   Name, Description, Price, StartDateAndTime, WebinarDuration, Language
FROM         dbo.Webinarium
```

CoursesInfo Wyświetla informacje o dostępnych kursach. Pokazuje cenę za cały kurs, zaliczkę, date kursu, rodzaj kursu i język w jakim będzie prowadzony.

```
CREATE VIEW [dbo].[CoursesInfo]
AS
SELECT   Name, Description, Prepayment, FullPayment, CourseStartDate, Type,
Language
FROM         dbo.Courses
```

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

```
CREATE VIEW [dbo].[StudiesInfo]
AS
SELECT        Name, Syllabus, NumberOfSeats, EntryFee, Language
FROM            dbo.Studies
```

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

```
SELECT dbo.Courses.Name, CAPM.CourseID, CAPM.MeetingID, CASE WHEN
CNRU.NumberRegistered = 0 THEN NULL ELSE CAST(CAPM.NumberAttended AS FLOAT) /
CNRU.NumberRegistered END AS Attendance, CAPM.NumberAttended,
CNRU.NumberRegistered
FROM   dbo.CourseAttendancePerMeeting() 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.

```
SELECT dbo.Courses.CourseID, dbo.Courses.Name,
AVG(dbo.AttendancePerCourseMeeting.Attendance) AS AverageAttendance
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
```

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

```
SELECT dbo.Webinarium.WebinarID, dbo.Webinarium.Name, CASE WHEN NumberRegistered =
0 THEN NULL ELSE CAST(WAPM.NumberAttended AS FLOAT) / NumberRegistered END AS
Attendance
FROM   dbo.WebinariumAttendancePerMeeting() AS WAPM INNER JOIN
            dbo.Webinarium ON WAPM.WebinarID = dbo.Webinarium.WebinarID LEFT
OUTER JOIN
                (SELECT WebinarID, COUNT(UserID) AS NumberRegistered
                FROM    dbo.Registration
                GROUP BY WebinarID) AS Subquery ON dbo.Webinarium.WebinarID =
Subquery.WebinarID
WHERE (dbo.Webinarium.EndDateAndTime < GETDATE())
```

BilocationReport Wyświetla raport bilokacji czyli wszystkich użytkowników, którym nakladają sie realizowane szkolenia. Widok zwraca funkcje której sygnatura jest podana dalej.

```
SELECT UserID
FROM dbo.GetBilocationReport() AS getReport
```

CoursesInfo Wyświetla informacje o kursach

```
SELECT        Name, Description, Prepayment, FullPayment, CourseStartDate, Type,
Language
FROM            dbo.Courses
```

FutureCourses Wyświetla przyszłe kursy

```
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
```

FutureEventsRegistrationNumber Zwraca liczbe osób zapisanych na przyszłe wydarzenia

```
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.OrderID 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

```
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
ORDER BY dbo.Studies.StudiumID
```

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;

    -- Sprawdź, czy UserID jest przypisany do kursu o podanym CourseID
    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
        -- UserID jest przypisany do kursu, więc dodaj rekord do tabeli
CourseAttendance
        INSERT INTO dbo.CourseAttendance (MeetingID, UserID)
        VALUES (@MeetingID, @UserID);
    END
    ELSE
    BEGIN
        --zwrócenie komunikatu o błędzie
        PRINT 'UserID is not assigned to the specified CourseID.';
    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
        --
        PRINT 'UserID is not assigned to the specified Study Meeting.';
    END
END
GO
```

AddWebinarAttendance Procedura dodaje użytkownika do listy obecności na podanym 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
        PRINT 'MeetingID or UserID invalid';
    END
END
GO
```

AddRegistrationAdmin Procedura rejestracje 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
```

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
```

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
```

AddRegistrationUser Dodaje rejestracje użytkownika

```
CREATE PROCEDURE [dbo].[AddRegistrationUser]
    @EventType TEXT,
    @UserID INT,
    @RegistrationDate DATE,
    @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,
        GETDATE(),
        0.0,
        'in_progess',
        @CourseID,
        @WebinarID,
        @StudiumID
    );
END
```

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
    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,
GatheringDate, @RegID from Gatherings g
        where g.StudiumID = @StudiumID
        -- znajduje i dodaje wszystkie zjazdy
    END
    ELSE IF @CourseID IS NOT NULL
    BEGIN
        DECLARE @id2 INT;
        select @Amount1 = Prepayment, @DueDate1 = CourseStartDate 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 = CourseEndDate 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:

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 u
    ON s.UserID = u.UserID
    WHERE @CourseID = reg.CourseID
    )
    SELECT t.UserID as ID, t.FirstName as 'First Name', t.LastName as 'Last Name',
cm.StartDateAndTime as 'Date',
        CASE
            WHEN ca.UserID IS NOT NULL THEN 'PRESENT'
            ELSE 'ABSENT'
        END AS AttendanceStatus
    FROM t
    LEFT JOIN CourseMeetings cm
    ON cm.CourseID = @CourseID
    LEFT JOIN CourseAttendance ca
    ON cm.MeetingID = ca.MeetingID AND t.UserID = ca.UserID
)
```

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 u
    ON s.UserID = u.UserID
    WHERE @StudiumID = reg.StudiumID
    )
    SELECT t.UserID as ID, t.FirstName as 'First Name', t.LastName as 'Last Name',
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
)
```

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 ID, t.FirstName as 'First Name', t.LastName as 'Last Name',
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
)
```

CourseAttendancePerMeeting Zwraca id meetingu, id kursu do którego należy i liczbę obenych studentów. Funkcja użyta w widoku AttendancePerCourseMeeting

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

```
    SELECT CourseMeetings.CourseID, CourseMeetings.MeetingID,
COUNT(CourseAttendance.UserID) AS NumberAttended
    FROM CourseMeetings JOIN CourseAttendance on
CourseMeetings.MeetingID=CourseAttendance.MeetingID
    GROUP BY CourseMeetings.CourseID, CourseMeetings.MeetingID
)
```

CourseNumberRegisteredUsers Zwraca id kursu, liczbę użytkowników zarejestrowanych na dany kurs. Funkcja użyta w widoku AttendancePerCourseMeeting.

```
CREATE FUNCTION [dbo].[CourseNumberRegisteredUsers]()
RETURNS TABLE
AS
RETURN
(
    SELECT Courses.CourseID, COUNT(Registration.RegistrationID) AS
NumberRegistered
    FROM Registration RIGHT OUTER JOIN Courses on
Registration.CourseID=Courses.CourseID
    GROUP BY Courses.CourseID
)
```

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)
)
```

GetFutureEvents Funckaj zwraca przszłe wydarzenia

```
CREATE FUNCTION [dbo].[GetFutureEvents]()
RETURNS TABLE
AS
RETURN
(
    -- Zapytanie UNION trzech tabel
    (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 st
    WHERE st.StartDate > GETDATE())
)
GO
```

GetPastEvents Funkcja zwraca eventy które już sie wydarzyły

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

RegisteredUsersEvens Funkcja przyjmuje ID usera i zwraca eventy na które jest zapisany

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

GetFutureEventsParticipants Zwraca tablice uzytkownikow ktorzy sa zapisani na jakies przyszle wydarzenia

```
CREATE FUNCTION [dbo].[GetFutureEventsParticipants]()
RETURNS TABLE
AS
RETURN
(
    --zwraca tablice uzytkownikow ktorzy sa zapisani na jakies przyszle wydarzenia
    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
```