Bazy Danych: Projekt

Raport

Zespół 5: Furgała Tomasz, Łukasz Kluza, Mateusz Sacha

1. Administrator

- Usuwanie webinaru, administrator może usunąć dostępne nagranie webinaru gdy uzna to za stosowne.
- Zarządzanie użytkownikami, administrator ma możliwość edycji kont innych użytkowników.
- Generowanie raportów, administrator generuję raporty zawierająca aktualne statystki.

2. Gość

- Założenie konta, użytkownik może założyć konto, które umożliwia mu korzystanie z systemu
- Przeglądanie kursów, użytkownik ma możliwość zapoznania się z aktualną ofertą kursów i szkoleń.

3. Zalogowany użytkownik

- Zapis na webinar, kurs lub studia, użytkownik może zapisać się na wybraną przez siebie usługę.
- Płatność za usługi, dokonuje opłaty by móc wziąć udział w webinarze, kursie lub studiach oraz wykupuje późniejszy dostęp do materiałów.
- Przeglądanie listy, możliwość przeglądania listy usług, na które dany użytkownik jest zapisany.
- Odbiera dyplom, użytkownik może odebrać dyplom, gdy zostanie on wystawiony przez administratora.

4. Koordynator

- Odraczanie płatności, dyrektor szkoły ma możliwość odroczenia płatności na określony czas.
- Wgląd do kursów oraz webinarów, dyrektor ma możliwość wglądu do danych o kursach i webinarach prowadzonych przez jego pracowników
- Zatwierdzanie programu studiów, dyrektor ma dostęp do ułożonych przez pracowników sylabusów przed opublikowaniem ich oraz możliwość zatwierdzania i wprowadzania poprawek do nich
- Zatwierdzanie nowych kursów i webinarów, dyrektor zatwierdza bądź odrzuca każdy nowy kurs, webinar, stworzony przez jego pracowników

5. Menadżer

- Zarządzaniem limitem miejsc, menadżer ustala maksymalną liczbę osób która może uczestniczyć w danym webinarze, szkoleniu
- Wystawianie dyplomów, menadżer wystawia dyplom użytkownikowi, który spełnił wszystkie regulaminowe przesłanki co to do tego.
- Zarządzanie ofertą, menadżer ma możliwość edycji obecnej oferty jak i możliwość dodawania nowych kursów, szkoleń.

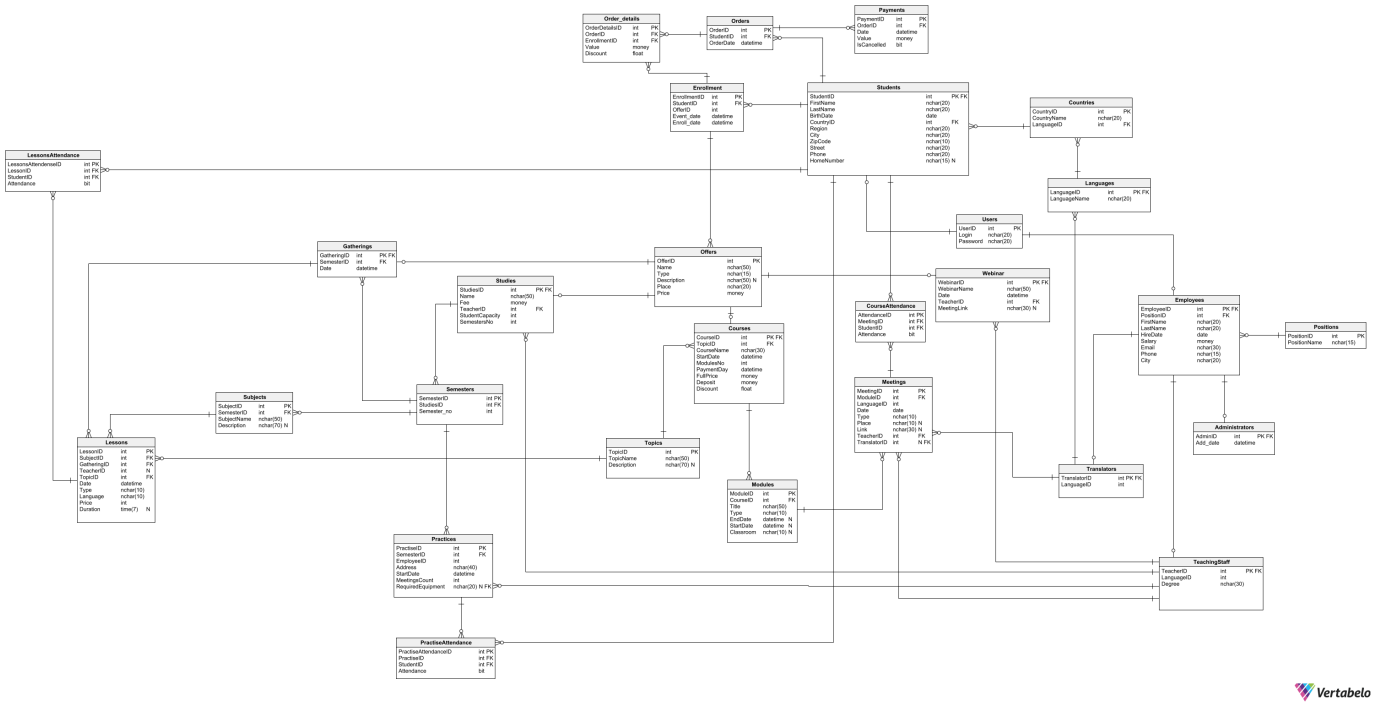
6. Prowadzący/Wykładowca

- Dostęp do swoich webinarów, każdy prowadzący ma nielimitowany czasowo dostęp do nagrań wszystkich swoich webinarów
- Możliwość edycji modułów kursu, prowadzący mają możliwość wprowadzania poprawek oraz modyfikacji materiałów znajdujących się na prowadzonych przez siebie kursach
- Dostęp do systemu ocen i obecności, prowadzący ma dostęp do systemu, w którym może swobodnie zapisywać oraz zmieniać oceny i obecności uczestników jego kursów
- Ułożenie sylabusu, prowadzący musi ułożyć sylabus do każdego z prowadzonych przez siebie przedmiotów w określonym terminie przed rozpoczęciem studiów

7. System

- Generowanie linków do płatności, system sam, automatycznie generuje link do płatności, gdy użytkownik chce opłacić zamówienie.
- Wysyłanie powiadomień, uczestnik spotkania dostaje powiadomienia, gdy rozpoczyna się spotkanie, w którym ma uczestniczyć.
- Powiadomienie o zapłacie, użytkownik do dostaje przypomnienie o konieczności zapłaty tydzień przed ostatecznym terminem dokonania płatności, dotyczy to także zaliczek.

Diagram bazy danych:



Tabele:

1. Offers:

Tabela zawiera informacje o wszystkich wydarzeniach jakie są oferowane. Zawiera idetyfikator wydarzenia (OfferID), nazwe, opis oraz typ (Name, Description, Type), typ określa czy jest to webinar, kurs, studia czy pojedyńcza lekcja. Dodatkowo miejsce wydarzenia oraz jego całkowity koszt (Place, Price).

```
CREATE TABLE [dbo].[Offers](
    [OfferID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nchar](50) NOT NULL,
    [Type] [nchar](15) NOT NULL,
    [Description] [nchar](50) NULL,
    [Place] [nchar](20) NOT NULL,
    [Price] [money] NOT NULL,
 CONSTRAINT [PK_Offers] PRIMARY KEY CLUSTERED
(
    [OfferID] 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].[Offers]  WITH CHECK ADD  CONSTRAINT [CHK_Name_Length] CHECK  ((len([Name])>=(5)))

ALTER TABLE [dbo].[Offers] CHECK CONSTRAINT [CHK_Name_Length]

ALTER TABLE [dbo].[Offers]  WITH CHECK ADD  CONSTRAINT [CHK_Price_NonNegative] CHECK  (([Price]>=(0)))

ALTER TABLE [dbo].[Offers] CHECK CONSTRAINT [CHK_Price_NonNegative]

ALTER TABLE [dbo].[Offers]  WITH CHECK ADD  CONSTRAINT [CHK_Type_Values] CHECK  (([Type]='Gathering' OR
[Type]='Lesson' OR [Type]='Studies' OR [Type]='Courses' OR [Type]='Webinar'))

ALTER TABLE [dbo].[Offers] CHECK CONSTRAINT [CHK_Type_Values]
```

2. Webinar:

Tabela zawiera informacje o webianrach, zawiera klucz główny (WebinarID), nazwę oraz datę rozpoczęcia (WebinarName, Date), inforamcje o osbie, która to prowadzi (TeacherID) i link do webinaru (MeetingLink).

```
CREATE TABLE [dbo].[Webinar](
    [WebinarID] [int] NOT NULL,
    [WebinarName] [nchar](50) NOT NULL,
    [Date] [datetime] NOT NULL,
    [TeacherID] [int] NOT NULL,
    [MeetingLink] [nchar](30) NULL,
 CONSTRAINT [PK_Webinar] 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]

ALTER TABLE [dbo].[Webinar]  WITH CHECK ADD  CONSTRAINT [FK_Webinar_Offers] FOREIGN KEY([WebinarID])
REFERENCES [dbo].[Offers] ([OfferID])

ALTER TABLE [dbo].[Webinar] CHECK CONSTRAINT [FK_Webinar_Offers]

ALTER TABLE [dbo].[Webinar]  WITH CHECK ADD  CONSTRAINT [FK_Webinar_TeachingStaff] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[TeachingStaff] ([TeacherID])

ALTER TABLE [dbo].[Webinar] CHECK CONSTRAINT [FK_Webinar_TeachingStaff]

ALTER TABLE [dbo].[Webinar]  WITH CHECK ADD  CONSTRAINT [CHK_Webinar_WebinarName_Length] CHECK
((len([WebinarName])>(5)))

ALTER TABLE [dbo].[Webinar] CHECK CONSTRAINT [CHK_Webinar_WebinarName_Length]
```

3. Studies:

Tabela zawiera informacje o studiach, zawiera klucz główny (StudiesID), kierunku studiów oraz opłacie za nie (Name, Fee), koorynatorze, maksymalnej ilości studentów na danym studium (MEnagerID, StudentCapacity).

```
CREATE TABLE [dbo].[Studies](
    [StudiesID] [int] NOT NULL,
    [Name] [nchar](50) NOT NULL,
    [Fee] [money] NOT NULL,
    [MenagerID] [int] NOT NULL,
    [StudentCapacity] [int] NOT NULL,
 CONSTRAINT [PK_Studies_1] PRIMARY KEY CLUSTERED
(
    [StudiesID] 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]S

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [FK_Studies_Employees] FOREIGN KEY([MenagerID])
REFERENCES [dbo].[Employees] ([EmployeeID])

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

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [FK_Studies_Offers] FOREIGN KEY([StudiesID])
REFERENCES [dbo].[Offers] ([OfferID])

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

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [CHK_Fee_NonNegative] CHECK  (([Fee]>=(0)))

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

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [CHK_StudentCapacity_Minimum] CHECK
(([StudentCapacity]>=(10)))

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

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [CHK_Studies_Name_Length] CHECK  ((len([Name])>(5)))
```

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

4. Courses:

Tabela zawiera spis wszystkich kursów z kluczem głównym (CourseID), posiada informację o temacie kursu oraz jego nazwie (TopicID, CourseName), a także dacie rozpoczęcia, ilości modułów z których kurs się składa i dacie zapłaty (StartDate, ModulesNo, PaymentDay), całkowitej kwocie jaką należy za kurs zapłacić, kwocie zaliczki oraz zniżce (FullPrice, Deposit, Discount).

```
CREATE TABLE [dbo].[Courses](
    [CourseID] [int] NOT NULL,
    [TopicID] [int] NOT NULL,
    [CourseName] [nchar](30) NOT NULL,
    [StartDate] [datetime] NOT NULL,
    [ModulesNo] [int] NOT NULL,
    [PaymentDay] [datetime] NOT NULL,
    [FullPrice] [money] NOT NULL,
    [Deposit] [money] NOT NULL,
    [Discount] [float] NOT 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]

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [FK_Courses_Offers] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Offers] ([OfferID])

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [FK_Courses_Topics] FOREIGN KEY([TopicID])
REFERENCES [dbo].[Topics] ([TopicID])

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CHK_Courses_CourseName_Length] CHECK
((len([CourseName])>(5)))

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CHK_Deposit_Range] CHECK  (([Deposit]>=(0) AND
[Deposit]<=[FullPrice]))

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CHK_Discount_Range] CHECK  (([Discount]>=(0) AND
[Discount]<=(1)))

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CHK_FullPrice_NonNegative] CHECK  (([FullPrice]>=
(0)))

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CHK_ModulesNo_Positive] CHECK  (([ModulesNo]>(0)))

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CHK_PaymentDay_BeforeStart] CHECK  (([PaymentDay]
<=dateadd(day,(-3),[StartDate])))

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

5. Gatherings:

Tabela zawiera informacje o zjazdach, posaida klucz główny (GatheringID) i semestr, w ramach którego odbywa się dany zjazd oraz datę w której zjazd się odbywa (SemestrID, Date).

```
CREATE TABLE [dbo].[Gatherings](
    [GatheringID] [int] NOT NULL,
    [Semester] [int] NOT NULL,
    [Date] [datetime] NOT NULL,
 CONSTRAINT [PK_Gatherings] PRIMARY KEY CLUSTERED
(
    [GatheringID] 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].[Gatherings]  WITH CHECK ADD  CONSTRAINT [FK_Gatherings_Offers] FOREIGN KEY([GatheringID])
REFERENCES [dbo].[Offers] ([OfferID])

ALTER TABLE [dbo].[Gatherings] CHECK CONSTRAINT [FK_Gatherings_Offers]

ALTER TABLE [dbo].[Gatherings]  WITH CHECK ADD  CONSTRAINT [FK_Gatherings_Semesters] FOREIGN KEY([Semester])
REFERENCES [dbo].[Semesters] ([SemesterID])

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

6. Semesters:

W tabeli znajdują się informacje o wszystkich semestrach na wszystkich kierunkach studiów, klucz główny to (SemesterID), zawiera też informacje o kierunku studiów na którym semestr się znajduje, numerze semestru(StudiesID, Semester_no).

```
CREATE TABLE [dbo].[Semesters](
    [SemesterID] [int] NOT NULL,
    [StudiesID] [int] NOT NULL,
    [Semester_no] [int] NOT NULL,
 CONSTRAINT [PK_Semesters] PRIMARY KEY CLUSTERED
(
    [SemesterID] 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].[Semesters]  WITH CHECK ADD  CONSTRAINT [FK_Semesters_Studies] FOREIGN KEY([StudiesID])
REFERENCES [dbo].[Studies] ([StudiesID])

ALTER TABLE [dbo].[Semesters] CHECK CONSTRAINT [FK_Semesters_Studies]

ALTER TABLE [dbo].[Semesters]  WITH CHECK ADD  CONSTRAINT [CHK_Semester_no_Positive] CHECK  (([Semester_no]>
(0)))

ALTER TABLE [dbo].[Semesters] CHECK CONSTRAINT [CHK_Semester_no_Positive]
```

7. Practices:

Tabela zawiera dane o praktykach, posiada klucz główny (PractiseID), semestrze na którym się odbywają i pracowniku, który je prowadzi (SemesterID, EmployeeID), posiada informacje o miejscu, w kótrym praktyki się odbywają, dacie rozpoczęcia, ilości spotkań oraz potrzebnym wyposażeniu (Address, StartDate, MeetingsCount, RequiredEquipment).

```
CREATE TABLE [dbo].[Practices](
    [PractiseID] [int] NOT NULL,
    [SemesterID] [int] NOT NULL,
    [EmployeeID] [int] NOT NULL,
    [Address] [nchar](40) NOT NULL,
```

```
    [StartDate] [datetime] NOT NULL,
    [MeetingsCount] [int] NOT NULL,
    [RequiredEquipment] [nchar](20) NULL,
 CONSTRAINT [PK_Practices] PRIMARY KEY CLUSTERED
(
    [PractiseID] 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].[Practices]  WITH CHECK ADD  CONSTRAINT [FK_Practices_Semesters] FOREIGN KEY([SemesterID])
REFERENCES [dbo].[Semesters] ([SemesterID])

ALTER TABLE [dbo].[Practices] CHECK CONSTRAINT [FK_Practices_Semesters]

ALTER TABLE [dbo].[Practices]  WITH CHECK ADD  CONSTRAINT [FK_Practices_TeachingStaff] FOREIGN
KEY([EmployeeID])
REFERENCES [dbo].[TeachingStaff] ([TeacherID])

ALTER TABLE [dbo].[Practices] CHECK CONSTRAINT [FK_Practices_TeachingStaff]

ALTER TABLE [dbo].[Practices]  WITH CHECK ADD  CONSTRAINT [CHK_MeetingsCount_Positive] CHECK
(([MeetingsCount]>(0)))

ALTER TABLE [dbo].[Practices] CHECK CONSTRAINT [CHK_MeetingsCount_Positive]
```

8. PractiseAttendance:

Tabela posiada informacje o obecności studentów na praktykach, posiada klucz główny (PractiseAttendanceID), dla każdego studenta przypisuje czy był obecny na danych praktykach, na które jest zapisany (PractiseID, StudentID, Attendance).

```
CREATE TABLE [dbo].[PractiseAttendance](
    [PractiseAttendanceID] [int] NOT NULL,
    [PractiseID] [int] NOT NULL,
    [StudentID] [int] NOT NULL,
    [Attendance] [bit] NOT NULL,
 CONSTRAINT [PK_PractiseAttendance] PRIMARY KEY CLUSTERED
(
    [PractiseAttendanceID] 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].[PractiseAttendance]  WITH CHECK ADD  CONSTRAINT [FK_PractiseAttendance_Lessons] FOREIGN
KEY([StudentID])
REFERENCES [dbo].[Lessons] ([LessonID])

ALTER TABLE [dbo].[PractiseAttendance] CHECK CONSTRAINT [FK_PractiseAttendance_Lessons]

ALTER TABLE [dbo].[PractiseAttendance]  WITH CHECK ADD  CONSTRAINT [FK_PractiseAttendance_Practices] FOREIGN
KEY([PractiseID])
REFERENCES [dbo].[Practices] ([PractiseID])

ALTER TABLE [dbo].[PractiseAttendance] CHECK CONSTRAINT [FK_PractiseAttendance_Practices]

ALTER TABLE [dbo].[PractiseAttendance]  WITH CHECK ADD  CONSTRAINT [FK_PractiseAttendance_Students] FOREIGN
KEY([StudentID])
REFERENCES [dbo].[Students] ([StudentID])

ALTER TABLE [dbo].[PractiseAttendance] CHECK CONSTRAINT [FK_PractiseAttendance_Students]
```

9. Subjects:

Tabela zawiera informacje o przedmiotach występujących w semestrach z kluczem głównym (SubjectID), przypisuje przemiot do określonego semestru, posiada nazwę przedmiotu oraz jego opis (SemesterID, SubjectName, Description).

```
CREATE TABLE [dbo].[Subjects](
    [SubjectID] [int] NOT NULL,
    [SemesterID] [int] NOT NULL,
    [SubjectName] [nchar](50) NOT NULL,
    [Description] [nchar](70) NULL,
 CONSTRAINT [PK_Subjects] PRIMARY KEY CLUSTERED
(
    [SubjectID] 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].[Subjects]  WITH CHECK ADD  CONSTRAINT [FK_Subjects_Semesters] FOREIGN KEY([SemesterID])
REFERENCES [dbo].[Semesters] ([SemesterID])

ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [FK_Subjects_Semesters]

ALTER TABLE [dbo].[Subjects]  WITH CHECK ADD  CONSTRAINT [CHK_Subjects_SubjectName_Length] CHECK
((len([SubjectName])>(5)))

ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [CHK_Subjects_SubjectName_Length]
```

10. Lessons:

Tabela zawiera informacje o lekcjach zarówno tych na studiach, oraz tych możliwych do kupienia pojedynczo, posida klucz główny (LessonID), przedmiot i zjazd do którego jest przypisana dana lekcja, oraz nauczyciela który ją prowadzi (SubjectID, GatheringID, TeacherID) zawiera temat, datę, typ, język prowadzenia, cenę i czas trwania (TopicID, Date, Type, Language, Price, Duration).

```
CREATE TABLE [dbo].[Lessons](
    [LessonID] [int] NOT NULL,
    [SubjectID] [int] NOT NULL,
    [GatheringID] [int] NOT NULL,
    [TeacherID] [int] NOT NULL,
    [TopicID] [int] NOT NULL,
    [Date] [datetime] NOT NULL,
    [Type] [nchar](10) NOT NULL,
    [Language] [nchar](10) NOT NULL,
    [Price] [int] NOT NULL,
    [Duration] [time](7) NULL,
 CONSTRAINT [PK_Lessons] PRIMARY KEY CLUSTERED
(
    [LessonID] 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].[Lessons]  WITH CHECK ADD  CONSTRAINT [FK_Lessons_Gatherings] FOREIGN KEY([GatheringID])
REFERENCES [dbo].[Gatherings] ([GatheringID])

ALTER TABLE [dbo].[Lessons] CHECK CONSTRAINT [FK_Lessons_Gatherings]

ALTER TABLE [dbo].[Lessons]  WITH CHECK ADD  CONSTRAINT [FK_Lessons_Subjects] FOREIGN KEY([SubjectID])
REFERENCES [dbo].[Subjects] ([SubjectID])

ALTER TABLE [dbo].[Lessons] CHECK CONSTRAINT [FK_Lessons_Subjects]

ALTER TABLE [dbo].[Lessons]  WITH CHECK ADD  CONSTRAINT [FK_Lessons_TeachingStaff] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[TeachingStaff] ([TeacherID])

ALTER TABLE [dbo].[Lessons] CHECK CONSTRAINT [FK_Lessons_TeachingStaff]

ALTER TABLE [dbo].[Lessons]  WITH CHECK ADD  CONSTRAINT [FK_Lessons_Topics] FOREIGN KEY([TopicID])
REFERENCES [dbo].[Topics] ([TopicID])

ALTER TABLE [dbo].[Lessons] CHECK CONSTRAINT [FK_Lessons_Topics]
```

```
ALTER TABLE [dbo].[Lessons]  WITH CHECK ADD  CONSTRAINT [CHK_Lessons_Type] CHECK  (([Type]='online' OR
[Type]='hybrid' OR [Type]='stationary'))

ALTER TABLE [dbo].[Lessons] CHECK CONSTRAINT [CHK_Lessons_Type]
```

11. LessonsAttendance:

Tabela posiada informacje o obecności studentów na lekcjach, posiada klucz główny (LessonsAttendenseID), dla każdego studenta przypisuje czy był obecny na danej lekcji, na którą jest zapisany (LessonID, StudentID, Attendance).

```
CREATE TABLE [dbo].[LessonsAttendance](
    [LessonsAttendenseID] [int] NOT NULL,
    [LessonID] [int] NOT NULL,
    [StudentID] [int] NOT NULL,
    [Attendance] [bit] NOT NULL,
 CONSTRAINT [PK_LessonsAttendance] PRIMARY KEY CLUSTERED
(
    [LessonsAttendenseID] 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].[LessonsAttendance]  WITH CHECK ADD  CONSTRAINT [FK_LessonsAttendance_Lessons] FOREIGN
KEY([LessonID])
REFERENCES [dbo].[Lessons] ([LessonID])

ALTER TABLE [dbo].[LessonsAttendance] CHECK CONSTRAINT [FK_LessonsAttendance_Lessons]

ALTER TABLE [dbo].[LessonsAttendance]  WITH CHECK ADD  CONSTRAINT [FK_LessonsAttendance_Students] FOREIGN
KEY([StudentID])
REFERENCES [dbo].[Students] ([StudentID])

ALTER TABLE [dbo].[LessonsAttendance] CHECK CONSTRAINT [FK_LessonsAttendance_Students]

ALTER TABLE [dbo].[LessonsAttendance]  WITH CHECK ADD  CONSTRAINT [FK_LessonsAttendance_Students1] FOREIGN
KEY([StudentID])
REFERENCES [dbo].[Students] ([StudentID])

ALTER TABLE [dbo].[LessonsAttendance] CHECK CONSTRAINT [FK_LessonsAttendance_Students1]
```

12. Topics:

Tabela posiada dane o tematach kursów, bądź lekcji, posiada klucz główny (TopicID) oraz nazwę tematu i jego opis (TopicName, Description).

```
CREATE TABLE [dbo].[Topics](
    [TopicID] [int] NOT NULL,
    [TopicName] [nchar](50) NOT NULL,
    [Description] [nchar](70) NULL,
 CONSTRAINT [PK_Topics] PRIMARY KEY CLUSTERED
(
    [TopicID] 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].[Topics]  WITH CHECK ADD  CONSTRAINT [CHK_Topics_TopicName_Length] CHECK
((len([TopicName])>(5)))

ALTER TABLE [dbo].[Topics] CHECK CONSTRAINT [CHK_Topics_TopicName_Length]
```

13. Modules:

Tabela zawiera wszystkie moduły, znajdujące się kursach, posiada klucz główny (ModuleID), informacje o kursie, do którego moduł należy oraz jego tytule i typie (CourseID, Title, Type), a także dacie zakończenia i rozpoczęcia oraz klasie, w której się odbywa (EndDate, StartDate, Classroom).

```
CREATE TABLE [dbo].[Modules](
    [ModuleID] [int] IDENTITY(1,1) NOT NULL,
    [CourseID] [int] NOT NULL,
    [Title] [nchar](50) NOT NULL,
    [Type] [nchar](10) NOT NULL,
    [EndDate] [datetime] NULL,
    [StartDate] [datetime] NULL,
    [Classroom] [nchar](10) NULL,
 CONSTRAINT [PK_Modules] PRIMARY KEY CLUSTERED
(
    [ModuleID] 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].[Modules]  WITH CHECK ADD  CONSTRAINT [FK_Modules_Courses] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Courses] ([CourseID])

ALTER TABLE [dbo].[Modules] CHECK CONSTRAINT [FK_Modules_Courses]

ALTER TABLE [dbo].[Modules]  WITH CHECK ADD  CONSTRAINT [CHK_Modules_Date_Order] CHECK  (([EndDate]>
[StartDate]))

ALTER TABLE [dbo].[Modules] CHECK CONSTRAINT [CHK_Modules_Date_Order]

ALTER TABLE [dbo].[Modules]  WITH CHECK ADD  CONSTRAINT [CHK_Modules_Title_Length] CHECK  ((len([Title])>
(5)))

ALTER TABLE [dbo].[Modules] CHECK CONSTRAINT [CHK_Modules_Title_Length]

ALTER TABLE [dbo].[Modules]  WITH CHECK ADD  CONSTRAINT [CHK_Modules_Type_Values] CHECK  (([Type]='online'
OR [Type]='hybrid' OR [Type]='stationary'))

ALTER TABLE [dbo].[Modules] CHECK CONSTRAINT [CHK_Modules_Type_Values]
```

14. Meetings:

Tabela zawiera dane o spotkaniach odbywających się w ramach konkretnego modułu, posiada klucz główny (MeetingID), przypisuje spotkanie do modułu, zawiera datę odbycia się i język prowadzenia oraz typ (ModuleID, Date, LanguageID, Type), miejsce odbywania się modułu, link do ewentualnego spotlania online, nauczyciela prowadzącego i tłumacza (Place, Link, TeacherID, TranslatorID).

```
CREATE TABLE [dbo].[Meetings](
    [MeetingID] [int] IDENTITY(1,1) NOT NULL,
    [ModuleID] [int] NOT NULL,
    [LanguageID] [int] NOT NULL,
    [Date] [date] NOT NULL,
    [Type] [nchar](10) NOT NULL,
    [Place] [nchar](10) NULL,
    [Link] [nchar](30) NULL,
    [TeacherID] [int] NOT NULL,
    [TranslatorID] [int] NULL,
 CONSTRAINT [PK_Meetings] 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].[Meetings]  WITH CHECK ADD  CONSTRAINT [FK_Meetings_Modules] FOREIGN KEY([ModuleID])
REFERENCES [dbo].[Modules] ([ModuleID])
```

```
ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [FK_Meetings_Modules]

ALTER TABLE [dbo].[Meetings]  WITH CHECK ADD  CONSTRAINT [FK_Meetings_TeachingStaff] FOREIGN
KEY([TeacherID])
REFERENCES [dbo].[TeachingStaff] ([TeacherID])

ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [FK_Meetings_TeachingStaff]

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

ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [FK_Meetings_Translators]

ALTER TABLE [dbo].[Meetings]  WITH CHECK ADD  CONSTRAINT [CHK_Meetings_Type_Values] CHECK  (([Type]='online'
OR [Type]='hybrid' OR [Type]='stationary'))

ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [CHK_Meetings_Type_Values]
```

15. CourseAttendace:

Tabela posiada informacje o obecności studentów na spotkaniach w donym module kursu, posiada klucz główny (AttendanceID), dla każdego studenta przypisuje czy był obecny na danym spotkaniu, na które jest zapisany (MeetingID, StudentID, Attendance).

```
CREATE TABLE [dbo].[CourseAttendance](
    [AttendanceID] [int] NOT NULL,
    [MeetingID] [int] NOT NULL,
    [StudentID] [int] NOT NULL,
    [Attendance] [bit] NOT NULL,
 CONSTRAINT [PK_Attendance] PRIMARY KEY CLUSTERED
(
    [AttendanceID] 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].[CourseAttendance]  WITH CHECK ADD  CONSTRAINT [FK_Attendance_Meetings] FOREIGN
KEY([MeetingID])
REFERENCES [dbo].[Meetings] ([MeetingID])

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

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

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

16. Orders:

Tabela przypisuje zamówienie do określonego studenta, posiada klucz główny (OrderID), studenta, do którego należy zamówienie, datę jego złożenia (StudentID, OrderDate).

```
CREATE TABLE [dbo].[Orders](
    [OrderID] [int] NOT NULL,
    [StudentID] [int] NOT NULL,
    [OrderDate] [datetime] NOT NULL,
 CONSTRAINT [PK_Cart] PRIMARY KEY CLUSTERED
(
    [OrderID] 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].[Orders]  WITH CHECK ADD  CONSTRAINT [FK_Orders_Students] FOREIGN KEY([StudentID])
REFERENCES [dbo].[Students] ([StudentID])

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Students]
```

17. Order_Details:

Tabela zawiera szczegółowe informacje o konkretnym zamówieniu, posiada klucz główny (OrderDetailsID), przypisuje zamówienie do złożonego zamówienia, który się w nim znajdu (OrderID, EnrollmentID), wartość produktu i zniżke(Value, Discount), zniażka jest wartoscia typu float z zakresu od 0 do 1.

```
CREATE TABLE [dbo].[Order_details](
    [OrderDetailsID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
    [OfferID] [int] NOT NULL,
    [Value] [money] NOT NULL,
    [Discount] [float] NOT NULL,
 CONSTRAINT [PK_Cart_details] PRIMARY KEY CLUSTERED
(
    [OrderDetailsID] 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].[Order_details]  WITH CHECK ADD  CONSTRAINT [FK_Cart_details_Cart] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])

ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT [FK_Cart_details_Cart]

ALTER TABLE [dbo].[Order_details]  WITH CHECK ADD  CONSTRAINT [FK_Order_details_Offers] FOREIGN
KEY([OfferID])
REFERENCES [dbo].[Offers] ([OfferID])

ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT [FK_Order_details_Offers]

ALTER TABLE [dbo].[Order_details]  WITH CHECK ADD  CONSTRAINT [CHK_OrderDetails_Discount_Range] CHECK
(([Discount]>=(0) AND [Discount]<=(1)))

ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT [CHK_OrderDetails_Discount_Range]

ALTER TABLE [dbo].[Order_details]  WITH CHECK ADD  CONSTRAINT [CHK_OrderDetails_Value_NonNegative] CHECK
(([Value]>=(0)))

ALTER TABLE [dbo].[Order_details] CHECK CONSTRAINT [CHK_OrderDetails_Value_NonNegative]
```

18. Payments:

Tabela zawiera dane o płatnościach, posiada klucz główny (PaymentID), łączy płatność z określonym zamówieniem(OrderID), zawiera datę, wartość oraz status płatności (Date, Value, IsCancelled), status jest typu bit.

```
CREATE TABLE [dbo].[Payments](
    [PaymentID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
    [Date] [datetime] NOT NULL,
    [Value] [money] NOT NULL,
    [IsCancelled] [bit] NOT NULL,
 CONSTRAINT [PK_Payments] PRIMARY KEY CLUSTERED
(
    [PaymentID] 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].[Payments]  WITH CHECK ADD  CONSTRAINT [FK_Payments_Cart] FOREIGN KEY([OrderID])
```

```
REFERENCES [dbo].[Orders] ([OrderID])

ALTER TABLE [dbo].[Payments] CHECK CONSTRAINT [FK_Payments_Cart]

ALTER TABLE [dbo].[Payments]  WITH CHECK ADD  CONSTRAINT [CHK_Payments_Value_Positive] CHECK  (([Value]>
(0)))

ALTER TABLE [dbo].[Payments] CHECK CONSTRAINT [CHK_Payments_Value_Positive]
```

19. Users:

Tabela zawiera wszystkich użytkowników z całej bazy danych, posiada klucz główny (UserID), do tego dla każdego użytkownika przypisuje login i hasło (Login, Password).

```
CREATE TABLE [dbo].[Users](
    [UserID] [int] NOT NULL,
    [Login] [nchar](20) NOT NULL,
    [Password] [nchar](20) 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],
 CONSTRAINT [UQ_Users_Login] UNIQUE NONCLUSTERED
(
    [Login] 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].[Users]  WITH CHECK ADD  CONSTRAINT [CHK_Users_Login_Length] CHECK  ((len([Login])>=(5)))

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CHK_Users_Login_Length]

ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT [CHK_Users_Password_Digit] CHECK  ((patindex('%[0-
9]%',[Password])>(0)))

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CHK_Users_Password_Digit]

ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT [CHK_Users_Password_Length] CHECK  ((len([Password])>=
(8)))

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CHK_Users_Password_Length]
```

20. Students:

Tabela posiada wszystkch zarejestrowanych studentów, zawiera klucz główny (StudentID). Przechowuje informacje o studentach takie jak: imię, nazwisko, datę urodzenia (FirstName, LastName, BirthDate), z jakiego kraju pochodzi i dane adresowe (CountryID, Country, Region, City, ZipCode, Street), numer prywatnego i domowego telefonu (Phone, HomeNumber).

```
CREATE TABLE [dbo].[Students](
    [StudentID] [int] NOT NULL,
    [FirstName] [nchar](20) NOT NULL,
    [LastName] [nchar](20) NOT NULL,
    [BirthDate] [date] NOT NULL,
    [CountryID] [int] NOT NULL,
    [Region] [nchar](20) NOT NULL,
    [City] [nchar](20) NOT NULL,
    [ZipCode] [nchar](10) NOT NULL,
    [Street] [nchar](20) NOT NULL,
    [Phone] [nchar](20) NOT NULL,
    [HomeNumber] [nchar](15) NULL,
 CONSTRAINT [PK_Students] PRIMARY KEY CLUSTERED
(
```

```
    [StudentID] 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_Countries] FOREIGN KEY([CountryID])
REFERENCES [dbo].[Countries] ([CountryID])

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [FK_Students_Countries]

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

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

ALTER TABLE [dbo].[Students]  WITH CHECK ADD  CONSTRAINT [CHK_Students_BirthDate] CHECK  (([BirthDate]
<=getdate()))

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

21. Employees:

Tabela zawiera o wszystkich pracownikach, posiada klucz główny (EmployeeID) oraz inforamcaje o pracowniku takie jak: pozycję, imię, nazwisko (PositionID, FirstName, LastName), datę zatrudnienia, pensje, email, numer telefonu oraz miasto (HireDate, Salary, Email, Phone, City), dodatkowo informację czy dany pracownik wciąż dla nas pracuje(IsActive).

```
CREATE TABLE [dbo].[Employees](
    [EmployeeID] [int] NOT NULL,
    [PositionID] [int] NOT NULL,
    [FirstName] [nchar](20) NOT NULL,
    [LastName] [nchar](20) NOT NULL,
    [HireDate] [date] NOT NULL,
    [Salary] [money] NOT NULL,
    [Email] [nchar](30) NOT NULL,
    [Phone] [nchar](15) NOT NULL,
    [City] [nchar](20) NOT NULL,
    [IsActive] [bit] NOT NULL,
 CONSTRAINT [PK_Employees] PRIMARY KEY CLUSTERED
(
    [EmployeeID] 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],
 CONSTRAINT [UQ_Employees_Email] UNIQUE NONCLUSTERED
(
    [Email] 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].[Employees]  WITH CHECK ADD  CONSTRAINT [FK_Employees_Position] FOREIGN KEY([PositionID])
REFERENCES [dbo].[Positions] ([PositionID])

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [FK_Employees_Position]

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

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

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT [CHK_Employees_Email_Format] CHECK
((charindex('@',[Email])>(0)))

ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CHK_Employees_Email_Format]

ALTER TABLE [dbo].[Employees]  WITH CHECK ADD  CONSTRAINT [CHK_Employees_Salary] CHECK  (([Salary]>(0)))
```

```
ALTER TABLE [dbo].[Employees] CHECK CONSTRAINT [CHK_Employees_Salary]
```

22. TeachingStaff:

Tabela zawiera inforamacje o kadrze nauczycielskiej, posiada klucz główny (TeacherID) oraz informajce o tym w jakim języku prowadzi zajęcia i jego stopień naukowy (LanguageID, Degree).

```
CREATE TABLE [dbo].[TeachingStaff](
    [TeacherID] [int] NOT NULL,
    [LanguageID] [int] NOT NULL,
    [Degree] [nchar](30) NOT NULL,
 CONSTRAINT [PK_TeachingStaff] 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]

ALTER TABLE [dbo].[TeachingStaff]  WITH CHECK ADD  CONSTRAINT [FK_TeachingStaff_Employees] FOREIGN
KEY([TeacherID])
REFERENCES [dbo].[Employees] ([EmployeeID])

ALTER TABLE [dbo].[TeachingStaff] CHECK CONSTRAINT [FK_TeachingStaff_Employees]

ALTER TABLE [dbo].[TeachingStaff]  WITH CHECK ADD  CONSTRAINT [CK_TeachingStaff_Degree] CHECK
(([Degree]='professor' OR [Degree]='doctor' OR [Degree]='master' OR [Degree]='bachelor' OR [Degree]='none'))

ALTER TABLE [dbo].[TeachingStaff] CHECK CONSTRAINT [CK_TeachingStaff_Degree]
```

23. Translators:

Tabela zawiera inforamacje o tłumaczach, posiada klucz główny (TranslatorID) oraz informacje o języku z którego tłumaczy (LanguageID).

```
CREATE TABLE [dbo].[Translators](
    [TranslatorID] [int] NOT NULL,
    [LanguageID] [int] NOT NULL,
 CONSTRAINT [PK_Translators] PRIMARY KEY CLUSTERED
(
    [TranslatorID] 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].[Translators]  WITH CHECK ADD  CONSTRAINT [FK_Translators_Employees] FOREIGN
KEY([TranslatorID])
REFERENCES [dbo].[Employees] ([EmployeeID])

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

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

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

24. Administrators:

Tabela zawiera inforamacja o admnistarotach zawiera klucz głowny (AdminID) oraz data otrzymania uprawnień (Add_date).

```
CREATE TABLE [dbo].[Administrators](
    [AdminID] [int] NOT NULL,
    [Add_date] [datetime] NOT NULL,
```

```
 CONSTRAINT [PK_Administrators_1] 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].[Administrators]  WITH CHECK ADD  CONSTRAINT [FK_Administrators_Employees] FOREIGN
KEY([AdminID])
REFERENCES [dbo].[Employees] ([EmployeeID])

ALTER TABLE [dbo].[Administrators] CHECK CONSTRAINT [FK_Administrators_Employees]
```

25. Countries:

Tabela zawiera informacje o krajach, posiada klucz główny (CountryID), nazwę kraju i język (CountryName, LanguageID).

```
CREATE TABLE [dbo].[Countries](
    [CountryID] [int] NOT NULL,
    [CountryName] [nchar](20) NOT NULL,
    [LanguageID] [int] NOT NULL,
 CONSTRAINT [PK_Countries2] PRIMARY KEY CLUSTERED
(
    [CountryID] 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].[Countries]  WITH CHECK ADD  CONSTRAINT [FK_Countries_Languages] FOREIGN KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])

ALTER TABLE [dbo].[Countries] CHECK CONSTRAINT [FK_Countries_Languages]

ALTER TABLE [dbo].[Countries]  WITH CHECK ADD  CONSTRAINT [CHK_Countries_CountryName_Length] CHECK
((len([CountryName])>=(3)))

ALTER TABLE [dbo].[Countries] CHECK CONSTRAINT [CHK_Countries_CountryName_Length]
```

26. Languages:

Tabela zawiera informacje o językach, posiada klucz główny (LanguageID) oraz nazwę języka (LanguageName).

```
CREATE TABLE [dbo].[Languages](
    [LanguageID] [int] NOT NULL,
    [LanguageName] [nchar](20) NOT NULL,
 CONSTRAINT [PK_Languages] PRIMARY KEY CLUSTERED
(
    [LanguageID] 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].[Languages]  WITH CHECK ADD  CONSTRAINT [CHK_Languages_LanguageName_Length] CHECK
((len([LanguageName])>=(3)))

ALTER TABLE [dbo].[Languages] CHECK CONSTRAINT [CHK_Languages_LanguageName_Length]
```

27. Position

Tabela zawiera informacje o stanowiskach, posiada klucz główny (PositionID) oraz nazwę stanowski w postaci znakowej (PositionName).

```
CREATE TABLE [dbo].[Positions](
    [PositionID] [int] NOT NULL,
```

```
    [PositionName] [nchar](15) NOT NULL,
 CONSTRAINT [PK_Position] PRIMARY KEY CLUSTERED
(
    [PositionID] 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].[Positions]  WITH CHECK ADD  CONSTRAINT [CHK_Positions_PositionName] CHECK
(([PositionName]='director' OR [PositionName]='administrator' OR [PositionName]='educator' OR
[PositionName]='menager'))

ALTER TABLE [dbo].[Positions] CHECK CONSTRAINT [CHK_Positions_PositionName]
```

Widoki:

1. AttendanceMeetingView

Widok przedstawiający obecność studentów na spotkaniach. Dla każdego kursu podaje sumę obecności, łączną liczbę spotkań oraz procentową obecność. Umożliwia analizę uczestnictwa studentów w ramach konkretnych kursów i modułów.

```
CREATE VIEW [dbo].[AttendanceMeetingView] AS
SELECT
    c.CourseID,
    c.CourseName,
    a.StudentID,
    s.FirstName,
    s.LastName,
    m.ModuleID,
    m.Title,
    SUM(CAST(a.Attendance AS INT)) AS Attendance,
    COUNT(CAST(a.Attendance AS INT) * 100) AS AllMeeting,
    AVG(CAST(a.Attendance AS INT) * 100) AS AttendancePercentage
FROM dbo.Courses AS c
    INNER JOIN dbo.Modules AS m ON m.CourseID = c.CourseID
    INNER JOIN dbo.Meetings AS me ON me.ModuleID = m.ModuleID
    INNER JOIN dbo.CourseAttendance AS a ON a.MeetingID = me.MeetingID INNER JOIN dbo.Students AS s ON
a.StudentID = s.StudentID
GROUP BY c.CourseID, c.CourseName, a.StudentID, s.FirstName, s.LastName, m.ModuleID, m.Title
```

	CourseID	CourseName	StudentID	FirstName	LastName	ModuleID	Title	Attendance	AllMeeting	Att…
1	13	Java Programming Course	20	Sakura	Tanaka	1	Module 1: Introduction to Java	1	1	100
2	13	Java Programming Course	21	Sophie	Dupont	1	Module 1: Introduction to Java	1	1	100
3	13	Java Programming Course	22	Marco	Rossi	1	Module 1: Introduction to Java	1	1	100
4	13	Java Programming Course	20	Sakura	Tanaka	2	Module 2: Data Types and Control Structures	0	1	0
5	13	Java Programming Course	21	Sophie	Dupont	2	Module 2: Data Types and Control Structures	1	1	100
6	13	Java Programming Course	22	Marco	Rossi	2	Module 2: Data Types and Control Structures	0	1	0
7	13	Java Programming Course	20	Sakura	Tanaka	3	Module 3: Object-Oriented Programming	1	1	100
8	13	Java Programming Course	21	Sophie	Dupont	3	Module 3: Object-Oriented Programming	1	1	100
9	13	Java Programming Course	22	Marco	Rossi	3	Module 3: Object-Oriented Programming	0	1	0

2. CoursesPass

Widok ten identyfikuje, czy studenci zaliczyli kurs na podstawie procentowej obecności w poszczególnych modułach. Dla każdego kursu podaje procentową obecność, łączną liczbę modułów oraz status "Pass" lub "Fail" w zależności od spełnienia warunku procentowej obecności (80% lub więcej). Umożliwia monitorowanie postępów studentów i ocenę ich osiągnięć w kontekście kursów.

```
CREATE VIEW [dbo].[CoursesPass] As
SELECT
    amv.CourseID,
    amv.CourseName,
    amv.StudentID,
    s.FirstName,
    s.LastName,
    COUNT(amv.ModuleID) * 100 / c.ModulesNo AS AttendancePercentage,
    c.ModulesNo,
    CASE WHEN ((COUNT(amv.ModuleID) * 100) / c.odulesNo) >= 80 THEN 'Pass' ELSE 'Fail' END AS Result
```

```
FROM dbo.AttendanceMeetingView AS amv
    INNER JOIN dbo.Courses AS c ON amv.CourseID = c.CourseID
    INNER JOIN dbo.Students AS s ON amv.StudentID = s.StudentID
WHERE amv.AttendancePercentage = 100
GROUP BY amv.CourseID, amv.CourseName, amv.StudentID, s.FirstName, s.LastName, c.ModulesNo
```

	CourseID	CourseName	StudentID	FirstName	LastName	AttendancePercentage	ModulesNo	Result
1	13	Java Programming Course	20	Sakura	Tanaka	50	4	Fail
2	13	Java Programming Course	21	Sophie	Dupont	75	4	Fail
3	13	Java Programming Course	22	Marco	Rossi	25	4	Fail

3. ConflictingTranslatorMeetings

Widok [ConflictingTranslatorMeetings] identyfikuje konfliktowe spotkania tłumaczy, prezentując informacje o dwóch spotkaniach o różnych identyfikatorach (ModuleID1 i ModuleID2), które mają tę samą datę (MeetingDate) oraz dotyczą tego samego tłumacza (PersonID). Dodatkowo, widok dostarcza imię (FirstName) i nazwisko (LastName) tłumacza za pomocą danych pobranych z tabeli pracowników (Employees).

```
CREATE VIEW [dbo].[ConflictingTranslatorMeetings] AS
SELECT
    M1.ModuleID AS ModuleID1,
    M2.ModuleID AS ModuleID2,
    M1.Date AS MeetingDate,
    M1.TranslatorID AS PersonID,
    T.FirstName,
    T.LastName
FROM
    Meetings M1
JOIN
    Meetings M2 ON M1.TranslatorID = M2.TranslatorID
JOIN
    Employees T ON M1.TranslatorID = T.EmployeeID
WHERE
    M1.MeetingID <> M2.MeetingID
    AND M1.Date = M2.Date
    AND M1.MeetingID < M2.MeetingID
```

	ModuleID1	ModuleID2	MeetingDate	PersonID	FirstName	LastName
1	6	7	2023-04-21	20	Olivia	Williams

4. CourseProfitView

Widok ten zawiera informacje o studentach zapisanych na kursy, prezentując identyfikator studenta (StudentID), imię (FirstName) i nazwisko (LastName) studenta, identyfikator kursu (CourseID), nazwę kursu (CourseName), opis kursu (CourseDescription), miejsce kursu (CoursePlace), oraz datę rozpoczęcia kursu (StartDate).

```
CREATE VIEW [dbo].[CourseProfitView] AS
SELECT
    c.CourseID,
    c.CourseName,
    ISNULL((
        SELECT COUNT(od.OrderDetailsID)
        FROM Order_details od
        WHERE od.OfferID = c.CourseID
    ), 0) AS Students_number,
    ISNULL((
        SELECT SUM(od.Value)
        FROM Order_details od
        WHERE od.OrderID IN (SELECT p.OrderID FROM Payments p)
        AND od.OfferID = c.CourseID
    ), 0) AS Profit,
    c.ModulesNo,
    FORMAT(C.StartDate, 'dd-MM-yyyy') AS StartDate
```

```
FROM
    Courses c
```

	CourseID	CourseName	Students_number	Profit	ModulesNo	StartDate
1	13	Java Programming Course	2	399.98	4	10-01-2023
2	14	Data Science Course	1	399.99	3	15-02-2023
3	15	Digital Marketing Course	1	249.99	5	20-03-2023
4	16	Web Development Bootcamp	1	499.99	3	25-04-2023

5. EnrolledStudentsToCourses

Widok przedstawia informacje o studentach zapisanych na kursy. Zawiera identyfikator studenta (StudentID), imię (FirstName) i nazwisko (LastName) studenta, identyfikator kursu (CourseID), nazwę kursu (CourseName), opis kursu (CourseDescription), miejsce kursu (CoursePlace), oraz datę rozpoczęcia kursu (StartDate).

```
CREATE VIEW [dbo].[EnrolledStudentsToCourses] AS
SELECT
    S.StudentID,
    S.FirstName,
    S.LastName,
    O.OfferID AS CourseID,
    O.Name AS CourseName,
    O.Description AS CourseDescription,
    O.Place AS CoursePlace,
    FORMAT(C.StartDate, 'dd-MM-yyyy') AS StartDate
FROM dbo.Students AS S
    INNER JOIN dbo.Orders AS Ord ON S.StudentID = Ord.StudentID
    INNER JOIN dbo.Order_details AS Od ON Ord.OrderID = Od.OrderID
    INNER JOIN dbo.Offers AS O ON Od.OfferID = O.OfferID
    INNER JOIN dbo.Courses AS C ON O.OfferID = C.CourseID
WHERE  (O.Type = 'Courses')
```

	StudentID	FirstName	LastName	CourseID	CourseName	CourseDescription	CoursePlace	StartDate
1	17	John	Smith	13	Java Programming Course	Comprehensive Java programming course	CityM, Boulevard 123	10-01-2023
2	20	Sakura	Tanaka	15	Digital Marketing Course	Strategies and techniques in digital marketing	CityO, Lane 789	20-03-2023
3	21	Sophie	Dupont	16	Web Development Bootcamp	Intensive web development training program	CityP, Square 012	25-04-2023
4	21	Sophie	Dupont	14	Data Science Fundamentals Course	Fundamental concepts of Data Science	CityN, Avenue 456	15-02-2023
5	33	Carlos	Fernandez	13	Java Programming Course	Comprehensive Java programming course	CityM, Boulevard 123	10-01-2023

6. EnrolledStudentsToGatherings

Widok [EnrolledStudentsToGatherings] dostarcza informacje o studentach zapisanych na spotkania. Prezentuje identyfikator studenta (StudentID), imię (FirstName) i nazwisko (LastName) studenta, identyfikator spotkania (GatheringID), nazwę spotkania (GatheringName), opis spotkania (GatheringDescription), miejsce spotkania (GatheringPlace), oraz datę spotkania (Date).

```
CREATE VIEW [dbo].[EnrolledStudentsToGatherings] AS
SELECT
    S.StudentID,
    S.FirstName,
    S.LastName,
    O.OfferID AS GatheringID,
    O.Name AS GatheringName,
    O.Description AS GatheringDescription,
    O.Place AS GatheringPlace,
    FORMAT(G.Date, 'dd-MM-yyyy') AS Date
FROM
    dbo.Students S
    INNER JOIN dbo.Orders Ord ON S.StudentID = Ord.StudentID
    INNER JOIN dbo.Order_details Od ON Ord.OrderID = Od.OrderID
    INNER JOIN dbo.Offers O ON Od.OfferID = O.OfferID
    INNER JOIN dbo.Gatherings G ON O.OfferID = G.GatheringID
WHERE
    O.Type = 'Gathering';
```

	StudentID	FirstName	LastName	GatheringID	GatheringName	GatheringDescription	GatheringPlace	Date
1	22	Marco	Rossi	19	Science Fiction Fans Meetup	Discussion and presentations on sci-fi	CityS, Lane 789	25-05-2023
2	23	Mei	Wong	17	Tech Enthusiasts Gathering	Various tech-related sessions	CityQ, Street 123	15-03-2023
3	19	James	Brown	11	English for Business Communication	Improving English communication skills	CityK, Lane 789	28-05-2023
4	22	Marco	Rossi	9	Introduction to Programming	Basic programming concepts	CityI, Street 123	18-03-2023
5	23	Mei	Wong	12	Introduction to Machine Learning	Basic concepts of machine learning	CityL, Square 012	01-07-2023
6	17	John	Smith	20	Art and Creativity Symposium	Artistic workshops and discussions	CityT, Square 012	30-06-2023
7	30	Juan	Lopez	18	Health and Wellness Expo	Wellness sessions and workshops	CityR, Avenue 456	20-04-2023
8	21	Sophie	Dupont	10	Advanced Data Structures	In-depth study of data structures	CityJ, Avenue 456	21-04-2023
9	21	Sophie	Dupont	11	English for Business Communication	Improving English communication skills	CityK, Lane 789	28-05-2023
10	29	Sophie	Müller	19	Science Fiction Fans Meetup	Discussion and presentations on sci-fi	CityS, Lane 789	25-05-2023

7. EnrolledStudentsToStudies

Widok dostarcza informacje o studentach zapisanych na studia. Prezentuje identyfikator studenta (StudentID), imię (FirstName) i nazwisko (LastName) studenta, identyfikator studiów (StudiesID), nazwę oferty studiów (OfferName), opis oferty studiów (OfferDescription), miejsce oferty studiów (OfferPlace), oraz datę rozpoczęcia studiów (StartDate).

```
CREATE VIEW [dbo].[EnrolledStudentsToStudies] AS
SELECT
    S.StudentID,
    S.FirstName,
    S.LastName,
    St.StudiesID,
    O.Name AS OfferName,
    O.Description AS OfferDescription,
    O.Place AS OfferPlace,
    MIN(G.[Date]) AS StartDate
FROM
    dbo.Students S
    LEFT JOIN dbo.Orders Ord ON S.StudentID = Ord.StudentID
    LEFT JOIN dbo.Order_details Od ON Ord.OrderID = Od.OrderID
    LEFT JOIN dbo.Offers O ON Od.OfferID = O.OfferID
    LEFT JOIN dbo.Studies St ON O.OfferID = St.StudiesID
    LEFT JOIN dbo.Semesters Sem ON St.StudiesID = Sem.StudiesID
    LEFT JOIN dbo.Gatherings G ON Sem.SemesterID = G.SemesterID
WHERE
    O.Type = 'Studies'
GROUP BY S.StudentID, S.FirstName, S.LastName, St.StudiesID, O.OfferID, O.Name, O.Description, O.Place;
```

	Studen…	FirstName	LastName	StudiesID	OfferName	OfferDescription	OfferPlace	StartDate
1	19	James …	Brown	5	Computer Science Bachelor Program	Bachelor studies in Computer Science	CityE, Boulevard 123	2023-03-15 00:00:00.000
2	20	Sakura …	Tanaka	6	Data Analytics Master Program	Master studies in Data Analytics	CityF, Square 456	NULL
3	20	Sakura …	Tanaka	7	Business Administration PhD Program	PhD studies in Business Administration	CityG, Avenue 789	NULL
4	21	Sophie …	Dupont	8	Artificial Intelligence Certificate Program	Certificate program in AI	CityH, Lane 012	NULL
5	22	Marco …	Rossi	5	Computer Science Bachelor Program	Bachelor studies in Computer Science	CityE, Boulevard 123	2023-03-15 00:00:00.000
6	28	Hiroshi …	Yamamoto	5	Computer Science Bachelor Program	Bachelor studies in Computer Science	CityE, Boulevard 123	2023-03-15 00:00:00.000

8. EnrolledStudentsToWebinars

Widok dostarcza informacje o studentach zapisanych na webinary. Prezentuje identyfikator studenta (StudentID), imię (FirstName) i nazwisko (LastName) studenta, identyfikator webinaru (WebinarID), nazwę spotkania (GatheringName), opis spotkania (GatheringDescription), miejsce spotkania (GatheringPlace), oraz datę webinaru (Date). Zastosowanie tego widoku ułatwia monitorowanie uczestnictwa studentów w webinarach, umożliwiając identyfikację zapisanych osób oraz szczegółowe informacje o danym wydarzeniu edukacyjnym.

```
CREATE VIEW [dbo].[EnrolledStudentsToWebinars] AS
SELECT
    S.StudentID,
    S.FirstName,
    S.LastName,
    O.OfferID AS WebinarID,
    O.Name AS GatheringName,
    O.Description AS GatheringDescription,
    O.Place AS GatheringPlace,
    W.[Date]
FROM
    dbo.Students S
    INNER JOIN dbo.Orders Ord ON S.StudentID = Ord.StudentID
```

```
    INNER JOIN dbo.Order_details Od ON Ord.OrderID = Od.OrderID
    INNER JOIN dbo.Offers O ON Od.OfferID = O.OfferID
    INNER JOIN dbo.Webinar W ON O.OfferID = W.WebinarID
WHERE
    O.Type = 'Webinar';
```

	StudentID	FirstName	LastName	WebinarID	GatheringName	GatheringDescription	GatheringPlace	Date
1	17	John	Smith	1	Webinar on Data Science Basics	Introduction to Data Science	CityA, Street 123	2023-01-15 00:00:00.000
2	18	Maria	Rodriguez	2	Webinar: Mastering Python	Explore Python programming	CityB, Avenue 456	2023-02-20 00:00:00.000
3	18	Maria	Rodriguez	3	Webinar: Machine Learning Fundamentals	Understanding basics of Machine Learning	CityC, Lane 789	2023-03-25 00:00:00.000
4	29	Sophie	Müller	4	Webinar: Cybersecurity Essentials	Essential tips for Cybersecurity	CityD, Square 012	2023-04-30 00:00:00.000
5	23	Mei	Wong	1	Webinar on Data Science Basics	Introduction to Data Science	CityA, Street 123	2023-01-15 00:00:00.000
6	30	Juan	Lopez	2	Webinar: Mastering Python	Explore Python programming	CityB, Avenue 456	2023-02-20 00:00:00.000

9. ListOfDebtors

Widok [ListOfDebtors] przedstawia szczegółowe informacje o osobach, które wzięły udział w różnych wydarzeniach, ale jeszcze nie uregulowały swoich płatności, pozostając w stanie zadłużenia. Zidentyfikowani dłużnicy są grupowani według identyfikatora studenta (StudentID) oraz oferty (OfferID), a informacje obejmują imię i nazwisko studenta (Student_name), identyfikator oferty (OfferID), nazwę oferty (Name) oraz kwotę zadłużenia (Debt). Widok uwzględnia różne rodzaje wydarzeń, takie jak spotkania (Gatherings), kursy (Courses), webinary (Webinar) oraz studia (Studies).

```
WITH t AS (
    SELECT
        o.OrderID,
        CASE
            WHEN EXISTS (SELECT 1 FROM Payments as p WHERE o.OrderID = p.OrderID AND p.CancelDate IS NULL)
THEN 1
            ELSE 0
        END AS OrderStatus
    FROM
        Orders as o
)

SELECT
    s.StudentID,
    TRIM(s.FirstName) + ' ' + TRIM(s.LastName) AS Student_name,
    o.OfferID,
    o.Name,
    FORMAT((d.Value*(1-d.Discount)-P.Value), '0.00') AS Debt
FROM
    Gatherings as g
INNER JOIN
    Offers as o ON g.GatheringID = o.OfferID
INNER JOIN
    Order_details as d ON d.OfferID = o.OfferID
INNER JOIN
    t ON t.OrderID = d.OrderID
INNER JOIN
    Orders as r ON r.OrderID = d.OrderID
INNER JOIN
    Students as s ON s.StudentID = r.StudentID
INNER JOIN
    Payments AS P ON r.OrderID = P.OrderID
WHERE
    t.OrderStatus = 0 AND g.Date < GETDATE()

GROUP BY
    s.StudentID, s.FirstName, s.LastName, o.OfferID, o.Name, P.Value, d.Value, d.Discount

UNION

SELECT
    s.StudentID,
    TRIM(s.FirstName) + ' ' + TRIM(s.LastName) AS Student_name,
    o.OfferID,
    o.Name,
    FORMAT((d.Value*(1-d.Discount)-P.Value), '0.00') AS Debt
```

```
FROM
    Courses as c
INNER JOIN
    Offers as o ON c.CourseID = o.OfferID
INNER JOIN
    Order_details as d ON d.OfferID = o.OfferID
INNER JOIN
    t ON t.OrderID = d.OrderID
INNER JOIN
    Orders as r ON r.OrderID = d.OrderID
INNER JOIN
    Students as s ON s.StudentID = r.StudentID
INNER JOIN
    Payments AS P ON r.OrderID = P.OrderID
WHERE
    t.OrderStatus = 0 AND c.StartDate < GETDATE()
GROUP BY
    s.StudentID, s.FirstName, s.LastName, o.OfferID, o.Name, P.Value, d.Value, d.Discount

UNION

SELECT
    s.StudentID,
    TRIM(s.FirstName) + ' ' + TRIM(s.LastName) AS Student_name,
    o.OfferID,
    o.Name,
    FORMAT((d.Value*(1-d.Discount)-P.Value), '0.00') AS Debt
FROM
    Webinar as w
INNER JOIN
    Offers as o ON w.WebinarID = o.OfferID
INNER JOIN
    Order_details as d ON d.OfferID = o.OfferID
INNER JOIN
    t ON t.OrderID = d.OrderID
INNER JOIN
    Orders as r ON r.OrderID = d.OrderID
INNER JOIN
    Students as s ON s.StudentID = r.StudentID
INNER JOIN
    Payments AS P ON r.OrderID = P.OrderID
WHERE
    t.OrderStatus = 0 AND w.Date < GETDATE()
    GROUP BY
    s.StudentID, s.FirstName, s.LastName, o.OfferID, o.Name, P.Value, d.Value, d.Discount

UNION

SELECT
    s.StudentID,
    TRIM(s.FirstName) + ' ' + TRIM(s.LastName) AS Student_name,
    o.OfferID,
    o.Name,
    FORMAT((d.Value*(1-d.Discount)-P.Value), '0.00') AS Debt
FROM
    Studies as sd
INNER JOIN
    Offers as o ON sd.StudiesID = o.OfferID
INNER JOIN
    Semesters as se ON se.StudiesID=sd.StudiesID
INNER JOIN
    Gatherings as g ON g.SemesterID = se.SemesterID
INNER JOIN
    Order_details as d ON d.OfferID = o.OfferID
INNER JOIN
    t ON t.OrderID = d.OrderID
INNER JOIN
    Orders as r ON r.OrderID = d.OrderID
INNER JOIN
```

```
    Students as s ON s.StudentID = r.StudentID
INNER JOIN
    Payments AS P ON r.OrderID = P.OrderID
GROUP BY
    s.StudentID, s.FirstName, s.LastName, o.OfferID, o.Name, P.Value, t.OrderStatus, d.Value, d.Discount
HAVING min(g.Date) < GETDATE() AND t.OrderStatus = 0;
```

	StudentID	Student_name	OfferID	Name	Debt
1	28	Hiroshi Yamamoto	5	Computer Science Bachelor Program	3955.00

10. OrdersPaymentsView

Widok [OrdersPaymentsView] dostarcza kompleksowych informacji na temat płatności związanych z zamówieniami. Prezentuje identyfikator zamówienia (OrderID), łączną wartość zamówienia (Value), opłaconą kwotę (Paid), kwotę do zapłaty (ToPay), datę anulowania płatności (CancelDate), datę zamówienia (OrderDate), identyfikator studenta (StudentID), oraz imię i nazwisko osoby składającej zamówienie (Orderer_name).

```
CREATE VIEW [dbo].[OrdersPaymentsView] AS
SELECT
    Ord.OrderID,
    FORMAT(SUM(OD.Value*(1-OD.Discount)), '0.00') AS Value,
    FORMAT(P.Value, '0.00') AS Paid,
    FORMAT(
        IIF(P.CancelDate IS NOT NULL, 0.00, SUM(OD.Value*(1-OD.Discount))-P.Value), '0.00'
    ) AS ToPay,
    P.CancelDate,
    FORMAT(OrderDate, 'dd-MM-yyyy') AS OrderDate,
    ord.StudentID,
    TRIM(FirstName) + ' ' + TRIM(LastName) AS Orderer_name
FROM
    Orders AS Ord
INNER JOIN
    Order_details AS OD ON Ord.OrderID = OD.OrderID
INNER JOIN
    Payments AS P ON Ord.OrderID = P.OrderID
INNER JOIN
    Students AS s ON s.StudentID = ord.StudentID
GROUP BY
    Ord.OrderID, P.CancelDate, P.Value, OrderDate, ord.StudentID, FirstName, LastName
```

	OrderID	Value	Paid	ToPay	CancelDate	OrderDate	StudentID	Orderer_name
1	1	350.72	350.72	0.00	NULL	01-12-2022	17	John Smith
2	2	75.98	75.98	0.00	NULL	02-12-2022	18	Maria Rodriguez
3	3	4283.99	4283.99	0.00	NULL	03-12-2022	19	James Brown
4	4	16174.99	16174.99	0.00	NULL	04-12-2022	20	Sakura Tanaka
5	5	2331.98	2331.98	0.00	NULL	05-12-2022	21	Sophie Dupont
6	6	5056.98	5056.98	0.00	NULL	06-12-2022	22	Marco Rossi
7	7	102.47	102.47	0.00	NULL	06-11-2022	23	Mei Wong
8	8	331.98	331.98	0.00	NULL	07-11-2022	21	Sophie Dupont
9	9	99.99	12.00	87.99	NULL	09-07-2022	33	Carlos Fernandez
10	9	99.99	12.00	0.00	2022-07-16	09-07-2022	33	Carlos Fernandez
11	10	4000.00	45.00	0.00	2022-11-13	07-11-2022	28	Hiroshi Yamamoto
12	11	65.58	65.58	0.00	NULL	05-12-2022	30	Juan Lopez
13	12	48.48	48.48	0.00	NULL	07-01-2022	29	Sophie Müller

11. ProfitInfo

Widok analizuje zamówienia dla ofert, grupując je według identyfikatora oferty (OfferID), typu oferty (Type), daty zamówienia (OrderDate), identyfikatora studenta (StudentID), oraz imienia i nazwiska zamawiającego (Orderer_name). Dla każdej grupy prezentuje liczbę wszystkich zamówień (AllOrders) oraz całkowity zysk (Profit). Dodatkowo pokazuje date dokonania zamówienia i dane osoby je składającej.

```
CREATE VIEW [dbo].[ProfitInfo] AS
SELECT
    Order_details.OfferID,
    Type,
    COUNT(Order_details.OrderID) AS AllOrders,
    SUM(Value) AS Profit,
    FORMAT(OrderDate, 'dd-MM-yyyy') AS OrderDate,
    Orders.StudentID,
    TRIM(FirstName) + ' ' + TRIM(LastName) AS Orderer_name
FROM dbo.Order_details
JOIN Offers ON Offers.OfferID = Order_details.OfferID
JOIN Orders ON Orders.OrderID = Order_details.OrderID
JOIN Students ON Students.StudentID = Orders.StudentID
GROUP BY Order_details.OfferID, Type, OrderDate, Orders.StudentID, FirstName, LastName
```

	OfferID	Type	AllOrders	Profit	OrderDate	StudentID	Orderer_name
1	1	Webinar	1	29.99	06-11-2022	23	Mei Wong
2	1	Webinar	1	29.99	01-12-2022	17	John Smith
3	2	Webinar	1	39.99	02-12-2022	18	Maria Rodriguez
4	2	Webinar	1	39.99	05-12-2022	30	Juan Lopez
5	3	Webinar	1	49.99	02-12-2022	18	Maria Rodriguez
6	4	Webinar	1	34.99	07-01-2022	29	Sophie Müller
7	5	Studies	1	5000.00	07-11-2022	28	Hiroshi Yamamoto
8	5	Studies	1	5000.00	03-12-2022	19	James Brown
9	5	Studies	1	5000.00	06-12-2022	22	Marco Rossi
10	6	Studies	1	7000.00	04-12-2022	20	Sakura Tanaka
11	7	Studies	1	10000.00	04-12-2022	20	Sakura Tanaka
12	8	Studies	1	2500.00	05-12-2022	21	Sophie Dupont
13	9	Gathering	1	49.99	06-12-2022	22	Marco Rossi
14	10	Gathering	1	59.99	05-12-2022	21	Sophie Dupont
15	11	Gathering	1	39.99	07-11-2022	21	Sophie Dupont
16	11	Gathering	1	39.99	03-12-2022	19	James Brown
17	12	Gathering	1	69.99	06-11-2022	23	Mei Wong
18	13	Courses	1	99.99	09-07-2022	33	Carlos Fernandez
19	13	Courses	1	299.99	01-12-2022	17	John Smith
20	14	Courses	1	399.99	07-11-2022	21	Sophie Dupont
21	15	Courses	1	249.99	04-12-2022	20	Sakura Tanaka
22	16	Courses	1	499.99	05-12-2022	21	Sophie Dupont
23	17	Gathering	1	19.99	06-11-2022	23	Mei Wong
24	18	Gathering	1	29.99	05-12-2022	30	Juan Lopez
25	19	Gathering	1	14.99	07-01-2022	29	Sophie Müller
26	19	Gathering	1	14.99	06-12-2022	22	Marco Rossi
27	20	Gathering	1	24.99	01-12-2022	17	John Smith

12. StudentPracticesCompletionStatus

Widok analizuje podsumowanie praktyk studenckich, korzystając z tymczasowej tabeli (PracticeCounts), aby określić liczbę wszystkich praktyk dla każdego studenta. Główne zapytanie prezentuje identyfikator studenta, imię, nazwisko, liczbę ukończonych praktyk, łączną liczbę praktyk, wynik (Pass/Fail) w zależności od tego, czy student ukończył wszystkie praktyki, oraz średnią frekwencję.

```
CREATE VIEW [dbo].[StudentPracticesCompletionStatus] AS
WITH PracticeCounts AS (
    SELECT
        in_t.StudentID,
        COUNT(in_t.PractiseID) AS TotalPracticesCount
    FROM
        StudentPracticesSummaryByPractiseID in_t
    GROUP BY
        in_t.StudentID
)

SELECT
    out_t.StudentID,
    out_t.FirstName,
    out_t.LastName,
    COUNT(out_t.PractiseID) AS CompletedPracticesCount,
    PracticeCounts.TotalPracticesCount,
    CASE
        WHEN COUNT(out_t.PractiseID) = PracticeCounts.TotalPracticesCount
            THEN 'Pass'
        ELSE 'Fail'
    END AS Result,
    FORMAT(SUM(CAST(out_t.Attendance AS FLOAT)) / PracticeCounts.TotalPracticesCount, '0.00') AS Attendance
FROM
    StudentPracticesSummaryByPractiseID out_t
JOIN
    PracticeCounts ON out_t.StudentID = PracticeCounts.StudentID
WHERE
    CompletedAllPractices = 'True'
GROUP BY
    out_t.StudentID, out_t.FirstName, out_t.LastName, PracticeCounts.TotalPracticesCount;
```

	StudentID	FirstName	LastName	CompletedPracticesCount	TotalPracticesCount	Result	Attendance
1	19	James	Brown	4	4	Pass	1.00
2	22	Marco	Rossi	1	4	Fail	0.25
3	28	Hiroshi	Yamamoto	3	4	Fail	0.75

13. StudentPracticesSummaryByPractiseID

Zapytanie to generuje raport na temat uczestnictwa studentów w praktykach zawodowych. Dla każdego studenta i praktyki, prezentuje identyfikator studenta, imię, nazwisko, identyfikator praktyki, informację czy student ukończył wszystkie zajęcia praktyczne ('True' lub 'False'), oraz procentowe obliczenie frekwencji studenta w praktyce.

```
CREATE VIEW [dbo].[StudentPracticesSummaryByPractiseID] AS
SELECT
    PA.StudentID,
    S.FirstName,
    S.LastName,
    PA.PractiseID,
    CASE WHEN SUM(CAST(PA.Attendance AS INT)) = COUNT(PA.Attendance)
         THEN 'True'
         ELSE 'False'
    END AS CompletedAllPractices,
    FORMAT(SUM(CAST(PA.Attendance AS FLOAT)) / COUNT(PA.Attendance), '0.00') AS Attendance
FROM
    PractiseAttendance PA
JOIN
    Students S ON PA.StudentID = S.StudentID
GROUP BY
    PA.StudentID, PA.PractiseID, S.FirstName, S.LastName;
```

	StudentID	FirstName	LastName	PractiseID	CompletedAllPractices	Attendance
1	19	James	Brown	1	True	1.00
2	22	Marco	Rossi	1	True	1.00
3	28	Hiroshi	Yamamoto	1	False	0.40
4	19	James	Brown	2	True	1.00
5	22	Marco	Rossi	2	False	0.00
6	28	Hiroshi	Yamamoto	2	True	1.00
7	19	James	Brown	3	True	1.00
8	22	Marco	Rossi	3	False	0.00
9	28	Hiroshi	Yamamoto	3	True	1.00
10	19	James	Brown	4	True	1.00
11	22	Marco	Rossi	4	False	0.00
12	28	Hiroshi	Yamamoto	4	True	1.00

14. StudentsEnrolmentInfo

Widok przedstawia informacje o zapisach studentów, uwzględniając identyfikator studenta, imię, nazwisko, liczbę unikalnych wydarzeń, do których się zapisali, oraz numer telefonu.

```
CREATE VIEW [dbo].[StudentsEnrolmentInfo] AS
SELECT
    dbo.Students.StudentID,
    dbo.Students.FirstName,
    dbo.Students.LastName,
    COUNT(DISTINCT dbo.Order_details.OfferID) AS Num_of_events,
    dbo.Students.Phone
FROM
    dbo.Users
INNER JOIN
    dbo.Students ON dbo.Users.UserID = dbo.Students.StudentID
INNER JOIN
    dbo.Orders ON dbo.Orders.StudentID = dbo.Students.StudentID
INNER JOIN
    dbo.Order_details ON dbo.Order_details.OrderID = dbo.Orders.OrderID
GROUP BY
    dbo.Students.StudentID,
    dbo.Students.FirstName,
    dbo.Students.LastName,
    dbo.Students.Phone
```

	StudentID	FirstName	LastName	Num_of_events	Phone
1	17	John	Smith	3	+1 555-123-4567
2	18	Maria	Rodriguez	2	+52 55-7890-1234
3	19	James	Brown	2	+1 416-555-7890
4	20	Sakura	Tanaka	3	+81 90-1234-5678
5	21	Sophie	Dupont	5	+33 1 23 45 67 89
6	22	Marco	Rossi	3	+39 06 1234 5678
7	23	Mei	Wong	3	+86 10 1234 5678
8	28	Hiroshi	Yamamoto	1	+81 90-9876-5432
9	29	Sophie	Müller	2	+49 89 1234 5678
10	30	Juan	Lopez	2	+34 91 987 65 43
11	33	Carlos	Fernandez	1	+34 93 987 65 43

15. StudiesProfitView

Widok prezentuje kompleksowe informacje o dochodach i zapisanych studentach dla różnych studiów. Obejmuje identyfikator studium, nazwę, liczbę studentów, całkowity dochód, ilość semestrów oraz imię i nazwisko menedżera studium.

```
SELECT
    s.StudiesID,
    s.Name,
    ISNULL((
        SELECT COUNT(od.OrderDetailsID)
        FROM Order_details od
        WHERE od.OfferID = s.StudiesID
    ), 0) AS Students_number,
    ISNULL((
        SELECT SUM(od.Value)
        FROM Order_details od
        WHERE od.OrderID IN (SELECT p.OrderID FROM Payments p)
        AND od.OfferID = s.StudiesID
    ), 0) AS Profit,
    ISNULL((
        SELECT COUNT(sem.SemesterID)
        FROM Semesters sem
        WHERE sem.StudiesID = s.StudiesID
    ), 0) AS Semesters_number,
    TRIM(e.FirstName) + ' ' + TRIM(e.LastName) AS Menager_name
FROM
    Studies s
LEFT JOIN TeachingStaff t ON t.TeacherID = s.MenagerID
LEFT JOIN Employees e ON e.EmployeeID = t.TeacherID;
```

	StudiesID	Name	Students_number	Profit	Semesters_number	Menager_name
1	5	Computer Science	3	15000.00	4	Michael Taylor
2	6	Data Analytics	1	7000.00	0	Emma Miller
3	7	Business Administration (PhD)	1	10000.00	0	Christopher Anderson
4	8	Artificial Intelligence Certificate	1	2500.00	0	Olivia Moore

16. WebinarProfitView

Widok prezentuje szczegółowe informacje o poszczególnych webinarach, uwzględniając identyfikator, nazwę, liczbę uczestników, całkowity dochód oraz datę wydarzenia. Dodatkowo, dostarcza imię i nazwisko prowadzącego.

```
CREATE VIEW [dbo].[WebinarProfitView] AS
SELECT
    w.WebinarID,
    w.WebinarName,
    ISNULL((
        SELECT COUNT(od.OrderDetailsID)
        FROM Order_details od
        WHERE od.OfferID = w.WebinarID
    ), 0) AS Students_number,
    ISNULL((
        SELECT SUM(od.Value)
        FROM Order_details od
        WHERE od.OrderID IN (SELECT p.OrderID FROM Payments p)
        AND od.OfferID = w.WebinarID
    ), 0) AS Profit,
    FORMAT(w.Date, 'dd-MM-yyyy') AS Event_date,
    TRIM(e.FirstName) + ' ' + TRIM(e.LastName) AS Teacher_name
FROM
    Webinar w
JOIN TeachingStaff t ON t.TeacherID = w.TeacherID
JOIN Employees e ON e.EmployeeID = t.TeacherID
```

	WebinarID	WebinarName	Students_number	Profit	Event_date	Teacher_name
1	1	Webinar on Data Science Basics	2	59.98	15-01-2023	Alice Johnson
2	2	Mastering Python	2	79.98	20-02-2023	Alice Johnson
3	3	Machine Learning Fundamentals	1	49.99	25-03-2023	David Brown
4	4	Cybersecurity Essentials	1	34.99	30-04-2023	Sophia Williams

Procedury:

1. AddLessonAttendance

Procedura ta pozwala na dodanie konkretnemu użytkownikowi obecności na danej lekcji, przed wykonaniem polecenia dodawania sprawdza także czy lekcja o podanym ID istnieje oraz czy uczeń o podanym ID istnieje.

```
CREATE PROCEDURE [dbo].[AddLessonAttendance]
    @LessonID INT,
    @StudentID INT,
    @IsPresent BIT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Lessons WHERE LessonID = @LessonID) AND EXISTS (SELECT 1 FROM Students WHERE
StudentID = @StudentID)
    BEGIN
        INSERT INTO LessonsAttendance(LessonID, StudentID, Attendance)
        VALUES (@LessonID, @StudentID, @IsPresent);
    END
END;
```

2. AddMeetingAttendance

Procedura ta pozwala na dodanie konkretnemu użytkownikowi obecności na danym spotkaniu, przed wykonaniem polecenia dodawania sprawdza także czy spotkanie o podanym ID istnieje oraz czy uczeń o podanym ID istnieje.

```
CREATE PROCEDURE [dbo].[AddMeetingAttendance]
    @MeetingID INT,
    @StudentID INT,
    @IsPresent BIT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Meetings WHERE MeetingID = @MeetingID) AND EXISTS (SELECT 1 FROM Students WHERE
StudentID = @StudentID)
    BEGIN
        INSERT INTO CourseAttendance (MeetingID, StudentID, Attendance)
        VALUES (@MeetingID, @StudentID, @IsPresent);
    END
END;
```

3. AddPractiseAttendance

Procedura ta pozwala na dodanie konkretnemu użytkownikowi obecności na danych praktykach, przed wykonaniem polecenia dodawania sprawdza także czy praktyki o podanym ID istnieją oraz czy uczeń o podanym ID istnieje.

```
CREATE PROCEDURE [dbo].[AddPractiseAttendance]
    @PractiseID INT,
    @StudentID INT,
    @IsPresent BIT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Practices WHERE PractiseID = @PractiseID) AND EXISTS (SELECT 1 FROM Students
WHERE StudentID = @StudentID)
    BEGIN
        INSERT INTO PractiseAttendance(PractiseID, StudentID, Attendance)
        VALUES (@PractiseID, @StudentID, @IsPresent);
```

```
    END
END;
```

4. AddNewOrder

Procedura ta umożliwa dodatnie do tabeli Orders nowego zamówienia dla studenta o podanym ID, jako datę zamówienia wstawia aktualną datę.

```
CREATE PROCEDURE [dbo].[AddNewOrder]
    @StudentID INT
AS
BEGIN
    INSERT INTO Orders (StudentID, OrderDate)
    VALUES (@StudentID, GETDATE());

END;
```

5. AddOrderDetails

Procedura ta pozwala na dodanie szczegółów do konkretnego zamówienia, przyjmuje argumenty takie jak: numer zamówienia, nummer oferty zamówionego produktu, koszt tego produktu i ewentualną zniżkę, przed dodaniem do tabeli upewnia się czy suma wartości pozostałych kupionych produktów oraz tego wstawianego nie przekracza przypadkiem kwoty która została zapłacona za zamówienia.

```
CREATE PROCEDURE [dbo].[AddOrderDetails]
    @OrderID INT,
    @OfferID INT,
    @Value MONEY,
    @Discount FLOAT
AS
BEGIN
    DECLARE @OrderTotalMoney MONEY;
    DECLARE @PaymentTotalMoney MONEY;

    IF EXISTS (SELECT 1 FROM Orders WHERE OrderID = @OrderID)
    BEGIN

        SELECT @OrderTotalMoney = SUM(Value * (1 - Discount))
        FROM Order_details
        WHERE OrderID = @OrderID;

        SET @OrderTotalMoney = @OrderTotalMoney + (@Value * (1 - @Discount));

        SELECT @PaymentTotalMoney = Value
        FROM Payments
        WHERE OrderID = @OrderID;

        IF @OrderTotalMoney <= @PaymentTotalMoney
        BEGIN
            INSERT INTO Order_details (OrderID, OfferID, Value, Discount)
            VALUES (@OrderID, @OfferID, @Value, @Discount);
        END
    END
END;
```

6. AddPayment

Procedura ta pozwala na dodanie nowego rekordu w tabeli Payments, dla konkretnego zamówieniam daty oraz kwoty oraz dla ewentualnej daty odroczenia płatności. Procedura sprawdza także czy podane ID zamówienia istnieje w tabeli z zamówieniami.

```
CREATE PROCEDURE [dbo].[AddPayment]
    @OrderID INT,
    @Date DATETIME,
```

```
    @Value MONEY,
    @CancelDate DATETIME
AS
BEGIN

    IF EXISTS (SELECT 1 FROM Orders WHERE OrderID = @OrderID)
    BEGIN

        INSERT INTO Payments (OrderID, Date, Value, CancelDate)
        VALUES (@OrderID, @Date, @Value, @CancelDate);

    END
END;
```

7. GetOrdersPaymentsByStudentID

```
CREATE PROCEDURE [dbo].[GetOrdersPaymentsByStudentID]
    @StudentID INT
AS
BEGIN
    SELECT
        Ord.OrderID,
        SUM(ROUND(OD.Value*(1-OD.Discount),2)) AS Value,
        P.Value AS Paid,
        ROUND(SUM(ROUND(ROUND(OD.Value*(1-OD.Discount),2),2))-P.Value,2) AS ToPay,
        P.CancelDate
    FROM
        Orders AS Ord
    INNER JOIN
        Order_details AS OD ON Ord.OrderID = OD.OrderID
    INNER JOIN
        Payments AS P ON Ord.OrderID = P.OrderID
    WHERE
        Ord.StudentID = @StudentID
    GROUP BY
        Ord.OrderID, P.CancelDate, P.Value;
END;
```

8. GetStudentPracticeCompletionStatus

```
CREATE PROCEDURE [dbo].[GetStudentPracticeCompletionStatus]
    @StudentID INT
AS
BEGIN
    SELECT *
    FROM StudentPracticesCompletionStatus
    WHERE StudentID = @StudentID;
END;
```

9. GetStudentPracticeSummary

```
CREATE PROCEDURE [dbo].[GetStudentPracticeSummary]
    @StudentID INT
AS
BEGIN
    SELECT
        PA.StudentID,
        S.FirstName,
        S.LastName,
        PA.PractiseID,
        CASE WHEN SUM(CAST(PA.Attendance AS INT)) = COUNT(PA.Attendance)
             THEN 'True'
```

```
            ELSE 'False'
        END AS CompletedAllPractices
    FROM
        PractiseAttendance PA
    JOIN
        Students S ON PA.StudentID = S.StudentID
    WHERE
        PA.StudentID = @StudentID
    GROUP BY
        PA.StudentID, PA.PractiseID, S.FirstName, S.LastName;
END;
```

10. MeetingsByTeacher

```
CREATE Procedure [dbo].[MeetingsByTeacher]
    @TeacherID INT
AS
Begin
select * from Meetings as m
where m.TeacherID=@TeacherID
End
```

11. OrdersByStudentID

```
CREATE PROCEDURE [dbo].[OrdersByStudentID]
    @StudentID int
AS
BEGIN
    SELECT O.OfferID, O.Name, O.Type, O.Place, OD.Value*(1-OD.Discount) AS Value, P.Value AS Paid, OD.Value-
P.Value AS ToPay, P.CancelDate FROM Orders AS Ord
    INNER JOIN Order_details AS OD ON Ord.OrderID = OD.OrderID
    INNER JOIN Offers AS O on O.OfferID = OD.OfferID
    INNER JOIN Payments AS P on Ord.OrderID =P.PaymentID
    WHERE Ord.StudentID = @StudentID
END;
```

12. UpdateLessonAttendance

Proceudra ta umożliwia zmianę statusu obecności danego ucznia na danej lekcji, przed wykonaniem polecenia sprawdza czy modyfikowany rekord obecności faktycznie istnieje.

```
CREATE PROCEDURE [dbo].[UpdateLessonAttendance]
    @LessonID INT,
    @StudentID INT,
    @NewAttendance BIT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM LessonsAttendance WHERE LessonID = @LessonID AND StudentID = @StudentID)
    BEGIN
        UPDATE LessonsAttendance
        SET Attendance = @NewAttendance
        WHERE LessonID = @LessonID AND StudentID = @StudentID;
    END
END;
```

13. UpdateMeetingAttendance

Proceudra ta umożliwia zmianę statusu obecności danego ucznia na danym spotkaniu, przed wykonaniem polecenia sprawdza czy modyfikowany rekord obecności faktycznie istnieje.

```
CREATE PROCEDURE [dbo].[UpdateMeetingAttendance]
    @MeetingID INT,
    @StudentID INT,
    @NewAttendance BIT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM CourseAttendance WHERE MeetingID = @MeetingID AND StudentID = @StudentID)
    BEGIN
        UPDATE CourseAttendance
        SET Attendance = @NewAttendance
        WHERE MeetingID = @MeetingID AND StudentID = @StudentID;
    END
END;
```

14. UpdatePractiseAttendance

Proceudra ta umożliwia zmianę statusu obecności danego ucznia na danych praktykach, przed wykonaniem polecenia sprawdza czy modyfikowany rekord obecności faktycznie istnieje.

```
CREATE PROCEDURE [dbo].[UpdatePractiseAttendance]
    @PractiseID INT,
    @StudentID INT,
    @NewAttendance BIT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM PractiseAttendance WHERE PractiseID = @PractiseID AND StudentID = @StudentID)
    BEGIN
        UPDATE PractiseAttendance
        SET Attendance = @NewAttendance
        WHERE PractiseID = @PractiseID AND StudentID = @StudentID;
    END
END;
```

14. GetProfitInTimeRange

Proceudra przedstawia dochód z poszczególnych kursów w zadanym przedziale czasowym.

```
CREATE PROCEDURE GetProfitInTimeRange
    @From DATE,
    @To DATE
AS
BEGIN
    SELECT
        c.CourseName,
        ISNULL((
            SELECT SUM(od.Value)
            FROM Order_details od
            INNER JOIN Payments p ON od.OrderID = p.OrderID
            WHERE od.OfferID = c.CourseID AND p.[Date] BETWEEN @From AND @To
        ), 0) AS Profit
    FROM Courses c;
END;
```

15. EnrolledStudentsToCoursesInTimeRange

Proceudra przedstawia studentów zapisanych na poszczególne kursy w zadanym przedziale czasowym.

```
CREATE PROCEDURE EnrolledStudentsToCoursesInTimeRange
    @From DATE,
    @To DATE
AS
BEGIN
```

```
    SELECT *
    FROM StudentCourseDetailsView
    WHERE StartDate BETWEEN @From AND @To;
END;
```

16.EnrolledStudentsToGatheringsInTimeRange

Proceudra przedstawia studentów zapisanych na poszczególne zjazdy w zadanym przedziale czasowym.

```
CREATE PROCEDURE EnrolledStudentsToGatheringsInTimeRange
    @From DATE,
    @To DATE
AS
BEGIN
    SELECT *
    FROM EnrolledStudentsToGatherings
    WHERE [Date] BETWEEN @From AND @To;
END;
```

17.EnrolledStudentsToStudiesInDateRange

Proceudra przedstawia studentów zapisanych na poszczególne studia w zadanym przedziale czasowym.

```
CREATE PROCEDURE EnrolledStudentsToStudiesInDateRange
    @From DATE,
    @To DATE
AS
BEGIN
    SELECT *
    FROM dbo.EnrolledStudentsToStudies
    WHERE StartDate BETWEEN @From AND @To;
END;
```

18.EnrolledStudentsToWebinarsInDateRange

Proceudra przedstawia studentów zapisanych na poszczególne webinary w zadanym przedziale czasowym.

```
CREATE PROCEDURE EnrolledStudentsToWebinarsInDateRange
    @From DATE,
    @To DATE
AS
BEGIN
    SELECT *
    FROM dbo.EnrolledStudentsToWebinars
    WHERE [Date] BETWEEN @From AND @To;
END;
```

19. UpdateMeetingDate

Procedura UpdateMeetingDate umożliwia aktualizację daty spotkania o określonym MeetingID na nową datę @NewMeetingDate. Natomiast procedura AddNewOffer pozwala na dodanie nowej oferty do bazy danych, sprawdzając wcześniej, czy oferta o podanej nazwie już istnieje, aby uniknąć konfliktów. Jeśli oferta istnieje, procedura zwraca błąd, w przeciwnym razie dodaje nową ofertę z określonymi parametrami, takimi jak Name, Type, Description, Place, Price i DiscountToStudents.

```
CREATE PROCEDURE [dbo].[UpdateMeetingDate]
    @MeetingID INT,
    @NewMeetingDate DATE
AS
```

```
BEGIN
    SET NOCOUNT ON;

    UPDATE Meetings
    SET Date = @NewMeetingDate
    WHERE MeetingID = @MeetingID;
END;
```

20. AddNewOffer

```
CREATE PROCEDURE [dbo].[AddNewOffer]
    @Name NVARCHAR(50),
    @Type NVARCHAR(15),
    @Description NVARCHAR(50),
    @Place NVARCHAR(20),
    @Price money,
    @DiscountToStudents DECIMAL(5, 2)
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (SELECT 1 FROM Offers WHERE Name = @Name)
    BEGIN
        THROW 50000, 'Oferta o podanej nazwie już istnieje.', 1;
        RETURN;
    END

    INSERT INTO Offers (Name, Type, Description, Place, Price, DiscountToStudents)
    VALUES (@Name, @Type, @Description, @Place, @Price, @DiscountToStudents);

END;
```

21. AddNewCourse

Procedura AddNewCourse służy do dodawania nowego kursu do bazy danych. Najpierw wywołuje procedurę AddNewOffer, aby dodać nową ofertę typu 'Courses', a następnie pobiera identyfikator nowo dodanej oferty. Następnie dodaje kurs z odpowiednimi parametrami, takimi jak TopicID, CourseName, StartDate, ModulesNo, PaymentDay, FullPrice, Deposit i Discount.

```
CREATE PROCEDURE [dbo].[AddNewCourse]
    @CourseName NVARCHAR(30),
    @TopicID Int,
    @CourseDescription NVARCHAR(50),
    @CoursePlace NVARCHAR(20),
    @Price MONEY,
    @DiscountToStudents DECIMAL(5, 2),
    @StartDate DATE,
    @EndDate DATE,
    @Deposit Money,
    @PaymentDay Date
AS
BEGIN
    SET NOCOUNT ON;

    EXEC AddNewOffer @CourseName, 'Courses', @CourseDescription, @CoursePlace, @Price, @DiscountToStudents;

    DECLARE @NewOfferID INT;
    SET @NewOfferID = (SELECT SCOPE_IDENTITY());

    INSERT INTO Courses (CourseID, TopicID, CourseName, StartDate, ModulesNo, PaymentDay, FullPrice,
Deposit, Discount)
    VALUES (@NewOfferID, @TopicID, @CourseName, @StartDate, 0, @PaymentDay, @Price, @Deposit,
@DiscountToStudents);
END;
```

22. AddNewModule

Procedura AddNewModule służy do dodawania nowego modułu związanego z określonym kursem. Przed dodaniem modułu sprawdza istnienie kursu o podanym CourseID, a następnie pobiera datę rozpoczęcia kursu i dodaje nowy moduł z odpowiednimi parametrami, takimi jak Title, Type, EndDate i Classroom.

```
CREATE PROCEDURE [dbo].[AddNewModule]
    @CourseID INT,
    @Title NVARCHAR(50),
    @Type NVARCHAR(10),
    @EndDate DATE,
    @Classroom NVARCHAR(10)
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
    BEGIN
        THROW 50000, 'Podane CourseID nie istnieje w Courses.', 1;
        RETURN;
    END

    DECLARE @StartDate DATE;
    SET @StartDate = (SELECT StartDate FROM Courses WHERE CourseID = @CourseID);

    INSERT INTO Modules (CourseID, Title, Type, StartDate, EndDate, Classroom)
    VALUES (@CourseID, @Title, @Type, @StartDate, @EndDate, @Classroom);
END;
```

23. AddNewMeeting

Ta procedura przechowuje informacje o nowym spotkaniu w bazie danych. Przyjmuje różne parametry, takie jak ModuleID, LanguageID, Date, Type, Place, Link, Title, TeacherID i TranslatorID. Przed dodaniem spotkania sprawdza istnienie modułu, nauczyciela i tłumacza (jeśli podano TranslatorID), a w przypadku braku zapisuje wartość NULL w TranslatorID.

```
CREATE PROCEDURE [dbo].[AddNewMeeting]
    @ModuleID INT,
    @LanguageID INT,
    @Date date,
    @Type NVARCHAR(10),
    @Place NVARCHAR(10),
    @Link NVARCHAR(30),
    @Title NVARCHAR(50),
    @TeacherID INT,
    @TranslatorID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
    BEGIN
        THROW 50000, 'Moduł o podanym ModuleID nie istnieje.', 1;
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM TeachingStaff WHERE TeacherID = @TeacherID)
    BEGIN
        THROW 50000, 'Nauczyciel o podanym TeacherID nie istnieje.', 1;
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM Translators WHERE TranslatorID = @TranslatorID)
    BEGIN
        SET @TranslatorID = NULL;
    END
```

```
    INSERT INTO Meetings (ModuleID, LanguageID, Date, Type, Place, Link, TeacherID, TranslatorID)
    VALUES (@ModuleID, @LanguageID, @Date, @Type, @Place, @Link, @TeacherID, @TranslatorID);

END;
```

Funkcje:

1. CourseEnrolmentsNumber

Ta funkcja zwraca ilość użytkowników aktualnie zapisanych na określony kurs o identyfikatorze (CourseID). Wykorzystuje informacje o zapisach, płatnościach i szczegółach zamówienia, filtrując rezultaty dla konkretnego kursu, a także sprawdzając, czy zamówienie nie zostało anulowane.

```
Create FUNCTION [dbo].[CourseEnrolmentsNumber](@CourseID INT)
RETURNS INT
AS
BEGIN

    DECLARE @Enrolments INT;

    select @Enrolments = count(o.StudentID) from Orders as o
    inner join Payments as p on p.OrderID=o.OrderID
    inner join Order_details as d on d.OrderID=p.OrderID
    inner join Offers as f on f.OfferID=d.OfferID
    inner join Courses as c on c.CourseID=f.OfferID
    group by c.CourseID,p.CancelDate
    having p.CancelDate is Null and c.CourseID=@CourseID

    RETURN @Enrolments;
END;
```

2. IsStudyEnrollmentPossible

Ta funkcja sprawdza, czy istnieje możliwość zapisania się na studium o określonym identyfikatorze (StudyID), porównując aktualną liczbę zapisanych studentów (wykorzystując funkcję dbo.StudyEnrollmentsNumber) do pojemności studium. Jeżeli istnieje dostępna przestrzeń, zwraca wartość 1, w przeciwnym razie 0.

```
CREATE FUNCTION [dbo].[IsStudyEnrollmentPossible] (@StudyID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @Capacity INT;

    SELECT @Capacity = s.StudentCapacity
    FROM Studies as s
    WHERE s.StudiesID = @StudyID;

    IF (dbo.StudyEnrollmentsNumber(@StudyID) < @Capacity)
        RETURN 1;
    RETURN 0;
END;
```

3. StudyEnrollmentsNumber

Ta funkcja zwraca ilość obecnie zapisanych studentów na studium o określonym identyfikatorze (StudyID). Wykorzystuje do tego liczbę zamówień (Orders), płatności (Payments), szczegóły zamówienia (Order_details), oferty (Offers) i samego studium (Studies). Funkcja uwzględnia tylko te zapisy, które nie zostały anulowane (CancelDate is Null) i dotyczą danego studium.

```
CREATE FUNCTION [dbo].[StudyEnrollmentsNumber] (@StudyID INT)
RETURNS INT
AS
BEGIN
```

```
    DECLARE @Enrollments INT;

    select @Enrollments = count(o.StudentID) from Orders as o
    inner join Payments as p on p.OrderID=o.OrderID
    inner join Order_details as d on d.OrderID=p.OrderID
    inner join Offers as f on f.OfferID=d.OfferID
    inner join Studies as s on s.StudiesID=f.OfferID
    group by p.CancelDate, s.StudiesID
    having p.CancelDate is Null and s.StudiesID=@StudyID

    RETURN @Enrollments;
END;
```

4. WebinarEnrolmentsNumber

Ta funkcja zwraca liczbę zapisanych studentów na webinar o określonym identyfikatorze (WebinarID). Wykorzystuje do tego liczbę zamówień (Orders), płatności (Payments), szczegóły zamówienia (Order_details), oferty (Offers) i samego webinaru (Webinar). Funkcja uwzględnia tylko te zapisy, które nie zostały anulowane (CancelDate is Null) i dotyczą danego webinaru.

```
CREATE FUNCTION [dbo].[WebinarEnrolmentsNumber](@WebinarID INT)
RETURNS INT
AS
BEGIN

    DECLARE @Enrolments INT;

    select @Enrolments = count(o.StudentID) from Orders as o
    inner join Payments as p on p.OrderID=o.OrderID
    inner join Order_details as d on d.OrderID=p.OrderID
    inner join Offers as f on f.OfferID=d.OfferID
    inner join Webinar as w on w.WebinarID=f.OfferID
    group by w.WebinarID,p.CancelDate
    having p.CancelDate is Null and w.WebinarID=@WebinarID

    RETURN @Enrolments;
END
```

Triggery:

1. CheckStudentCountOnStudies

Ten trigger sprawdza, czy liczba studentów zapisanych na studium przekracza maksymalną pojemność studium po dodaniu lub aktualizacji zamówienia. Jeśli liczba przekracza pojemność, wyświetla komunikat o błędzie i wykonuje rollback transakcji, uniemożliwiając przekroczenie limitu pojemności studium.

```
CREATE TRIGGER [dbo].[CheckStudentCountOnStudies]
ON [dbo].[Order_details]
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @StudiesID INT,
            @NewStudentCount INT,
            @MaxStudentCapacity INT;

    SELECT @StudiesID = o.OfferID
    FROM inserted i
    INNER JOIN Offers o ON i.OfferID = o.OfferID;

    SELECT @NewStudentCount = COUNT(*)
    FROM Order_details od
    WHERE od.OfferID = @StudiesID;

    SELECT @MaxStudentCapacity = s.StudentCapacity
    FROM Studies s
```

```
    WHERE s.StudiesID = @StudiesID;

    IF (@NewStudentCount > @MaxStudentCapacity)
    BEGIN
        RAISERROR('Liczba studentów przekracza maksymalną pojemność studium!', 16, 1);
        ROLLBACK;
    END
END;
```

2. UpdateModulesNumber

Ten trigger automatycznie aktualizuje liczbę modułów (ModulesNo) w tabeli Courses po dodaniu nowego modułu. Działa na zasadzie zliczania liczby modułów przypisanych do danego kursu i aktualizuje odpowiedni rekord w tabeli Courses.

```
CREATE TRIGGER [dbo].[UpdateModulesNumber]
ON [dbo].[Modules]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CourseID INT;

    SELECT @CourseID = CourseID
    FROM inserted;

    UPDATE Courses
    SET ModulesNo = (
        SELECT ISNULL(COUNT(ModuleID), 0)
        FROM Modules
        WHERE CourseID = @CourseID
    )
    WHERE CourseID = @CourseID;
END;

ALTER TABLE [dbo].[Modules] ENABLE TRIGGER [UpdateModulesNumber]
```

3. CheckAndUpdateModuleDates

Ten trigger sprawdza i aktualizuje daty modułów (StartDate i EndDate) w tabeli Modules po zmianie daty spotkania (MeetingDate) w tabeli Meetings. Jeśli spotkanie już się odbyło, trigger zwraca błąd i blokuje transakcję, w przeciwnym razie aktualizuje odpowiednie daty modułów.

```
CREATE TRIGGER [dbo].[CheckAndUpdateModuleDates]
ON [dbo].[Meetings]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ModuleID INT;
    DECLARE @NewMeetingDate DATE;
    DECLARE @StartDate DATE;
    DECLARE @EndDate DATE;
    DECLARE @OldMeetingDate DATE;

    SELECT
        @ModuleID = m.ModuleID,
        @NewMeetingDate = i.Date,
        @OldMeetingDate = m.Date,
        @StartDate = mo.StartDate,
        @EndDate = mo.EndDate
    FROM
        inserted i
        INNER JOIN Modules mo ON i.ModuleID = mo.ModuleID
```

```
        INNER JOIN Meetings m ON i.ModuleID = m.ModuleID;

    IF @OldMeetingDate > GETDATE()
        BEGIN
            THROW 50000, 'Nie można zmienić daty spotkania, ponieważ wydarzenie się już odbyło.', 1;
            ROLLBACK;
        END
    ELSE
        BEGIN
            IF @NewMeetingDate < @StartDate
            BEGIN
                UPDATE Modules
                SET StartDate = @NewMeetingDate
                WHERE ModuleID = @ModuleID;
            END;

            IF @NewMeetingDate > @EndDate
            BEGIN
                UPDATE Modules
                SET EndDate = @NewMeetingDate
                WHERE ModuleID = @ModuleID;
            END;
        END;
END;

ALTER TABLE [dbo].[Meetings] ENABLE TRIGGER [CheckAndUpdateModuleDates]
```

4. UpdateCourseStartDate

Trigger sprawdza i aktualizuje datę rozpoczęcia kursu (StartDate) w tabeli Courses po zmianie daty rozpoczęcia modułu (NewStartDate) w tabeli Modules. Jeśli nowa data rozpoczęcia modułu jest późniejsza niż data zakończenia kursu, trigger zwraca błąd i blokuje transakcję, w przeciwnym razie aktualizuje datę rozpoczęcia kursu.

```
CREATE TRIGGER [dbo].[UpdateCourseStartDate]
ON [dbo].[Modules]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CourseID INT;
    DECLARE @NewStartDate DATE;
    DECLARE @EndDate DATE;

    SELECT
        @CourseID = i.CourseID,
        @NewStartDate = i.StartDate,
        @EndDate = i.EndDate
    FROM
        inserted i

    IF @NewStartDate > @EndDate
    BEGIN
        THROW 50000, 'Nowa data rozpoczęcia modułu nie może być późniejsza niż data zakończenia kursu.', 1;
        ROLLBACK;
    END
    ELSE
    BEGIN
        UPDATE Courses
        SET StartDate = @NewStartDate
        WHERE CourseID = @CourseID;
    END;
END;

ALTER TABLE [dbo].[Modules] ENABLE TRIGGER [UpdateCourseStartDate]
```