

PODSTAWY BAZ DANYCH

PROJEKT

Wojciech Burzak

01.2024r.

UŻYTKOWNICY SYSTEMU

- Administrator systemowy
- Koordynator oferty
- Dyrektor
- Wykładowca
- Użytkownik zalogowany
- Użytkownik niezalogowany
- Tłumacz
- Koordynator kursów
- Koordynator webinarów
- Koordynator studiów
- Księgowość

FUNKCJE UŻYTKOWNIKÓW W SYSTEMIE

Administrator systemowy

1. Tworzy i usuwa konta użytkowników.
2. Tworzy kopie zapasowe bazy danych .
3. Przywraca usunięte dane z bazy danych
4. Aktualizuje bazę danych
5. Może skasować przechowywane nagrania.

Koordynator oferty

1. Tworzenie oferty edukacyjnej.
2. Zarządzanie ofertą edukacyjną.

Dyrektor

1. Decyduje o odroczeniu płatności dla stałych klientów.
2. Ogląda raporty finansowe.

Wykładowca

1. Prowadzenie zajęć (webinarium, kursów, studiów).
2. Dodawanie i edycja informacji dotyczących prowadzonych zajęć.
3. Sprawdzanie obecności, wyników egzaminów, wystawianie ocen uczestnikom.
4. Posiadanie dostępu do harmonogramu zajęć, dostępności sal.

Użytkownik zalogowany

1. Przeglądanie spisu dostępnych kursów, webinarów oraz studiów.
2. Przeglądanie sylabusów prowadzonych studiów.
3. Przeglądanie cen dostępnych ofert.
4. Rejestracja na kursy, webinaria, studia.
5. Dodawanie ofert do koszyka.
6. Posiadanie dostępu do harmonogramu zajęć, informacji o sali.
7. Posiadanie dostępu do informacji dotyczących zaliczenia poszczególnych etapów nauki.
8. Dokonywanie płatności za wybrane usługi, sprawdzanie statusu płatności.
9. Oglądanie nagrań i materiałów z przeprowadzonych kursów i webinarów.
10. Ocenianie i recenzowanie kursów, webinarów lub studiów oraz ich prowadzących.
11. Korzystanie z bezpłatnych webinarów.
12. Zgłaszanie problemów.
13. Możliwość edycji swoich danych.

Użytkownik niezalogowany

1. Przeglądanie spisu dostępnych kursów, webinarów oraz studiów.
2. Przeglądanie sylabusów prowadzonych studiów.
3. Sprawdzanie cen dostępnych ofert.
4. Możliwość przeglądania opinii.
5. Możliwość założenia konta.

Tłumacz

1. Tłumaczenie na żywo webinarów i kursów na język polski.
2. Posiada informacje o spotkaniach, które tłumaczy.

Koordynator studiów

1. Zarządzanie dostępnością sal.

2. Tworzenie planu zajęć.
3. Przydzielanie prowadzących oraz tłumaczy do konkretnych grup zajęciowych.

Koordynator webinarów

1. Organizacja nowych webinarów, ustalanie daty, platformy.
2. Udostępnia użytkownikom nagrania z prowadzonych zajęć na okres 30 dni.
3. Tworzenie harmonogramu prowadzonych spotkań.
4. Przydzielanie prowadzących oraz tłumaczy

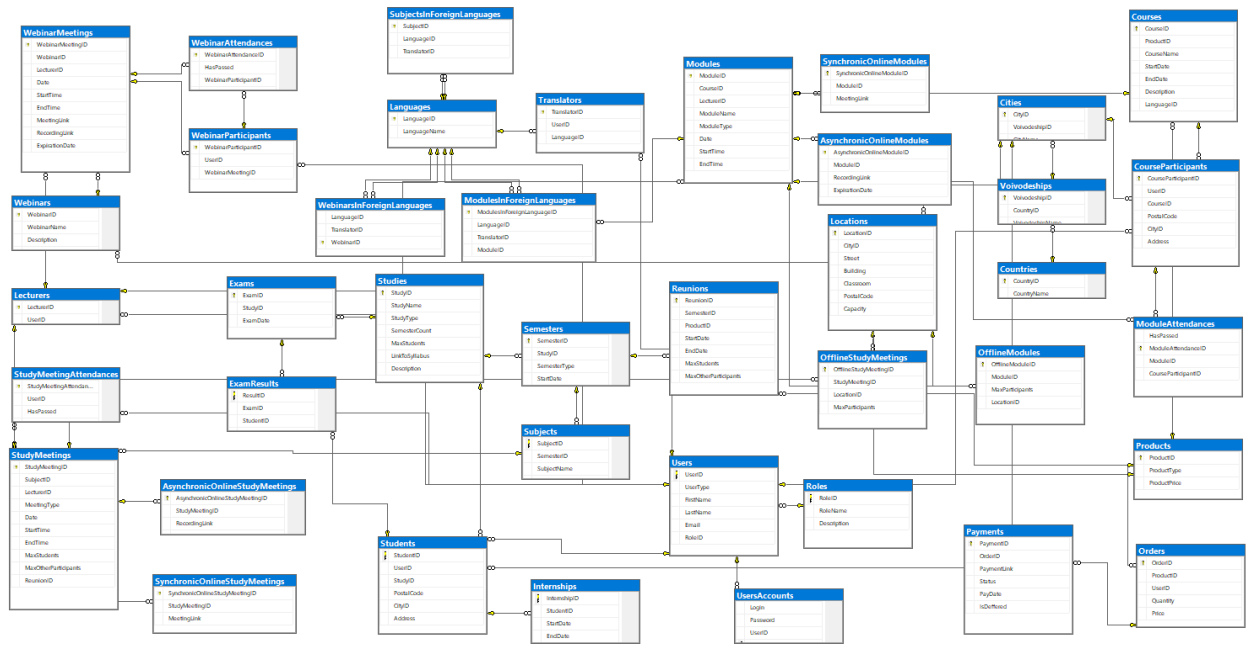
Koordynator kursów

1. Tworzenie, edycja kursów, w tym ustalanie dat, miejsc (online lub offline), wykładowców.
2. Możliwość przeglądania i zarządzania rejestracjami na kursy, listami uczestników, obsługa rezygnacji.
3. Posiadanie dostępu do listy uczestników, statystyki frekwencji, oceny kursów.
4. Komunikacja z uczestnikami kursów.

Księgowość

1. Tworzenie raportów finansowych
2. Wysyłanie upomnień klientom o zalegających płatnościach.

DIAGRAM BAZY DANYCH



OPISY TABEL

Tabela "AsynchronousOnlineModules"

Przeznaczenie: Przechowuje informacje o modułach online dostępnych asynchronicznie, wraz z linkami do nagrań i datami ich wygaśnięcia.

Opis pól:

- **AsynchronousOnlineModuleID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator modułu.
- **ModuleID (int, NULL):** Klucz obcy odnoszący się do tabeli Modules, identyfikujący moduł.
- **RecordingLink (nvarchar(255), NOT NULL):** Link do nagrania modułu.
- **ExpirationDate (datetime, NOT NULL):** Data wygaśnięcia dostępu do nagrania.

```
CREATE TABLE [dbo].[AsynchronousOnlineModules] (  
    [AsynchronousOnlineModuleID] INT IDENTITY (1, 1) NOT NULL,  
    [ModuleID] INT NULL,  
    [RecordingLink] NVARCHAR (255) NOT NULL,  
    [ExpirationDate] DATETIME NOT NULL,  
    PRIMARY KEY CLUSTERED ([AsynchronousOnlineModuleID] ASC),  
    CONSTRAINT [CHK_RecordingLink_Format] CHECK ([RecordingLink] like  
'https://%.com'),  
    FOREIGN KEY ([ModuleID]) REFERENCES [dbo].[Modules] ([ModuleID])  
) ;
```

Warunki integralnościowe:

RecordingLink musi być w formie zaczynającej się od https:// i kończącej na .com

```
ALTER TABLE AsynchronousOnlineModules
```

```
ADD CONSTRAINT CHK_RecordingLink_Format CHECK (RecordingLink LIKE 'https://%.com')
```

Tabela "AsynchronousOnlineStudyMeetings"

Przeznaczenie: Zawiera informacje o spotkaniach studyjnych online dostępnych asynchronicznie, wraz z linkami do nagrań i datami ich wygaśnięcia.

Opis pól:

- **AsynchronousOnlineStudyMeetingID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator spotkania studyjnego online.
- **StudyMeetingID (int, NULL):** Klucz obcy odnoszący się do tabeli StudyMeetings, identyfikujący spotkanie studyjne.
- **RecordingLink (nvarchar(255), NOT NULL):** Link do nagrania spotkania studyjnego.
- **ExpirationDate (datetime, NOT NULL):** Data wygaśnięcia dostępu do nagrania.

```
CREATE TABLE [dbo].[AsynchronousOnlineStudyMeetings] (  
    [AsynchronousOnlineStudyMeetingID] INT          IDENTITY (1, 1) NOT NULL,  
    [StudyMeetingID] INT          NULL,  
    [RecordingLink] NVARCHAR (255) NOT NULL,  
    [ExpirationDate] DATETIME     NOT NULL,  
    PRIMARY KEY CLUSTERED ([AsynchronousOnlineStudyMeetingID] ASC),  
    CONSTRAINT [CHK_RecordingLink_Format_StudyMeetings] CHECK ([RecordingLink] like  
'https://%.com'),  
    FOREIGN KEY ([StudyMeetingID]) REFERENCES [dbo].[StudyMeetings]  
    ([StudyMeetingID])  
);
```

Warunki integralnościowe:

RecordingLink musi być w formie zaczynającej się od https:// i kończącej na .com

```
ALTER TABLE AsynchronousOnlineStudyMeetings
```

```
ADD CONSTRAINT CHK_RecordingLink_Format_StudyMeetings CHECK (RecordingLink LIKE  
'https://%.com')
```

Tabela "Cities"

Przeznaczenie: Tabela **Cities** jest przeznaczona do przechowywania informacji o miastach.

Opis pól:

- **CityID** (int, IDENTITY(1,1), NOT NULL): Unikalny identyfikator dla każdego miasta.
- **VoivodeshipID** (int, NULL): Klucz obcy odnoszący się do tabeli **Voivodeships**, identyfikator województwa, w którym znajduje się miasto. Pole może być puste, jeśli miasto nie jest jeszcze przypisane do województwa.
- **CityName** (nvarchar(60), NOT NULL): Nazwa miasta przechowywana jako ciąg znaków o maksymalnej długości 60 znaków. Pole nie może być puste.

```
CREATE TABLE [dbo].[Cities] (  
  
    [CityID]          INT          IDENTITY (1, 1) NOT NULL,  
  
    [VoivodeshipID]   INT          NULL,  
  
    [CityName]        NVARCHAR (60) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([CityID] ASC),  
  
    FOREIGN KEY ([VoivodeshipID]) REFERENCES [dbo].[Voivodeships] ([VoivodeshipID])  
  
);
```


Tabela "Countries"

Przeznaczenie: Tabela **Countries** służy do przechowywania informacji o państwach. Każdy rekord w tej tabeli reprezentuje jedno państwo.

Opis pól:

- **CountryID** (int, IDENTITY(1,1), NOT NULL): Unikalny identyfikator dla każdego państwa.
- **CountryName** (nvarchar(30), NOT NULL): Nazwa państwa przechowywana jako ciąg znaków o maksymalnej długości 30 znaków. Pole nie może być puste.

```
CREATE TABLE [dbo].[Countries] (  
  
    [CountryID] INT IDENTITY (1, 1) NOT NULL,  
  
    [CountryName] NVARCHAR (30) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([CountryID] ASC)  
  
);
```

Tabela "CourseParticipants"

Przeznaczenie: Tabela CourseParticipants przechowuje informacje o uczestnikach kursów, łącząc ich z odpowiednimi kursami i adresami.

Opis pól:

- **CourseParticipantID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator uczestnika kursu.
- **UserID (int, NULL):** Klucz obcy odnoszący się do tabeli Users, identyfikujący użytkownika jako uczestnika kursu.
- **CourseID (int, NULL):** Klucz obcy odnoszący się do tabeli Courses, identyfikujący kurs, w którym uczestniczy osoba.
- **PostalCode (nvarchar(10), NOT NULL):** Kod pocztowy adresu uczestnika kursu.
- **CityID (int, NULL):** Klucz obcy odnoszący się do tabeli Cities, identyfikujący miasto, w którym mieszka uczestnik.
- **Address (nvarchar(30), NOT NULL):** Adres zamieszkania uczestnika kursu.

```
CREATE TABLE [dbo].[CourseParticipants] (  
  
    [CourseParticipantID] INT          IDENTITY (1, 1) NOT NULL,  
  
    [UserID]              INT          NULL,  
  
    [CourseID]            INT          NULL,  
  
    [PostalCode]          NVARCHAR (6) NOT NULL,  
  
    [CityID]              INT          NULL,  
  
    [Address]             NVARCHAR (30) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([CourseParticipantID] ASC),  
  
    FOREIGN KEY ([CityID]) REFERENCES [dbo].[Cities] ([CityID]),  
  
    FOREIGN KEY ([CourseID]) REFERENCES [dbo].[Courses] ([CourseID]),  
  
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Users] ([UserID])  
  
) ;
```

Tabela "Courses"

Przeznaczenie: Tabela Courses służy do przechowywania informacji o różnych kursach oferowanych przez instytucję. Umożliwia ona zarządzanie kursami, ich harmonogramem oraz szczegółami.

Opis pól:

- **CourseID (int, IDENTITY(1,1), NOT NULL):** Klucz główny tabeli, przydzielany sekwencyjnie dla każdego kursu.
- **ProductID (int, NULL):** Klucz obcy powiązany z tabelą Products, wskazujący produkt, który odpowiada temu kursowi.
- **CourseName (nvarchar(50), NOT NULL):** Nazwa kursu, która może mieć maksymalnie 50 znaków.
- **StartDate (datetime, NOT NULL):** Data rozpoczęcia kursu.
- **EndDate (datetime, NOT NULL):** Data zakończenia kursu.
- **Description (nvarchar(max), NULL):** Opis kursu, który może zawierać maksymalną liczbę znaków.

```
CREATE TABLE [dbo].[Courses] (  
  
    [CourseID]      INT                IDENTITY (1, 1) NOT NULL,  
  
    [ProductID]     INT                NULL,  
  
    [CourseName]    NVARCHAR (50)     NOT NULL,  
  
    [StartDate]     DATETIME           NOT NULL,  
  
    [EndDate]       DATETIME           NOT NULL,  
  
    [Description]   NVARCHAR (MAX)     NULL,  
  
    PRIMARY KEY CLUSTERED ([CourseID] ASC),  
  
    CONSTRAINT [CHK_EndDate_Greater] CHECK ([EndDate]>[StartDate]),  
  
    FOREIGN KEY ([ProductID]) REFERENCES [dbo].[Products] ([ProductID])  
  
);
```

Warunki integralnościowe:

EndDate musi być większe od StartDate

```
ALTER TABLE Courses
```

```
ADD CONSTRAINT CHK_EndDate_Greater CHECK (EndDate > StartDate)
```

Tabela "ExamResults"

Przeznaczenie: Tabela ExamResults przechowuje wyniki egzaminów studentów, pozwalając na śledzenie ich postępów i wyników.

Opis pól:

- **ResultID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator wyniku egzaminu.
- **ExamID (int, NULL):** Klucz obcy odnoszący się do tabeli Exams, identyfikujący egzamin.
- **StudentID (int, NULL):** Klucz obcy odnoszący się do tabeli Students, identyfikujący studenta, którego dotyczy wynik.
- **HasPassed (bit, NOT NULL):** Wartość wskazująca, czy student zdał egzamin (1) czy nie (0).

```
CREATE TABLE [dbo].[ExamResults] (  
  
    [ResultID] INT IDENTITY (1, 1) NOT NULL,  
  
    [ExamID] INT NULL,  
  
    [StudentID] INT NULL,  
  
    [HasPassed] BIT DEFAULT ((0)) NULL,  
  
    PRIMARY KEY CLUSTERED ([ResultID] ASC),  
  
    FOREIGN KEY ([ExamID]) REFERENCES [dbo].[Exams] ([ExamID]),  
  
    FOREIGN KEY ([StudentID]) REFERENCES [dbo].[Students] ([StudentID])  
  
);
```

Warunki integralnościowe:

HasPassed może być NULLem (w przypadku gdy nie ma wprowadzonej jeszcze daty egzaminu)

```
ALTER TABLE ExamResults  
ALTER COLUMN HasPassed BIT NULL;
```

Tabela "Exams"

Przeznaczenie: Tabela Exams przechowuje informacje o egzaminach, w tym ich daty i powiązania z określonymi studiami.

Opis pól:

- **ExamID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator egzaminu.
- **StudyID (int, NULL):** Klucz obcy odnoszący się do tabeli Studies, identyfikujący studia, w ramach których odbywa się egzamin.
- **ExamDate (datetime, NOT NULL):** Data i czas przeprowadzenia egzaminu.

```
CREATE TABLE [dbo].[Exams] (  
  
    [ExamID] INT IDENTITY (1, 1) NOT NULL,  
  
    [StudyID] INT NULL,  
  
    [ExamDate] DATETIME NULL,  
  
    PRIMARY KEY CLUSTERED ([ExamID] ASC),  
  
    FOREIGN KEY ([StudyID]) REFERENCES [dbo].[Studies] ([StudyID])  
  
);
```

Warunki integralnościowe:

ExamDate może być NULlem (egzaminy odbywają się w dalekiej przyszłości)

```
ALTER TABLE Exams
```

```
ALTER COLUMN ExamDate DATETIME NULL;
```

Tabela "Internships"

Przeznaczenie: Tabela Internships przechowuje informacje o stażach studentów, w tym ich daty rozpoczęcia i zakończenia oraz status.

Opis pól:

- **InternshipID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator stażu.
- **StudentID (int, NULL):** Klucz obcy odnoszący się do tabeli Students, identyfikujący studenta odbywającego staż.
- **StartDate (datetime, NOT NULL):** Data rozpoczęcia stażu.
- **EndDate (datetime, NOT NULL):** Data zakończenia stażu.
- **IsCompleted (bit, NOT NULL):** Wartość wskazująca, czy staż został zakończony (1) czy nie (0).

```
CREATE TABLE [dbo].[Internships] (  
  
    [InternshipID] INT          IDENTITY (1, 1) NOT NULL,  
  
    [StudentID]     INT          NULL,  
  
    [StartDate]     DATETIME NOT NULL,  
  
    [EndDate]       DATETIME NOT NULL,  
  
    [IsCompleted]   BIT          DEFAULT ((0)) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([InternshipID] ASC),  
  
    FOREIGN KEY ([StudentID]) REFERENCES [dbo].[Students] ([StudentID]);
```

Tabela "Languages"

Przeznaczenie: Tabela Languages zawiera informacje o językach, które są używane w różnych modułach i kursach oferowanych przez instytucję.

Opis pól:

- **LanguageID** (int, IDENTITY(1,1), NOT NULL): Jest to główny klucz tabeli, który jest numerem przypisanym automatycznie kolejnym językom dodawanym do bazy danych.
- **LanguageName** (nvarchar(30), NOT NULL): Nazwa języka, przechowywana jako tekst. Nazwa ta jest ograniczona do 30 znaków i nie może być pusta.

```
CREATE TABLE [dbo].[Languages] (  
  
    [LanguageID]      INT          IDENTITY (1, 1) NOT NULL,  
  
    [LanguageName]    NVARCHAR (30) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([LanguageID] ASC)  
  
);
```

Tabela "Lecturers"

Przeznaczenie: Tabela Lecturers przechowuje dane dotyczące wykładowców. Każdy rekord reprezentuje jednego wykładowcę i jest powiązany z użytkownikiem w systemie.

Opis pól:

- **LecturerID** (int, IDENTITY(1,1), NOT NULL): Główny klucz tabeli, numer przypisany każdemu wykładowcy w sposób sekwencyjny.
- **UserID** (int, NULL): Klucz obcy łączący wykładowcę z rekordem użytkownika. Może być pusty, jeśli wykładowca nie jest jeszcze powiązany z użytkownikiem.

```
CREATE TABLE [dbo].[Lecturers] (  
  
    [LecturerID] INT IDENTITY (1, 1) NOT NULL,  
  
    [UserID] INT NULL,  
  
    PRIMARY KEY CLUSTERED ([LecturerID] ASC),  
  
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Users] ([UserID])  
  
);
```


Tabela "Locations"

Przeznaczenie: Tabela Locations przechowuje informacje o lokalizacjach, w których odbywają się kursy i inne wydarzenia edukacyjne.

Opis pól:

- **LocationID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator lokalizacji.
- **CityID (int, NULL):** Klucz obcy odnoszący się do tabeli Cities, identyfikujący miasto, w którym znajduje się lokalizacja.
- **Street (nvarchar(30), NOT NULL):** Ulica, na której znajduje się lokalizacja.
- **Building (nvarchar(30), NOT NULL):** Numer budynku.
- **Classroom (int, NOT NULL):** Numer sali lub klasy.
- **PostalCode (nvarchar(10), NOT NULL):** Kod pocztowy lokalizacji.
- **Capacity (INT, NULL):** Pojemność sali wyrażona liczbą miejsc.

```
CREATE TABLE [dbo].[Locations] (  
  
    [LocationID] INT          IDENTITY (1, 1) NOT NULL,  
  
    [CityID]      INT          NULL,  
  
    [Street]      NVARCHAR (30) NOT NULL,  
  
    [Building]    INT          NULL,  
  
    [Classroom]   INT          NOT NULL,  
  
    [PostalCode]  NVARCHAR (10) NOT NULL,  
  
    [Capacity]    INT          NULL,  
  
    PRIMARY KEY CLUSTERED ([LocationID] ASC),  
  
    FOREIGN KEY ([CityID]) REFERENCES [dbo].[Cities] ([CityID])  
  
);
```

Tabela "ModuleAttendances"

Przeznaczenie: Tabela ModuleAttendances śledzi obecność i wyniki uczestników w różnych modułach kursów.

Opis pól:

- **ModuleAttendanceID** (int, IDENTITY(1,1), NOT NULL): Unikalny identyfikator obecności na module.
- **ModuleID** (int, NULL): Klucz obcy odnoszący się do tabeli Modules, identyfikujący konkretny moduł.
- **CourseParticipantID** (int, NULL): Klucz obcy odnoszący się do tabeli CourseParticipants, identyfikujący uczestnika kursu.
- **HasPassed** (bit, NOT NULL): Wartość wskazująca, czy uczestnik zdał moduł (1) czy nie (0).

```
CREATE TABLE [dbo].[ModuleAttendances] (  
  
    [ModuleID]                INT NOT NULL,  
  
    [CourseParticipantID] INT NULL,  
  
    [HasPassed]              BIT DEFAULT ((0)) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([ModuleID] ASC),  
  
    FOREIGN KEY ([CourseParticipantID]) REFERENCES [dbo].[CourseParticipants]  
        ([CourseParticipantID])  
  
);
```

Tabela "Modules"

Przeznaczenie: Tabela Modules służy do przechowywania informacji o modułach kursów oferowanych przez instytucję. Każdy rekord reprezentuje oddzielny moduł kursu, który może być dostępny online lub stacjonarnie.

Opis pól:

- **ModuleID** (int, IDENTITY(1,1), NOT NULL): To główny klucz tabeli, nadający każdemu modułowi unikalny numer identyfikacyjny, przydzielany automatycznie.
- **CourseID** (int, NULL): Klucz obcy wskazujący kurs, do którego moduł należy. Pole może być puste, jeśli moduł nie jest przypisany do żadnego kursu.
- **LecturerID** (int, NULL): Klucz obcy wskazujący wykładowcę odpowiedzialnego za moduł. Może być puste, jeśli moduł jeszcze nie ma przypisanego wykładowcy.
- **ModuleName** (nvarchar(50), NOT NULL): Nazwa modułu, przechowywana jako tekst o długości do 50 znaków.
- **ModuleType** (nvarchar(20), NOT NULL): Typ modułu, np. "online", "stacjonarny", "hybrydowy" itp.
- **Date** (datetime, NOT NULL): Data przeprowadzenia modułu.
- **StartTime** (time(7), NOT NULL): Czas rozpoczęcia modułu.
- **EndTime** (time(7), NOT NULL): Czas zakończenia modułu.

```
CREATE TABLE [dbo].[Modules] (  
  
    [ModuleID]      INT              IDENTITY (1, 1) NOT NULL,  
  
    [CourseID]      INT              NULL,  
  
    [LecturerID]    INT              NULL,  
  
    [ModuleName]    NVARCHAR (50) NOT NULL,  
  
    [ModuleType]    NVARCHAR (20) NOT NULL,  
  
    [Date]          DATETIME         NOT NULL,  
  
    [StartTime]     TIME (7)         NOT NULL,  
  
    [EndTime]       TIME (7)         NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([ModuleID] ASC),  
  
    CONSTRAINT [CHK_Modules_ModuleType] CHECK ([ModuleType]='hybrid' OR  
[ModuleType]='stationary' OR [ModuleType]='synchronic online' OR  
[ModuleType]='asynchronic online'),  
  
    CONSTRAINT [CHK_Modules_StartEndTime] CHECK ([StartTime]<[EndTime]),  
  
    FOREIGN KEY ([CourseID]) REFERENCES [dbo].[Courses] ([CourseID]),  
  
    FOREIGN KEY ([LecturerID]) REFERENCES [dbo].[Lecturers] ([LecturerID])
```

);

Warunki integralnościowe:

ModuleType ma być wartością ze zbioru 'asynchronic onlie', 'synchronic onlie', 'stationary', 'hybrid'

ALTER TABLE Modules

ADD CONSTRAINT CHK_Modules_ModuleType CHECK (ModuleType IN ('asynchronic online', 'synchronic online', 'stationary', 'hybrid'));

Czas rozpoczęcia ma być wcześniej niż czas zakończenia

ALTER TABLE Modules

ADD CONSTRAINT CHK_Modules_StartEndTime CHECK (StartTime < EndTime);

Tabela "ModulesInForeignLanguages"

Przeznaczenie: Tabela ModulesInForeignLanguages przechowuje informacje o modułach kursów dostępnych w obcych językach, wraz z tłumaczami.

Opis pól:

- **ModuleForeignLanguageID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator modułu w obcym języku.
- **ModuleID (int, NULL):** Klucz obcy odnoszący się do tabeli Modules, identyfikujący moduł.
- **LanguageID (int, NULL):** Klucz obcy odnoszący się do tabeli Languages, identyfikujący język, w którym dostępny jest moduł.
- **TranslatorID (int, NULL):** Klucz obcy odnoszący się do tabeli Translators, identyfikujący tłumacza.

```
CREATE TABLE [dbo].[ModulesInForeignLanguages] (  
  
    [ModuleID]          INT IDENTITY (1, 1) NOT NULL,  
  
    [LanguageID]       INT NULL,  
  
    [TranslatorID]     INT NULL,  
  
    PRIMARY KEY CLUSTERED ([ModuleID] ASC),  
  
    FOREIGN KEY ([LanguageID]) REFERENCES [dbo].[Languages] ([LanguageID]),  
  
    FOREIGN KEY ([TranslatorID]) REFERENCES [dbo].[Languages] ([LanguageID])  
  
);
```

Tabela "OfflineModules"

Przeznaczenie: Tabela OfflineModules przechowuje informacje o modułach kursów, które odbywają się stacjonarnie w określonych lokalizacjach.

Opis pól:

- **OfflineModuleID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator modułu offline.
- **ModuleID (int, NULL):** Klucz obcy odnoszący się do tabeli Modules, identyfikujący moduł.
- **MaxParticipants (int, NULL):** Maksymalna liczba uczestników, którzy mogą wziąć udział w module.
- **LocationID (int, NULL):** Klucz obcy odnoszący się do tabeli Locations, identyfikujący lokalizację, w której odbywa się moduł.

```
CREATE TABLE [dbo].[OfflineModules] (  
  
    [OfflineModuleID] INT IDENTITY (1, 1) NOT NULL,  
  
    [ModuleID] INT NULL,  
  
    [MaxParticipants] INT NULL,  
  
    [LocationID] INT NULL,  
  
    PRIMARY KEY CLUSTERED ([OfflineModuleID] ASC),  
  
    FOREIGN KEY ([LocationID]) REFERENCES [dbo].[Locations] ([LocationID]),  
  
    FOREIGN KEY ([ModuleID]) REFERENCES [dbo].[Modules] ([ModuleID])  
  
) ;
```

Tabela "OfflineStudyMeetings"

Przeznaczenie: Tabela OfflineStudyMeetings przechowuje informacje o spotkaniach edukacyjnych odbywających się stacjonarnie.

Opis pól:

- **OfflineStudyMeetingID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator spotkania edukacyjnego offline.
- **StudyMeetingID (int, NULL):** Klucz obcy odnoszący się do tabeli StudyMeetings, identyfikujący spotkanie.
- **LocationID (int, NULL):** Klucz obcy odnoszący się do tabeli Locations, identyfikujący lokalizację spotkania.

```
CREATE TABLE [dbo].[OfflineStudyMeetings] (  
  
    [OfflineStudyMeetingID] INT IDENTITY (1, 1) NOT NULL,  
  
    [StudyMeetingID] INT NULL,  
  
    [LocationID] INT NULL,  
  
    PRIMARY KEY CLUSTERED ([OfflineStudyMeetingID] ASC),  
  
    FOREIGN KEY ([LocationID]) REFERENCES [dbo].[Locations] ([LocationID]),  
  
    FOREIGN KEY ([StudyMeetingID]) REFERENCES [dbo].[StudyMeetings]  
    ([StudyMeetingID])  
  
);
```

Tabela "Orders"

Przeznaczenie: Tabela Orders przechowuje informacje o zamówieniach produktów i kursów przez użytkowników.

Opis pól:

- **OrderID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator zamówienia.
- **ProductID (int, NULL):** Klucz obcy odnoszący się do tabeli Products, identyfikujący zamówiony produkt.
- **UserID (int, NULL):** Klucz obcy odnoszący się do tabeli Users, identyfikujący użytkownika, który złożył zamówienie.
- **Quantity (int, NULL):** Ilość zamówionych produktów.
- **Price (money, NULL):** Całkowita cena zamówienia.

```
CREATE TABLE [dbo].[Orders] (  
  
    [OrderID]    INT        IDENTITY (1, 1) NOT NULL,  
  
    [ProductID] INT        NULL,  
  
    [UserID]     INT        NULL,  
  
    [Quantity]   INT        CONSTRAINT [DF_Orders_Quantity] DEFAULT ((1)) NULL,  
  
    [Price]      MONEY NULL,  
  
    PRIMARY KEY CLUSTERED ([OrderID] ASC),  
  
    CONSTRAINT [CHK_Orders_Quantity] CHECK ([Quantity]>(0)),  
  
    FOREIGN KEY ([ProductID]) REFERENCES [dbo].[Products] ([ProductID]),  
  
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Products] ([ProductID])  
  
);
```

Warunki integralnościowe:

Defaultowo ilość w zamówieniu jest równa 1

```
ALTER TABLE Orders
```

```
ADD CONSTRAINT DF_Orders_Quantity DEFAULT 1 FOR Quantity;
```

Żeby zamówienie miało sens ilość ma być większa niż 0

```
ALTER TABLE Orders
```

```
ADD CONSTRAINT CHK_Orders_Quantity CHECK (Quantity > 0);
```


Tabela "Payments"

Przeznaczenie: Tabela Payments śledzi płatności związane z zamówieniami, w tym ich status i datę.
Opis pól:

- **PaymentID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator płatności.
- **OrderID (int, NULL):** Klucz obcy odnoszący się do tabeli Orders, identyfikujący zamówienie, którego dotyczy płatność.
- **PaymentLink (nvarchar(255), NOT NULL):** Link do platformy płatności online.
- **Status (bit, NULL):** Status płatności (np. zapłacone, oczekujące).
- **PayDate (datetime, NOT NULL):** Data dokonania płatności.
- **IsDeferred (bit, NULL):** Wartość wskazująca, czy płatność została odroczone (1) czy nie (0).

```
CREATE TABLE [dbo].[Payments] (  
  
    [PaymentID] INT IDENTITY (1, 1) NOT NULL,  
  
    [OrderID] INT NULL,  
  
    [PaymentLink] NVARCHAR (255) NOT NULL,  
  
    [Status] BIT DEFAULT ((0)) NULL,  
  
    [PayDate] DATETIME CONSTRAINT [DF_Payments_PayDate] DEFAULT  
(getdate()) NOT NULL,  
  
    [IsDeferred] BIT DEFAULT ((0)) NULL,  
  
    PRIMARY KEY CLUSTERED ([PaymentID] ASC),  
  
    CONSTRAINT [CHK_Payments_PaymentLink] CHECK ([PaymentLink] like  
'https://%.com'),  
  
    FOREIGN KEY ([OrderID]) REFERENCES [dbo].[Orders] ([OrderID])  
  
) ;
```

Warunki integralnościowe:

PaymentLink musi być w formie zaczynającej się od https:// i kończącej na .com

ALTER TABLE Payments

ADD CONSTRAINT CHK_Payments_PaymentLink CHECK (PaymentLink LIKE 'https://%.com');

Defaultowo PayDate ustawiony na dzisiaj

ALTER TABLE Payments

ADD CONSTRAINT DF_Payments_PayDate DEFAULT GETDATE() FOR PayDate;

Tabela "Products"

Przeznaczenie: Tabela Products służy do przechowywania informacji o różnych produktach oferowanych przez instytucję, takich jak kursy, webinary, i inne materiały edukacyjne.

Opis pól:

- **ProductID (int, IDENTITY(1,1), NOT NULL):** Klucz główny tabeli, unikalny identyfikator produktu, przydzielany sekwencyjnie.
- **ProductType (nvarchar(20), NULL):** Typ produktu, określający jego kategorię, np. "kurs", "webinar", "materiał".
- **ProductPrice (money, NOT NULL):** Cena produktu. Domyślnie ustawiona na 0.

```
CREATE TABLE [dbo].[Products] (  
  
    [ProductID]      INT          IDENTITY (1, 1) NOT NULL,  
  
    [ProductType]    NVARCHAR (20) NULL,  
  
    [ProductPrice]   MONEY        DEFAULT ((0)) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([ProductID] ASC),  
  
    CONSTRAINT [CHK_Products_ProductType] CHECK ([ProductType]='reunion' OR  
[ProductType]='course' OR [ProductType]='webinar')  
  
);
```

Warunki integralnościowe:

ProductType ma się zawierać w zbiorze 'webinar', 'course', 'reunion'

```
ALTER TABLE Products  
ADD CONSTRAINT CHK_Products_ProductType CHECK (ProductType IN ('webinar',  
'course', 'reunion'));
```

Tabela "Reunions"

Przeznaczenie: Przechowuje informacje o zjazdach, w tym ich daty rozpoczęcia i zakończenia oraz maksymalną liczbę uczestników.

Opis pól:

- **ReunionID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator zjazdu.
- **SemesterID (int, NULL):** Klucz obcy odnoszący się do tabeli Semesters, identyfikujący semestr, w ramach którego odbywa się zjazd.
- **ProductID (int, NULL):** Klucz obcy odnoszący się do tabeli Products, identyfikujący produkt związany z zjazdem.
- **StartDate (datetime, NOT NULL):** Data rozpoczęcia zjazdu.
- **EndDate (datetime, NOT NULL):** Data zakończenia zjazdu.
- **MaxStudents (int, NOT NULL):** Maksymalna liczba studentów.
- **MaxOtherParticipants (int, NULL):** Maksymalna liczba innych uczestników.

```
CREATE TABLE [dbo].[Reunions] (  
  
    [ReunionID]            INT          IDENTITY (1, 1) NOT NULL,  
  
    [SemesterID]           INT          NULL,  
  
    [ProductID]            INT          NULL,  
  
    [StartDate]            DATETIME NOT NULL,  
  
    [EndDate]              DATETIME NOT NULL,  
  
    [MaxStudents]          INT          NOT NULL,  
  
    [MaxOtherParticipants] INT          NULL,  
  
    PRIMARY KEY CLUSTERED ([ReunionID] ASC),  
  
    CONSTRAINT [CHK_Reunions_EndDate] CHECK ([EndDate]>[StartDate]),  
  
    FOREIGN KEY ([ProductID]) REFERENCES [dbo].[Products] ([ProductID]),  
  
    FOREIGN KEY ([SemesterID]) REFERENCES [dbo].[Semesters] ([SemesterID])  
  
);
```

Warunki integralnościowe:

EndDate ma być większy od StartDate

```
ALTER TABLE Reunions
```

```
ADD CONSTRAINT CHK_Reunions_EndDate CHECK (EndDate > StartDate);
```

Tabela "Roles"

Przeznaczenie: Zawiera informacje o rolach użytkowników w systemie, wraz z ich opisami.

Opis pól:

- **RoleID** (int, IDENTITY(1,1), NOT NULL): Unikalny identyfikator roli.
- **RoleName** (nvarchar(30), NOT NULL): Nazwa roli.
- **Description** (nvarchar(max), NULL): Opis roli.

```
CREATE TABLE [dbo].[Roles] (  
  
    [RoleID]          INT          IDENTITY (1, 1) NOT NULL,  
  
    [RoleName]        NVARCHAR (30) NOT NULL,  
  
    [Description]     NVARCHAR (MAX) NULL,  
  
    PRIMARY KEY CLUSTERED ([RoleID] ASC),  
  
    CONSTRAINT [CHK_Roles_RoleName] CHECK ([RoleName]='WebinarParticipants'  
OR [RoleName]='CourseParticipants' OR [RoleName]='Students' OR  
[RoleName]='Participant' OR [RoleName]='Accounting' OR [RoleName]='Webinar  
Coordinator' OR [RoleName]='Study Coordinator' OR [RoleName]='Course  
Coordinator' OR [RoleName]='Secretariat' OR [RoleName]='Translator' OR  
[RoleName]='Lecturer' OR [RoleName]='Director' OR [RoleName]='Offer  
Coordinator' OR [RoleName]='System Administrator')  
  
);
```

Warunki integralnościowe:

Dziedzina ról użytkowników

```
ALTER TABLE Roles  
ADD CONSTRAINT CHK_Roles_RoleName CHECK (RoleName IN ('System  
Administrator', 'Offer Coordinator', 'Director', 'Lecturer', 'Translator',  
'Secretariat', 'Course Coordinator', 'Study Coordinator', 'Webinar  
Coordinator', 'Accounting', 'Participant', 'Students', 'CourseParticipants',  
'WebinarParticipants'))
```

Tabela "Semesters"

Przeznaczenie: Tabela Semesters przechowuje informacje o semestrach w roku akademickim, w tym ich typie, dacie rozpoczęcia i zakończenia.

Opis pól:

- **SemesterID (int, IDENTITY(1,1), NOT NULL):** Klucz główny tabeli, unikalny identyfikator semestru, przydzielany sekwencyjnie.
- **StudyID (int, NULL):** Klucz obcy odnoszący się do tabeli Studies, identyfikujący studia, do których należy semestr.
- **SemesterType (nvarchar(20), NOT NULL):** Typ semestru, np. "zimowy", "letni".
- **StartDate (datetime, NOT NULL):** Data rozpoczęcia semestru.
- **EndDate (datetime, NOT NULL):** Data zakończenia semestru.

```
CREATE TABLE [dbo].[Semesters] (  
  
    [SemesterID]    INT                IDENTITY (1, 1) NOT NULL,  
  
    [StudyID]       INT                NULL,  
  
    [SemesterType]  NVARCHAR (20) NOT NULL,  
  
    [StartDate]     DATETIME           NOT NULL,  
  
    [EndDate]       DATETIME           NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([SemesterID] ASC),  
  
    CONSTRAINT [CHK_Semesters_EndDate] CHECK ([EndDate]>[StartDate]),  
  
    CONSTRAINT [CHK_Semesters_SemesterType] CHECK ([SemesterType]='winter' OR  
[SemesterType]='summer'),  
  
    FOREIGN KEY ([StudyID]) REFERENCES [dbo].[Studies] ([StudyID])  
  
);
```

Warunki integralnościowe:

Typ semestru może być tylko zimowy lub letni

```
ALTER TABLE Semesters
```

```
ADD CONSTRAINT CHK_Semesters_SemesterType CHECK (SemesterType IN ('summer',  
'winter'));
```

EndDate ma być większy od StartDate

```
ALTER TABLE Semesters
```

```
ADD CONSTRAINT CHK_Semesters_EndDate CHECK (EndDate > StartDate);
```

Tabela "Students"

Przeznaczenie: Tabela Students przechowuje informacje o studentach uczestniczących w różnych programach edukacyjnych. Każdy rekord reprezentuje jednego studenta.

Opis pól:

- **StudentID (int, IDENTITY(1,1), NOT NULL):** Klucz główny tabeli, unikalny identyfikator studenta, przydzielany sekwencyjnie.
- **UserID (int, NULL):** Klucz obcy odnoszący się do tabeli Users, identyfikujący użytkownika, który jest studentem.
- **StudyID (int, NULL):** Klucz obcy odnoszący się do tabeli Studies, identyfikujący program studiów, w którym uczestniczy student.
- **PostalCode (nvarchar(10), NOT NULL):** Kod pocztowy adresu zamieszkania studenta.
- **CityID (int, NULL):** Klucz obcy odnoszący się do tabeli Cities, identyfikujący miasto zamieszkania studenta.
- **Address (nvarchar(30), NOT NULL):** Adres zamieszkania studenta, składający się z ulicy oraz numeru mieszkania.

```
CREATE TABLE [dbo].[Students] (  
  
    [StudentID] INT IDENTITY (1, 1) NOT NULL,  
  
    [UserID] INT NULL,  
  
    [StudyID] INT NULL,  
  
    [PostalCode] NVARCHAR (6) NOT NULL,  
  
    [CityID] INT NULL,  
  
    [Address] NVARCHAR (30) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([StudentID] ASC) ,  
  
    FOREIGN KEY ([CityID]) REFERENCES [dbo].[Cities] ([CityID]) ,  
  
    FOREIGN KEY ([StudyID]) REFERENCES [dbo].[Studies] ([StudyID]) ,  
  
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Users] ([UserID])  
  
);
```

Tabela "Studies"

Przeznaczenie: Tabela Studies przechowuje informacje o różnych programach studiów oferowanych przez instytucję.

Opis pól:

- **StudyID (int, IDENTITY(1,1), NOT NULL):** Klucz główny tabeli, unikalny identyfikator programu studiów, przydzielany sekwencyjnie.
- **StudyName (nvarchar(50), NOT NULL):** Nazwa programu studiów.
- **StudyType (nvarchar(20), NOT NULL):** Typ programu studiów, np. "pełnoetatowe", "zaoczne".
- **SemesterCount (int, NOT NULL):** Liczba semestrów w programie studiów.
- **MaxStudents (int, NOT NULL):** Maksymalna liczba studentów w programie.
- **LinkToSyllabus (nvarchar(255), NOT NULL):** Link do programu nauczania.
- **Description (nvarchar(max), NULL):** Opis programu studiów.

```
CREATE TABLE [dbo].[Studies] (  
  
    [StudyID]          INT                IDENTITY (1, 1) NOT NULL,  
  
    [StudyName]        NVARCHAR (50)     NOT NULL,  
  
    [StudyType]        NVARCHAR (20)     NOT NULL,  
  
    [SemesterCount]    INT                NOT NULL,  
  
    [MaxStudents]      INT                NOT NULL,  
  
    [LinkToSyllabus]   NVARCHAR (255)    NOT NULL,  
  
    [Description]      NVARCHAR (MAX)    NULL,  
  
    PRIMARY KEY CLUSTERED ([StudyID] ASC),  
  
    CONSTRAINT [CHK_Studies_LinkToSyllabus] CHECK ([LinkToSyllabus] like  
'https://% .com'),  
  
    CONSTRAINT [CHK_Studies_SemesterCount] CHECK ([SemesterCount]>(2))  
  
);
```

Warunki integralnościowe:

Semestrów ma być co najmniej 3 (studia są kilkuletnie)

```
ALTER TABLE Studies
```

```
ADD CONSTRAINT CHK_Studies_SemesterCount CHECK (SemesterCount > 2);
```

Link do sylabusu ma zaczynać się od 'https://' i kończyć się na '.com'

```
ALTER TABLE Studies
```

```
ADD CONSTRAINT CHK_Studies_LinkToSyllabus CHECK (LinkToSyllabus LIKE  
'https://% .com');
```

Tabela "StudyMeetingAttendances"

Przeznaczenie: Tabela StudyMeetingAttendances przechowuje informacje o obecnościach studentów na spotkaniach i wykładach.

Opis pól:

- **StudyMeetingAttendanceID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator obecności.
- **UserID (int, NULL):** Klucz obcy odnoszący się do tabeli Users, identyfikujący uczestnika spotkania.
- **HasPassed (bit, NOT NULL):** Informacja, czy użytkownik zaliczył spotkanie.

```
CREATE TABLE [dbo].[StudyMeetingAttendances] (  
  
    [StudyMeetingAttendanceID] INT IDENTITY (1, 1) NOT NULL,  
  
    [UserID] INT NULL,  
  
    [HasPassed] BIT DEFAULT ((0)) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([StudyMeetingAttendanceID] ASC),  
  
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Users] ([UserID])  
  
);
```


Tabela "StudyMeetings"

Przeznaczenie: Tabela StudyMeetings przechowuje informacje o organizowanych spotkaniach, wykładach i zajęciach.

Opis pól:

- **StudyMeetingID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator spotkania.
- **SubjectID (int, NULL):** Klucz obcy odnoszący się do tabeli Subjects, identyfikujący przedmiot spotkania.
- **LecturerID (int, NULL):** Klucz obcy odnoszący się do tabeli Lecturers, identyfikujący prowadzącego spotkanie.
- **MeetingType (nvarchar(20), NOT NULL):** Typ spotkania, np. "wykład", "ćwiczenia".
- **Date (datetime, NOT NULL):** Data i godzina spotkania.
- **StartTime (time(7), NOT NULL):** Godzina rozpoczęcia spotkania.
- **EndTime (time(7), NOT NULL):** Godzina zakończenia spotkania.
- **MaxStudents (int, NOT NULL):** Maksymalna liczba studentów.
- **MaxOtherParticipants (int, NOT NULL):** Maksymalna liczba innych uczestników.

```
CREATE TABLE [dbo].[StudyMeetings] (  
  
    [StudyMeetingID]          INT          IDENTITY (1, 1) NOT NULL,  
  
    [SubjectID]               INT          NULL,  
  
    [LecturerID]              INT          NULL,  
  
    [MeetingType]             NVARCHAR (20) NOT NULL,  
  
    [Date]                    DATETIME     NOT NULL,  
  
    [StartTime]                TIME (7)     NOT NULL,  
  
    [EndTime]                  TIME (7)     NOT NULL,  
  
    [MaxStudents]              INT          NOT NULL,  
  
    [MaxOtherParticipants]    INT          NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([StudyMeetingID] ASC),  
  
    CONSTRAINT [CHK_MeetingType] CHECK ([MeetingType]='hybrid' OR  
[MeetingType]='stationary' OR [MeetingType]='synchronic online' OR  
[MeetingType]='asynchronic online'),  
  
    CONSTRAINT [CHK_StartTime_EndTime] CHECK ([StartTime]<[EndTime]),  
  
    FOREIGN KEY ([LecturerID]) REFERENCES [dbo].[Lecturers] ([LecturerID]),  
  
    FOREIGN KEY ([SubjectID]) REFERENCES [dbo].[Subjects] ([SubjectID])
```

);

Warunki integralnościowe:

Typ spotkania zawiera się w dziedzinie 'asynchronic online', 'synchronic online', 'stationary', 'hybrid'

ALTER TABLE StudyMeetings

ADD CONSTRAINT CHK_MeetingType CHECK (MeetingType IN ('asynchronic online', 'synchronic online', 'stationary', 'hybrid'))

Godzina rozpoczęcia ma być mniejsza niż godzina zakończenia

ALTER TABLE StudyMeetings

ADD CONSTRAINT CHK_StartTime_EndTime
CHECK (StartTime < EndTime)

Tabela "Subjects"

Przeznaczenie: Tabela Subjects zawiera informacje o przedmiotach nauczanych w ramach semestrów. Umożliwia zarządzanie przedmiotami i ich opisami w obrębie systemu edukacyjnego instytucji.

Opis pól:

- **SubjectID (int, IDENTITY(1,1), NOT NULL):** Główny klucz tabeli, numer przydzielany sekwencyjnie każdemu przedmiotowi.
- **SemesterID (int, NULL):** Klucz obcy odnoszący się do semestru, w ramach którego przedmiot jest nauczany. Pole może być puste, jeśli przedmiot nie jest przypisany do żadnego semestru.
- **SubjectName (nvarchar(30), NOT NULL):** Nazwa przedmiotu, może zawierać do 30 znaków.
- **Description (nvarchar(max), NULL):** Opis przedmiotu, może zawierać maksymalną możliwą liczbę znaków.

```
CREATE TABLE [dbo].[Subjects] (  
  
    [SubjectID]    INT                IDENTITY (1, 1) NOT NULL,  
  
    [SemesterID]   INT                NULL,  
  
    [SubjectName]  NVARCHAR (30)     NOT NULL,  
  
    [Description]  NVARCHAR (MAX)    NULL,  
  
    PRIMARY KEY CLUSTERED ([SubjectID] ASC),  
  
    FOREIGN KEY ([SemesterID]) REFERENCES [dbo].[Semesters] ([SemesterID])  
  
);
```

Tabela "SubjectsInForeignLanguages"

Przeznaczenie: Przechowuje informacje o przedmiotach nauczanych w obcych językach, wraz z przypisanymi tłumaczami.

Opis pól:

- **SubjectInForeignLanguagesID** (int, IDENTITY(1,1), NOT NULL): Unikalny identyfikator wpisu, służący jako klucz główny tabeli.
- **SubjectID** (int, NULL): Klucz obcy odnoszący się do tabeli **Subjects**, identyfikujący przedmiot. Wskazuje, który przedmiot jest nauczany w języku obcym.
- **LanguageID** (int, NULL): Klucz obcy odnoszący się do tabeli Languages, identyfikujący język, w którym nauczany jest przedmiot.
- **TranslatorID** (int, NULL): Klucz obcy odnoszący się do tabeli Translators, identyfikujący tłumacza.

```
CREATE TABLE [dbo].[SubjectsInForeignLanguages] (  
  
    [SubjectID]                INT NULL,  
  
    [LanguageID]              INT NULL,  
  
    [TranslatorID]            INT NULL,  
  
    [SubjectInForeignLanguagesID] INT IDENTITY (1, 1) NOT NULL,  
  
    CONSTRAINT [PK_SubjectsInForeignLanguages] PRIMARY KEY CLUSTERED  
    ([SubjectInForeignLanguagesID] ASC) ,  
  
    FOREIGN KEY ([LanguageID]) REFERENCES [dbo].[Languages] ([LanguageID]) ,  
  
    FOREIGN KEY ([TranslatorID]) REFERENCES [dbo].[Languages] ([LanguageID]) ,  
  
    CONSTRAINT [FK_SubjectsInForeignLanguages_Subjects] FOREIGN KEY ([SubjectID])  
    REFERENCES [dbo].[Subjects] ([SubjectID])  
  
) ;
```

Tabela "SynchronicOnlineModules"

Przeznaczenie: Tabela SynchronicOnlineModules przechowuje informacje o modułach online prowadzonych w czasie rzeczywistym.

Opis pól:

- **SynchronicOnlineModuleID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator modułu online.
- **ModuleID (int, NULL):** Klucz obcy odnoszący się do tabeli Modules, identyfikujący moduł.
- **MeetingLink (nvarchar(255), NOT NULL):** Link do spotkania online.

```
CREATE TABLE [dbo].[SynchronicOnlineModules] (  
  
    [SynchronicOnlineModuleID] INT          IDENTITY (1, 1) NOT NULL,  
  
    [ModuleID]                INT           NULL,  
  
    [MeetingLink]             NVARCHAR (255) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([SynchronicOnlineModuleID] ASC),  
  
    CONSTRAINT [CHK_MeetingLink_SyncOnlineMod] CHECK ([MeetingLink] like  
'https://%.com'),  
  
    FOREIGN KEY ([ModuleID]) REFERENCES [dbo].[Modules] ([ModuleID])  
  
);
```

Warunki integralnościowe:

Link do spotkania ma zaczynać się od 'https://' i kończyć się na '.com'

```
ALTER TABLE SynchronicOnlineModules  
ADD CONSTRAINT CHK_MeetingLink_SyncOnlineMod CHECK (MeetingLink LIKE  
'https://%.com')
```

Tabela "SynchronicOnlineStudyMeetings"

Przeznaczenie: Tabela SynchronicOnlineStudyMeetings przechowuje informacje o spotkaniach studyjnych online prowadzonych w czasie rzeczywistym.

Opis pól:

- **SynchronicOnlineStudyMeetingID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator spotkania studyjnego online.
- **StudyMeetingID (int, NULL):** Klucz obcy odnoszący się do tabeli StudyMeetings, identyfikujący spotkanie studyjne.
- **MeetingLink (nvarchar(255), NOT NULL):** Link do spotkania online.

```
CREATE TABLE [dbo].[SynchronicOnlineStudyMeetings] (  
  
    [SynchronicOnlineStudyMeetingID] INT          IDENTITY (1, 1) NOT NULL,  
  
    [StudyMeetingID] INT          NULL,  
  
    [MeetingLink] NVARCHAR (255) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([SynchronicOnlineStudyMeetingID] ASC),  
  
    CONSTRAINT [CHK_MeetingLink_SyncOnlineStudy] CHECK ([MeetingLink] like  
'https://% .com'),  
  
    FOREIGN KEY ([StudyMeetingID]) REFERENCES [dbo].[StudyMeetings]  
    ([StudyMeetingID])  
  
);
```

Warunki integralnościowe:

Link do spotkania ma zaczynać się od 'https://' i kończyć się na '.com'

```
ALTER TABLE SynchronicOnlineStudyMeetings  
ADD CONSTRAINT CHK_MeetingLink_SyncOnlineStudy CHECK (MeetingLink LIKE  
'https://% .com')
```

Tabela "Translators"

Przeznaczenie: Tabela Translators przechowuje dane o tłumaczach, którzy mogą pracować nad tłumaczeniem materiałów kursowych na różne języki.

Opis pól:

- **TranslatorID (int, IDENTITY(1,1), NOT NULL):** Klucz główny tabeli, unikalny identyfikator tłumacza, przydzielany sekwencyjnie.
- **UserID (int, NULL):** Klucz obcy odnoszący się do tabeli Users, identyfikujący użytkownika, który jest tłumaczem.
- **LanguageID (int, NULL):** Klucz obcy odnoszący się do tabeli Languages, identyfikujący język, w którym tłumacz specjalizuje się.

```
CREATE TABLE [dbo].[Translators] (  
  
    [TranslatorID] INT IDENTITY (1, 1) NOT NULL,  
  
    [UserID] INT NULL,  
  
    [LanguageID] INT NULL,  
  
    PRIMARY KEY CLUSTERED ([TranslatorID] ASC),  
  
    FOREIGN KEY ([LanguageID]) REFERENCES [dbo].[Languages] ([LanguageID]),  
  
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Users] ([UserID])  
  
);
```

Tabela "Users"

Przeznaczenie: Tabela Users służy do przechowywania informacji o użytkownikach systemu, takich jak studenci, wykładowcy czy administratorzy. Każdy rekord reprezentuje jedną osobę z unikalnym identyfikatorem.

Opis pól:

- **UserID (int, IDENTITY(1,1), NOT NULL):** Klucz główny tabeli, unikalny identyfikator użytkownika, przydzielany sekwencyjnie.
- **UserType (nvarchar(30), NOT NULL):** Typ użytkownika, określający rolę w systemie, np. "student", "wykładowca", "admin".
- **FirstName (nvarchar(30), NOT NULL):** Imię użytkownika.
- **LastName (nvarchar(30), NOT NULL):** Nazwisko użytkownika.
- **Email (nvarchar(100), NOT NULL):** Adres email użytkownika. Musi być unikalny w systemie.
- **RoleID (int, NULL):** Klucz obcy odnoszący się do tabeli Roles, identyfikujący rolę przypisaną użytkownikowi.

```
CREATE TABLE [dbo].[Users] (  
  
    [UserID]          INT                IDENTITY (1, 1) NOT NULL,  
  
    [UserType]        NVARCHAR (30)      NOT NULL,  
  
    [FirstName]       NVARCHAR (30)      NOT NULL,  
  
    [LastName]        NVARCHAR (30)      NOT NULL,  
  
    [Email]           NVARCHAR (100)     NOT NULL,  
  
    [RoleID]          INT                NULL,  
  
    PRIMARY KEY CLUSTERED ([UserID] ASC),  
  
    CONSTRAINT [CHK_Email_Format] CHECK ([Email] like '%@%'),  
  
    CONSTRAINT [FK_Users_Roles] FOREIGN KEY ([RoleID]) REFERENCES [dbo].[Roles]  
    ([RoleID]),  
  
    CONSTRAINT [UC_Email] UNIQUE NONCLUSTERED ([Email] ASC)  
  
);
```

Warunki integralnościowe:

W Emailu znajduje się @

```
ALTER TABLE Users
```

```
ADD CONSTRAINT CHK_Email_Format CHECK (Email LIKE '%@%')
```


Tabela "UsersAccounts"

Przeznaczenie: Tabela UsersAccounts przechowuje informacje o kontach użytkowników, w tym o ich loginach i hasłach. Umożliwia to zarządzanie dostępem do systemu.

Opis pól:

- **AccountID** (INT, IDENTITY(1,1), NOT NULL): Jest to klucz główny tabeli, będący unikalnym identyfikatorem dla każdego konta użytkownika.
- **UserID** (INT, NULL): Pole przechowuje klucz obcy, który odnosi się do tabeli Users.
- **Login** (nvarchar(100), NOT NULL): Login użytkownika, unikalny dla każdego konta.
- **Password** (nvarchar(30), NOT NULL): Hasło użytkownika. Musi mieć więcej niż 7 znaków.

```
CREATE TABLE [dbo].[UsersAccounts] (  
  
    [UserID] INT IDENTITY (1, 1) NOT NULL,  
  
    [Login] NVARCHAR (100) NOT NULL,  
  
    [Password] NVARCHAR (30) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([UserID] ASC),  
  
    CHECK (len([Password])>(7)),  
  
    CONSTRAINT [FK_UsersAccounts_Users] FOREIGN KEY ([UserID]) REFERENCES  
[dbo].[Users] ([UserID]),  
  
    CONSTRAINT [UC_Login] UNIQUE NONCLUSTERED ([Login] ASC)  
  
);
```

Tabela "Voivodeships"

Przeznaczenie: Tabela **Voivodeships** służy do przechowywania informacji o województwach.

Opis pól:

- **VoivodeshipID** (int, IDENTITY(1,1), NOT NULL): Unikalny identyfikator dla każdego województwa.
- **CountryID** (int, NULL): Klucz obcy odnoszący się do tabeli **Countries**, identyfikator państwa, w którym znajduje się województwo. Pole może być puste, jeśli województwo nie jest jeszcze przypisane do państwa.
- **VoivodeshipName** (nvarchar(30), NOT NULL): Nazwa województwa przechowywana jako ciąg znaków o maksymalnej długości 30 znaków. Pole nie może być puste.

```
CREATE TABLE [dbo].[Voivodeships] (  
  
    [VoivodeshipID]      INT                IDENTITY (1, 1) NOT NULL,  
  
    [CountryID]          INT                NULL,  
  
    [VoivodeshipName]    NVARCHAR (30) NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([VoivodeshipID] ASC),  
  
    FOREIGN KEY ([CountryID]) REFERENCES [dbo].[Countries] ([CountryID])  
  
);
```

Tabela "WebinarAttendances"

Przeznaczenie: Zawiera informacje o obecności i wynikach uczestników webinarów, w tym dane o uczestnikach, spotkaniach, a także o tym, czy uczestnik zaliczył webinar.

Opis pól:

- **WebinarAttendanceID (INT, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator obecności na webinarze. Jest to kolumna autoinkrementowana, co oznacza, że każdy nowy rekord otrzymuje wartość większą o 1 od poprzedniego.
- **HasPassed (BIT, NOT NULL):** Pole typu bitowe wskazujące, czy uczestnik zaliczył webinar. Wartości mogą być 1 (zaliczony) lub 0 (niezaliczony). Kolumna ta nie może być pusta (NOT NULL).
- **WebinarParticipantID (INT, NULL):** Klucz obcy odnoszący się do tabeli WebinarParticipants, identyfikujący uczestnika webinaru. Może być pusta, co oznacza, że rekord obecności może nie być powiązany z konkretnym uczestnikiem.
- **WebinarMeetingID (INT, NULL):** Klucz obcy odnoszący się do tabeli WebinarMeetings, identyfikujący spotkanie webinarowe. Może być pusta, co oznacza, że rekord obecności może nie być powiązany z konkretnym spotkaniem.

```
CREATE TABLE [dbo].[WebinarAttendances] (  
  
    [WebinarAttendanceID] INT IDENTITY (1, 1) NOT NULL,  
  
    [HasPassed] BIT NOT NULL,  
  
    [WebinarParticipantID] INT NULL,  
  
    [WebinarMeetingID] INT NULL,  
  
    PRIMARY KEY CLUSTERED ([WebinarAttendanceID] ASC),  
  
    CONSTRAINT [FK_WebinarMeetingID] FOREIGN KEY ([WebinarMeetingID]) REFERENCES  
[dbo].[WebinarMeetings] ([WebinarMeetingID]),  
  
    CONSTRAINT [FK_WebinarParticipantID] FOREIGN KEY ([WebinarParticipantID])  
REFERENCES [dbo].[WebinarParticipants] ([WebinarParticipantID])  
  
);
```

Tabela "WebinarMeetings"

Przeznaczenie: Zawiera informacje o spotkaniach webinarowych, w tym ich daty, czasy, linki do spotkań i nagrań oraz daty wygaśnięcia nagrań.

Opis pól:

- **WebinarMeetingID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator spotkania webinarowego.
- **WebinarID (int, NULL):** Klucz obcy odnoszący się do tabeli Webinars, identyfikujący webinar.
- **LecturerID (int, NULL):** Klucz obcy odnoszący się do tabeli Lecturers, identyfikujący wykładowcę.
- **Date (datetime, NOT NULL):** Data spotkania.
- **StartTime (time(7), NOT NULL):** Czas rozpoczęcia spotkania.
- **EndTime (time(7), NOT NULL):** Czas zakończenia spotkania.
- **MeetingLink (nvarchar(255), NOT NULL):** Link do spotkania webinarowego.
- **RecordingLink (nvarchar(255), NOT NULL):** Link do nagrania spotkania.
- **ExpirationDate (datetime, NOT NULL):** Data wygaśnięcia dostępu do nagrania.

```
CREATE TABLE [dbo].[WebinarMeetings] (  
  
    [WebinarMeetingID] INT          IDENTITY (1, 1) NOT NULL,  
  
    [WebinarID]          INT          NULL,  
  
    [LecturerID]         INT          NULL,  
  
    [Date]               DATETIME    NOT NULL,  
  
    [StartTime]          TIME (7)     NOT NULL,  
  
    [EndTime]           TIME (7)     NOT NULL,  
  
    [MeetingLink]        NVARCHAR (255) NOT NULL,  
  
    [RecordingLink]      NVARCHAR (255) NOT NULL,  
  
    [ExpirationDate]     DATETIME    NOT NULL,  
  
    PRIMARY KEY CLUSTERED ([WebinarMeetingID] ASC),  
  
    CONSTRAINT [CHK_MeetingLink_WebinarMeetings] CHECK ([MeetingLink] like  
'https://%.com'),  
  
    CONSTRAINT [CHK_RecordingLink_WebinarMeetings] CHECK ([RecordingLink] like  
'https://%.com'),  
  
    CONSTRAINT [CHK_StartTime_EndTime_WebinarMeetings] CHECK  
    ([StartTime]<[EndTime]),  
  
    FOREIGN KEY ([LecturerID]) REFERENCES [dbo].[Lecturers] ([LecturerID]),
```

```
FOREIGN KEY ([WebinarID]) REFERENCES [dbo].[Webinars] ([WebinarID])

);

Warunki integralnościowe:

Godzina rozpoczęcia Spotkania jest mniejsza niż godzina zakończenia

ALTER TABLE WebinarMeetings
ADD CONSTRAINT CHK_StartTime_EndTime_WebinarMeetings CHECK (StartTime < EndTime)

Link do spotkania zaczyna się na 'https://' oraz kończy na '.com'

ALTER TABLE WebinarMeetings
ADD CONSTRAINT CHK_MeetingLink_WebinarMeetings CHECK (MeetingLink LIKE
'https://%.com')

Link do nagrania zaczyna się na 'https://' oraz kończy na '.com'

ALTER TABLE WebinarMeetings
ADD CONSTRAINT CHK_RecordingLink_WebinarMeetings CHECK (RecordingLink LIKE
'https://%.com')
```

Tabela "WebinarParticipants"

Przeznaczenie: Przechowuje informacje o uczestnikach webinarów, łącząc ich z odpowiednimi spotkaniami webinarowymi.

Opis pól:

- **WebinarParticipantID (int, IDENTITY(1,1), NOT NULL):** Unikalny identyfikator uczestnika webinaru.
- **UserID (int, NULL):** Klucz obcy odnoszący się do tabeli Users, identyfikujący użytkownika.
- **WebinarMeetingID (int, NULL):** Klucz obcy odnoszący się do tabeli WebinarMeetings, identyfikujący spotkanie webinarowe.

```
CREATE TABLE [dbo].[WebinarParticipants] (  
  
    [WebinarParticipantID] INT IDENTITY (1, 1) NOT NULL,  
  
    [UserID] INT NULL,  
  
    [WebinarMeetingID] INT NULL,  
  
    PRIMARY KEY CLUSTERED ([WebinarParticipantID] ASC),  
  
    FOREIGN KEY ([UserID]) REFERENCES [dbo].[Users] ([UserID]),  
  
    FOREIGN KEY ([WebinarMeetingID]) REFERENCES [dbo].[WebinarMeetings]  
    ([WebinarMeetingID])  
  
);
```

Tabela "Webinars"

Przeznaczenie: Zawiera informacje o webinarach, w tym ich nazwy i opisy.

Opis pól:

- **WebinarID** (int, IDENTITY(1,1), NOT NULL): Unikalny identyfikator webinaru.
- **WebinarName** (nvarchar(50), NOT NULL): Nazwa webinaru.
- **Description** (nvarchar(max), NULL): Opis webinaru.

```
CREATE TABLE [dbo].[Webinars] (  
  
    [WebinarID] INT IDENTITY (1, 1) NOT NULL,  
  
    [WebinarName] NVARCHAR (50) NOT NULL,  
  
    [Description] NVARCHAR (MAX) NULL,  
  
    [ProductID] INT NULL,  
  
    [LanguageID] INT NULL,  
  
    PRIMARY KEY CLUSTERED ([WebinarID] ASC),  
  
    CONSTRAINT [FK_Webinars_Languages] FOREIGN KEY ([LanguageID]) REFERENCES  
[dbo].[Languages] ([LanguageID]),  
  
    CONSTRAINT [FK_Webinars_Products] FOREIGN KEY ([ProductID]) REFERENCES  
[dbo].[Products] ([ProductID])  
  
) ;
```

Tabela "WebinarsInForeignLanguages"

Przeznaczenie: Przechowuje informacje o webinarach dostępnych w obcych językach, wraz z przypisanymi tłumaczami.

Opis pól:

- **WebinarInForeignLanguagesID** (INT, IDENTITY(1,1), NOT NULL): Jest to unikalny identyfikator dla każdego wpisu w tabeli.
- **LanguageID** (int, NULL): Klucz obcy odnoszący się do tabeli Languages, identyfikujący język, w którym dostępny jest webinar.
- **TranslatorID** (int, NULL): Klucz obcy odnoszący się do tabeli Translators, identyfikujący tłumacza.
- **WebinarID** (INT, NULL): Jest to klucz obcy, który odnosi się do tabeli Webinars.

```
CREATE TABLE [dbo].[WebinarsInForeignLanguages] (  
  
    [WebinarInForeignLanguagesID] INT IDENTITY (1, 1) NOT NULL,  
  
    [WebinarID] INT NULL,  
  
    [LanguageID] INT NULL,  
  
    [TranslatorID] INT NULL,  
  
    CONSTRAINT [PK_WebinarsInForeignLanguages] PRIMARY KEY CLUSTERED  
    ([WebinarInForeignLanguagesID] ASC),  
  
    FOREIGN KEY ([LanguageID]) REFERENCES [dbo].[Languages] ([LanguageID]),  
  
    FOREIGN KEY ([TranslatorID]) REFERENCES [dbo].[Languages] ([LanguageID]),  
  
    CONSTRAINT [FK_WebinarsInForeignLanguages_Webinars] FOREIGN KEY ([WebinarID])  
    REFERENCES [dbo].[Webinars] ([WebinarID])
```

WIDOKI

1. Zestawienie przychodów dla każdego wydarzenia (webinaru/kursu/studium)
-Bianka Marciniak


```

CREATE VIEW IncomeSummary AS
SELECT
    SUM(Orders.Price) AS Income,
    Orders.ProductID,
    Products.ProductType,
    w.WebinarName AS EventName
FROM Orders
INNER JOIN Webinars AS w ON w.ProductID = Orders.ProductID
INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
INNER JOIN Products ON Products.ProductID = Orders.ProductID
WHERE Payments.Status = 1 AND Payments.IsDeferred = 0
GROUP BY Orders.ProductID, Products.ProductType, w.WebinarName

UNION

SELECT
    SUM(Orders.Price) AS Income,
    Orders.ProductID,
    Products.ProductType,
    c.CourseName AS EventName
FROM Orders
INNER JOIN Courses AS c ON c.ProductID = Orders.ProductID
INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
INNER JOIN Products ON Products.ProductID = Orders.ProductID
WHERE Payments.Status = 1 AND Payments.IsDeferred = 0
GROUP BY Orders.ProductID, Products.ProductType, c.CourseName

UNION

SELECT
    SUM(Orders.Price) AS Income,
    Orders.ProductID,
    Products.ProductType,
    s.StudyName AS EventName
FROM Orders
INNER JOIN Reunions AS r ON r.ProductID = Orders.ProductID
INNER JOIN Semesters AS ss ON ss.SemesterID = r.SemesterID
INNER JOIN Studies AS s ON s.StudyID = ss.StudyID
INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
INNER JOIN Products ON Products.ProductID = Orders.ProductID
WHERE Payments.Status = 1 AND Payments.IsDeferred = 0
GROUP BY Orders.ProductID, Products.ProductType, s.StudyName;

```

2. Zestawienie przychodów dla każdego webinaru -Bianka Marciniak

```
CREATE VIEW WebinarsIncome AS
SELECT
    SUM(Orders.Price) AS Income,
    Orders.ProductID,
    w.WebinarName AS WebinarName
FROM Orders
INNER JOIN Webinars AS w ON w.ProductID = Orders.ProductID
INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
INNER JOIN Products ON Products.ProductID = Orders.ProductID
WHERE Payments.Status = 1 AND Payments.IsDeffered = 0
GROUP BY Orders.ProductID, w.WebinarName
```

3. Zestawienie przychodów dla każdego kursu -Bianka Marciniak

```
CREATE VIEW CoursesIncome AS
SELECT
    SUM(Orders.Price) AS Income,
    Orders.ProductID,
    c.CourseName AS CourseName
FROM Orders
INNER JOIN Courses AS c ON c.ProductID = Orders.ProductID
INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
INNER JOIN Products ON Products.ProductID = Orders.ProductID
WHERE Payments.Status = 1 AND Payments.IsDeffered = 0
GROUP BY Orders.ProductID, Products.ProductType, c.CourseName
```

4. Zestawienie przychodów dla każdego zjazdu studyjnego - Bianka Marciniak

```
CREATE VIEW ReunionsIncome AS
SELECT
    Orders.ProductID,
    SUM(Orders.Price) AS Income,
    s.StudyName AS StudyName
FROM Orders
INNER JOIN Reunions AS r ON r.ProductID = Orders.ProductID
INNER JOIN Semesters AS ss ON ss.SemesterID = r.SemesterID
```

```

INNER JOIN Studies AS s ON s.StudyID = ss.StudyID
INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
INNER JOIN Products ON Products.ProductID = Orders.ProductID
WHERE Payments.Status = 1 AND Payments.IsDeferred = 0
GROUP BY Orders.ProductID, s.StudyName;

```

5. Ogólna lista „dłużników” – osoby, które skorzystały z usług, ale nie uiściły opłat. -Julia Grela

```

CREATE VIEW Debtors AS
SELECT
    Orders.UserID,
    u.FirstName,
    u.LastName,
    w.WebinarName AS EventName,
    wm.Date AS EventDate,
    p.ProductType
FROM
    Orders
    INNER JOIN Products p on p.ProductID=Orders.ProductID
    INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
    INNER JOIN Users AS u ON u.UserID = Orders.UserID
    INNER JOIN Webinars w ON w.ProductID = Orders.ProductID
    INNER JOIN WebinarMeetings wm ON wm.WebinarID = w.WebinarID
WHERE
    Payments.Status = 0 AND
    wm.Date < GETDATE()

UNION

SELECT
    Orders.UserID,
    u.FirstName,
    u.LastName,
    c.CourseName AS EventName,
    c.StartDate AS EventDate,
    p.ProductType
FROM
    Orders
    INNER JOIN Products p on p.ProductID=Orders.ProductID
    INNER JOIN Payments ON Payments.OrderID = Orders.OrderID
    INNER JOIN Users AS u ON u.UserID = Orders.UserID
    INNER JOIN Courses c ON c.ProductID = Orders.ProductID

```

WHERE

```
Payments.Status = 0 AND  
c.EndDate < GETDATE()
```

UNION

SELECT

```
Orders.UserID,  
u.FirstName,  
u.LastName,  
s.StudyName AS EventName,  
r.StartDate AS EventDate,  
p.ProductType
```

FROM

```
Orders  
INNER JOIN Products p ON p.ProductID=Orders.ProductID  
INNER JOIN Payments ON Payments.OrderID = Orders.OrderID  
INNER JOIN Users AS u ON u.UserID = Orders.UserID  
INNER JOIN Reunions r ON r.ProductID = Orders.ProductID  
INNER JOIN Semesters sr ON sr.SemesterID = r.SemesterID  
INNER JOIN Studies s ON s.StudyID = sr.StudyID
```

WHERE

```
Payments.Status = 0 AND  
r.EndDate < GETDATE();
```

6. Lista dłużników dla webinarów -Julia Grela

```
CREATE VIEW WebinarDebtors AS  
SELECT Orders.UserID, u.FirstName, u.LastName, w.WebinarName, wm.Date  
FROM Orders  
INNER JOIN Payments ON Payments.OrderID=Orders.OrderID  
INNER JOIN Users as u ON u.UserID = Orders.UserID  
INNER JOIN Webinars w ON w.ProductID = Orders.ProductID  
INNER JOIN WebinarMeetings wm ON wm.WebinarID=w.WebinarID  
WHERE Payments.Status = 0 AND wm.Date < getDate()  
GROUP BY Orders.UserID, u.FirstName, u.LastName,w.WebinarName, wm.Date;
```

7. Lista dłużników dla kursów - Julia Grela

```
CREATE VIEW CourseDebtors AS  
SELECT Orders.UserID, u.FirstName, u.LastName, c.CourseName, c.StartDate  
FROM Orders  
INNER JOIN Payments ON Payments.OrderID=Orders.OrderID
```

```

INNER JOIN Users as u on u.UserID = Orders.UserID
INNER JOIN Courses c on c.ProductID = Orders.ProductID
WHERE Payments.Status = 0 AND c.EndDate < getDate()
GROUP BY Orders.UserID, u.FirstName, u.LastName, c.CourseName, c.StartDate;

```

8. Lista dłużników dla studiów - Julia Grela

```

CREATE VIEW StudyDebtors AS
SELECT Orders.UserID, u.FirstName, u.LastName, s.StudyName, r.StartDate
FROM Orders
INNER JOIN Payments on Payments.OrderID=Orders.OrderID
INNER JOIN Users as u on u.UserID = Orders.UserID
INNER JOIN Reunions r on r.ProductID = Orders.ProductID
INNER JOIN Semesters sr on sr.SemesterID=r.SemesterID
INNER JOIN Studies s on s.StudyID=sr.StudyID
WHERE Payments.Status = 0 AND r.EndDate < getDate()
GROUP BY Orders.UserID, u.FirstName, u.LastName, s.StudyName, r.StartDate;

```

9. Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia (z informacją, czy wydarzenie jest stacjonarnie, czy zdalnie) - Wojciech Burzak

```

CREATE VIEW FutureParticipants AS
SELECT count(wb.WebinarParticipantID) as NumberOfPeople, Webinars.WebinarName as
EventName, 'online' as mode, p.ProductType, w.Date
FROM WebinarParticipants wb

INNER JOIN WebinarMeetings as w on w.WebinarMeetingID = wb.WebinarMeetingID
INNER JOIN Webinars on Webinars.WebinarID =w.WebinarID
INNER JOIN Products as p on p.ProductID=Webinars.ProductID
WHERE w.Date > getDate()
GROUP BY Webinars.WebinarName, p.ProductType, w.Date

UNION

SELECT count(cp.CourseParticipantID) as NumberOfPeople, c.CourseName as EventName,
m.ModuleType, 'course module' as ProductType, m.Date
FROM CourseParticipants cp
INNER JOIN Courses as c on c.CourseID = cp.CourseID
INNER JOIN Products as p on p.ProductID=c.ProductID
INNER JOIN Modules as m on m.CourseID=c.CourseID
WHERE c.StartDate > getDate()
GROUP BY c.CourseName, m.ModuleType, m.Date

UNION

```

```

SELECT count(st.StudentID) as NumberOfPeople, s.StudyName as EventName,
sm.MeetingType, p.ProductType, sm.Date
FROM Students st
INNER JOIN Studies as s on s.StudyID=st.StudyID
INNER JOIN Semesters as sr on sr.SemesterID = s.StudyID
INNER JOIN Reunions as r on r.SemesterID = sr.SemesterID
INNER JOIN Subjects as sb on sb.SemesterID=sr.SemesterID
INNER JOIN StudyMeetings as sm on sm.SubjectID=sb.SubjectID
INNER JOIN Products as p on p.ProductID=r.ProductID
WHERE r.StartDate > getDate()
GROUP BY s.StudyName, sm.MeetingType, p.ProductType, sm.Date

```

10. Liczba zapisanych osób na przyszłe webinary - Wojciech Burzak

```

CREATE VIEW FutureWebinarParticipants AS
SELECT count(wb.WebinarParticipantID) as NumberOfPeople, Webinars.WebinarName,
'online' as mode, w.Date as WebinarDate
FROM WebinarParticipants wb
INNER JOIN WebinarMeetings as w on w.WebinarMeetingID = wb.WebinarMeetingID
INNER JOIN Webinars on Webinars.WebinarID =w.WebinarID
WHERE w.Date > getDate()
GROUP BY Webinars.WebinarName, w.Date

```

11. Liczba zapisanych osób na przyszłe kursy - Wojciech Burzak

```

CREATE VIEW FutureCourseParticipants AS
SELECT count(cp.CourseParticipantID) as NumberOfPeople, c.CourseName, c.StartDate
FROM CourseParticipants cp
INNER JOIN Courses as c on c.CourseID = cp.CourseID
WHERE c.StartDate > getDate()
GROUP BY c.CourseName, c.StartDate

```

12. Liczba zapisanych osób na przyszłe zjazdy studyjne (studenci + osoby z zewnątrz) - Wojciech Burzak

```

CREATE VIEW FutureStudyMeetingParticipants AS
SELECT
COUNT(DISTINCT COALESCE(s.StudentID, u.UserID)) AS NumberOfPeople,
st.StudyName as Studies,
sub.SubjectName,
sm.Date

```

```

FROM
    StudyMeetingAttendances sma
LEFT JOIN Students s ON s.StudentID = sma.UserID
LEFT JOIN Users AS u ON u.UserID = sma.UserID
INNER JOIN StudyMeetings sm ON sm.StudyMeetingID = sma.StudyMeetingID
INNER JOIN Subjects sub ON sub.SubjectID = sm.SubjectID
INNER JOIN Semesters AS sem ON sem.SemesterID = sub.SemesterID
INNER JOIN Studies AS st ON st.StudyID = sem.StudyID
WHERE
    sm.Date > GETDATE()
GROUP BY
    st.StudyName,
    sub.SubjectName,
    sm.Date;

```

13. Ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach. - Bianka Marciniak

Frekwencję liczymy ze wzoru : $(\text{obecnych} \times 100\%) / (\text{obecnych} + \text{nieobecnych})$

```

CREATE VIEW AttendanceRate AS
SELECT
    'StudyMeeting' AS EventType,
    sm.StudyMeetingID AS EventID,
    sub.SubjectName AS EventName,
    sm.MeetingType AS MeetingType,
    COUNT(DISTINCT CASE WHEN sma.HasPassed = 1 THEN sma.UserID END) AS
PresentParticipants,
    COUNT(DISTINCT CASE WHEN sma.HasPassed = 0 THEN sma.UserID END) AS
AbsentParticipants,
    CASE
        WHEN COUNT(DISTINCT CASE WHEN sma.HasPassed = 1 THEN sma.UserID END) +
COUNT(DISTINCT CASE WHEN sma.HasPassed = 0 THEN sma.UserID END) > 0
        THEN (COUNT(DISTINCT CASE WHEN sma.HasPassed = 1 THEN sma.UserID END) * 100)
/ (COUNT(DISTINCT CASE WHEN sma.HasPassed = 1 THEN sma.UserID END) + COUNT(DISTINCT
CASE WHEN sma.HasPassed = 0 THEN sma.UserID END))
        ELSE NULL
    END AS AttendanceRate
FROM
    StudyMeetings sm
INNER JOIN
    Subjects sub ON sub.SubjectID = sm.SubjectID
LEFT JOIN

```

```

StudyMeetingAttendances sma ON sma.StudyMeetingID = sm.StudyMeetingID
WHERE
    sm.Date < GETDATE()
GROUP BY
    sm.StudyMeetingID, sub.SubjectName, sm.MeetingType

UNION

SELECT
    'WebinarMeeting' AS EventType,
    wm.WebinarMeetingID AS EventID,
    w.WebinarName AS EventName,
    'online' AS MeetingType,
    COUNT(DISTINCT CASE WHEN wa.HasPassed = 1 THEN wa.WebinarParticipantID END) AS
PresentParticipants,
    COUNT(DISTINCT CASE WHEN wa.HasPassed = 0 THEN wa.WebinarParticipantID END) AS
AbsentParticipants,
    CASE
        WHEN COUNT(DISTINCT CASE WHEN wa.HasPassed = 1 THEN wa.WebinarParticipantID
END) + COUNT(DISTINCT CASE WHEN wa.HasPassed = 0 THEN wa.WebinarParticipantID END)
> 0
        THEN (COUNT(DISTINCT CASE WHEN wa.HasPassed = 1 THEN wa.WebinarParticipantID
END) * 100) / (COUNT(DISTINCT CASE WHEN wa.HasPassed = 1 THEN
wa.WebinarParticipantID END) + COUNT(DISTINCT CASE WHEN wa.HasPassed = 0 THEN
wa.WebinarParticipantID END))
        ELSE NULL
    END AS AttendanceRate
FROM
    WebinarMeetings wm
INNER JOIN
    Webinars w ON w.WebinarID = wm.WebinarID
LEFT JOIN
    WebinarAttendances wa ON wa.WebinarMeetingID = wm.WebinarMeetingID
WHERE
    wm.Date < GETDATE()
GROUP BY
    wm.WebinarMeetingID, w.WebinarName

UNION

SELECT
    'Module' AS EventType,
    m.ModuleID AS EventID,
    c.CourseName AS EventName,
    m.ModuleType AS MeetingType,

```



```

COUNT(DISTINCT CASE WHEN ma.HasPassed = 1 THEN ma.CourseParticipantID END) AS
PresentParticipants,
COUNT(DISTINCT CASE WHEN ma.HasPassed = 0 THEN ma.CourseParticipantID END) AS
AbsentParticipants,
CASE
    WHEN COUNT(DISTINCT CASE WHEN ma.HasPassed = 1 THEN ma.CourseParticipantID
END) + COUNT(DISTINCT CASE WHEN ma.HasPassed = 0 THEN ma.CourseParticipantID END) >
0
    THEN (COUNT(DISTINCT CASE WHEN ma.HasPassed = 1 THEN ma.CourseParticipantID
END) * 100) / (COUNT(DISTINCT CASE WHEN ma.HasPassed = 1 THEN
ma.CourseParticipantID END) + COUNT(DISTINCT CASE WHEN ma.HasPassed = 0 THEN
ma.CourseParticipantID END))
    ELSE NULL
END AS AttendanceRate
FROM
    Modules m
INNER JOIN
    Courses c ON c.CourseID = m.CourseID
LEFT JOIN
    ModuleAttendances ma ON ma.ModuleID = m.ModuleID
WHERE
    m.Date < GETDATE()
GROUP BY
    m.ModuleID, c.CourseName, m.ModuleType;

```

14. Lista obecności dla każdego szkolenia z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie -Bianka Marciniak

```

CREATE VIEW AttendanceList AS
SELECT
    ma.CourseParticipantID,
    u.FirstName,
    u.LastName,
    m.ModuleName AS EventName,
    m.Date AS EventDate,
    ma.HasPassed AS WasPresent,
    'Module' AS EventType,
    'CourseParticipant' AS ParticipantType
FROM
    ModuleAttendances ma
    INNER JOIN CourseParticipants AS cp ON cp.CourseParticipantID =
ma.CourseParticipantID

```

```

INNER JOIN Users AS u ON u.UserID = cp.UserID
INNER JOIN Modules m ON m.ModuleID = ma.ModuleID
WHERE
    m.Date < GETDATE()

UNION

SELECT
    sma.UserID,
    u.FirstName,
    u.LastName,
    sub.SubjectName AS EventName,
    sm.Date AS EventDate,
    sma.HasPassed AS WasPresent,
    'StudyMeeting' AS EventType,
CASE
    WHEN s.StudentID IS NOT NULL AND u.UserID IS NOT NULL THEN 'Student'
    WHEN u.UserID IS NOT NULL THEN 'OtherParticipant'
END AS ParticipantType
FROM
    StudyMeetingAttendances sma
LEFT JOIN Students s ON s.StudentID = sma.UserID
LEFT JOIN Users AS u ON u.UserID = sma.UserID
INNER JOIN StudyMeetings sm ON sm.StudyMeetingID = sma.StudyMeetingID
INNER JOIN Subjects sub ON sub.SubjectID = sm.SubjectID
WHERE
    sm.Date < GETDATE()

UNION

SELECT
    wa.WebinarParticipantID,
    u.FirstName,
    u.LastName,
    w.WebinarName AS EventName,
    wm.Date AS EventDate,
    wa.HasPassed AS WasPresent,
    'Webinar' AS EventType,
    'WebinarParticipant' AS ParticipantType
FROM
    WebinarAttendances wa
    INNER JOIN WebinarParticipants AS wp ON wp.WebinarParticipantID =
wa.WebinarParticipantID
    INNER JOIN Users AS u ON u.UserID = wp.UserID
    INNER JOIN WebinarMeetings AS wm ON wm.WebinarMeetingID = wa.WebinarMeetingID

```

```
INNER JOIN Webinars AS w ON w.WebinarID = wm.WebinarID
WHERE
    wm.Date < GETDATE();
```

15. Lista obecności na webinarach - Julia Grela

```
CREATE VIEW WebinarAttendanceList AS
```

```
SELECT
```

```
    wa.WebinarParticipantID,
```

```
    u.FirstName,
```

```
    u.LastName,
```

```
    w.WebinarName,
```

```
    wm.Date,
```

```
wa.HasPassed AS WasPresent
```

```
FROM
```

```
    WebinarAttendances wa
```

```
    INNER JOIN WebinarParticipants AS wp ON
```

```
wp.WebinarParticipantID=wa.WebinarParticipantID
```

```
    INNER JOIN Users AS u ON u.UserID = wp.UserID
```

```
INNER JOIN WebinarMeetings AS wm ON wm.WebinarMeetingID=wa.WebinarMeetingID
```

```
INNER JOIN Webinars AS w ON w.WebinarID=wm.WebinarID
```

```
WHERE
```

```
    wm.Date < GETDATE();
```

16. Lista obecności na modułach - Julia Grela

```
CREATE VIEW ModuleAttendanceList AS
```

```

SELECT
    ma.CourseParticipantID,
    u.FirstName,
    u.LastName,
    m.ModuleName,
    m.Date,
    ma.HasPassed AS WasPresent
FROM
    ModuleAttendances ma
    INNER JOIN CourseParticipants AS cp ON
cp.CourseParticipantID=ma.CourseParticipantID
    INNER JOIN Users AS u ON u.UserID = cp.UserID
    INNER JOIN Modules m ON m.ModuleID = ma.ModuleID

WHERE
    m.Date < GETDATE()

```

17. Lista obecności na spotkaniach studyjnych (rozdzielenie na studentów i osoby z zewnątrz) - Julia Grela

```

CREATE VIEW StudyMeetingAttendanceList AS
SELECT
    CASE
        WHEN s.StudentID IS NOT NULL AND u.UserID IS NOT NULL THEN 'Student'
        WHEN u.UserID IS NOT NULL THEN 'OtherParticipant'
    END AS ParticipantType,
    COALESCE(s.StudentID, u.UserID) AS ParticipantID,
    u.FirstName,
    u.LastName,
    sub.SubjectName AS EventName,
    sm.Date AS EventDate,
    sma.HasPassed AS WasPresent
FROM
    StudyMeetingAttendances sma
LEFT JOIN Students s ON s.StudentID = sma.UserID
LEFT JOIN Users AS u ON u.UserID = sma.UserID
INNER JOIN StudyMeetings sm ON sm.StudyMeetingID = sma.StudyMeetingID
INNER JOIN Subjects sub ON sub.SubjectID = sm.SubjectID
WHERE
    sm.Date < GETDATE();

```

18. Lista szkoleń kolidujących ze sobą czasowo - Wojciech Burzak

```

CREATE VIEW OverlappingMeetings AS
WITH OverlappingMeetings AS (
    SELECT

```

```

        'Webinar' AS MeetingType,
        DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00', WM.StartTime), WM.Date) AS
StartDateTime,
        DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00', WM.EndTime), WM.Date) AS
EndDateTime,
        WM.WebinarMeetingID AS MeetingID,
        w.WebinarName,
        NULL AS ModuleName,
        NULL AS Subject
FROM
        WebinarMeetings WM
INNER JOIN Webinars w ON w.WebinarID = wm.WebinarID
UNION ALL
SELECT
        'StudyMeeting' AS MeetingType,
        DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00', SM.StartTime), SM.Date) AS
StartDateTime,
        DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00', SM.EndTime), SM.Date) AS
EndDateTime,
        SM.StudyMeetingID AS MeetingID,
        NULL AS WebinarName,
        NULL AS ModuleName,
        SUB.SubjectName
FROM
        StudyMeetings SM
INNER JOIN Subjects sub ON SM.SubjectID = sub.SubjectID

UNION ALL
SELECT
        'Module' AS MeetingType,
        DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00', M.StartTime), M.Date) AS
StartDateTime,
        DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00', M.EndTime), M.Date) AS
EndDateTime,
        M.ModuleID AS MeetingID,
        NULL AS WebinarName,
        M.ModuleName,
        NULL AS Subject
FROM
        Modules M
)
SELECT
        OM1.MeetingType AS MeetingType1,
        OM1.MeetingID AS MeetingID1,
        OM1.StartDateTime AS StartDateTimel,

```

```

    OM1.EndDateTime AS EndDateTime1,
    OM1.WebinarName AS WebinarName1,
    OM1.ModuleName AS ModuleName1,
    OM1.Subject AS Subject1,
    OM2.MeetingType AS MeetingType2,
    OM2.MeetingID AS MeetingID2,
    OM2.StartDateTime AS StartDateTime2,
    OM2.EndDateTime AS EndDateTime2,
    OM2.WebinarName AS WebinarName2,
    OM2.ModuleName AS ModuleName2,
    OM2.Subject AS Subject2
FROM
    OverlappingMeetings OM1
INNER JOIN
    OverlappingMeetings OM2 ON OM1.MeetingID < OM2.MeetingID
    AND OM1.StartDateTime < OM2.EndDateTime
    AND OM1.EndDateTime > OM2.StartDateTime;

```

20. Lista kierunków studiów wraz z listą przedmiotów - Julia Grela

```

CREATE VIEW StudySubjects AS

SELECT

    s.StudyName,

    s.SemesterCount,

    STRING_AGG(sb.SubjectName, ', ') AS Subjects

FROM Studies s

INNER JOIN Semesters ss ON ss.StudyID = s.StudyID

INNER JOIN Subjects sb ON sb.SemesterID = ss.SemesterID

GROUP BY s.StudyName, s.SemesterCount;

```

21. Lista kursów wraz z rozpiską modułów i datami kiedy się odbywają -Bianka Marciniak

```

CREATE VIEW CoursesAndModules AS

```

```

SELECT c.CourseName, c.StartDate AS CourseStartDate, m.ModuleName, m.Date AS
ModuleDate, m.ModuleType, m.StartTime, m.EndTime

FROM Courses c

INNER JOIN Modules AS m ON m.CourseID=c.CourseID

GROUP BY c.CourseName, c.StartDate, m.ModuleName, m.Date, m.ModuleType,
m.StartTime, m.EndTime

```

22. Lista prowadzących webinary - Bianka Marciniak

```

CREATE VIEW WebinarLecturers AS

SELECT l.LecturerID, u.FirstName, u.LastName, u.Email, w.WebinarName, wm.Date

from WebinarMeetings wm

inner join Webinars as w on w.WebinarID=wm.WebinarID

inner join Lecturers as l on l.LecturerID=wm.LecturerID

inner join Users as u on u.UserID=l.UserID

group by l.LecturerID, u.FirstName, u.LastName, u.Email, w.WebinarName, wm.Date

```

23. Lista prowadzących moduły - Bianka Marciniak

```

CREATE VIEW ModulesLecturers AS

SELECT l.LecturerID, u.FirstName, u.LastName, u.Email, c.CourseName, m.ModuleName,
m.Date

from Modules m

inner join Courses as c on c.CourseID=m.CourseID

inner join Lecturers as l on l.LecturerID=m.LecturerID

inner join Users as u on u.UserID=l.UserID

group by l.LecturerID, u.FirstName, u.LastName, u.Email, c.CourseName,
m.ModuleName, m.Date

```

24. Lista prowadzących spotkania studyjne - Bianka Marciniak

```
CREATE VIEW StudyMeetingLecturers AS

SELECT l.LecturerID, u.FirstName, u.LastName, u.Email, s.StudyName, sb.SubjectName,
sm.Date as MeetingDate

from StudyMeetings sm

inner join Subjects sb on sb.SubjectID=sm.SubjectID

inner join Semesters as sem on sem.SemesterID=sb.SemesterID

inner join Studies as s on s.StudyID=sem.StudyID

inner join Lecturers as l on l.LecturerID=sm.LecturerID

inner join Users as u on u.UserID=l.UserID

group by l.LecturerID, u.FirstName, u.LastName, u.Email, s.StudyName,
sb.SubjectName, sm.Date
```

25. Webinary, na które zapisana jest dana osoba (ich data oraz czy osoba za nie zapłaciła) - Wojciech Burzak

```
CREATE VIEW WebinarParticipantsPayments AS

SELECT

    u.UserID,

    u.FirstName,

    u.LastName,

    w.WebinarID,

    w.WebinarName,

    wm.Date,

    p.Status,

    pr.ProductPrice AS WebinarPrice
```



```
FROM WebinarParticipants wp

INNER JOIN WebinarMeetings wm ON wm.WebinarMeetingID=wp.WebinarMeetingID

INNER JOIN Webinars w ON w.WebinarID = wm.WebinarID

LEFT JOIN Users u ON wp.UserID = u.UserID

INNER JOIN Orders o ON o.ProductID=w.ProductID

INNER JOIN Payments p ON p.OrderID=o.OrderID

INNER JOIN Products pr ON pr.ProductID=w.ProductID
```

26. Kursy, na które zapisana jest dana osoba (ich data oraz czy osoba za nie zapłaciła) - Wojciech Burzak

```
CREATE VIEW CourseParticipantsPayments AS

SELECT

    u.UserID,

    u.FirstName,

    u.LastName,

    c.CourseID,

    c.CourseName,

    c.StartDate,

    p.Status,

    pr.ProductPrice AS CoursePrice

FROM CourseParticipants cp

INNER JOIN Courses c ON c.CourseID = cp.CourseID

INNER JOIN Users u ON cp.UserID = u.UserID

INNER JOIN Orders o ON o.ProductID=c.ProductID

INNER JOIN Payments p ON p.OrderID=o.OrderID

INNER JOIN Products pr ON pr.ProductID=c.ProductID
```

27. Lista adresów, na jakie trzeba wysłać wygenerowane certyfikaty ukończenia studiów bądź kursu (wyświetlone są jedynie adresy osób, które zaliczyły kurs bądź zdały egzamin końcowy na studiach) - Bianka Marciniak

```
CREATE VIEW CertificatesToSend AS
SELECT
    st.StudentID AS ParticipantID,
    u.FirstName + ' ' + u.LastName AS ParticipantName,
    st.Address + ', ' + c.CityName + ', ' + co.CountryName AS Address,
    s.StudyName AS CourseOrStudyName,
    'Student' AS ParticipantType
FROM Students st
INNER JOIN ExamResults er ON er.StudentID = st.StudentID
INNER JOIN Studies s ON s.StudyID = st.StudyID
INNER JOIN Users u ON u.UserID = st.UserID
INNER JOIN Cities c ON c.CityID = st.CityID
INNER JOIN Voivodeships v ON v.VoivodeshipID = c.VoivodeshipID
INNER JOIN Countries co ON co.CountryID = v.CountryID
WHERE er.HasPassed = 1

UNION ALL

SELECT
    cp.CourseParticipantID AS ParticipantID,
    u.FirstName + ' ' + u.LastName AS ParticipantName,
    MAX(cp.Address + ', ' + ct.CityName + ', ' + co.CountryName) AS Address,
    c.CourseName AS CourseOrStudyName,
    'CourseParticipant' AS ParticipantType
FROM CourseParticipants cp
INNER JOIN Users u ON u.UserID = cp.UserID
INNER JOIN Courses c ON c.CourseID = cp.CourseID
INNER JOIN Modules m ON m.CourseID = c.CourseID
INNER JOIN ModuleAttendances ma ON ma.ModuleID = m.ModuleID AND
ma.CourseParticipantID = cp.CourseParticipantID
INNER JOIN Cities ct ON ct.CityID = cp.CityID
INNER JOIN Voivodeships v ON v.VoivodeshipID = ct.VoivodeshipID
INNER JOIN Countries co ON co.CountryID = v.CountryID
GROUP BY
    cp.CourseParticipantID,
    u.FirstName,
    u.LastName,
    c.CourseName;
```

28. Widok na praktyki studenta (czy zostały zaliczone, kiedy się odbywają/odbyły, na jakich studiach jest dany student) - Julia Grela

```
CREATE VIEW StudentsInternships AS
SELECT st.StudentID, u.FirstName + ' ' + u.LastName AS StudentName, s.StudyName,
i.StartDate AS StartOfInternship, i.EndDate AS EndOfInternship, i.IsCompleted
FROM Students st
INNER JOIN Internships AS i ON i.StudentID=st.StudentID
inner join Studies AS s ON s.StudyID=st.StudyID
inner join Users AS u ON u.UserID=st.UserID
```

29. Lista studentów i nazwy studiów na jakie są zapisani - Julia Grela

```
CREATE VIEW StudentStudy AS
SELECT st.StudentID, u.FirstName + ' ' + u.LastName AS StudentName, s.StudyName
FROM Students st
inner join Studies AS s ON s.StudyID=st.StudyID
inner join Users AS u ON u.UserID=st.UserID
```

30. Lista osób zapisanych na kursy, pokazuje status zaliczenia danego kursu - Julia Grela

```
CREATE VIEW CoursePassingStatus AS
SELECT
    cp.CourseParticipantID,
    u.FirstName + ' ' + u.LastName AS ParticipantName,
    c.CourseName,
    COUNT(DISTINCT m.ModuleID) AS TotalModules,
    SUM(CASE WHEN ma.HasPassed = 1 THEN 1 ELSE 0 END) AS PassedModules,
    CASE WHEN COUNT(DISTINCT m.ModuleID) > 0 AND
        SUM(CASE WHEN ma.HasPassed = 1 THEN 1 ELSE 0 END) / COUNT(DISTINCT
m.ModuleID) >= 0.8
        THEN 'Passed'
        ELSE 'Not Passed'
    END AS CourseStatus
FROM CourseParticipants cp
INNER JOIN Users u ON u.UserID = cp.UserID
INNER JOIN Courses c ON c.CourseID = cp.CourseID
INNER JOIN Modules m ON m.CourseID = c.CourseID
LEFT JOIN ModuleAttendances ma ON ma.ModuleID = m.ModuleID AND
ma.CourseParticipantID = cp.CourseParticipantID
GROUP BY
```

```
cp.CourseParticipantID,  
u.FirstName,  
u.LastName,  
c.CourseName
```

31. Lista płatności odroczonych, liczba dni zmiany data zapłaty jest stała, równa 7 dni od złożenia zamówienia - Bianka Marciniak

```
CREATE VIEW DefferedPayments AS  
SELECT  
    u.FirstName + ' ' + u.LastName AS Name,  
    o.Price,  
    DATEADD(DAY, 7, p.PayDate) AS UpdatedPayDate  
FROM Orders o  
INNER JOIN Users u ON u.UserID = o.UserID  
INNER JOIN Payments p ON p.OrderID = o.OrderID  
WHERE p.IsDeffered = 1;
```

32. Lista loginów oraz haseł użytkowników wraz z imieniem, nazwiskiem, rolą tych osób w systemie - Wojciech Burzak

```
CREATE VIEW LoginPassword AS  
SELECT u.UserID, u.FirstName, u.LastName, r.RoleName, u.Email, ua.Login,  
ua.Password  
FROM UsersAccounts ua  
INNER JOIN Users AS u ON u.UserID=ua.UserID  
INNER JOIN UserRole AS ur ON ur.UserID=ua.UserID  
INNER JOIN Roles AS r ON r.RoleID=ur.RoleID
```

FUNKCJE

1. Zliczanie liczby kursantów dla danego kursu (Bianka Marciniak)

```
CREATE FUNCTION GetCourseParticipantCount (@CourseID INT)
RETURNS INT
AS
BEGIN
    RETURN (SELECT COUNT(*) FROM CourseParticipants WHERE CourseID = @CourseID)
END
```

Wywołanie:

```
SELECT dbo.GetCourseParticipantCount(@CourseID) AS ParticipantCount;
```

2. Zliczanie wartości całego zamówienia (Bianka Marciniak)

```
CREATE FUNCTION CalculateOrderTotal (@OrderID INT)
RETURNS MONEY
AS
BEGIN
    RETURN (SELECT SUM(Quantity * Price) FROM Orders WHERE OrderID = @OrderID)
END
```

Wywołanie:

```
SELECT dbo.CheckModuleAvailability(@ModuleID) AS AvailableSpots;
```

3. Sprawdzanie ile jeszcze osób może zapisać się na Moduł (Wojciech Burzak)

```
CREATE FUNCTION CheckModuleAvailability (@ModuleID INT)
RETURNS INT
AS
BEGIN
    DECLARE @MaxParticipants INT;
    SELECT @MaxParticipants = MaxParticipants FROM OfflineModules WHERE ModuleID =
@ModuleID;
    RETURN @MaxParticipants - (SELECT COUNT(*) FROM ModuleAttendances WHERE ModuleID
= @ModuleID)
END
```

Wywołanie:

```
SELECT dbo.CheckModuleAvailability(@ModuleID) AS AvailableSpots;
```

4. Sprawdzanie czy student zdał egzamin (Bianka Marciniak)

```
CREATE FUNCTION HasStudentPassedExam (@StudentID INT, @ExamID INT)
RETURNS BIT
AS
BEGIN
    RETURN (SELECT TOP 1 HasPassed FROM ExamResults WHERE StudentID = @StudentID AND
ExamID = @ExamID)
END
```

Wywołanie:

```
SELECT dbo.HasStudentPassedExam(@StudentID, @ExamID) AS PassedExam;
```

5. Sprawdzanie ile jest w sumie studentów (Bianka Marciniak)

```
CREATE FUNCTION GetTotalStudentCount ()
RETURNS INT
AS
BEGIN
    RETURN (SELECT COUNT(*) FROM Students)
END
```

Wywołanie:

```
SELECT dbo.GetTotalStudentCount () AS TotalStudentCount;
```

6. Sprawdzanie czy student ma jakieś niezapłacone transakcje (Wojciech Burzak)

```
CREATE FUNCTION HasStudentPendingPayments (@StudentID INT)
RETURNS BIT
AS
BEGIN
    RETURN (SELECT CASE WHEN SUM(CASE WHEN Payments.Status = 0 THEN 1 ELSE 0 END) >
0 THEN 1 ELSE 0 END
FROM Payments
INNER JOIN Orders ON Payments.OrderID = Orders.OrderID
INNER JOIN Students ON Orders.UserID = Students.UserID
WHERE Students.StudentID = @StudentID)
END
```

Wywołanie:

```
SELECT dbo.HasStudentPendingPayments (@StudentID) AS HasPendingPayments;
```

7. Sprawdzanie ile kursów prowadzi dany wykładowca (Bianka Marciniak)

```
CREATE FUNCTION GetNumberOfCoursesByLecturer (@LecturerID INT)
```

```

RETURNS INT
AS
BEGIN
    RETURN (SELECT COUNT(*)
            FROM Courses
            INNER JOIN Modules ON Courses.CourseID = Modules.CourseID
            WHERE Modules.LecturerID = @LecturerID)
END

```

Wywołanie:

```
SELECT dbo.GetNumberOfCoursesByLecturer(@LecturerID) AS NumberOfCourses;
```

8. Zwraca ile modułów ma podany kurs (Wojciech Burzak)

```

CREATE FUNCTION GetNumberOfModulesInCourse(@CourseID INT)
RETURNS INT
AS
BEGIN
    RETURN (SELECT COUNT(*) FROM Modules WHERE CourseID = @CourseID)
END

```

Wywołanie:

```
SELECT dbo.GetNumberOfModulesInCourse(@CourseID) AS NumberOfModules;
```

9. Zwraca najpopularniejszy kurs (Wojciech Burzak)

```

CREATE FUNCTION GetMostPopularCourse()
RETURNS INT
AS
BEGIN
    RETURN (SELECT TOP 1 CourseID FROM CourseParticipants
            GROUP BY CourseID
            ORDER BY COUNT(*) DESC)
END

```

Wywołanie:

```
SELECT dbo.GetMostPopularCourse() AS MostPopularCourseID;
```

10. Zwraca długość (w minutach) trwania wszystkich modułów w danym kursie (Wojciech Burzak)

```

CREATE FUNCTION CalculateTotalCourseDuration(@CourseID INT)
RETURNS INT

```

```

AS
BEGIN
    RETURN (SELECT SUM(DATEDIFF(minute, StartTime, EndTime))
            FROM Modules
            WHERE CourseID = @CourseID)
END

```

Wywołanie:

```
SELECT dbo.CalculateTotalCourseDuration(@CourseID) AS TotalCourseDuration;
```

11. Sprawdza czy dany kurs ma jakieś moduły online (Wojciech Burzak)

```

CREATE FUNCTION HasOnlineModules(@CourseID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @HasOnline BIT;
    SELECT @HasOnline = CASE WHEN COUNT(*) > 0 THEN 1 ELSE 0 END
    FROM Modules
    WHERE CourseID = @CourseID AND ModuleType IN ('synchronic online', 'asynchronic
online')
    RETURN @HasOnline
END

```

Wywołanie:

```
SELECT dbo.HasOnlineModules(@CourseID) AS HasOnlineModules;
```

12. Zwraca ilość niezapłaconych transakcji dla danego użytkownika (Julia Grela)

```

CREATE FUNCTION GetPendingPaymentsCountByUser(@UserID INT)
RETURNS INT
AS
BEGIN
    RETURN (SELECT COUNT(*) FROM Payments
            INNER JOIN Orders ON Payments.OrderID = Orders.OrderID
            WHERE Orders.UserID = @UserID AND Payments.Status = 0)
END

```

Wywołanie:

```
SELECT dbo.GetPendingPaymentsCountByUser(@UserID) AS PendingPaymentsCount;
```

13. Sprawdza status zamówienia (Julia Grela)


```

CREATE FUNCTION GetPaymentStatus(@OrderID INT)
RETURNS BIT
AS
BEGIN
    RETURN (SELECT TOP 1 Status FROM Payments WHERE OrderID = @OrderID ORDER BY
PayDate DESC)
END
Wywołanie:
SELECT dbo.GetPaymentStatus(@OrderID) AS PaymentStatus;

```

14. Sprawdza czy link do nagrania się nie przedawnił (Wojciech Burzak)

```

CREATE FUNCTION IsWebinarRecordingAvailable(@WebinarID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @Available BIT;
    SELECT @Available = CASE WHEN COUNT(*) > 0 THEN 1 ELSE 0 END
    FROM WebinarMeetings
    WHERE WebinarID = @WebinarID AND RecordingLink IS NOT NULL AND ExpirationDate >
GETDATE()
    RETURN @Available
END

```

Wywołanie:

```

SELECT dbo.IsWebinarRecordingAvailable(@WebinarID) AS RecordingAvailable;

```

15. Sprawdza ile zamówień było dla danego produktu (Wojciech Burzak)

```

CREATE FUNCTION GetOrderCountForProduct(@ProductID INT)
RETURNS INT
AS
BEGIN
    RETURN (SELECT COUNT(*) FROM Orders WHERE ProductID = @ProductID)
END

```

Wywołanie:

```

SELECT dbo.GetOrderCountForProduct(@ProductID) AS OrderCountForProduct;

```

16. Zwraca liczbę przyszłych webinarów (Bianka Marciniak)

```

CREATE FUNCTION GetActiveWebinarsCount()
RETURNS INT

```

```
AS
BEGIN
    RETURN (SELECT COUNT(*) FROM WebinarMeetings WHERE ExpirationDate > GETDATE())
END
```

Wywołanie:

```
SELECT dbo.GetActiveWebinarsCount() AS ActiveWebinarsCount;
```

17. Zwraca średnią cenę dla wszystkich produktów (Julia Grela)

```
CREATE FUNCTION GetOverallAverageProductPrice()
RETURNS MONEY
AS
BEGIN
    RETURN (SELECT AVG(ProductPrice) FROM Products)
END
```

Wywołanie:

```
SELECT dbo.GetOverallAverageProductPrice() AS OverallAveragePrice;
```

18. Zwraca średnią cenę dla kursów (Julia Grela)

```
CREATE FUNCTION GetAverageCoursePrice()
RETURNS MONEY
AS
BEGIN
    RETURN (SELECT AVG(ProductPrice) FROM Products WHERE ProductType = 'course')
END
```

Wywołanie:

```
SELECT dbo.GetAverageCoursePrice() AS AverageCoursePrice;
```

19. Zwraca średnią cenę dla webinarów (Julia Grela)

```
CREATE FUNCTION GetAverageWebinarPrice()
RETURNS MONEY
AS
BEGIN
    RETURN (SELECT AVG(ProductPrice) FROM Products WHERE ProductType = 'webinar')
END
```

Wywołanie:

```
SELECT dbo.GetAverageWebinarPrice() AS AverageWebinarPrice;
```

20. Zwraca średnią cenę dla zjazdów (Julia Grela)

```
CREATE FUNCTION GetAverageReunionPrice()
RETURNS MONEY
AS
BEGIN
    RETURN (SELECT AVG(ProductPrice) FROM Products WHERE ProductType = 'reunion')
END
```

Wywołanie:

```
SELECT dbo.GetAverageReunionPrice() AS AverageReunionPrice;
```

21. Funkcja sprawdzająca dostępność sali:

```
CREATE FUNCTION CheckRoomAvailability
(
    @LocationID INT,
    @DesiredDate DATETIME,
    @DesiredStartTime TIME,
    @DesiredEndTime TIME,
    @NumberOfParticipants INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsAvailable BIT = 1;
    DECLARE @LocationCapacity INT;

    SELECT @LocationCapacity = Capacity
    FROM [dbo].[Locations]
    WHERE LocationID = @LocationID;

    IF EXISTS (
        SELECT 1
        FROM [dbo].[OfflineModules] om
        INNER JOIN [dbo].[Modules] m ON om.ModuleID = m.ModuleID
        WHERE om.LocationID = @LocationID
            AND m.Date = CAST(@DesiredDate AS DATE)
            AND (m.StartTime < @DesiredEndTime AND m.EndTime > @DesiredStartTime)
            AND om.MaxParticipants > (@LocationCapacity - @NumberOfParticipants)
    )
    BEGIN
        SET @IsAvailable = 0;
    END

    IF EXISTS (
        SELECT 1
```

```

        FROM [dbo].[OfflineStudyMeetings] osm
        INNER JOIN [dbo].[StudyMeetings] sm ON osm.StudyMeetingID =
sm.StudyMeetingID
        WHERE osm.LocationID = @LocationID
            AND sm.Date = CAST(@DesiredDate AS DATE)
            AND (sm.StartTime < @DesiredEndTime AND sm.EndTime > @DesiredStartTime)
            AND osm.MaxParticipants > (@LocationCapacity - @NumberOfParticipants)
    )
BEGIN
    SET @IsAvailable = 0;
END

RETURN @IsAvailable;
END

```

Przykładowe wywołanie:

```

DECLARE @RoomAvailable BIT;
SET @RoomAvailable = dbo.CheckRoomAvailability(
    31,                -- ID lokacji
    '2024-02-02',      -- Żądana data
    '09:00',           -- Żądany czas rozpoczęcia
    '11:00',           -- Żądany czas zakończenia
    59                 -- Liczba uczestników
);
SELECT @RoomAvailable AS IsRoomAvailable;

```

22. Funkcja sprawdzająca dostępność wykładowcy:

```

CREATE FUNCTION IsLecturerAvailable
(
    @LecturerID INT,
    @DesiredDate DATETIME,
    @StartTime TIME,
    @EndTime TIME
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsAvailable BIT = 1;

    -- Sprawdzanie dostępności w tabeli Modules
    IF EXISTS (
        SELECT 1
        FROM [dbo].[Modules]
        WHERE LecturerID = @LecturerID
            AND [Date] = CAST(@DesiredDate AS DATE)
            AND ([StartTime] < @EndTime AND [EndTime] > @StartTime)
    )
    BEGIN
        SET @IsAvailable = 0;
    END

    -- Sprawdzanie dostępności w tabeli StudyMeetings
    IF EXISTS (

```

```

        SELECT 1
        FROM [dbo].[StudyMeetings]
        WHERE LecturerID = @LecturerID
            AND [Date] = CAST(@DesiredDate AS DATE)
            AND ([StartTime] < @EndTime AND [EndTime] > @StartTime)
    )
BEGIN
    SET @IsAvailable = 0;
END

-- Sprawdzanie dostępności w tabeli WebinarMeetings
IF EXISTS (
    SELECT 1
    FROM [dbo].[WebinarMeetings]
    WHERE LecturerID = @LecturerID
        AND [Date] = CAST(@DesiredDate AS DATE)
        AND ([StartTime] < @EndTime AND [EndTime] > @StartTime)
)
BEGIN
    SET @IsAvailable = 0;
END

RETURN @IsAvailable;
END

```

Przykładowe wywołanie:

```

DECLARE @IsLecturerFree BIT;
SET @IsLecturerFree = dbo.IsLecturerAvailable(
    1, -- ID wykładowcy
    '2024-03-31', -- Żądana data
    '09:00', -- Żądany czas rozpoczęcia
    '11:00' -- Żądany czas zakończenia
);
SELECT @IsLecturerFree AS IsLecturerAvailable;

```

23. Funkcja sprawdzająca dostępność tłumacza:

```

CREATE FUNCTION IsTranslatorAvailable
(
    @TranslatorID INT,
    @DesiredDate DATETIME,
    @DesiredStartTime TIME,
    @DesiredEndTime TIME
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsAvailable BIT = 1;
    -- Sprawdzanie dostępności w StudyMeetings
    IF EXISTS (
        SELECT 1 FROM [dbo].[StudyMeetings] sm
        JOIN [dbo].[SubjectsInForeignLanguages] sfl ON sm.SubjectID = sfl.SubjectID
        WHERE sfl.TranslatorID = @TranslatorID
    )

```

```

        AND sm.Date = CAST(@DesiredDate AS DATE)
        AND (sm.StartTime < @DesiredEndTime AND sm.EndTime > @DesiredStartTime)
    )
    BEGIN
        SET @IsAvailable = 0;
    END
-- Sprawdzanie dostępności w WebinarMeetings
    IF EXISTS (
        SELECT 1 FROM [dbo].[WebinarMeetings] wm
        JOIN [dbo].[WebinarsInForeignLanguages] wfl ON wm.WebinarID = wfl.WebinarID
        WHERE wfl.TranslatorID = @TranslatorID
        AND wm.Date = CAST(@DesiredDate AS DATE)
        AND (wm.StartTime < @DesiredEndTime AND wm.EndTime > @DesiredStartTime)
    )
    BEGIN
        SET @IsAvailable = 0;
    END
-- Sprawdzanie dostępności w Modules
    IF EXISTS (
        SELECT 1 FROM [dbo].[Modules] m
        JOIN [dbo].[ModulesInForeignLanguages] mfl ON m.ModuleID = mfl.ModuleID
        WHERE mfl.TranslatorID = @TranslatorID
        AND m.Date = CAST(@DesiredDate AS DATE)
        AND (m.StartTime < @DesiredEndTime AND m.EndTime > @DesiredStartTime)
    )
    BEGIN
        SET @IsAvailable = 0;
    END

    RETURN @IsAvailable;
END

```

Przykładowe wywołanie:

```

DECLARE @IsTranslatorFree BIT;
SET @IsTranslatorFree = dbo.IsTranslatorAvailable(
    5, -- ID tłumacza
    '2024-04-01', -- Żądana data
    '10:00:00', -- Żądany czas rozpoczęcia
    '12:00:00' -- Żądany czas zakończenia
);

SELECT @IsTranslatorFree AS IsTranslatorAvailable;

```

PROCEDURY

1. Dodanie produktu

Procedura dodania produktu do bazy:

```
CREATE PROCEDURE AddProduct
    @ProductType NVARCHAR(20),
    @ProductPrice MONEY,
    @ProductID INT OUTPUT
AS
BEGIN
    IF @ProductType NOT IN ('reunion', 'course', 'webinar')
    BEGIN
        RAISERROR ('Nieprawidłowy typ produktu', 16, 1);
        RETURN;
    END

    INSERT INTO [dbo].[Products] (ProductType, ProductPrice)
    VALUES (@ProductType, @ProductPrice);

    SET @ProductID = SCOPE_IDENTITY();
END
```

Przykładowe wywołanie procedury:

```
DECLARE @NewProductID INT; -- Zmienna dla ID produktu
-- Wywołanie procedury z bezpośrednim przekazaniem wartości:
EXEC AddProduct @ProductType = 'course', @ProductPrice = 100.00, @ProductID =
@NewProductID OUTPUT;
-- Wyświetlenie ID nowo dodanego produktu:
SELECT @NewProductID as NewProductId;
```

2. Dodanie kursu

Procedura dodania kursu do bazy:

```
CREATE PROCEDURE AddCourse
    @ProductID INT,
    @CourseName NVARCHAR(50),
    @StartDate DATETIME,
    @EndDate DATETIME,
    @Description NVARCHAR(MAX),
    @LanguageID INT
AS
BEGIN
    -- Sprawdzenie, czy data zakończenia jest późniejsza niż data rozpoczęcia kursu
    IF @EndDate <= @StartDate
    BEGIN
```

```

        RAISERROR ('Data zakończenia musi być późniejsza niż data rozpoczęcia.', 16,
1);

    RETURN;

END
-- Sprawdzenie, czy istnieje ProductID
IF NOT EXISTS (SELECT 1 FROM [dbo].[Products] WHERE [ProductID] = @ProductID)
BEGIN
    RAISERROR ('ProductID nie istnieje.', 16, 1);
    RETURN;
END
-- Wstawienie nowego kursu do tabeli Courses
INSERT INTO [dbo].[Courses] (ProductID, CourseName, StartDate, EndDate,
Description, LanguageID)
VALUES (@ProductID, @CourseName, @StartDate, @EndDate, @Description,
@LanguageID);
END

```

Przykładowe wywołanie:

```

EXEC AddCourse
    @ProductID = 96,
    @CourseName = 'Programowanie w Javie',
    @StartDate = '2024-10-01',
    @EndDate = '2024-10-31',
    @Description = 'Kurs programowania w języku Java dla początkujących.',
    @LanguageID = '2';

```

2. Dodanie Webinaru:

```

CREATE PROCEDURE AddWebinar
    @WebinarName NVARCHAR(50),
    @Description NVARCHAR(MAX),
    @ProductID INT,
    @LanguageID INT,
    @NewWebinarID INT OUTPUT
AS
BEGIN
    -- Sprawdzenie, czy ProductID i LanguageID istnieją w odpowiednich tabelach
    IF @ProductID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM [dbo].[Products] WHERE
ProductID = @ProductID)
    BEGIN
        RAISERROR ('Podany ProductID nie istnieje.', 16, 1);
        RETURN;
    END

```



```

        IF @LanguageID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM [dbo].[Languages] WHERE
LanguageID = @LanguageID)
        BEGIN
            RAISERROR ('Podany LanguageID nie istnieje.', 16, 1);
            RETURN;
        END
-- Dodanie informacji o webinarze do tabeli Webinars
INSERT INTO [dbo].[Webinars] (WebinarName, Description, ProductID, LanguageID)
VALUES (@WebinarName, @Description, @ProductID, @LanguageID);
-- Ustawienie identyfikatora nowo utworzonego webinaru
SET @NewWebinarID = SCOPE_IDENTITY();
END;
GO

```

Przykładowe wywołanie:

```

DECLARE @NewWebinarID INT;
EXEC AddWebinar
    @WebinarName = 'Webinar o SQL',
    @Description = 'Szczegółowe wprowadzenie do SQL',
    @ProductID = 91,
    @LanguageID = 2,
    @NewWebinarID = @NewWebinarID OUTPUT;

SELECT @NewWebinarID as IDNowegoWebinaru;

```

3. Dodanie kierunku studiów :

```

CREATE PROCEDURE AddStudy
    @StudyName NVARCHAR(50),
    @StudyType NVARCHAR(20),
    @SemesterCount INT,
    @MaxStudents INT,
    @LinkToSyllabus NVARCHAR(255),
    @Description NVARCHAR(MAX),
    @NewStudyID INT OUTPUT
AS
BEGIN
-- Sprawdzenie poprawności danych wejściowych
IF @SemesterCount <= 2
    BEGIN
        RAISERROR ('Liczba semestrów musi być większa niż 2.', 16, 1);
        RETURN;
    END

IF NOT @LinkToSyllabus LIKE 'https://%com'

```

```

BEGIN
    RAISERROR ('Link do sylabusa jest nieprawidłowy.', 16, 1);
    RETURN;
END
-- Dodanie informacji o studiach do tabeli Studies
INSERT INTO [dbo].[Studies] (StudyName, StudyType, SemesterCount, MaxStudents,
LinkToSyllabus, Description)
VALUES (@StudyName, @StudyType, @SemesterCount, @MaxStudents, @LinkToSyllabus,
@Description);
-- Ustawienie identyfikatora nowo utworzonych studiów
SET @NewStudyID = SCOPE_IDENTITY();
END;
GO

```

Przykładowe wywołanie:

```

DECLARE @NewStudyID INT;
EXEC AddStudy
    @StudyName = 'Psychologia',
    @StudyType = 'Offline',
    @SemesterCount = 6,
    @MaxStudents = 100,
    @LinkToSyllabus = 'https://psychologia_example.com',
    @Description = 'Opis programu studiów',
    @NewStudyID = @NewStudyID OUTPUT;

SELECT @NewStudyID as IDNowegoProgramuStudiów;

```

4. Dodanie modułów :

```

CREATE PROCEDURE AddModuleNew
    @CourseID INT,
    @LecturerID INT,
    @ModuleName NVARCHAR(50),
    @ModuleType NVARCHAR(20),
    @Date DATETIME,
    @StartTime TIME,
    @EndTime TIME,
    @LocationID INT = NULL,           -- Dla modułów offline
    @MaxParticipants INT = NULL,      -- Dla modułów offline
    @MeetingLink NVARCHAR(255) = NULL, -- Dla synchronicznych modułów online
    @RecordingLink NVARCHAR(255) = NULL, -- Dla asynchronicznych modułów online
    @ExpirationDate DATETIME = NULL  -- Dla asynchronicznych modułów online
AS
BEGIN
    -- Dodajemy podstawowe informacje o module
    INSERT INTO [dbo].[Modules] (CourseID, LecturerID, ModuleName, ModuleType,
Date, StartTime, EndTime)

```

```

VALUES (@CourseID, @LecturerID, @ModuleName, @ModuleType, @Date, @StartTime,
@EndTime);

DECLARE @ModuleID INT = SCOPE_IDENTITY();

-- Logika dla różnych typów modułów
IF @ModuleType = 'asynchronous online'
BEGIN
    INSERT INTO [dbo].[AsynchronousOnlineModules] (ModuleID, RecordingLink,
ExpirationDate)
VALUES (@ModuleID, @RecordingLink, @ExpirationDate);
END
ELSE IF @ModuleType = 'synchronic online'
BEGIN
    INSERT INTO [dbo].[SynchronicOnlineModules] (ModuleID, MeetingLink)
VALUES (@ModuleID, @MeetingLink);
END
ELSE IF @ModuleType = 'stationary'
BEGIN
    INSERT INTO [dbo].[OfflineModules] (ModuleID, MaxParticipants, LocationID)
VALUES (@ModuleID, @MaxParticipants, @LocationID);
END
END
END

```

Przykładowe wywołanie:

```

EXEC AddModuleNew
@CourseID = 69, -- ID kursu
@LecturerID = 20, -- ID wykładowcy
@ModuleName = 'JavaFX', -- Nazwa modułu
@ModuleType = 'stationary', -- Typ modułu ('stationary' dla modułu
stacjonarnego)
@Date = '2024-10-01', -- Data modułu
@StartTime = '09:00:00', -- Czas rozpoczęcia
@EndTime = '12:00:00', -- Czas zakończenia
@LocationID = 31, -- ID lokalizacji
@MaxParticipants = 30 -- Maksymalna liczba uczestników
-- Pozostałe parametry mogą być pominięte lub ustawione na NULL

```

5. Dodanie WebinarMeeting :

```

CREATE PROCEDURE AddWebinarMeeting
@WebinarID INT,
@LecturerID INT,
@Date DATETIME,
@StartTime TIME,
@EndTime TIME,
@MeetingLink NVARCHAR(255),
@RecordingLink NVARCHAR(255),
@NewWebinarMeetingID INT OUTPUT

```

```

AS
BEGIN
-- Obliczenie ExpirationDate jako 30 dni od StartTime
    DECLARE @ExpirationDate DATETIME = DATEADD(DAY, 30, @Date);
-- Sprawdzenie, czy WebinarID i LecturerID istnieją w odpowiednich tabelach
    IF NOT EXISTS (SELECT 1 FROM [dbo].[Webinars] WHERE WebinarID = @WebinarID)
    BEGIN
        RAISERROR ('Podany WebinarID nie istnieje.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM [dbo].[Lecturers] WHERE LecturerID = @LecturerID)
    BEGIN
        RAISERROR ('Podany LecturerID nie istnieje.', 16, 1);
        RETURN;
    END

-- Dodanie informacji o spotkaniu webinaru do tabeli WebinarMeetings
    INSERT INTO [dbo].[WebinarMeetings] (WebinarID, LecturerID, Date, StartTime,
EndTime, MeetingLink, RecordingLink, ExpirationDate)
        VALUES (@WebinarID, @LecturerID, @Date, @StartTime, @EndTime, @MeetingLink,
@RecordingLink, @ExpirationDate);
-- Ustawienie identyfikatora nowo utworzonego spotkania webinaru
    SET @NewWebinarMeetingID = SCOPE_IDENTITY();
END;
GO

```

Przykładowe wywołanie:

```

DECLARE @NewWebinarMeetingID INT;
EXEC AddWebinarMeeting
    @WebinarID = 31,
    @LecturerID = 15,
    @Date = '2024-01-01',
    @StartTime = '10:00:00',
    @EndTime = '12:00:00',
    @MeetingLink = 'https://meeting15example.com',
    @RecordingLink = 'https://recording15example.com',
    @NewWebinarMeetingID = @NewWebinarMeetingID OUTPUT;

```

6. Dodanie Zjazdu:

```

CREATE PROCEDURE AddReunion
    @SemesterID INT,

```

```

@ProductID INT,
@StartDate DATETIME,
@EndDate DATETIME,
@MaxStudents INT,
@MaxOtherParticipants INT = NULL,
@NewReunionID INT OUTPUT
AS
BEGIN
-- Sprawdzenie poprawności dat
IF @EndDate <= @StartDate
BEGIN
    RAISERROR ('Data zakończenia musi być późniejsza niż data rozpoczęcia.', 16,
1);
    RETURN;
END
-- Sprawdzenie, czy ProductID istnieje w tabeli Products
IF NOT EXISTS (SELECT 1 FROM [dbo].[Products] WHERE ProductID = @ProductID)
BEGIN
    RAISERROR ('Podany ProductID nie istnieje.', 16, 1);
    RETURN;
END
-- Sprawdzenie, czy SemesterID istnieje w tabeli Semesters
IF NOT EXISTS (SELECT 1 FROM [dbo].[Semesters] WHERE SemesterID = @SemesterID)
BEGIN
    RAISERROR ('Podany SemesterID nie istnieje.', 16, 1);
    RETURN;
END
-- Dodanie informacji o spotkaniu do tabeli Reunions
INSERT INTO [dbo].[Reunions] (SemesterID, ProductID, StartDate, EndDate,
MaxStudents, MaxOtherParticipants)
VALUES (@SemesterID, @ProductID, @StartDate, @EndDate, @MaxStudents,
@MaxOtherParticipants);
-- Ustawienie identyfikatora nowo utworzonego spotkania
SET @NewReunionID = SCOPE_IDENTITY();
END;
GO

```

Przykładowe wywołanie:

```

DECLARE @NewReunionID INT;
EXEC AddReunion
    @SemesterID = 1,
    @ProductID = 2,
    @StartDate = '2024-01-01',
    @EndDate = '2024-01-07',

```

```

    @MaxStudents = 100,
    @MaxOtherParticipants = 50,
    @NewReunionID = @NewReunionID OUTPUT;

SELECT @NewReunionID as IDNowegoSpotkania;

```

7. Dodanie przedmiotu:

```

CREATE PROCEDURE AddSubject
    @SemesterID INT,
    @SubjectName NVARCHAR(30),
    @Description NVARCHAR(MAX),
    @NewSubjectID INT OUTPUT
AS
BEGIN
    -- Sprawdzenie, czy SemesterID istnieje w tabeli Semesters
    IF @SemesterID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM [dbo].[Semesters] WHERE
SemesterID = @SemesterID)
        BEGIN
            RAISERROR ('Podany SemesterID nie istnieje.', 16, 1);
            RETURN;
        END
    -- Dodanie informacji o przedmiocie do tabeli Subjects
    INSERT INTO [dbo].[Subjects] (SemesterID, SubjectName, Description)
    VALUES (@SemesterID, @SubjectName, @Description);
    -- Ustawienie identyfikatora nowo utworzonego przedmiotu
    SET @NewSubjectID = SCOPE_IDENTITY();
END;
GO

```

Przykładowe wywołanie:

```

DECLARE @NewSubjectID INT;
EXEC AddSubject
    @SemesterID = 1,
    @Description = 'Podstawy programowania w języku Python.',
    @NewSubjectID = @NewSubjectID OUTPUT;

SELECT @NewSubjectID as IDNowegoPrzedmiotu;

```

8. Dodanie lokalizacji:

```

CREATE PROCEDURE AddLocation
    @CityID INT,
    @Street NVARCHAR(30),
    @Building INT,

```

```

@Classroom INT,
@PostalCode NVARCHAR(10),
@Capacity INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM [dbo].[Cities] WHERE [CityID] = @CityID)
    BEGIN
        RAISERROR ('CityID nie istnieje.', 16, 1);
        RETURN;
    END

    INSERT INTO [dbo].[Locations] (CityID, Street, Building, Classroom, PostalCode,
Capacity)
VALUES (@CityID, @Street, @Building, @Classroom, @PostalCode, @Capacity);
END;

```

Przykładowe wywołanie:

```

EXEC AddLocation
@CityID = 3,
@Street = 'Kawiorzy',
@Building = 12,
@Classroom = 313,
@PostalCode = '30-072',
@Capacity = 60;

```

8. Dodanie użytkownika do bazy:

```

CREATE PROCEDURE AddUser
@UserType NVARCHAR(30),
@FirstName NVARCHAR(30),
@LastName NVARCHAR(30),
@email NVARCHAR(100),
@RoleID INT,
@login NVARCHAR(100),
@Password NVARCHAR(30),
@NewUserID INT OUTPUT
AS
BEGIN
    -- Sprawdzenie, czy email jest unikalny
    IF EXISTS (SELECT 1 FROM [dbo].[Users] WHERE Email = @Email)
    BEGIN
        RAISERROR ('Podany email już istnieje w bazie danych.', 16, 1);
        RETURN;
    END

    -- Sprawdzenie, czy login jest unikalny

```

```

IF EXISTS (SELECT 1 FROM [dbo].[UsersAccounts] WHERE Login = @Login)
BEGIN
    RAISERROR ('Podany login już istnieje w bazie danych.', 16, 1);
    RETURN;
END

-- Dodanie użytkownika do tabeli Users
INSERT INTO [dbo].[Users] (UserType, FirstName, LastName, Email, RoleID)
VALUES (@UserType, @FirstName, @LastName, @Email, @RoleID);
-- Pobranie UserID dla nowo utworzonego użytkownika
SET @NewUserID = SCOPE_IDENTITY();
-- Dodanie konta użytkownika do tabeli UsersAccounts
INSERT INTO [dbo].[UsersAccounts] (Login, Password, UserID)
VALUES (@Login, @Password, @NewUserID);
END;
GO

```

Przykładowe wywołanie:

```

DECLARE @NewUserID INT;
EXEC AddUser
    @UserType = 'Logged User',
    @FirstName = 'Jessica',
    @LastName = 'Kowalska',
    @Email = 'jess.kowalska@example.com',
    @RoleID = 4,
    @Login = 'jkowalska123',
    @Password = 'StrongpasswordJess123',
    @NewUserID = @NewUserID OUTPUT;

SELECT @NewUserID as IDNowegoUzytkownika;

```

TRIGGERY

1. Trigger sprawdzający dostępność wykładowcy w danym czasie (dla tabeli Modules):

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

    -- Sprawdzamy każdy wstawiony wiersz
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE EXISTS (
            -- Sprawdzamy kolizje w tabeli Modules
            SELECT 1
            FROM [dbo].[Modules] m
            WHERE m.LecturerID = i.LecturerID
            AND m.Date = i.Date
            AND m.StartTime < i.EndTime
            AND m.EndTime > i.StartTime
            AND m.ModuleID <> i.ModuleID
        )
        OR EXISTS (
            -- Sprawdzamy kolizje w tabeli WebinarMeetings
            SELECT 1
            FROM [dbo].[WebinarMeetings] wm
            WHERE wm.LecturerID = i.LecturerID
            AND wm.Date = i.Date
            AND wm.StartTime < i.EndTime
            AND wm.EndTime > i.StartTime
        )
        OR EXISTS (
            -- Sprawdzamy kolizje w tabeli StudyMeetings
            SELECT 1
            FROM [dbo].[StudyMeetings] sm
            WHERE sm.LecturerID = i.LecturerID
            AND sm.Date = i.Date
            AND sm.StartTime < i.EndTime
            AND sm.EndTime > i.StartTime
        )
    )
    BEGIN
        RAISERROR ('Wykładowca jest już przypisany do innego modułu, webinaru, lub
spotkania studyjnego w tym czasie.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END
GO
```

2. Trigger sprawdzający dostępność, a także pojemność sali dla modułów offline:

```

CREATE TRIGGER trg_CheckRoomOccupancyAndCapacity
ON [dbo].[OfflineModules]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @LocationID INT, @ModuleID INT, @Date DATETIME, @StartTime TIME,
    @EndTime TIME, @MaxParticipants INT, @Capacity INT;

    SELECT @ModuleID = i.ModuleID, @LocationID = i.LocationID, @MaxParticipants =
    i.MaxParticipants
    FROM inserted i;

    SELECT @Date = m.Date, @StartTime = m.StartTime, @EndTime = m.EndTime
    FROM [dbo].[Modules] m
    WHERE m.ModuleID = @ModuleID;

    -- Sprawdzenie pojemności sali
    SELECT @Capacity = Capacity
    FROM [dbo].[Locations]
    WHERE LocationID = @LocationID;

    IF @MaxParticipants > @Capacity
    BEGIN
        RAISERROR ('Liczba uczestników przekracza pojemność sali.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END

    -- Sprawdzenie dostępności sali
    IF EXISTS (
        SELECT 1
        FROM [dbo].[OfflineModules] om
        JOIN [dbo].[Modules] m ON om.ModuleID = m.ModuleID
        WHERE om.LocationID = @LocationID
        AND m.Date = @Date
        AND m.StartTime < @EndTime
        AND m.EndTime > @StartTime
        AND om.ModuleID <> @ModuleID
    )
    BEGIN
        RAISERROR ('Sala jest już zajęta w tym czasie.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END
GO

```

3. Trigger sprawdzający dostępność wykładowcy w danym czasie (dla tabeli Webinars):

```

CREATE TRIGGER trg_CheckLecturerAvailabilityForWebinar

```

```

ON [dbo].[WebinarMeetings]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzamy każdy wstawiony wiersz
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE EXISTS (
            -- Sprawdzamy kolizje w tabeli Modules
            SELECT 1
            FROM [dbo].[Modules] m
            WHERE m.LecturerID = i.LecturerID
            AND m.Date = i.Date
            AND m.StartTime < i.EndTime
            AND m.EndTime > i.StartTime
        )
        OR EXISTS (
            -- Sprawdzamy kolizje w tabeli WebinarMeetings
            SELECT 1
            FROM [dbo].[WebinarMeetings] wm
            WHERE wm.LecturerID = i.LecturerID
            AND wm.Date = i.Date
            AND wm.StartTime < i.EndTime
            AND wm.EndTime > i.StartTime
            AND wm.WebinarMeetingID <> i.WebinarMeetingID
        )
        OR EXISTS (
            -- Sprawdzamy kolizje w tabeli StudyMeetings
            SELECT 1
            FROM [dbo].[StudyMeetings] sm
            WHERE sm.LecturerID = i.LecturerID
            AND sm.Date = i.Date
            AND sm.StartTime < i.EndTime
            AND sm.EndTime > i.StartTime
        )
    )
    BEGIN
        RAISERROR ('Wykładowca jest już przypisany do innego modułu, webinaru, lub
spotkania studyjnego w tym czasie.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END
GO

```

4. Trigger sprawdzający dostępność wykładowcy w danym czasie (dla tabeli StudyMeetings):

```

CREATE TRIGGER trg_CheckLecturerAvailabilityForStudyMeeting
ON [dbo].[StudyMeetings]
AFTER INSERT

```

```

AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzamy każdy wstawiony wiersz
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE EXISTS (
            -- Sprawdzamy kolizje w tabeli Modules
            SELECT 1
            FROM [dbo].[Modules] m
            WHERE m.LecturerID = i.LecturerID
            AND m.Date = i.Date
            AND m.StartTime < i.EndTime
            AND m.EndTime > i.StartTime
        )
        OR EXISTS (
            -- Sprawdzamy kolizje w tabeli WebinarMeetings
            SELECT 1
            FROM [dbo].[WebinarMeetings] wm
            WHERE wm.LecturerID = i.LecturerID
            AND wm.Date = i.Date
            AND wm.StartTime < i.EndTime
            AND wm.EndTime > i.StartTime
        )
        OR EXISTS (
            -- Sprawdzamy kolizje w tabeli StudyMeetings
            SELECT 1
            FROM [dbo].[StudyMeetings] sm
            WHERE sm.LecturerID = i.LecturerID
            AND sm.Date = i.Date
            AND sm.StartTime < i.EndTime
            AND sm.EndTime > i.StartTime
            AND sm.StudyMeetingID <> i.StudyMeetingID
        )
    )
    BEGIN
        RAISERROR ('Wykładowca jest już przypisany do innego modułu, webinaru, lub
spotkania studyjnego w tym czasie.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END
GO

```

5. Trigger przypisujący tłumacza do Webinaru:

```
CREATE TRIGGER trg_AssignTranslatorToWebinarMeeting
ON [dbo].[WebinarMeetings]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @WebinarMeetingID INT, @WebinarID INT, @LanguageID INT, @TranslatorID
    INT, @Date DATETIME, @StartTime TIME, @EndTime TIME;

    -- Pobranie danych z wstawionego rekordu
    SELECT @WebinarMeetingID = WebinarMeetingID, @WebinarID = WebinarID, @Date =
    Date, @StartTime = StartTime, @EndTime = EndTime FROM inserted;

    -- Pobranie LanguageID dla webinaru
    SELECT @LanguageID = LanguageID FROM [dbo].[Webinars] WHERE WebinarID =
    @WebinarID;

    -- Sprawdzamy, czy LanguageID jest różne od 1 (domyślnego języka)
    IF @LanguageID IS NOT NULL AND @LanguageID <> 1
    BEGIN
        -- Wyszukanie dostępnego tłumacza
        SELECT TOP 1 @TranslatorID = TranslatorID
        FROM [dbo].[Translators]
        WHERE LanguageID = @LanguageID AND dbo.IsTranslatorAvailable(TranslatorID,
        @Date, @StartTime, @EndTime) = 1;

        -- Jeśli znaleziono dostępnego tłumacza, dodajemy do
        WebinarsInForeignLanguages
        IF @TranslatorID IS NOT NULL
        BEGIN
            INSERT INTO [dbo].[WebinarsInForeignLanguages] (WebinarID, LanguageID,
            TranslatorID)
            VALUES (@WebinarID, @LanguageID, @TranslatorID);
        END
        ELSE
        BEGIN
            PRINT 'Brak dostępnego tłumacza dla wybranego webinaru i języka.';
        END
    END
END;
GO
```

6. Trigger przypisujący tłumacza do Modułu:

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

    DECLARE @ModuleID INT, @CourseLanguageID INT, @Date DATETIME, @StartTime TIME,
    @EndTime TIME, @TranslatorID INT;
    -- Iterujemy po wszystkich wstawionych modułach
    DECLARE cursor_modules CURSOR FOR
    SELECT ModuleID FROM inserted;

    OPEN cursor_modules;
    FETCH NEXT FROM cursor_modules INTO @ModuleID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Pobieranie LanguageID i sprawdzanie, czy kurs jest w języku obcym
        SELECT @CourseLanguageID = c.LanguageID
        FROM [dbo].[Courses] c
        JOIN [dbo].[Modules] m ON c.CourseID = m.CourseID
        WHERE m.ModuleID = @ModuleID AND c.LanguageID <> 1;

        IF @CourseLanguageID IS NOT NULL
        BEGIN
            -- Pobieranie daty i godzin modułu
            SELECT @Date = Date, @StartTime = StartTime, @EndTime = EndTime
            FROM [dbo].[Modules]
            WHERE ModuleID = @ModuleID;

            -- Znalezienie dostępnego tłumacza
            SELECT TOP 1 @TranslatorID = t.TranslatorID
            FROM [dbo].[Translators] t
            WHERE t.LanguageID = @CourseLanguageID
            AND dbo.IsTranslatorAvailable(t.TranslatorID, @Date, @StartTime,
            @EndTime) = 1;

            -- Przypisanie tłumacza do modułu
            IF @TranslatorID IS NOT NULL
            BEGIN
                INSERT INTO [dbo].[ModulesInForeignLanguages] (LanguageID,
                TranslatorID, ModuleID)
                VALUES (@CourseLanguageID, @TranslatorID, @ModuleID);
```

```

        END
    ELSE
    BEGIN
-- Tłumacz nie jest dostępny, czyli należy podjąć odpowiednie działania:
        RAISERROR ('Brak dostępnego tłumacza dla wybranego modułu i
języka.', 16, 1);
    END
END

        FETCH NEXT FROM cursor_modules INTO @ModuleID;
    END

    CLOSE cursor_modules;
    DEALLOCATE cursor_modules;
END;
GO

```

6. Trigger automatycznie uzupełniający tabelę Lecturers:

```

CREATE TRIGGER trg_AutoAddLecturer
ON [dbo].[Users]
AFTER INSERT
AS
BEGIN
-- Sprawdzamy, czy dodany użytkownik ma RoleID = 4
    IF EXISTS (SELECT 1 FROM inserted WHERE RoleID = 4)
    BEGIN
-- Dodajemy rekord do tabeli Lecturers
        INSERT INTO [dbo].[Lecturers] (UserID)
        SELECT UserID FROM inserted WHERE RoleID = 4;
    END
END;
GO

```

INDEKSY

CREATE INDEX IDX_Orders_ProductID ON dbo.Orders(ProductID);

CREATE INDEX IDX_Orders_UserID ON dbo.Orders(UserID);

CREATE INDEX IDX_CourseParticipants_UserID ON dbo.CourseParticipants(UserID);

CREATE INDEX IDX_CourseParticipants_CourseID ON
dbo.CourseParticipants(CourseID);

CREATE INDEX IDX_Modules_CourseID ON dbo.Modules(CourseID);

CREATE INDEX IDX_Modules_LecturerID ON dbo.Modules(LecturerID);

CREATE INDEX IDX_Users_Email ON dbo.Users(Email);

CREATE INDEX IDX_Payments_Status ON dbo.Payments(Status);

CREATE INDEX IDX_Courses_StartDate ON dbo.Courses(StartDate);

CREATE INDEX IDX_Courses_EndDate ON dbo.Courses(EndDate);

CREATE INDEX IDX_Modules_Date ON dbo.Modules(Date);

CREATE INDEX IDX_Orders_UserID_ProductID ON dbo.Orders(UserID, ProductID);

CREATE INDEX IDX_WebinarMeetings_Date_WebinarID ON
dbo.WebinarMeetings(Date, WebinarID);

CREATE INDEX IDX_Payments_OrderID ON dbo.Payments(OrderID);

CREATE INDEX IDX_Payments_PayDate ON dbo.Payments(PayDate);

CREATE INDEX IDX_Orders_Price ON dbo.Orders(Price);

CREATE INDEX IDX_Courses_LanguageID ON dbo.Courses(LanguageID);

CREATE INDEX IDX_WebinarMeetings_LecturerID ON
dbo.WebinarMeetings(LecturerID);


```
CREATE INDEX IDX_Users_FirstName_LastName ON dbo.Users(FirstName,  
LastName);
```

```
CREATE INDEX IDX_CourseParticipants_CityID ON dbo.CourseParticipants(CityID);
```

ROLE ORAZ UPRAWNIENIA

```
CREATE ROLE SystemAdministrator;
```

-- Nadawanie uprawnień dla roli SystemAdministrator

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::dbo TO
SystemAdministrator;

GRANT CREATE TABLE TO SystemAdministrator;

GRANT ALTER TO SystemAdministrator;

GRANT DROP TO SystemAdministrator;

GRANT CREATE VIEW TO SystemAdministrator;

GRANT CREATE PROCEDURE TO SystemAdministrator;

GRANT CREATE FUNCTION TO SystemAdministrator;

GRANT CREATE ROLE TO SystemAdministrator;

GRANT ALTER ANY ROLE TO SystemAdministrator;

GRANT DROP ANY ROLE TO SystemAdministrator;

GRANT ALTER ANY USER TO SystemAdministrator;

CREATE ROLE OfferCoordinator;

-- Nadawanie uprawnień dla roli OfferCoordinator

GRANT SELECT, INSERT, UPDATE ON dbo.Products TO OfferCoordinator;

GRANT SELECT, INSERT, UPDATE ON dbo.Courses TO OfferCoordinator;

GRANT SELECT, INSERT, UPDATE ON dbo.Webinars TO OfferCoordinator;

GRANT DELETE ON dbo.Products TO OfferCoordinator;

GRANT DELETE ON dbo.Courses TO OfferCoordinator;

GRANT DELETE ON dbo.Webinars TO OfferCoordinator;

CREATE ROLE Lecturer;

-- Nadawanie uprawnień dla roli Lecturer

GRANT SELECT ON dbo.Modules TO Lecturer;

GRANT SELECT ON dbo.Courses TO Lecturer;

GRANT SELECT ON dbo.CourseParticipants TO Lecturer;

GRANT UPDATE ON dbo.Modules TO Lecturer;

GRANT UPDATE ON dbo.CourseParticipants TO Lecturer;

GRANT INSERT, UPDATE ON dbo.ModuleAttendances TO Lecturer;

GRANT SELECT, UPDATE ON AttendanceRate TO Lecturer;

GRANT SELECT, UPDATE ON AttendanceRate TO Lecturer;

CREATE ROLE Translator;

-- Nadawanie uprawnień dla roli Translator

GRANT SELECT ON dbo.ModulesInForeignLanguages TO Translator;

GRANT SELECT ON dbo.WebinarsInForeignLanguages TO Translator;

GRANT SELECT ON dbo.SubjectsInForeignLanguages TO Translator;

CREATE ROLE StudyCoordinator;

-- Nadawanie uprawnień dla roli StudyCoordinator

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Studies TO StudyCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Semesters TO StudyCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Exams TO StudyCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Subjects TO StudyCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Reunions TO StudyCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Students TO StudyCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.StudyMeetingAttendances TO StudyCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.StudyMeetings TO StudyCoordinator;

GRANT SELECT, UPDATE ON dbo.CertificatesToSend TO StudyCoordinator;

GRANT SELECT ON dbo.OverlappingMeetings TO StudyCoordinator;

CREATE ROLE WebinarCoordinator;

-- Nadawanie uprawnień dla roli WebinarCoordinator

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Webinars TO WebinarCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.WebinarMeetings TO WebinarCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.WebinarParticipants TO WebinarCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.WebinarAttendances TO WebinarCoordinator;

GRANT SELECT, INSERT, UPDATE ON dbo.WebinarsInForeignLanguages TO WebinarCoordinator;

GRANT SELECT, UPDATE ON dbo.CertificatesToSend TO WebinarCoordinator;

GRANT SELECT ON dbo.OverlappingMeetings TO WebinarCoordinator

CREATE ROLE CourseCoordinator;

-- Nadawanie uprawnień dla roli CourseCoordinator

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Courses TO CourseCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.Modules TO CourseCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.CourseParticipants TO CourseCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.ModuleAttendances TO CourseCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.AsynchronicOnlineModules TO CourseCoordinator;

GRANT SELECT, INSERT, DELETE, UPDATE ON dbo.SynchronicOnlineModules TO CourseCoordinator;

GRANT SELECT, UPDATE ON dbo.CertificatesToSend TO CourseCoordinator;

GRANT SELECT ON dbo.OverlappingMeetings TO CourseCoordinator

CREATE ROLE Accounting;

-- Nadawanie uprawnień dla roli Accounting

GRANT SELECT, INSERT, UPDATE ON dbo.Payments TO Accounting;

GRANT SELECT ON dbo.Orders TO Accounting;

GRANT SELECT ON dbo.Products TO Accounting;

GRANT SELECT ON dbo.IncomeSummary TO Accounting;

GRANT SELECT ON dbo.WebinarsIncome TO Accounting;

GRANT SELECT ON dbo.CoursesIncome TO Accounting;

GRANT SELECT ON dbo.ReunionsIncome TO Accounting;

GRANT SELECT ON dbo.CourseDebtors TO Accounting;

GRANT SELECT ON dbo.Debtors TO Accounting;

GRANT SELECT ON dbo.WebinarDebtors TO Accounting;

GRANT SELECT ON dbo.CourseDebtors TO Accounting;

CREATE ROLE Director;

-- Nadawanie uprawnień dla roli Director

GRANT SELECT, UPDATE, INSERT, DELETE ON ALL TO Director;

GRANT SELECT, UPDATE ON dbo.Users TO Director;

GRANT SELECT, UPDATE ON dbo.Courses TO Director;

GRANT SELECT, UPDATE ON dbo.Studies TO Director;

GRANT SELECT, UPDATE ON dbo.Payments TO Director;

GRANT SELECT, UPDATE ON dbo.Orders TO Director;

GRANT SELECT, UPDATE ON dbo.Products TO Director;

GRANT CREATE ROLE, ALTER ANY ROLE TO Director;

GRANT ALTER ANY USER TO Director;

CASE - dodawanie nowego kursu

1.Dodanie produktu:

Procedura dodania produktu do bazy:

```
CREATE PROCEDURE AddProduct  
    @ProductType NVARCHAR(20),
```

```

        @ProductPrice MONEY,
        @ProductID INT OUTPUT
AS
BEGIN
    IF @ProductType NOT IN ('reunion', 'course', 'webinar')
    BEGIN
        RAISERROR ('Nieprawidłowy typ produktu', 16, 1);
        RETURN;
    END

    INSERT INTO [dbo].[Products] (ProductType, ProductPrice)
    VALUES (@ProductType, @ProductPrice);

    SET @ProductID = SCOPE_IDENTITY();
END

```

Przykładowe wywołanie procedury:

```

DECLARE @NewProductID INT; -- Zmienna dla ID produktu
-- Wywołanie procedury z bezpośrednim przekazaniem wartości:
EXEC AddProduct @ProductType = 'course', @ProductPrice = 100.00, @ProductID =
@NewProductID OUTPUT;
-- Wyświetlenie ID nowo dodanego produktu:
SELECT @NewProductID as NewProductId;

```

2. Dodanie kursu

Procedura dodania kursu do bazy:

```

CREATE PROCEDURE AddCourse
    @ProductID INT,
    @CourseName NVARCHAR(50),
    @StartDate DATETIME,
    @EndDate DATETIME,
    @Description NVARCHAR(MAX),
    @LanguageID INT
AS
BEGIN
    -- Sprawdzenie, czy data zakończenia jest późniejsza niż data rozpoczęcia kursu
    IF @EndDate <= @StartDate
    BEGIN
        RAISERROR ('Data zakończenia musi być późniejsza niż data rozpoczęcia.', 16,
1);
        RETURN;
    END

    -- Sprawdzenie, czy istnieje ProductID
    IF NOT EXISTS (SELECT 1 FROM [dbo].[Products] WHERE [ProductID] = @ProductID)
    BEGIN

```



```

        RAISERROR ('ProductID nie istnieje.', 16, 1);
    RETURN;
END
-- Wstawienie nowego kursu do tabeli Courses
INSERT INTO [dbo].[Courses] (ProductID, CourseName, StartDate, EndDate,
Description, LanguageID)
VALUES (@ProductID, @CourseName, @StartDate, @EndDate, @Description,
@LanguageID);
END

```

Przykładowe wywołanie:

```

EXEC AddCourse
    @ProductID = 96,
    @CourseName = 'Programowanie w Javie',
    @StartDate = '2024-10-01',
    @EndDate = '2024-10-31',
    @Description = 'Kurs programowania w języku Java dla początkujących.',
    @LanguageID = '2';

```

3. Dodanie modułów do kursu:

Procedura ta uzupełnia tabelę Modules, a także w zależności od typu kursu, uzupełnia tabelę AsynchronousOnlineModules, SynchronousOnlineModules lub OfflineModules:

```

CREATE PROCEDURE AddModuleNew
    @CourseID INT,
    @LecturerID INT,
    @ModuleName NVARCHAR(50),
    @ModuleType NVARCHAR(20),
    @Date DATETIME,
    @StartTime TIME,
    @EndTime TIME,
    @LocationID INT = NULL,           -- Dla modułów offline
    @MaxParticipants INT = NULL,     -- Dla modułów offline
    @MeetingLink NVARCHAR(255) = NULL, -- Dla synchronicznych modułów online
    @RecordingLink NVARCHAR(255) = NULL, -- Dla asynchronicznych modułów online
    @ExpirationDate DATETIME = NULL  -- Dla asynchronicznych modułów online
AS
BEGIN
    -- Dodajemy podstawowe informacje o module
    INSERT INTO [dbo].[Modules] (CourseID, LecturerID, ModuleName, ModuleType,
Date, StartTime, EndTime)
VALUES (@CourseID, @LecturerID, @ModuleName, @ModuleType, @Date, @StartTime,
@EndTime);

    DECLARE @ModuleID INT = SCOPE_IDENTITY();

    -- Logika dla różnych typów modułów
    IF @ModuleType = 'asynchronous online'
    BEGIN

```

```

        INSERT INTO [dbo].[AsynchronousOnlineModules] (ModuleID, RecordingLink,
ExpirationDate)
        VALUES (@ModuleID, @RecordingLink, @ExpirationDate);
    END
    ELSE IF @ModuleType = 'synchronic online'
    BEGIN
        INSERT INTO [dbo].[SynchronicOnlineModules] (ModuleID, MeetingLink)
        VALUES (@ModuleID, @MeetingLink);
    END
    ELSE IF @ModuleType = 'stationary'
    BEGIN
        INSERT INTO [dbo].[OfflineModules] (ModuleID, MaxParticipants, LocationID)
        VALUES (@ModuleID, @MaxParticipants, @LocationID);
    END
END

```

Przykładowe wywołanie:

```

EXEC AddModuleNew
    @CourseID = 69, -- ID kursu
    @LecturerID = 20, -- ID wykładowcy
    @ModuleName = 'JavaFX', -- Nazwa modułu
    @ModuleType = 'stationary', -- Typ modułu ('stationary' dla modułu
stacjonarnego)
    @Date = '2024-10-01', -- Data modułu
    @StartTime = '09:00:00', -- Czas rozpoczęcia
    @EndTime = '12:00:00', -- Czas zakończenia
    @LocationID = 31, -- ID lokalizacji
    @MaxParticipants = 30 -- Maksymalna liczba uczestników
    -- Pozostałe parametry mogą być pominięte lub ustawione na NULL

```

TRIGGER:

Trigger sprawdzający dostępność wykładowcy w danym czasie:

```

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

    -- Sprawdzamy każdy wstawiony wiersz
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE EXISTS (
            -- Sprawdzamy kolizje w tabeli Modules
            SELECT 1
            FROM [dbo].[Modules] m
            WHERE m.LecturerID = i.LecturerID
            AND m.Date = i.Date

```

```

        AND m.StartTime < i.EndTime
        AND m.EndTime > i.StartTime
        AND m.ModuleID <> i.ModuleID
    )
    OR EXISTS (
        -- Sprawdzamy kolizje w tabeli WebinarMeetings
        SELECT 1
        FROM [dbo].[WebinarMeetings] wm
        WHERE wm.LecturerID = i.LecturerID
        AND wm.Date = i.Date
        AND wm.StartTime < i.EndTime
        AND wm.EndTime > i.StartTime
    )
    OR EXISTS (
        -- Sprawdzamy kolizje w tabeli StudyMeetings
        SELECT 1
        FROM [dbo].[StudyMeetings] sm
        WHERE sm.LecturerID = i.LecturerID
        AND sm.Date = i.Date
        AND sm.StartTime < i.EndTime
        AND sm.EndTime > i.StartTime
    )
)
BEGIN
    RAISERROR ('Wykładowca jest już przypisany do innego modułu, webinaru, lub
spotkania studyjnego w tym czasie.', 16, 1);
    ROLLBACK TRANSACTION;
END
END
GO

```

Trigger sprawdzający dostępność, a także pojemność sali dla modułów offline:

```

CREATE TRIGGER trg_CheckRoomOccupancyAndCapacity
ON [dbo].[OfflineModules]
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @LocationID INT, @ModuleID INT, @Date DATETIME, @StartTime TIME,
@EndTime TIME, @MaxParticipants INT, @Capacity INT;

    SELECT @ModuleID = i.ModuleID, @LocationID = i.LocationID, @MaxParticipants =
i.MaxParticipants
    FROM inserted i;

    SELECT @Date = m.Date, @StartTime = m.StartTime, @EndTime = m.EndTime
    FROM [dbo].[Modules] m
    WHERE m.ModuleID = @ModuleID;

    -- Sprawdzenie pojemności sali
    SELECT @Capacity = Capacity
    FROM [dbo].[Locations]

```

```

WHERE LocationID = @LocationID;

IF @MaxParticipants > @Capacity
BEGIN
    RAISERROR ('Liczba uczestników przekracza pojemność sali.', 16, 1);
    ROLLBACK TRANSACTION;
    RETURN;
END

-- Sprawdzenie dostępności sali
IF EXISTS (
    SELECT 1
    FROM [dbo].[OfflineModules] om
    JOIN [dbo].[Modules] m ON om.ModuleID = m.ModuleID
    WHERE om.LocationID = @LocationID
    AND m.Date = @Date
    AND m.StartTime < @EndTime
    AND m.EndTime > @StartTime
    AND om.ModuleID <> @ModuleID
)
BEGIN
    RAISERROR ('Sala jest już zajęta w tym czasie.', 16, 1);
    ROLLBACK TRANSACTION;
END
END
GO

```

Trigger uzupełniający tabelę ModulesInForeignLanguages, jeżeli kurs jest w języku innym niż polski, po dodaniu do niego modułów automatycznie uzupełniana jest tabela ModulesInForeignLanguages. Przydzielany jest dostępny w danym czasie tłumacz.

```

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

    DECLARE @ModuleID INT, @CourseLanguageID INT, @Date DATETIME, @StartTime TIME,
    @EndTime TIME, @TranslatorID INT;
    -- Iterujemy po wszystkich wstawionych modułach
    DECLARE cursor_modules CURSOR FOR
    SELECT ModuleID FROM inserted;

    OPEN cursor_modules;
    FETCH NEXT FROM cursor_modules INTO @ModuleID;

    WHILE @@FETCH_STATUS = 0

```

```

BEGIN
-- Pobieranie LanguageID i sprawdzanie, czy kurs jest w języku obcym
SELECT @CourseLanguageID = c.LanguageID
FROM [dbo].[Courses] c
JOIN [dbo].[Modules] m ON c.CourseID = m.CourseID
WHERE m.ModuleID = @ModuleID AND c.LanguageID <> 1;

IF @CourseLanguageID IS NOT NULL
BEGIN
-- Pobieranie daty i godzin modułu
SELECT @Date = Date, @StartTime = StartTime, @EndTime = EndTime
FROM [dbo].[Modules]
WHERE ModuleID = @ModuleID;
-- Znalezienie dostępnego tłumacza
SELECT TOP 1 @TranslatorID = t.TranslatorID
FROM [dbo].[Translators] t
WHERE t.LanguageID = @CourseLanguageID
AND dbo.IsTranslatorAvailable(t.TranslatorID, @Date, @StartTime,
@EndTime) = 1;
-- Przypisanie tłumacza do modułu
IF @TranslatorID IS NOT NULL
BEGIN
INSERT INTO [dbo].[ModulesInForeignLanguages] (LanguageID,
TranslatorID, ModuleID)
VALUES (@CourseLanguageID, @TranslatorID, @ModuleID);
END
ELSE
BEGIN
-- Tłumacz nie jest dostępny, czyli należy podjąć odpowiednie działania:
RAISERROR ('Brak dostępnego tłumacza dla wybranego modułu i
języka.', 16, 1);
END
END

FETCH NEXT FROM cursor_modules INTO @ModuleID;
END

CLOSE cursor_modules;
DEALLOCATE cursor_modules;
END;
GO

```

Trigger działa z wykorzystaniem funkcji:

```

CREATE FUNCTION IsTranslatorAvailable
(

```

```

    @TranslatorID INT,
    @DesiredDate DATETIME,
    @DesiredStartTime TIME,
    @DesiredEndTime TIME
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsAvailable BIT = 1;
    -- Sprawdzanie dostępności w StudyMeetings
    IF EXISTS (
        SELECT 1 FROM [dbo].[StudyMeetings] sm
        JOIN [dbo].[SubjectsInForeignLanguages] sfl ON sm.SubjectID = sfl.SubjectID
        WHERE sfl.TranslatorID = @TranslatorID
        AND sm.Date = CAST(@DesiredDate AS DATE)
        AND (sm.StartTime < @DesiredEndTime AND sm.EndTime > @DesiredStartTime)
    )
    BEGIN
        SET @IsAvailable = 0;
    END

    -- Sprawdzanie dostępności w WebinarMeetings
    IF EXISTS (
        SELECT 1 FROM [dbo].[WebinarMeetings] wm
        JOIN [dbo].[WebinarsInForeignLanguages] wfl ON wm.WebinarID = wfl.WebinarID
        WHERE wfl.TranslatorID = @TranslatorID
        AND wm.Date = CAST(@DesiredDate AS DATE)
        AND (wm.StartTime < @DesiredEndTime AND wm.EndTime > @DesiredStartTime)
    )
    BEGIN
        SET @IsAvailable = 0;
    END

    -- Sprawdzanie dostępności w Modules
    IF EXISTS (
        SELECT 1 FROM [dbo].[Modules] m
        JOIN [dbo].[ModulesInForeignLanguages] mfl ON m.ModuleID = mfl.ModuleID
        WHERE mfl.TranslatorID = @TranslatorID
        AND m.Date = CAST(@DesiredDate AS DATE)
        AND (m.StartTime < @DesiredEndTime AND m.EndTime > @DesiredStartTime)
    )
    BEGIN
        SET @IsAvailable = 0;
    END

    RETURN @IsAvailable;
END

```

--dodawanie produktu

```
CREATE PROCEDURE AddProduct
    @ProductType NVARCHAR(20),
    @ProductPrice MONEY,
    @ProductID INT OUTPUT
AS
BEGIN
    IF @ProductType NOT IN ('reunion', 'course', 'webinar')
```

```

BEGIN
    RAISERROR ('Nieprawidłowy typ produktu', 16, 1);
    RETURN;
END

INSERT INTO [dbo].[Products] (ProductType, ProductPrice)
VALUES (@ProductType, @ProductPrice);

SET @ProductID = SCOPE_IDENTITY();
END

DECLARE @NewProductID INT;
EXEC AddProduct @ProductType = 'course', @ProductPrice = 50, @ProductID =
@NewProductID OUTPUT;

--dodawanie kursu

CREATE PROCEDURE AddCourse
    @ProductID INT,
    @CourseName NVARCHAR(50),
    @StartDate DATETIME,
    @EndDate DATETIME,
    @Description NVARCHAR(MAX)
AS
BEGIN
    IF @EndDate <= @StartDate
    BEGIN
        RAISERROR ('Data zakończenia musi być późniejsza niż data rozpoczęcia.', 16,
1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM [dbo].[Products] WHERE [ProductID] =
@ProductID)
    BEGIN
        RAISERROR ('ProductID nie istnieje.', 16, 1);
        RETURN;
    END

```



```
INSERT INTO [dbo].[Courses] (ProductID, CourseName, StartDate, EndDate,
Description)
VALUES (@ProductID, @CourseName, @StartDate, @EndDate, @Description);
END;
```

EXEC AddCourse

```
@ProductID = 93,
@CourseName = 'Data Science Essentials',
@StartDate = '2024-02-01',
@EndDate = '2024-03-01',
@Description = 'Explore the exciting world of Data Science with our Data Science
Essentials course.
```

This concise program is designed to introduce you to the core concepts of data analysis, visualization, and basic machine learning. Learn the fundamentals of Python programming, data manipulation using Pandas, and insightful data visualization with Matplotlib. Perfect for beginners aiming to get a solid foundation in Data Science.';

--dodawanie modułów

CREATE PROCEDURE AddModule

```
@CourseID INT,
@LecturerID INT,
@ModuleName NVARCHAR(50),
@ModuleType NVARCHAR(20),
@Date DATETIME,
@StartTime TIME(7),
@EndTime TIME(7)
```

AS

BEGIN

```
IF @ModuleType NOT IN ('hybrid', 'stationary', 'synchronic online', 'asynchronic
online')
```

BEGIN

```
RAISERROR ('Nieprawidłowy typ modułu.', 16, 1);
```

```
RETURN;
```

END

```
IF @StartTime >= @EndTime
```

BEGIN

```
        RAISERROR ('Czas rozpoczęcia musi być wcześniejszy niż czas zakończenia.',  
16, 1);  
    RETURN;  
END
```

```
INSERT INTO [dbo].[Modules] (CourseID, LecturerID, ModuleName, ModuleType,  
Date, StartTime, EndTime)  
VALUES (@CourseID, @LecturerID, @ModuleName, @ModuleType, @Date,  
@StartTime, @EndTime);  
END;
```

--3 rodzaje modułów

```
EXEC AddModule  
    @CourseID = 65,  
    @LecturerID = 2,  
    @ModuleName = 'Introduction to Data Science',  
    @ModuleType = 'stationary',  
    @Date = '2024-02-02',  
    @StartTime = '09:00:00',  
    @EndTime = '12:00:00';
```

```
EXEC AddModule  
    @CourseID = 65,  
    @LecturerID = 4,  
    @ModuleName = 'Python Fundamentals for Data Science',  
    @ModuleType = 'synchronic online',  
    @Date = '2024-02-10',  
    @StartTime = '10:00:00',  
    @EndTime = '12:00:00';
```

```
EXEC AddModule  
    @CourseID = 65,  
    @LecturerID = 4,  
    @ModuleName = 'Data Exploration with Pandas',  
    @ModuleType = 'asynchronic online',  
    @Date = '2024-02-12',  
    @StartTime = '18:00:00',  
    @EndTime = '21:00:00';
```

--dodawanie modułów w obcych językach

```
CREATE PROCEDURE AddModuleInForeignLanguage
    @LanguageID INT,
    @TranslatorID INT,
    @ModuleID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM [dbo].[Languages] WHERE [LanguageID] =
@LanguageID)
    BEGIN
        RAISERROR ('LanguageID nie istnieje.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM [dbo].[Languages] WHERE [LanguageID] =
@TranslatorID)
    BEGIN
        RAISERROR ('TranslatorID nie istnieje.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM [dbo].[Modules] WHERE [ModuleID] =
@ModuleID)
    BEGIN
        RAISERROR ('ModuleID nie istnieje.', 16, 1);
        RETURN;
    END

    INSERT INTO [dbo].[ModulesInForeignLanguages] (LanguageID, TranslatorID,
ModuleID)
    VALUES (@LanguageID, @TranslatorID, @ModuleID);
END
```

```
EXEC AddModuleInForeignLanguage
    @LanguageID = 2,
    @TranslatorID = 2,
    @ModuleID = 186;
```

--dodawanie lokalizacji

```
CREATE PROCEDURE AddLocation
```

```
    @CityID INT,  
    @Street NVARCHAR(30),  
    @Building INT,  
    @Classroom INT,  
    @PostalCode NVARCHAR(10)
```

```
AS
```

```
BEGIN
```

```
    IF NOT EXISTS (SELECT 1 FROM [dbo].[Cities] WHERE [CityID] = @CityID)
```

```
    BEGIN
```

```
        RAISERROR ('CityID nie istnieje.', 16, 1);
```

```
        RETURN;
```

```
    END
```

```
    INSERT INTO [dbo].[Locations] (CityID, Street, Building, Classroom, PostalCode)
```

```
    VALUES (@CityID, @Street, @Building, @Classroom, @PostalCode);
```

```
END;
```

```
EXEC AddLocation
```

```
    @CityID = 3,  
    @Street = 'Kawiory',  
    @Building = 11,  
    @Classroom = 413,  
    @PostalCode = '30-072';
```

**** poprawiłam procedurę, uzupełniłam o Capacity

```
CREATE PROCEDURE AddLocation
```

```
    @CityID INT,  
    @Street NVARCHAR(30),  
    @Building INT,  
    @Classroom INT,  
    @PostalCode NVARCHAR(10),  
    @Capacity INT
```

```
AS
```

```
BEGIN
```

```
    IF NOT EXISTS (SELECT 1 FROM [dbo].[Cities] WHERE [CityID] = @CityID)
```

```
    BEGIN
```

```

        RAISERROR ('CityID nie istnieje.', 16, 1);
    RETURN;
END

INSERT INTO [dbo].[Locations] (CityID, Street, Building, Classroom, PostalCode,
Capacity)
VALUES (@CityID, @Street, @Building, @Classroom, @PostalCode, @Capacity);
END;

EXEC AddLocation
    @CityID = 3,
    @Street = 'Kawiorz',
    @Building = 12,
    @Classroom = 313,
    @PostalCode = '30-072',
    @Capacity = 60;

```

--dodawanie modułu offline

```

CREATE PROCEDURE AddOfflineModule
    @ModuleID INT,
    @MaxParticipants INT,
    @LocationID INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM [dbo].[Modules] WHERE [ModuleID] =
@ModuleID)
    BEGIN
        RAISERROR ('ModuleID nie istnieje.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM [dbo].[Locations] WHERE [LocationID] =
@LocationID)
    BEGIN
        RAISERROR ('LocationID nie istnieje.', 16, 1);
        RETURN;
    END

```

```
INSERT INTO [dbo].[OfflineModules] (ModuleID, MaxParticipants, LocationID)
VALUES (@ModuleID, @MaxParticipants, @LocationID);
END;
```

```
EXEC AddOfflineModule
  @ModuleID = 185,
  @MaxParticipants = 20,
  @LocationID = 31;
```

--dodawanie modułu synchronicznego

```
CREATE PROCEDURE AddSynchronicOnlineModule
  @ModuleID INT,
  @MeetingLink NVARCHAR(255)
AS
BEGIN
  IF NOT @MeetingLink LIKE 'https://%.com'
  BEGIN
    RAISERROR ('MeetingLink nie spełnia wymagań formatu.', 16, 1);
    RETURN;
  END

  IF NOT EXISTS (SELECT 1 FROM [dbo].[Modules] WHERE [ModuleID] =
@ModuleID)
  BEGIN
    RAISERROR ('ModuleID nie istnieje.', 16, 1);
    RETURN;
  END

  INSERT INTO [dbo].[SynchronicOnlineModules] (ModuleID, MeetingLink)
  VALUES (@ModuleID, @MeetingLink);
END;
```

```
SELECT * FROM SynchronicOnlineModules
```

```
EXEC AddSynchronicOnlineModule
  @ModuleID = 186,
  @MeetingLink = 'https://examplemeeting.com';
```

--dodawanie asynchronicznego modułu

```

CREATE PROCEDURE AddAsynchronousOnlineModule
    @ModuleID INT,
    @RecordingLink NVARCHAR(255),
    @ExpirationDate DATETIME
AS
BEGIN
    IF NOT @RecordingLink LIKE 'https://%.com'
    BEGIN
        RAISERROR ('RecordingLink nie spełnia wymagań formatu.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM [dbo].[Modules] WHERE [ModuleID] =
@ModuleID)
    BEGIN
        RAISERROR ('ModuleID nie istnieje.', 16, 1);
        RETURN;
    END

    INSERT INTO [dbo].[AsynchronousOnlineModules] (ModuleID, RecordingLink,
ExpirationDate)
    VALUES (@ModuleID, @RecordingLink, @ExpirationDate);
END;

EXEC AddAsynchronousOnlineModule
    @ModuleID = 187,
    @RecordingLink = 'https://examplerecording.com',
    @ExpirationDate = '2024-02-12';

```

--aktualizowanie linku do synchronicznego modułu

```

CREATE PROCEDURE UpdateMeetingLinkToSynchronicOnlineModule
    @SynchronicOnlineModuleID INT,
    @NewMeetingLink NVARCHAR(255)
AS
BEGIN
    IF NOT @NewMeetingLink LIKE 'https://%.com'
    BEGIN
        RAISERROR ('MeetingLink nie spełnia wymagań formatu.', 16, 1);
        RETURN;
    END

```

```

IF NOT EXISTS (SELECT 1 FROM [dbo].[SynchronicOnlineModules] WHERE
[SynchronicOnlineModuleID] = @SynchronicOnlineModuleID)
BEGIN
    RAISERROR ('SynchronicOnlineModuleID nie istnieje.', 16, 1);
    RETURN;
END

UPDATE [dbo].[SynchronicOnlineModules]
SET [MeetingLink] = @NewMeetingLink
WHERE [SynchronicOnlineModuleID] = @SynchronicOnlineModuleID;
END;

```

```

EXEC UpdateMeetingLinkToSynchronicOnlineModule
    @SynchronicOnlineModuleID = 71,
    @NewMeetingLink = 'https://newexamplemeeting.com';

```

--updatowanie linku do synchronicznego modułu

```

CREATE PROCEDURE UpdateAsynchronicOnlineModule
    @AsynchronicOnlineModuleID INT,
    @NewRecordingLink NVARCHAR(255),
    @NewExpirationDate DATETIME
AS
BEGIN
    IF NOT @NewRecordingLink LIKE 'https://%.com'
    BEGIN
        RAISERROR ('RecordingLink nie spełnia wymagań formatu.', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM [dbo].[AsynchronicOnlineModules] WHERE
[AsynchronicOnlineModuleID] = @AsynchronicOnlineModuleID)
    BEGIN
        RAISERROR ('AsynchronicOnlineModuleID nie istnieje.', 16, 1);
        RETURN;
    END

    UPDATE [dbo].[AsynchronicOnlineModules]
    SET [RecordingLink] = @NewRecordingLink,
        [ExpirationDate] = @NewExpirationDate

```



```
WHERE [AsynchronousOnlineModuleID] = @AsynchronousOnlineModuleID;  
END;  
GO
```

```
SELECT * FROM AsynchronousOnlineModules
```

```
EXEC UpdateAsynchronousOnlineModule  
    @AsynchronousOnlineModuleID = 21,  
    @NewRecordingLink = 'https://updatedexamplerecording.com',  
    @NewExpirationDate = '2025-12-31';
```