Podstawy baz danych Projekt

Bernard Gawor, Olgierd Smyka, Stas Kochevenko

1. **Funkcjonalność systemu**

Użytkownicy

Klient bez konta:

- Wyświetlanie oferowanych usług
- Utworzenie konta indywidualnego bądź firmowego

Student (klient z kontem):

- Wyświetlanie oferowanych usług
- Możliwość zapisu na bezpłatne lub płatne programy edukacyjne oraz pojedyncze zajęcia na wybranych studiach poprzez uiszczenie opłaty
- Wyświetlanie wszystkich programów edukacyjnych, na które jest zapisany oraz szczegółów ich dotyczących
- Dostęp do materiałów dydaktycznych w kursach i na studiach
- Wyświetlanie harmonogramu zajęć, na które jest zapisany

Pracownik (nauczyciel/translator):

- Wyświetlanie wszystkich programów edukacyjnych, które prowadzi oraz szczegółów ich dotyczących
- Dostęp do materiałów dydaktycznych programów edukacyjnych oraz możliwość ich edycji
- Możliwość zatwierdzenia listy obecności studentów na zajęciach
- Możliwość wystawiania ocen z zajęć oraz egzaminów studentów w ramach studiów (tylko dla nauczyciela)
- Możliwość zatwierdzenia odrobienia nieobecności studenta

Pracownik systemowy:

- Zarządzanie harmonogramem zajęć, prowadzonych w ramach programów edukacyjnych
- Aktualizacja oferowanych usług
- Możliwość tworzenia raportów finansowych
- Dostęp do danych statystycznych
- Możliwość zmiany danych studenta
- Edycja listy pracowników

Dyrektor szkoły:

- Ma uprawnienia pracownika systemowego oraz ma dostęp do takich funkcji:
- Zmiana dostępu użytkownika do programu edukacyjnego

System

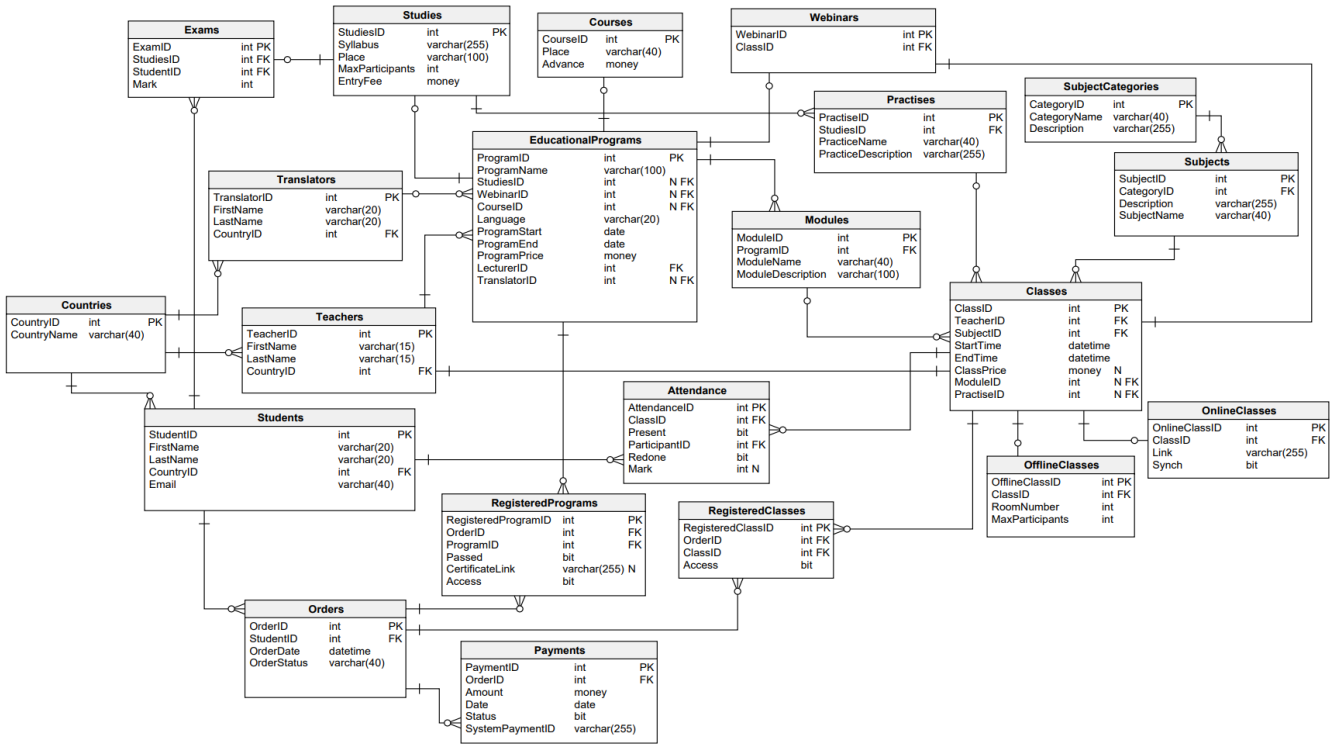
- Raporty finansowe – zestawienie przychodów dla każdego webinaru/kursu/studium
- Lista „dłużników" – osoby, które skorzystały z usług, ale nie uiściły opłat.
- Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia (z informacją, czy wydarzenie jest stacjonarnie, czy zdalnie)
- Ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach
- Lista obecności dla każdego szkolenia z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie
- Raport bilokacji: lista osób, które są zapisane na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo
- Przegląd listy studentów zapisanych z zewnątrz
- Przegląd szczegółów dotyczących konkretnych programów edukacyjnych
- Automatyczna aktualizacja dostępu do zakupionych programów edukacyjnych oraz pojedynczych zajęć po udanej transakcji
- Weryfikacja możliwości zmiany dostępu do programu edukacyjnego albo pojedynczych zajęć
- Automatyczna zmiana statusu zaliczenia programu edukacyjnego po otrzymaniu pozytywnej oceny z egzaminu

Specyfikacje

- Kursy i studia mogą odbywać się: online, stacjonarnie, hybrydowo
- Stacjonarne zajęcia kursów i studiów posiadają limit miejsc
- Webinary udostępniane są uczestnikom na okres 30 dni
- Zaliczenie kursu wymaga zaliczenia min. 80% modułów
- W przypadku studiów wymagane jest zaliczenie praktyk oraz frekwencja na poziomie minimum 80%, przy czym nieobecności mogą zostać odrobione poprzez uczestnictwo w zajęciach lub kursie komercyjnym o zbliżonej tematyce

- Tematyka programów studiów nie może być modyfikowana po ich rozpoczęciu
- Praktyki trwają 14 dni – wymagana jest tu 100% frekwencja
- Możliwość zapisania się na pojedyncze spotkania bez konieczności udziału w całym studium, przy tym cena jest inna
- Administrator ma możliwość zapisu klientów na nieopłacone szkolenia // W sytuacjach wyjątkowych
- Uczestnictwo w kursie wymaga wpłacenia zaliczki przy zapisie, oraz dopłaty całości kwoty najpóźniej 3 dni przed rozpoczęciem kursu
- Uczestnictwo w studium wymaga uiszczenia wpisowego oraz uiszczenia opłaty za dany zjazd najpóźniej 3 dni przed jego rozpoczęciem
- Szkolenia mogą być prowadzone w różnych ustalonych językach
- Wszystkie zajęcia online odbywają się na zewnętrznej platformie chmurowej
- System płatności jest dostarczany przez zewnętrzną firmę

2. Schemat Bazy Danych



Oferowane przez firmę usługi (różnego rodzaju kursy i szkolenia) łączy EducationalPrograms. Każdy rekord przedstawia albo studia (Studies), albo kurs (Courses) albo webinar (Webinars). Spis wszystkich poszczególnych zajęć (spotkań) znajduje się w tabeli Classes. Spotkania mogą być stacjonarne (OfflineClasses) lub niestacjonarne (OnlineClasses). Kursy (Courses) składają się z modułów (Modules). Pojedyncze zajęcia tych modułów mogą być prowadzone stacjonarnie lub niestacjonarnie. Studia podobnie do kursów składają się z modułów (Modules), oraz posiadają praktyki (Practises).

Studenci mogą składać zamówienia (Orders) i przeglądać listę programów (RegisteredPrograms) oraz pojedynczych spotkań (RegisteredClasses), na które są zapisane.

```
-- Table:  Translators
CREATE TABLE  Translators (
   TranslatorID int  NOT NULL,
   FirstName varchar(20)  NOT NULL,
   LastName varchar(20)  NOT NULL,
   CountryID int  NOT NULL,
   CONSTRAINT Translator_pk PRIMARY KEY  (TranslatorID)
);
```

```
-- Table: Attendance
-- Zawiera informacje dotyczące obecności konkretnych studentów z tabeli Students na zajęciach z tabeli Classes
CREATE TABLE Attendance (
    AttendanceID int  NOT NULL,
    ClassID int  NOT NULL,
    Present bit  NOT NULL DEFAULT 0,
    ParticipantID int  NOT NULL,
    Redone bit  NOT NULL DEFAULT 0,
    CONSTRAINT Attendance_pk PRIMARY KEY  (AttendanceID)
);


-- Table: Classes
-- Pojedyncze spotkanie w ramach programu edukacyjnego (albo konkretnego modułu w przypadku kursów lub studiów),
może być w formacie online lub offline
CREATE TABLE Classes (
    ClassID int  NOT NULL,
    TeacherID int  NOT NULL,
    SubjectID int  NOT NULL,
    StartTime datetime  NOT NULL,
    EndTime datetime  NOT NULL,
    ClassPrice money  NULL,
    ModuleID int  NULL,
    PractiseID int  NULL,
    CHECK (EndTime > StartTime),
    CHECK (ClassPrice >= 0),
    CONSTRAINT Classes_pk PRIMARY KEY  (ClassID)
);


-- Table: Countries
CREATE TABLE Countries (
    CountryID int  NOT NULL,
    CountryName int  NOT NULL UNIQUE,
    CONSTRAINT Countries_pk PRIMARY KEY  (CountryID)
);


-- Table: Courses
CREATE TABLE Courses (
    CourseID int  NOT NULL,
    Place varchar(40)  NOT NULL,
    Advance money  NOT NULL,
    CHECK (Advance >= 0),
    CONSTRAINT Courses_pk PRIMARY KEY  (CourseID)
);
```

```
-- Table: EducationalPrograms
-- Zawiera szczegóły konkretnego programu edukacyjnego, którym mogą być studia z tabeli Studies, kursy z tabeli
Courses lub Webinary z tabeli Webinars, w każdym rekorcie tylko jedna z trzech wartości: StudiesID, WebinarID,
CourseID nie jest NULL-em.
CREATE TABLE EducationalPrograms (
   ProgramID int  NOT NULL,
   ProgramName varchar(100)  NOT NULL UNIQUE,
   StudiesID int  NULL,
   WebinarID int  NULL,
   CourseID int  NULL,
   Language varchar(20)  NOT NULL,
   ProgramStart date  NOT NULL,
   ProgramEnd date  NOT NULL,
   ProgramPrice money  NOT NULL,
   LecturerID int  NOT NULL DEFAULT 'Polish',
   TranslatorID int  NULL,
   CHECK (ProgramEnd > ProgramStart),
   CHECK (ProgramPrice >= 0),
   CONSTRAINT EducationalPrograms_pk PRIMARY KEY  (ProgramID)
);


-- Table: Exams
-- Zawiera wyniki z egzaminów dla studentów (Tabela Students) zapisanych na studia(Tabela Studies)
CREATE TABLE Exams (
   ExamID int  NOT NULL,
   StudiesID int  NOT NULL,
   StudentID int  NOT NULL,
   Mark int  NOT NULL DEFAULT 0,
   CHECK(Mark >= 0 AND Mark <= 100),
   CONSTRAINT Exams_pk PRIMARY KEY  (ExamID)
);
```

```
-- Table: Modules
-- Zbiór zajęć na określony temat, nie tożsamy z pojęciem przedmiotu (jeden moduł może zawierać zajęcia z różnych
przedmiotów). Pozwalają na łączenie zajęć różnej formy kształcenia (stacjonarne, online asynchroniczne, online
synchroniczne, hybrydowe).
-- Dla przykładu:
-- Moduł “Programowanie w matematyce” mógłby obejmować szereg zajęć z przedmiotów matematycznych, na których
problemy rozwiązywane są przy pomocy pisanego kodu
CREATE TABLE Modules (
   ModuleID int  NOT NULL,
   ProgramID int  NOT NULL,
   ModuleName varchar(40)  NOT NULL,
   ModuleDescription varchar(100)  NOT NULL,
   CONSTRAINT Modules_pk PRIMARY KEY  (ModuleID)
);


-- Table: OfflineClasses
-- Podzbiór Classes: pojedyncze zajęcia, prowadzone w trybie offline (stacjonarnie), zawsze są podporządkowane
jednemu modułu zajęć.
CREATE TABLE OfflineClasses (
   OfflineClassID int  NOT NULL,
   ClassID int  NOT NULL,
   RoomNumber int  NOT NULL,
   MaxParticipants int  NOT NULL DEFAULT 0,
   Mark int  NULL,
   CHECK(Mark >= 0 AND Mark <= 100),
   CONSTRAINT OfflineClasses_pk PRIMARY KEY  (OfflineClassID)
);


-- Table: OnlineClasses
-- Podzbiór Classes: pojedyncze zajęcia, prowadzone w trybie online. Obejmują synchroniczne i asynchroniczne
moduły.
CREATE TABLE OnlineClasses (
   OnlineClassID int  NOT NULL,
   ClassID int  NOT NULL,
   Link varchar(255)  NOT NULL,
   Synch bit  NOT NULL DEFAULT 0,
   CONSTRAINT OnlineClasses_pk PRIMARY KEY  (OnlineClassID)
);
```

```
-- Table: Orders
-- Lista zamówień przez Studentów. Informacja o zakupionych programach oraz pojedynczych spotkaniach znajduje się
w tabelach RegisteredPrograms i RegisteredClasses odpowiednio.
CREATE TABLE Orders (
   OrderID int  NOT NULL,
   StudentID int  NOT NULL,
   OrderDate datetime  NOT NULL DEFAULT GETDATE(),
   OrderStatus varchar(40) NOT NULL DEFAULT 'NOT PAID',
   CHECK(OrderStatus IN ('NOT PAID', 'ENTRY PAID', 'FULL PAID'))
   CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);


-- Table: Payments
-- Spis płatność dokonanych w celu częściowego lub całkowitego opłacenia zamówienia z tabeli Orders. Kolumna
Status informuje czy płatność została zakończona sukcesem, natomiast kolumna SystemPaymentID zawiera link do
zewnętrznego systemu płatności.
CREATE TABLE Payments (
   PaymentID int  NOT NULL,
   OrderID int  NOT NULL,
   Amount money  NOT NULL,
   Date date  NOT NULL DEFAULT GETDATE(),
   Status bit  NOT NULL DEFAULT 0,
   CHECK (Amount >= 0),
   CONSTRAINT Payments_pk PRIMARY KEY  (PaymentID)
);


-- Table: Practises
-- Każde studia mogą zawierać wiele praktyk, tabela przetrzymuje opis i identyfikator danych praktyk. W Tabeli
Classes znajduje się pole PracticeID, które nie jest NULL-em w przypadku gdy dane zajęcia realizują dane
praktyki.
CREATE TABLE Practises (
   PractiseID int  NOT NULL,
   StudiesID int  NOT NULL,
   PracticeName varchar(40)  NOT NULL,
   PracticeDescription varchar(255)  NOT NULL,
   CONSTRAINT Practices_pk PRIMARY KEY  (PractiseID)
);
```

```sql
-- Table: RegisteredClasses
-- Lista zakupionych przez studentów pojedynczych classes (zjazdów w ramach studiów) z numerami zamówienia
CREATE TABLE RegisteredClasses (
    RegisteredClassID int  NOT NULL,
    OrderID int  NOT NULL,
    ClassID int  NOT NULL,
    Access bit NOT NULL DEFAULT 0,
    CONSTRAINT RegisteredClasses_pk PRIMARY KEY  (RegisteredClassID)
);


-- Table: RegisteredPrograms
-- Lista zakupionych przez studentów EducationalProgramów z numerami zamówienia
CREATE TABLE RegisteredPrograms (
    RegisteredProgramID int  NOT NULL,
    OrderID int  NOT NULL,
    ProgramID int  NOT NULL,
    Passed bit  NOT NULL DEFAULT 0,
    CertificateLink varchar(255),
    Access bit NOT NULL DEFAULT 0,
    CONSTRAINT RegisteredPrograms_pk PRIMARY KEY  (RegisteredProgramID)
);


-- Table: Students
CREATE TABLE Students (
    StudentID int  NOT NULL,
    FirstName varchar(20)  NOT NULL,
    LastName varchar(20)  NOT NULL,
    CountryID int  NOT NULL,
    Email varchar(40)  NOT NULL UNIQUE,
    CONSTRAINT Students_pk PRIMARY KEY  (StudentID)
);


-- Table: Studies
CREATE TABLE Studies (
    StudiesID int  NOT NULL,
    Syllabus varchar(255)  NOT NULL,
    Place varchar(100)  NOT NULL,
    MaxParticipants int  NOT NULL,
    EntryFee money  NOT NULL
    CHECK (EntryFee >= 0),
    CONSTRAINT Studies_pk PRIMARY KEY  (StudiesID)
);
```

```sql
-- Table: SubjectCategories
-- Zawiera kategorie różnych prowadzonych przedmiotów z tabeli Subjects
-- np. Matematyka(SubjectCategories) jest kategorią przedmiotu algebra(Subjects)
CREATE TABLE SubjectCategories (
    CategoryID int  NOT NULL,
    CategoryName varchar(40)  NOT NULL UNIQUE,
    Description varchar(255)  NOT NULL,
    CONSTRAINT SubjectCategories_pk PRIMARY KEY  (CategoryID)
);


-- Table: Subjects
CREATE TABLE Subjects (
    SubjectID int  NOT NULL,
    CategoryID int  NOT NULL,
    Description varchar(255)  NOT NULL,
    SubjectName varchar(40)  NOT NULL UNIQUE,
    CONSTRAINT Subjects_pk PRIMARY KEY  (SubjectID)
);


-- Table: Teachers
CREATE TABLE Teachers (
    TeacherID int  NOT NULL,
    FirstName varchar(15)  NOT NULL,
    LastName varchar(15)  NOT NULL,
    CountryID int  NOT NULL,
    CONSTRAINT Teachers_pk PRIMARY KEY  (TeacherID)
);


-- Table: Webinars
CREATE TABLE Webinars (
    WebinarID int  NOT NULL,
    ClassID int  NOT NULL,
    CONSTRAINT Webinars_pk PRIMARY KEY  (WebinarID)
);


-- foreign keys
-- Reference:  Translators_Countries (table:  Translators)
ALTER TABLE  Translators ADD CONSTRAINT  Translators_Countries
    FOREIGN KEY (CountryID)
    REFERENCES Countries (CountryID);


-- Reference: Attendance_Students (table: Attendance)
ALTER TABLE Attendance ADD CONSTRAINT Attendance_Students
    FOREIGN KEY (ParticipantID)
    REFERENCES Students (StudentID);


-- Reference: Classes_Attendance (table: Attendance)
ALTER TABLE Attendance ADD CONSTRAINT Classes_Attendance
    FOREIGN KEY (ClassID)
    REFERENCES Classes (ClassID);


-- Reference: Classes_Modules (table: Classes)
ALTER TABLE Classes ADD CONSTRAINT Classes_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID);


-- Reference: Classes_Practises (table: Classes)
ALTER TABLE Classes ADD CONSTRAINT Classes_Practises
    FOREIGN KEY (PractiseID)
    REFERENCES Practises (PractiseID);
```

```
-- Reference: Classes_Subjects (table: Classes)
ALTER TABLE Classes ADD CONSTRAINT Classes_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES Subjects (SubjectID);


-- Reference: Classes_Teachers (table: Classes)
ALTER TABLE Classes ADD CONSTRAINT Classes_Teachers
    FOREIGN KEY (TeacherID)
    REFERENCES Teachers (TeacherID);


-- Reference: EducationalPrograms_ Translators (table: EducationalPrograms)
ALTER TABLE EducationalPrograms ADD CONSTRAINT EducationalPrograms_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES  Translators (TranslatorID);


-- Reference: EducationalPrograms_Courses (table: EducationalPrograms)
ALTER TABLE EducationalPrograms ADD CONSTRAINT EducationalPrograms_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);


-- Reference: EducationalPrograms_Studies (table: EducationalPrograms)
ALTER TABLE EducationalPrograms ADD CONSTRAINT EducationalPrograms_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);


-- Reference: EducationalPrograms_Teachers (table: EducationalPrograms)
ALTER TABLE EducationalPrograms ADD CONSTRAINT EducationalPrograms_Teachers
    FOREIGN KEY (LecturerID)
    REFERENCES Teachers (TeacherID);


-- Reference: EducationalPrograms_Webinars (table: EducationalPrograms)
ALTER TABLE EducationalPrograms ADD CONSTRAINT EducationalPrograms_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES Webinars (WebinarID);


-- Reference: Exams_Students (table: Exams)
ALTER TABLE Exams ADD CONSTRAINT Exams_Students
    FOREIGN KEY (StudentID)
    REFERENCES Students (StudentID);


-- Reference: Exams_Studies (table: Exams)
ALTER TABLE Exams ADD CONSTRAINT Exams_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);


-- Reference: Modules_EducationalPrograms (table: Modules)
ALTER TABLE Modules ADD CONSTRAINT Modules_EducationalPrograms
    FOREIGN KEY (ProgramID)
    REFERENCES EducationalPrograms (ProgramID);


-- Reference: OfflineClasses_Classes (table: OfflineClasses)
ALTER TABLE OfflineClasses ADD CONSTRAINT OfflineClasses_Classes
    FOREIGN KEY (ClassID)
    REFERENCES Classes (ClassID);


-- Reference: OnlineClasses_Classes (table: OnlineClasses)
ALTER TABLE OnlineClasses ADD CONSTRAINT OnlineClasses_Classes
    FOREIGN KEY (ClassID)
    REFERENCES Classes (ClassID);
```

```sql
-- Reference: OrderClasses_Classes (table: RegisteredClasses)
ALTER TABLE RegisteredClasses ADD CONSTRAINT OrderClasses_Classes
    FOREIGN KEY (ClassID)
    REFERENCES Classes (ClassID);


-- Reference: OrderClasses_Orders (table: RegisteredClasses)
ALTER TABLE RegisteredClasses ADD CONSTRAINT OrderClasses_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);

-- Reference: OrderDetails_EducationalPrograms (table: RegisteredPrograms)
ALTER TABLE RegisteredPrograms ADD CONSTRAINT OrderDetails_EducationalPrograms
    FOREIGN KEY (ProgramID)
    REFERENCES EducationalPrograms (ProgramID);


-- Reference: OrderDetails_Orders (table: RegisteredPrograms)
ALTER TABLE RegisteredPrograms ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);


-- Reference: Orders_Students (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Orders_Students
    FOREIGN KEY (StudentID)
    REFERENCES Students (StudentID);


-- Reference: Payments_Orders (table: Payments)
ALTER TABLE Payments ADD CONSTRAINT Payments_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);


-- Reference: Practises_Studies (table: Practises)
ALTER TABLE Practises ADD CONSTRAINT Practises_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);


-- Reference: Students_Countries (table: Students)
ALTER TABLE Students ADD CONSTRAINT Students_Countries
    FOREIGN KEY (CountryID)
    REFERENCES Countries (CountryID);


-- Reference: Subjects_SubjectCategories (table: Subjects)
ALTER TABLE Subjects ADD CONSTRAINT Subjects_SubjectCategories
    FOREIGN KEY (CategoryID)
    REFERENCES SubjectCategories (CategoryID);


-- Reference: Teachers_Countries (table: Teachers)
ALTER TABLE Teachers ADD CONSTRAINT Teachers_Countries
    FOREIGN KEY (CountryID)
    REFERENCES Countries (CountryID);


-- Reference: Webinars_Classes (table: Webinars)
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Classes
    FOREIGN KEY (ClassID)
    REFERENCES Classes (ClassID);
```

3. **Widoki**

1. Raporty finansowe – zestawienie przychodów dla każdego webinaru/kursu/studium

```
-- Webinars
CREATE VIEW WebinarsRevenue AS
SELECT Webinars.WebinarID, EducationalPrograms.ProgramName, EducationalPrograms.ProgramStart AS WebinarStart,
EducationalPrograms.ProgramEnd AS WebinarEnd, COALESCE(SUM(Payments.Amount),0) AS Revenue
FROM Webinars
LEFT JOIN EducationalPrograms ON Webinars.WebinarID = EducationalPrograms.WebinarID
LEFT JOIN RegisteredPrograms ON RegisteredPrograms.ProgramID = EducationalPrograms.ProgramID
LEFT JOIN Orders ON Orders.OrderID = RegisteredPrograms.OrderID
LEFT JOIN Payments ON Orders.OrderID = Payments.OrderID
GROUP BY Webinars.WebinarID, EducationalPrograms.ProgramName, EducationalPrograms.ProgramStart,
EducationalPrograms.ProgramEnd
```

	WebinarID	ProgramName	WebinarStart	WebinarEnd	Revenue
1	11	3D Design	2024-02-07	2024-03-11	0.0000
2	7	Artificial Intelligence in Modern Engineering	2023-02-22	2023-05-27	34002.5200
3	6	Cybersecurity Essentials for Data Protection	2022-09-23	2022-11-09	36778.1300
4	1	Exploring the Cosmos: An Astronomy Journey	2021-10-31	2022-02-21	26198.0700
5	8	Journalism Ethics in Today's Media Landscape	2021-09-06	2021-11-26	17336.8200
6	3	Marine Ecosystem Preservation: Challenges and Solutions	2020-04-04	2020-05-31	27361.8500
7	10	Music Production Techniques: From Theory to Studio	2023-06-24	2023-08-30	40222.2000
8	9	Psychological Resilience in Stressful Environments	2022-08-06	2022-12-19	6636.6400
9	5	Societal Impacts of Economic Policies	2020-07-01	2020-08-04	14329.7700
10	4	The Chemistry Behind Sustainable Energy	2022-06-25	2022-10-06	7736.4300
11	2	Understanding Brain Plasticity in Neurology	2023-01-24	2023-02-27	15449.9000

```
--Courses
CREATE VIEW CoursesRevenue AS SELECT Courses.CourseID, EducationalPrograms.ProgramName,
EducationalPrograms.ProgramStart as CourseStart, EducationalPrograms.ProgramEnd as CourseEnd,
COALESCE(SUM(Payments.Amount),0) AS Revenue
FROM Courses
LEFT JOIN EducationalPrograms ON Courses.CourseID = EducationalPrograms.CourseID
LEFT JOIN RegisteredPrograms ON RegisteredPrograms.ProgramID = EducationalPrograms.ProgramID
LEFT JOIN Orders ON Orders.OrderID = RegisteredPrograms.OrderID
LEFT JOIN Payments ON Orders.OrderID = Payments.OrderID
GROUP BY Courses.CourseID, EducationalPrograms.ProgramName, EducationalPrograms.ProgramStart,
EducationalPrograms.ProgramEnd;
```

	CourseID	ProgramName	CourseStart	CourseEnd	Revenue
1	4	Arts, Creativity, and Culture	2022-11-04	2022-12-12	13937.7200
2	6	Business and Management Principles	2022-01-19	2022-03-10	30109.5700
3	7	Diverse Studies	2022-09-30	2022-10-18	16978.5500
4	1	Foundational Sciences	2023-04-14	2023-05-04	34402.0600
5	5	Health and Well-being	2023-10-15	2023-11-29	74149.0200
6	8	Processing meat: Butcher guide 1	2024-02-02	2024-02-22	0.0000
7	2	Social and Behavioral Studies	2021-04-03	2021-05-05	93006.7500
8	3	Technology and Innovation	2022-08-02	2022-09-09	44485.4600

```
-- Studies
CREATE VIEW StudiesRevenue AS SELECT Studies.StudiesID, EducationalPrograms.ProgramName,
EducationalPrograms.ProgramStart as StudiesStart, EducationalPrograms.ProgramEnd as StudiesEnd,
COALESCE(SUM(Payments.Amount),0) AS Revenue
FROM Studies
LEFT JOIN EducationalPrograms ON Studies.StudiesID = EducationalPrograms.StudiesID
LEFT JOIN RegisteredPrograms ON RegisteredPrograms.ProgramID = EducationalPrograms.ProgramID
LEFT JOIN Modules ON Modules.ProgramID = EducationalPrograms.ProgramID
LEFT JOIN Practises ON Practises.StudiesID = Studies.StudiesID
LEFT JOIN Classes ON Classes.ModuleID = Modules.ModuleID OR Classes.PractiseID = Practises.PractiseID
LEFT JOIN RegisteredClasses ON RegisteredClasses.ClassID = Classes.ClassID
LEFT JOIN Orders ON Orders.OrderID = RegisteredClasses.OrderID OR Orders.OrderID = RegisteredPrograms.OrderID
LEFT JOIN Payments ON Payments.OrderID = Orders.OrderID
```

```
GROUP BY Studies.StudiesID, EducationalPrograms.ProgramName, EducationalPrograms.ProgramStart,
EducationalPrograms.ProgramEnd;
```

	StudiesID	ProgramName	StudiesStart	StudiesEnd	Revenue
1	2	Public-key needs-based strategy	2022-08-20	2025-11-29	4851431.4800
2	5	Visionary 24/7 monitoring	2020-11-22	2022-03-17	3253533.4000
3	6	Synergistic actuating portal	2021-05-11	2024-05-07	5703829.9600
4	7	Upgradable discrete definition	2023-01-26	2026-07-09	8213574.0400
5	3	Reactive mission-critical database	2022-11-19	2024-11-12	4536044.2000
6	1	Synergized motivating capability	2021-04-12	2024-09-10	16941042.2800
7	4	Vision-oriented solution-oriented paradigm	2021-12-28	2025-08-07	7462093.8400

2. Lista „dłużników” – osoby, które skorzystały z usług, ale nie uiściły opłat.

```
CREATE VIEW Debtors AS
    SELECT S.*, ISNULL(OPS.ProgramsCost, 0) + ISNULL(OCS.ClassesCost, 0) - ISNULL(OP.Paid, 0) As Debt
    FROM Students S
    LEFT JOIN (
        SELECT O.StudentID, SUM(EP.ProgramPrice) AS ProgramsCost
        FROM Orders O
        LEFT JOIN RegisteredPrograms RP ON RP.OrderID = O.OrderID
        LEFT JOIN EducationalPrograms EP ON EP.ProgramID = RP.ProgramID
        GROUP BY O.StudentID
    ) OPS ON OPS.StudentID = S.StudentID
    LEFT JOIN (
        SELECT O.StudentID, SUM(C.ClassPrice) AS ClassesCost
        FROM Orders O
        LEFT JOIN RegisteredClasses RC ON RC.OrderID = O.OrderID
        LEFT JOIN Classes C ON C.ClassID = RC.ClassID
        GROUP BY O.StudentID
    ) OCS ON OCS.StudentID = S.StudentID
    LEFT JOIN (
        SELECT O.StudentID, SUM(P.Amount) AS Paid
        FROM Orders O
        LEFT JOIN Payments P ON P.OrderID = O.OrderID
        GROUP BY O.StudentID
    ) OP ON OP.StudentID = S.StudentID
    WHERE ISNULL(OPS.ProgramsCost, 0) + ISNULL(OCS.ClassesCost, 0) > ISNULL(OP.Paid, 0)
```

	StudentID	FirstName	LastName	CountryID	Email	Debt
1	11	Cynthia	Smith	9	CynSm96@gmail.com	2033.6100
2	12	John	Jackson	10	JoJac25@gmail.com	476.9400
3	13	Kelsey	Williams	5	KelWill84@gmail.com	9410.5600
4	15	John	Wong	9	JoWo60@gmail.com	6270.1200
5	21	Daniel	Quinn	4	DanQu4@gmail.com	689.1500
6	22	Allen	Lucas	7	AlLu26@gmail.com	711.8100
7	25	Denise	Rios	4	DenRi77@gmail.com	102.6100
8	29	Virginia	Lewis	10	VirgLe62@gmail.com	230.2500
9	30	Michael	Munoz	3	MicMu19@gmail.com	20.2200
10	32	Heather	Adams	5	HeaAd43@gmail.com	6.3900

3. Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia - realizowane w przyszłości zajęcia (z informacją, czy zajęcia są stacjonarnie, czy zdalnie).

```
CREATE VIEW NumOfInterestedInFutureClasses as
with
tab as (
   select Classes.ClassID, count(Students.StudentID) as NumOfInterested
   from Classes
       left outer join RegisteredClasses RC on Classes.ClassID = RC.ClassID and RC.Access = 'true'
       left outer join Orders on RC.OrderID = Orders.OrderID
       left outer join Students on Orders.StudentID = Students.StudentID
   where Classes.StartTime > getdate()
   group by Classes.ClassID)
select Classes.ClassID, Classes.StartTime, Teachers.FirstName + ' ' + Teachers.LastName as Teacher,
Subjects.SubjectName, tab.NumOfInterested, OfflineClassID, OnlineClassID
from Classes
   left outer join tab on tab.ClassID = Classes.ClassID
   left outer join OnlineClasses on Classes.ClassID = OnlineClasses.ClassID
   left outer join OfflineClasses on Classes.ClassID = OfflineClasses.ClassID
   left outer join Teachers on Classes.TeacherID = Teachers.TeacherID
   left outer join Subjects on Classes.SubjectID = Subjects.SubjectID
where Classes.StartTime > getdate()
```

	ClassID	StartTime	Teacher	SubjectName	NumOfInterested	OfflineClassID	OnlineClassID
1	269	2024-02-07 14:00:00.000	Lisa Boyd	Marine Biology	0	<null>	16
2	270	2024-02-07 14:00:00.000	Jeremy Walker	Astronomy	0	254	<null>
3	271	2024-02-07 14:00:00.000	Jeremy Walker	Astronomy	0	255	<null>

4. Ogólny raport dotyczący liczby zapisanych osób na jeszcze nie rozpoczęte programy edukacyjne wraz z datą rozpoczęcia

```
CREATE VIEW NumOfInterestedInFutureEducationalPrograms as
with tab as (
select EducationalPrograms.ProgramID, count(Students.StudentID) as NumOfInterested
from EducationalPrograms
   left outer join RegisteredPrograms RP on EducationalPrograms.ProgramID = RP.ProgramID and RP.Access = 'true'
   left outer join Orders on RP.OrderID = Orders.OrderID
   left outer join Students on Orders.StudentID = Students.StudentID
where EducationalPrograms.ProgramStart > getdate()
group by EducationalPrograms.ProgramID
) select tab.ProgramID, EducationalPrograms.ProgramName, tab.NumOfInterested, EducationalPrograms.ProgramStart
from EducationalPrograms
   join tab on EducationalPrograms.ProgramID = tab.ProgramID
```

	ProgramID	ProgramName	NumOfInterested	ProgramStart
1	25	Processing meat: Butcher guide 1	0	2024-02-02
2	26	3D Design	0	2024-02-07

5. Ogólny raport dotyczący listy osób listy osób zapisanych na stacjonarne zajęcia w ramach programu edukacyjnego.

```
CREATE VIEW OfflineParticipantsList as
select st.StudentID, st.FirstName + ' ' + st.LastName as Student, c.ClassID, m.ModuleName, sub.SubjectName,
t.FirstName + ' ' + t.LastName as Teacher, c.StartTime, c.EndTime, ofl.RoomNumber, rp.Access
from Students as st
    inner join Orders as od
        on st.StudentID = od.StudentID
    inner join RegisteredPrograms as rp
        on od.OrderID = rp.OrderID
    inner join EducationalPrograms as ep
        on rp.ProgramID = ep.ProgramID
    inner join Modules as m
        on ep.ProgramID = m.ProgramID
    inner join Classes as c
        on m.ModuleID = c.ModuleID
    inner join Subjects as sub
        on c.SubjectID = sub.SubjectID
    inner join Teachers as t
        on c.TeacherID = t.TeacherID
    inner join OfflineClasses as ofl
        on c.ClassID = ofl.ClassID
```

	StudentID	Student	ClassID	ModuleName	SubjectName	Teacher	StartTime	EndTime	Room...	Access
1	14	Michele Diaz	13	Information Sciences	Information Technology	James Robertson	2022-08-02 14:00…	2022-08-02 16:25…	30	false
2	53	Jessica Alexander	13	Information Sciences	Information Technology	James Robertson	2022-08-02 14:00…	2022-08-02 16:25…	30	false
3	54	Jonathan Allen	13	Information Sciences	Information Technology	James Robertson	2022-08-02 14:00…	2022-08-02 16:25…	30	• true
4	80	Brandon Sanchez	13	Information Sciences	Information Technology	James Robertson	2022-08-02 14:00…	2022-08-02 16:25…	30	false
5	81	Thomas Walsh	13	Information Sciences	Information Technology	James Robertson	2022-08-02 14:00…	2022-08-02 16:25…	30	• true
6	98	James Clarke	13	Information Sciences	Information Technology	James Robertson	2022-08-02 14:00…	2022-08-02 16:25…	30	false
7	32	Heather Adams	15	Media and Communication	Journalism	Jeremy Walker	2022-11-04 14:00…	2022-11-04 16:25…	33	false
8	35	Michael Mason	15	Media and Communication	Journalism	Jeremy Walker	2022-11-04 14:00…	2022-11-04 16:25…	33	false
9	36	Sierra Mendoza	15	Media and Communication	Journalism	Jeremy Walker	2022-11-04 14:00…	2022-11-04 16:25…	33	• true
10	45	Hannah Fischer	15	Media and Communication	Journalism	Jeremy Walker	2022-11-04 14:00…	2022-11-04 16:25…	33	• true

6. Lista obecności dla każdego szkolenia z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie.

```
CREATE VIEW AttendanceAllClasses as
SELECT C.ClassID, CONCAT(year(C.StartTime), '-', month(C.StartTime), '-', day(C.StartTime)) as Date,
Students.FirstName + ' ' + Students.LastName as Student, A.Present
```

```
FROM Classes C
LEFT OUTER JOIN Attendance A on C.ClassID = A.ClassID
LEFT OUTER JOIN Students on Students.StudentID = A.ParticipantID
```

	ClassID	Date	Student	Present
1	21	2021-4-12	Jessica Gates	• true
2	22	2021-4-12	Jessica Gates	• true
3	23	2021-4-12	Jessica Gates	• true
4	24	2021-4-12	Jessica Gates	• true
5	25	2021-4-12	Jessica Gates	• true
6	26	2021-4-12	Jessica Gates	• true
7	27	2021-4-12	Jessica Gates	• true
8	28	2021-4-12	Jessica Gates	• true
9	29	2021-4-12	Jessica Gates	• true
10	30	2021-4-12	Jessica Gates	• true

7. Raport bilokacji: Lista kolidujących się zajęć wraz z informacją o studencie, ID zajęć oraz kolidyjącymi się terminami.

```
CREATE VIEW BilocationsList as select distinct s.StudentID, s.FirstName + ' ' + s.LastName as Student, a.ClassID
as a_ClassID, a.StartTime as a_StartTime, a.EndTime as a_EndTime, b.ClassID as b_ClassID, b.StartTime as
b_StartTime, b.EndTime as b_EndTime from Students as s
       inner join Orders as o
           on s.StudentID = o.StudentID
       inner join RegisteredPrograms as rp
           on o.OrderID = rp.OrderID
       inner join Modules as m
           on rp.ProgramID = m.ProgramID
       inner join Classes as a
           on m.ModuleID = a.ModuleID
       cross join Classes as b
where a.ClassID < b.ClassID and ((a.StartTime BETWEEN b.StartTime and b.EndTime) or (b.StartTime BETWEEN
a.StartTime and a.EndTime) or (a.EndTime BETWEEN b.StartTime and b.EndTime) or (b.EndTime BETWEEN a.StartTime and
a.EndTime))
```

	StudentID	Student	a_ClassID	a_StartTime	a_EndTime	b_ClassID	b_StartTime	b_EndTime
1	11	Cynthia Smith	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
2	24	Nancy Maxwell	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
3	27	Kimberly Martin	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
4	29	Virginia Lewis	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
5	33	Chad Lyons	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
6	37	Thomas Johnson	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
7	48	Sharon Jackson	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
8	72	Darren Moore	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
9	90	Mark Winters	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	166	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000
10	11	Cynthia Smith	165	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000	167	2020-11-22 13:00:00.000	2020-11-22 15:25:00.000

8. Liczba osób dla każdego zakończonego już wydarzenia

```
CREATE VIEW NumberOfParticipations as select c.ClassID, c.TeacherID, t.FirstName + ' ' + t.LastName as Teacher,
sub.SubjectName, c.StartTime, c.EndTime, count(s.StudentID) as StudentsAmount
   from Classes as c
       left join Attendance as a
           on c.ClassID = a.ClassID
       left join Students as s
           on a.ParticipantID = s.StudentID
       left join Teachers as t
           on c.TeacherID = t.TeacherID
       left join Subjects as sub
           on c.SubjectID = sub.SubjectID
   where c.EndTime < getdate()
group by c.ClassID, c.TeacherID, t.FirstName + ' ' + t.LastName, sub.SubjectName, c.SubjectID, c.StartTime,
c.EndTime
```

	ClassID	TeacherID	Teacher	SubjectName	StartTime	EndTime	StudentsAmount
1	1	10	James Robertson	Environmental Science	2021-10-31 14:00:00.000	2021-10-31 17:00:00.000	9
2	2	14	Patrick Velasquez	Neuroscience	2023-01-24 14:00:00.000	2023-01-24 17:00:00.000	0
3	3	11	Jodi Norris	Marine Biology	2020-04-04 14:00:00.000	2020-04-04 17:00:00.000	7
4	4	4	Lisa Boyd	Chemistry	2022-06-25 14:00:00.000	2022-06-25 17:00:00.000	0
5	5	8	Jeff Murillo	Economics	2020-07-01 14:00:00.000	2020-07-01 17:00:00.000	6
6	6	1	Jessica Ramos	Cybersecurity	2022-09-23 14:00:00.000	2022-09-23 17:00:00.000	0
7	7	11	Jodi Norris	Engineering	2023-02-22 14:00:00.000	2023-02-22 17:00:00.000	0
8	8	8	Jeff Murillo	Journalism	2021-09-06 14:00:00.000	2021-09-06 17:00:00.000	6
9	9	9	Wendy Douglas	Psychology	2022-08-06 14:00:00.000	2022-08-06 17:00:00.000	0
10	10	1	Jessica Ramos	Music Production	2023-06-24 14:00:00.000	2023-06-24 17:00:00.000	0

9. Dane o każdym przeprowadzonym egzaminie, które zawierają ocenę, ID studenta, ID studiów, ID programu edukacyjnego oraz terminy rozpoczęcia & zakończenia danych studiów

```
CREATE VIEW ExamDetails as select ex.ExamID, ex.StudentID, S2.FirstName + ' ' + S2.LastName as Student, ex.Mark,
ex.StudiesID, ep.ProgramName, ep.ProgramStart, ep.ProgramEnd from exams as ex
   inner join Studies as s
       on ex.StudiesID = s.StudiesID
   inner join dbo.Students S2
       on ex.StudentID = S2.StudentID
   inner join dbo.EducationalPrograms EP
       on s.StudiesID = EP.StudiesID
```

	ExamID	StudentID	Student	Mark	StudiesID	ProgramName	ProgramStart	ProgramEnd
1	0	10	Jessica Gates	33	1	Synergized motivating capability	2021-04-12	2024-09-10
2	1	11	Cynthia Smith	66	5	Visionary 24/7 monitoring	2020-11-22	2022-03-17
3	2	13	Kelsey Williams	50	6	Synergistic actuating portal	2021-05-11	2024-05-07
4	3	15	John Wong	58	1	Synergized motivating capability	2021-04-12	2024-09-10
5	4	17	Tracy Campbell	94	1	Synergized motivating capability	2021-04-12	2024-09-10
6	5	18	Peter Jones	50	6	Synergistic actuating portal	2021-05-11	2024-05-07
7	6	19	James Bentley	76	1	Synergized motivating capability	2021-04-12	2024-09-10
8	7	20	Brian Day	47	1	Synergized motivating capability	2021-04-12	2024-09-10
9	8	21	Daniel Quinn	94	1	Synergized motivating capability	2021-04-12	2024-09-10
10	9	23	Kevin Hanson	32	4	Vision-oriented solution-oriented paradigm	2021-12-28	2025-08-07

10. Lista przedmiotów prowadzonych w ramach modułów pewnych studiów wraz z informacją o kategorii oraz prowadzącym zajęcia z danego przedmiotu

```
create view StudiesSubjectsInfo as select ed.ProgramID, ed.ProgramName, m.ModuleName, sub.SubjectName,
sc.CategoryName, t.FirstName + ' ' + t.LastName as Teacher
   from Studies as s
       inner join EducationalPrograms as ed
           on s.StudiesID = ed.StudiesID
       inner join Modules as m
           on ed.ProgramID = m.ProgramID
       inner join Classes as c
           on m.ModuleID = c.ModuleID
       inner join Subjects as sub
           on c.SubjectID = sub.SubjectID
       inner join SubjectCategories as sc
           on sub.CategoryID = sc.CategoryID
       inner join Teachers as t
           on c.TeacherID = t.TeacherID
```

	ProgramID	ProgramName	ModuleName	SubjectName	CategoryName	Teacher
1	18	Synergized motivating capability	Enough her would any.	Comparative Literature	Arts and Humanities	Jeff Murillo
2	18	Synergized motivating capability	Enough her would any.	Interior Design	Arts and Humanities	Jessica Ramos
3	18	Synergized motivating capability	Enough her would any.	Environmental Science	Natural Sciences	Patrick Velasquez
4	18	Synergized motivating capability	Food per.	Drama/Theater	Arts and Humanities	Jessica Ramos
5	18	Synergized motivating capability	Food per.	Creative Writing	Arts and Humanities	Steven Jenkins
6	18	Synergized motivating capability	Food per.	Music Production	Arts and Humanities	Wendy Douglas
7	18	Synergized motivating capability	Food per.	Culinary Arts	Other	Patrick Velasquez
8	18	Synergized motivating capability	Food per.	Technical Writing	Arts and Humanities	Jessica Ramos
9	18	Synergized motivating capability	Either senior.	Machine Learning	Engineering and Technology	Joseph Gonzalez
10	18	Synergized motivating capability	Either senior.	Political Science	Social Sciences	Wendy Douglas

11. Lista przedmiotów prowadzonych w ramach modułów pewnych kursów wraz z informacją o kategorii oraz prowadzącym zajęcia z danego przedmiotu

```
create view CoursesSubjectsInfo as select ed.ProgramID, ed.ProgramName, m.ModuleName, sub.SubjectName,
sc.CategoryName, t.FirstName + ' ' + t.LastName as Teacher
  from Courses
      inner join EducationalPrograms as ed
          on Courses.CourseID = ed.CourseID
      inner join Modules as m
          on ed.ProgramID = m.ProgramID
      inner join Classes as c
          on m.ModuleID = c.ModuleID
      inner join Subjects as sub
          on c.SubjectID = sub.SubjectID
      inner join SubjectCategories as sc
          on sub.CategoryID = sc.CategoryID
      inner join Teachers as t
          on c.TeacherID = t.TeacherID
```

	ProgramID	ProgramName	ModuleName	SubjectName	CategoryName	Teacher
1	11	Foundational Sciences	Nature and Environment	Environmental Science	Natural Sciences	Steven Jenkins
2	12	Social and Behavioral Studies	Societal Systems	Political Science	Social Sciences	Cameron Sims
3	13	Technology and Innovation	Information Sciences	Information Technology	Engineering and Technology	James Robertson
4	14	Arts, Creativity, and Culture	Creative Arts Overview	Creative Writing	Arts and Humanities	James Robertson
5	14	Arts, Creativity, and Culture	Media and Communication	Journalism	Arts and Humanities	Jeremy Walker
6	14	Arts, Creativity, and Culture	Design and Expression	Graphic Design	Arts and Humanities	Joseph Gonzalez
7	16	Business and Management Principles	Economics and Finance	Supply Chain Management	Business and Management	James Robertson
8	17	Diverse Studies	Hospitality and Services	Hospitality Management	Other	Anna Ruiz
9	17	Diverse Studies	Hospitality and Services	Culinary Arts	Other	Wendy Douglas
10	17	Diverse Studies	Sports and Health Sciences	Kinesiology	Other	Steven Jenkins

12. Lista przyszłych webinarów, widok wyświetla nazwę przedmiotu, imię i nazwisko nauczyciela, który prowadzi webinar, czas trwania oraz czas dostępu do webinaru

```
create view WebinarsInfo as select w.WebinarID, c.ClassID, s.SubjectName, t.FirstName + ' ' + t.LastName as
TeacherName, c.StartTime as WebinarStart, c.EndTime as WebinarEnd, ed.ProgramStart as AccessStart, ed.ProgramEnd
as AccessEnd
   from webinars as w
       inner join dbo.Classes C on C.ClassID = w.ClassID
       inner join dbo.OnlineClasses OC on C.ClassID = OC.ClassID
       inner join dbo.Teachers T on C.TeacherID = T.TeacherID
       inner join EducationalPrograms as Ed ON w.WebinarID = Ed.WebinarID
       inner join Subjects as s on C.SubjectID = s.SubjectID
```

	WebinarID	ClassID	SubjectName	TeacherName	WebinarStart	WebinarEnd	AccessStart	AccessEnd
1	1	1	Environmental Science	James Robertson	2021-10-31 14:00:00.000	2021-10-31 17:00:00.000	2021-10-31	2022-02-21
2	2	2	Neuroscience	Patrick Velasquez	2023-01-24 14:00:00.000	2023-01-24 17:00:00.000	2023-01-24	2023-02-27
3	3	3	Marine Biology	Jodi Norris	2020-04-04 14:00:00.000	2020-04-04 17:00:00.000	2020-04-04	2020-05-31
4	4	4	Chemistry	Lisa Boyd	2022-06-25 14:00:00.000	2022-06-25 17:00:00.000	2022-06-25	2022-10-06
5	5	5	Economics	Jeff Murillo	2020-07-01 14:00:00.000	2020-07-01 17:00:00.000	2020-07-01	2020-08-04
6	6	6	Cybersecurity	Jessica Ramos	2022-09-23 14:00:00.000	2022-09-23 17:00:00.000	2022-09-23	2022-11-09
7	7	7	Engineering	Jodi Norris	2023-02-22 14:00:00.000	2023-02-22 17:00:00.000	2023-02-22	2023-05-27
8	8	8	Journalism	Jeff Murillo	2021-09-06 14:00:00.000	2021-09-06 17:00:00.000	2021-09-06	2021-11-26
9	9	9	Psychology	Wendy Douglas	2022-08-06 14:00:00.000	2022-08-06 17:00:00.000	2022-08-06	2022-12-19
10	10	10	Music Production	Jessica Ramos	2023-06-24 14:00:00.000	2023-06-24 17:00:00.000	2023-06-24	2023-08-30
11	11	269	Marine Biology	Lisa Boyd	2024-02-07 14:00:00.000	2024-02-07 15:30:00.000	2024-02-07	2024-03-11

13. Lista programów edukacyjnych, na które są zapisane studenty, wraz z numerem zamówienia, w którym dany program był zamówiony, datą rozpoczęcia i zakończenia programu oraz informacją o zaliczeniu.

```
create view StudentsPrograms as select s.StudentID, s.FirstName + ' ' + s.LastName as Student,
rp.RegisteredProgramID, ep.ProgramID, ep.ProgramName, ep.ProgramStart, ep.ProgramEnd, rp.Access, rp.Passed
    from Students as s
        inner join Orders as o
            on s.StudentID = o.StudentID
        inner join RegisteredPrograms as rp
            on o.OrderID = rp.OrderID
        inner join EducationalPrograms as ep
            on rp.ProgramID = ep.ProgramID
```

	StudentID	Student	RegisteredProgramID	ProgramID	ProgramName	ProgramStart	ProgramEnd	Access	Passed
1	10	Jessica Gates	1	10	Music Production …	2023-06-24	2023-08-30	• true	false
2	10	Jessica Gates	2	18	Synergized motiva…	2021-04-12	2024-09-10	• true	false
3	11	Cynthia Smith	3	22	Visionary 24/7 mo…	2020-11-22	2022-03-17	false	• true
4	12	John Jackson	4	4	The Chemistry Beh…	2022-06-25	2022-10-06	• true	false
5	13	Kelsey Williams	5	10	Music Production …	2023-06-24	2023-08-30	• true	• true
6	13	Kelsey Williams	6	20	Reactive mission-…	2022-11-19	2024-11-12	• true	• true
7	13	Kelsey Williams	7	23	Synergistic actua…	2021-05-11	2024-05-07	• true	• true
8	14	Michele Diaz	8	13	Technology and In…	2022-08-02	2022-09-09	false	• true
9	15	John Wong	9	18	Synergized motiva…	2021-04-12	2024-09-10	• true	• true
10	15	John Wong	10	11	Foundational Scie…	2023-04-14	2023-05-04	• true	• true

14. Lista osób, które są zapisane na dane zajęcia "z zewnątrz". Widok wyświetla imię i nazwisko studenta, numer zamówienia, w którym dane spotkanie było zamówione, numer spotkania, status dostępu, nazwę przedmiotu dla danego spotkania, nazwę modulu, w ramach którego jest prowadzone spotkanie, oraz nazwę praktyki (o ile to spotkanie jest praktyką)

```
create view StudentsOuterClasses as select s.StudentID, s.FirstName + ' ' + s.LastName as Student, o.OrderID,
c.ClassID, m.ModuleName, sub.SubjectName, t.FirstName + ' ' + t.LastName as Teacher, c.StartTime, c.EndTime,
ofl.RoomNumber, rc.Access
   from Students as s
       inner join Orders as o
           on s.StudentID = o.StudentID
       inner join RegisteredClasses as rc
           on o.OrderID = rc.OrderID
       inner join Classes as c
           on rc.ClassID = c.ClassID
       inner join Modules as m
           on c.ModuleID = m.ModuleID
       left join Practises as p
           on c.PractiseID = p.PractiseID
       inner join Subjects as sub
           on c.SubjectID = sub.SubjectID
       inner join Teachers as t
           on c.TeacherID = t.TeacherID
       inner join OfflineClasses as ofl
           on c.ClassID = ofl.ClassID
```

	Stude...	Student	O...	ClassID	ModuleName	SubjectName	Tea...	StartTime	EndTime	Room...	Access
1	2	Aaron Leon	2	182	Who rich least.	Cybersecurity	Patrick Ve…	2020-11-22 13:00…	2020-11-22 15:25…	86	false
2	7	Sheri Crawford	7	129	Prepare recognize low either.	Artificial Intel…	Wendy Doug…	2021-12-28 13:00…	2021-12-28 15:25…	77	• true
3	19	James Bentley	19	74	Somebody and land.	Child Development	James Robe…	2022-08-20 13:00…	2022-08-20 15:25…	76	false
4	29	Virginia Lewis	29	74	Somebody and land.	Child Development	James Robe…	2022-08-20 13:00…	2022-08-20 15:25…	76	false
5	29	Virginia Lewis	29	74	Somebody and land.	Child Development	James Robe…	2022-08-20 13:00…	2022-08-20 15:25…	76	false
6	29	Virginia Lewis	29	74	Somebody and land.	Child Development	James Robe…	2022-08-20 13:00…	2022-08-20 15:25…	76	false
7	29	Virginia Lewis	29	74	Somebody and land.	Child Development	James Robe…	2022-08-20 13:00…	2022-08-20 15:25…	76	false
8	30	Michael Munoz	30	74	Somebody and land.	Child Development	James Robe…	2022-08-20 13:00…	2022-08-20 15:25…	76	false
9	4	Brian Morales	4	165	Than.	Cybersecurity	Wendy Doug…	2020-11-22 13:00…	2020-11-22 15:25…	76	• true
10	12	John Jackson	12	81	Several ever decide skin.	Kinesiology	Jodi Norris	2022-08-20 13:00…	2022-08-20 15:25…	57	false

4. **Procedury**

1. Dodanie nowego Studenta

```
CREATE PROCEDURE AddStudent(
   @firstName VARCHAR(20),
   @lastName VARCHAR(20),
   @countryID INT,
   @email VARCHAR(40)
)
AS
BEGIN
   BEGIN TRY
       SET NOCOUNT ON;
       IF EXISTS(SELECT * FROM Students WHERE Email = @email)
            THROW 52034, N'Email already in use', 1;
       IF NOT EXISTS(SELECT * FROM Countries WHERE CountryID = @CountryID)
            THROW 52034, N'Country does not exist in the Countries table', 1;
       INSERT INTO Students (FirstName, LastName, CountryID, Email)
       VALUES (@firstName, @lastName, @countryID, @email);
       PRINT 'Student added successfully.';
   END TRY
   BEGIN CATCH
       DECLARE @Message NVARCHAR(1000) = N'error: ' + ERROR_MESSAGE();
       THROW 52034, @Message, 1;
   END CATCH
END
```

2. Dodanie nowego kursu

```
CREATE PROCEDURE AddCourse
   @Place varchar(40),
   @Advance money,
   @ProgramName varchar(100),
   @Language varchar(20),
   @ProgramStart date,
   @ProgramEnd date,
   @ProgramPrice money,
   @LecturerID int,
   @TranslatorID int
AS
BEGIN
BEGIN TRY
   SET NOCOUNT ON;

   IF NOT EXISTS (SELECT 1 FROM Teachers WHERE TeacherID = @LecturerID)
   BEGIN
       THROW 50000, 'TeacherID does not exist in the Teachers table.', 1;
   END;
   IF NOT EXISTS (SELECT 1 FROM Translators WHERE TranslatorID = @TranslatorID)
   BEGIN
       THROW 50000, 'TranslatorID does not exist in the Translators table.', 1;
   END;

   INSERT INTO Courses (Place, Advance)
   VALUES (@Place, @Advance);


   INSERT INTO EducationalPrograms (ProgramName, CourseID, Language, ProgramStart, ProgramEnd, ProgramPrice,
LecturerID, TranslatorID)
   VALUES (@NewProgramID, @ProgramName, @NewCourseID, @Language, @ProgramStart, @ProgramEnd, @ProgramPrice,
@LecturerID, @TranslatorID);
   PRINT 'Course added successfully.';
END TRY
BEGIN CATCH
    DECLARE @Message NVARCHAR(1000) = N'error: ' + ERROR_MESSAGE();
    THROW 52011, @Message, 1;
```

```
END CATCH
END;
```

3. Usuwanie danych studenta

```
ALTER PROCEDURE DeleteStudent(@studentID INT)
AS
BEGIN
   BEGIN TRY
       IF NOT EXISTS(
                SELECT *
                FROM Students
                WHERE StudentID = @studentID
           )
           BEGIN
               THROW 52000, N'There is no student with given ID', 1;
           END
           DECLARE @an NVARCHAR(10) = 'xxxxxxxx'
           UPDATE Students
               SET FirstName   = @an,
               LastName    = @an,
               Email     = @an
           WHERE StudentID = @studentID
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
       THROW 52000, @msg, 1;
   END CATCH
END
```

4. Aktualizacja danych studenta (tylko email albo country)

```
CREATE PROCEDURE ChangeStudentData(
  @studentID int,
  @countryID int = NULL,
  @email varchar(40) = NULL)
AS
BEGIN
   SET NOCOUNT ON;
   IF @countryID IS NOT NULL
   BEGIN
       IF @countryID in (select CountryID from Countries)
           UPDATE Students SET CountryID = @countryID WHERE StudentID = @studentID
   END
   IF @email IS NOT NULL
   BEGIN
       UPDATE Students SET Email = @Email WHERE StudentID = @studentID
       PRINT 'Student data changed successfully.';
   END
END
```

5. Dodanie nowego nauczyciela

```
CREATE PROCEDURE AddTeacher(
  @firstName VARCHAR(20),
  @lastName VARCHAR(20),
  @countryID INT
)
AS
BEGIN
  BEGIN TRY
    IF NOT EXISTS(SELECT * FROM Countries WHERE CountryID = @CountryID)
        THROW 52034, N'Country does not exist in the Countries table', 1;
    INSERT INTO Teachers (FirstName, LastName, CountryID)
    VALUES (@firstName, @lastName, @countryID)
    PRINT 'Teacher added successfully.';
```

```
  END TRY
  BEGIN CATCH
        DECLARE @Message NVARCHAR(1000) = N'error: ' + ERROR_MESSAGE();
        THROW 52011, @Message, 1;
  END CATCH
END
```

6. Oznaczenie odrobienia nieobecności na zajęciach przez studenta

```
create procedure redoAttendance @classID int, @studentID int
as
begin
   set nocount on
   begin try
       if not exists
           (
           select ClassID, ParticipantID
               from Attendance
           WHERE ClassID = @classID and ParticipantID = @studentID
           )
       begin;
           throw 52000, N'The student was not registered for the class with the given ID', 1
       end
       if exists
           (
           select ClassID, ParticipantID
               from Attendance
           WHERE ClassID = @classID and ParticipantID = @studentID and Redone = 1
           )
       begin;
           throw 52000, N'Attendance had already been made up earlier', 1
       end
       update Attendance
       set Redone = 1
       where ClassID = @classID and ParticipantID = @studentID
       print 'Attendance was successfully set as redone!'
   end try
   begin catch
       declare @error varchar(1000)= 'Error when setting attendance as made up: ' + ERROR_MESSAGE();
       throw 77777, @error, 1
   end catch
end
```

7. Dodanie zamówienia

```
CREATE PROCEDURE AddOrder(
   @Studentid INT
)
AS
BEGIN
  BEGIN TRY
      IF NOT EXISTS(SELECT * FROM Students WHERE StudentID = @Studentid)
      BEGIN
          THROW 52011, N'There is no student with such id', 1;
      END
      BEGIN
          INSERT INTO Orders (StudentID, OrderDate)
          VALUES (@Studentid, GETDATE());

          PRINT 'Order added successfully.';
       END
  END TRY
  BEGIN CATCH
      DECLARE @Message NVARCHAR(1000) = N'error: ' + ERROR_MESSAGE();
      THROW 52011, @Message, 1;
  END CATCH
END
```

8. Dodanie pojedynczych zajęć do zamówienia

```
CREATE PROCEDURE RegisterClass(
   @OrderID INT,
   @ClassID INT
)
AS
BEGIN
  BEGIN TRY
       BEGIN TRAN REGISTER_CLASS
       IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID)
        THROW 52313, N'There is no Order with such id', 1;


       IF NOT EXISTS(SELECT * FROM Classes WHERE ClassID = @ClassID)
        THROW 52313, N'There is no Class with such id', 1;


       IF EXISTS (SELECT * FROM RegisteredClasses WHERE OrderID = @OrderID and ClassID = @ClassID)
        THROW 52313, N'Student has already registered for this class', 1;

       INSERT INTO RegisteredClasses (OrderID, ClassID)
       VALUES (@OrderID, @ClassID);

       COMMIT TRAN REGISTER_CLASS
       PRINT 'Class was added successfully!'
  END TRY
  BEGIN CATCH
      ROLLBACK TRAN REGISTER_CLASS
      DECLARE @Message NVARCHAR(1000) = N'error: ' + ERROR_MESSAGE();
      THROW 52011, @Message, 1;
  END CATCH
END
```

9. Dodawanie programu edukacyjnego do zamówienia

```
CREATE PROCEDURE RegisterProgram(
 @OrderID INT,
 @ProgramID INT,
 @Passed AS bit = FALSE,
 @CertificateLink AS varchar(255) = NULL
)
AS
BEGIN
BEGIN TRY
     BEGIN TRAN REGISTER_PROGRAM
     IF NOT EXISTS(SELECT * FROM Orders WHERE OrderID = @OrderID)
      THROW 52313, N'There is no Order with such id', 1;

     IF NOT EXISTS(SELECT * FROM EducationalPrograms WHERE ProgramID = @ProgramID)
      THROW 52313, N'There is no EducationalProgram with such id', 1;

     IF EXISTS (SELECT * FROM RegisteredPrograms WHERE OrderID = @OrderID and ProgramID = @ProgramID)
       THROW 52313, N'Student has already registered for this educational program', 1;

     INSERT INTO RegisteredPrograms (OrderID, ProgramID, Passed, CertificateLink)
     VALUES (@OrderID, @ProgramID, @Passed, @CertificateLink);
     COMMIT TRAN REGISTER_PROGRAM
     PRINT 'Program was added successfully!'
END TRY
BEGIN CATCH
    ROLLBACK TRAN REGISTER_PROGRAM
    DECLARE @Message NVARCHAR(1000) = N'error: ' + ERROR_MESSAGE();
    THROW 52011, @Message, 1;
END CATCH
END;
```

10. Dodanie nowego pojedynczego niestacjonarnego zajęcia

```
CREATE PROCEDURE AddOnlineClass
 @Link varchar(255),
 @Synch bit,
 @TeacherID int,
 @SubjectID int,
 @StartTime datetime,
 @EndTime datetime,
 @ClassPrice money = NULL,
 @ModuleID int = NULL,
 @PractiseID int = NULL,
 @NewClassID int OUTPUT

AS
BEGIN
 SET NOCOUNT ON;
 BEGIN TRY
     IF NOT EXISTS (SELECT 1 FROM Teachers WHERE TeacherID = @TeacherID)
     BEGIN
         THROW 50000, 'TeacherID does not exist in the Teachers table.', 1;
     END;

     IF NOT EXISTS (SELECT 1 FROM Subjects WHERE SubjectID = @SubjectID)
     BEGIN
         THROW 50000, 'SubjectID does not exist in the Subjects table.', 1;
     END;

     IF (@ModuleID IS NOT NULL AND @PractiseID IS NOT NULL)
     BEGIN
         THROW 50000, 'Can`t define both ModuleID and PractiseID', 1;
     END;

     IF @ModuleID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
     BEGIN
         THROW 50000, 'ModuleID does not exist in the Modules table.', 1;
     END;

     IF @PractiseID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Practises WHERE PractiseID = @PractiseID)
     BEGIN
         THROW 50000, 'PractiseID does not exist in the Practises table.', 1;
     END;

     INSERT INTO Classes (TeacherID, SubjectID, StartTime, EndTime, ClassPrice)
     VALUES (@TeacherID, @SubjectID, @StartTime, @EndTime, @ClassPrice);

     SELECT @NewClassID = ISNULL(MAX(ClassID), 0)

     INSERT INTO OnlineClasses (ClassID, Link, Synch)
     VALUES (@NewClassID, @Link, @Synch)

     IF @ModuleID IS NOT NULL
     BEGIN
         UPDATE Classes SET ModuleID = @ModuleID WHERE ClassID = @NewClassID
     END;

     IF @PractiseID IS NOT NULL
     BEGIN
         UPDATE Classes SET PractiseID = @PractiseID WHERE ClassID = @NewClassID
     END;
     PRINT 'OnlineClass added successfully.';
 END TRY
 BEGIN CATCH
     DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
     THROW 52000, @msg, 1;
 END CATCH
END;
```

11. Dodanie nowego pojedynczego stacjonarnego zajęcia

```
CREATE PROCEDURE AddOfflineClass
  @RoomNumber int,
  @MaxParticipants int,
  @TeacherID int,
  @SubjectID int,
  @StartTime datetime,
  @EndTime datetime,
  @ClassPrice money = NULL,
  @ModuleID int,
  @PractiseID int = NULL,
  @NewClassID int OUTPUT

AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
      IF NOT EXISTS (SELECT 1 FROM Teachers WHERE TeacherID = @TeacherID)
      BEGIN
          THROW 50000, 'TeacherID does not exist in the Teachers table.', 1;
      END;


      IF NOT EXISTS (SELECT 1 FROM Subjects WHERE SubjectID = @SubjectID)
      BEGIN
          THROW 50000, 'SubjectID does not exist in the Subjects table.', 1;
      END;


      IF @ModuleID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
      BEGIN
          THROW 50000, 'ModuleID does not exist in the Modules table.', 1;
      END;


      IF @PractiseID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Practises WHERE PractiseID = @PractiseID)
      BEGIN
          THROW 50000, 'PractiseID does not exist in the Practises table.', 1;
      END;


      -- Sprawdzenie, czy w ramach tych studiów można dodać zajęcia z taką maksymalną ilością miejsc
      -- (musi ona być większa bądź równa od maksymalnej ilości miejsc dla studiów, żeby wszystkie studenty
      -- mogli się zmieszcić, a dodatkowo mogą pojawić się miejsca dla osób "z zewnątrz"
      DECLARE @MinParticipants INT;
      SET @MinParticipants = dbo.CalculateMinClassParticipantsForStudies (@ModuleID);


      IF @MaxParticipants < @MinParticipants
      BEGIN
          THROW 50000, 'MaxParticipants of class should be greater or equal to MaxParticipants of studies within
which classes take place.', 1;
      END;

      INSERT INTO Classes (TeacherID, SubjectID, StartTime, EndTime, ClassPrice, ModuleID)
      VALUES (@TeacherID, @SubjectID, @StartTime, @EndTime, @ClassPrice, @ModuleID);

      SELECT @NewClassID = ISNULL(MAX(ClassID), 0)

      INSERT INTO OfflineClasses (ClassID, RoomNumber, MaxParticipants)
      VALUES (@NewClassID, @RoomNumber, @MaxParticipants)


      IF @PractiseID IS NOT NULL
      BEGIN
          UPDATE Classes SET PractiseID = @PractiseID WHERE ClassID = @NewClassID
      END;
```

```
        PRINT 'OfflineClass added successfully.';
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END;
```

12. Dodanie nowego webinaru

```
CREATE PROCEDURE AddWebinar
    @ProgramName varchar(100),
    @Language varchar(20),
    @ProgramStart date,
    @ProgramEnd date,
    @ProgramPrice money,
    @LecturerID int,
    @TranslatorID int = NULL,


    @Link varchar(255),
    @Synch bit,
    @SubjectID int,
    @StartTime datetime,
    @EndTime datetime,
    @ClassPrice money = NULL,
    @ModuleID int = NULL,
    @PractiseID int = NULL


AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        DECLARE @NewClassID int;
        DECLARE @NewWebinarID int;

        IF NOT EXISTS (SELECT 1 FROM Teachers WHERE TeacherID = @LecturerID)
        BEGIN
            THROW 50000, 'LecturerID does not exist in the Teachers table.', 1;
        END;


        IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Translators WHERE TranslatorID = @TranslatorID)
        BEGIN
            THROW 50000, 'TranslatorID does not exist in the Translators table.', 1;
        END;


        EXEC AddOnlineClass @Link, @Synch, @LecturerID, @SubjectID, @StartTime, @EndTime, @ClassPrice, @ModuleID,
@PractiseID, @NewClassID OUTPUT

        INSERT INTO Webinars (ClassID)
        VALUES (@NewClassID);

        SELECT @NewWebinarID = ISNULL(MAX(WebinarID), 0)

        INSERT INTO EducationalPrograms (ProgramName, WebinarID, Language, ProgramStart, ProgramEnd, ProgramPrice,
LecturerID, TranslatorID)
        VALUES (@ProgramName, @NewWebinarID, @Language, @ProgramStart, @ProgramEnd, @ProgramPrice, @LecturerID,
@TranslatorID);


        COMMIT TRAN
        PRINT 'Webinar added successfully.';
    END TRY
```

```
  BEGIN CATCH
      ROLLBACK TRANSACTION
      DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
      THROW 52000, @msg, 1;
  END CATCH
END;
```

13. Dodanie nowego kursu

```
CREATE PROCEDURE AddCourse
  @ProgramName varchar(100),
  @Language varchar(20),
  @ProgramStart date,
  @ProgramEnd date,
  @ProgramPrice money,
  @LecturerID int,
  @TranslatorID int = NULL,
  @Place varchar(20),
  @Advance money


AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      DECLARE @NewCourseID int;

      IF NOT EXISTS (SELECT 1 FROM Teachers WHERE TeacherID = @LecturerID)
      BEGIN
          THROW 50000, 'LecturerID does not exist in the Teachers table.', 1;
      END;


      IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Translators WHERE TranslatorID = @TranslatorID)
      BEGIN
          THROW 50000, 'TranslatorID does not exist in the Translators table.', 1;
      END;


      INSERT INTO Courses (Place, Advance)
      VALUES (@Place, @Advance);

      SELECT @NewCourseID = ISNULL(MAX(CourseID), 0)

      INSERT INTO EducationalPrograms (ProgramName, CourseID, Language, ProgramStart, ProgramEnd, ProgramPrice,
LecturerID, TranslatorID)
      VALUES (@ProgramName, @NewCourseID, @Language, @ProgramStart, @ProgramEnd, @ProgramPrice, @LecturerID,
@TranslatorID);

      COMMIT TRAN
      PRINT 'Course added successfully.';
  END TRY
  BEGIN CATCH
      ROLLBACK TRANSACTION
      DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
      THROW 52000, @msg, 1;
  END CATCH
END;
```

14. Dodanie nowych studiów

```
CREATE PROCEDURE AddStudies
  @ProgramName varchar(100),
```

```
    @Language varchar(20),
    @ProgramStart date,
    @ProgramEnd date,
    @ProgramPrice money,
    @LecturerID int,
    @TranslatorID int = NULL,
    @Syllabus varchar(255),
    @Place varchar(20),
    @MaxParticipants int,
    @EntryFee money


AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        DECLARE @NewStudiesID int;

        IF NOT EXISTS (SELECT 1 FROM Teachers WHERE TeacherID = @LecturerID)
        BEGIN
            THROW 50000, 'LecturerID does not exist in the Teachers table.', 1;
        END;

        IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Translators WHERE TranslatorID = @TranslatorID)
        BEGIN
            THROW 50000, 'TranslatorID does not exist in the Translators table.', 1;
        END;

        INSERT INTO Studies (Syllabus, Place, MaxParticipants, EntryFee)
        VALUES (@Syllabus, @Place, @MaxParticipants, @EntryFee);

        SELECT @NewStudiesID = ISNULL(MAX(StudiesID), 0)

        INSERT INTO EducationalPrograms (ProgramName, StudiesID, Language, ProgramStart, ProgramEnd, ProgramPrice,
LecturerID, TranslatorID)
        VALUES (@ProgramName, @NewStudiesID, @Language, @ProgramStart, @ProgramEnd, @ProgramPrice, @LecturerID,
@TranslatorID);

        COMMIT TRAN
        PRINT 'Studies added successfully.';


    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION
        DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
        THROW 52000, @msg, 1;
    END CATCH
END;
```

15. Zmiana szczegółów programu edukacyjnego

```
CREATE PROCEDURE UpdateEducationalProgram
   @ProgramID INT,
   @NewProgramName VARCHAR(100) = NULL,
   @NewLanguage VARCHAR(20) = NULL,
   @NewProgramStart DATE = NULL,
   @NewProgramEnd DATE = NULL,
   @NewProgramPrice MONEY = NULL,
   @NewLecturerID INT = NULL,
   @NewTranslatorID INT = NULL,
   @NewSyllabus VARCHAR(255) = NULL,
   @NewStudiesPlace VARCHAR(100) = NULL,
   @NewMinParticipants INT = NULL,
   @NewEntryFee MONEY = NULL,
   @NewCoursesPlace VARCHAR(40) = NULL,
   @NewAdvance MONEY = NULL,
   @NewClassID INT = NULL
AS
BEGIN
 BEGIN TRY
   SET NOCOUNT ON;
   DECLARE @IsCourse BIT, @IsWebinar BIT, @IsStudies BIT


   -- Sprawdzenie typu programu na podstawie ProgramID
   SELECT @IsCourse = IIF(EXISTS (SELECT 1 FROM EducationalPrograms WHERE ProgramID = @ProgramID and CourseID IS
NOT NULL), 1, 0),
          @IsWebinar = IIF(EXISTS (SELECT 1 FROM EducationalPrograms WHERE ProgramID = @ProgramID and WebinarID
IS NOT NULL), 1, 0),
          @IsStudies = IIF(EXISTS (SELECT 1 FROM EducationalPrograms WHERE ProgramID = @ProgramID and StudiesID
IS NOT NULL), 1, 0)


   IF @IsStudies = 1 AND (@NewSyllabus IS NOT NULL OR @NewStudiesPlace IS NOT NULL OR @NewMinParticipants IS NOT
NULL OR @NewEntryFee IS NOT NULL)
   BEGIN
       UPDATE Studies
       SET Syllabus = ISNULL(@NewSyllabus, Syllabus),
           Place = ISNULL(@NewStudiesPlace, Place),
           MaxParticipants = ISNULL(@NewMinParticipants, MaxParticipants),
           EntryFee = ISNULL(@NewEntryFee, EntryFee)
       FROM Studies
       JOIN EducationalPrograms ON Studies.StudiesID = EducationalPrograms.StudiesID
       WHERE EducationalPrograms.ProgramID = @ProgramID;
   END
   ELSE IF @IsStudies = 0 AND (@NewSyllabus IS NOT NULL OR @NewStudiesPlace IS NOT NULL OR @NewMinParticipants IS
NOT NULL OR @NewEntryFee IS NOT NULL)
       THROW 52313, N'EducationalProgram is not ranked in Studies', 10;


   ELSE IF @IsCourse = 1 AND (@NewCoursesPlace IS NOT NULL OR @NewAdvance IS NOT NULL)
   BEGIN
       UPDATE Courses
       SET Place = ISNULL(@NewCoursesPlace, Place),
           Advance = ISNULL(@NewAdvance, Advance)
       FROM Courses
       JOIN EducationalPrograms ON Courses.CourseID = EducationalPrograms.CourseID
       WHERE EducationalPrograms.ProgramID = @ProgramID;
   END
   ELSE IF @IsCourse = 0 AND (@NewCoursesPlace IS NOT NULL OR @NewAdvance IS NOT NULL)
       THROW 52313, N'EducationalProgram is not ranked in Courses', 10;


   ELSE IF @IsWebinar = 1 AND @NewClassID IS NOT NULL
   BEGIN
       IF NOT EXISTS (SELECT 1 FROM Classes WHERE ClassID = @NewClassID)
           THROW 52314, N'NewClassID does not exist in Classes', 10;
       UPDATE Webinars
```

```
        SET ClassID = @NewClassID
        FROM Webinars
        JOIN EducationalPrograms ON Webinars.WebinarID = EducationalPrograms.WebinarID
        WHERE EducationalPrograms.ProgramID = @ProgramID;
    END
    ELSE IF @IsWebinar = 0 AND @NewClassID IS NOT NULL
        THROW 52313, N'EducationalProgram is not ranked in Webinars', 10;


    IF (@NewProgramName IS NOT NULL OR @NewLanguage IS NOT NULL OR @NewProgramStart IS NOT NULL OR
        @NewProgramEnd IS NOT NULL OR @NewProgramPrice IS NOT NULL OR @NewLecturerID IS NOT NULL OR
        @NewTranslatorID IS NOT NULL)
    BEGIN
        IF @NewLecturerID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Teachers WHERE TeacherID = @NewLecturerID)
            THROW 52314, N'NewLecturerID does not exist in Teachers', 10;
        IF @NewTranslatorID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Translators WHERE TranslatorID =
@NewTranslatorID)
            THROW 52314, N'NewTranslatorID does not exist in Translators', 10;
        UPDATE EducationalPrograms
        SET ProgramName = ISNULL(@NewProgramName, ProgramName),
            Language = ISNULL(@NewLanguage, Language),
            ProgramStart = ISNULL(@NewProgramStart, ProgramStart),
            ProgramEnd = ISNULL(@NewProgramEnd, ProgramEnd),
            ProgramPrice = ISNULL(@NewProgramPrice, ProgramPrice),
            LecturerID = ISNULL(@NewLecturerID, LecturerID),
            TranslatorID = ISNULL(@NewTranslatorID, TranslatorID)
        WHERE ProgramID = @ProgramID
    END
 END TRY
 BEGIN CATCH
    DECLARE @Message NVARCHAR(1000) = N'error: ' + ERROR_MESSAGE();
    THROW 123456, @Message, 10;
 END CATCH
END;
```

16. Dodanie Płatności do złożonego zamówienia

```
CREATE PROCEDURE AddPayment
    @OrderID INT,
    @SystemPaymentID VARCHAR(255),
    @PayFull Bit
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS (SELECT * FROM Orders WHERE OrderID = @OrderID)
        BEGIN
            THROW 51234, N'There is no order with such ID', 1;
        END
        BEGIN
            DECLARE @price INT;
            IF @PayFull = 1
            BEGIN
            SELECT @price = dbo.CalculateFullPriceForOrder(@OrderID)
            END

            ELSE
            BEGIN
            SELECT @price = dbo.CalculateEntryPriceForOrder(@OrderID)
            END

            FROM Payments
            INSERT INTO Payments (OrderId, Amount, Date, Status, SystemPaymentID )
            VALUES (@OrderID, @price, GETDATE(), 0, @SystemPaymentID)
        END
    END TRY


    BEGIN CATCH
```

```
        DECLARE @ErrorMessage NVARCHAR(1000) = N'Error: ' + ERROR_MESSAGE();
        THROW 52011, @ErrorMessage, 1;
    END CATCH
END;
```

17. Dodanie nowych offline zajęć w ramach studiów

```
CREATE PROCEDURE AddStudiesOfflineClasses
  @StudiesID int,
  @RoomNumber int,
  @MaxParticipants int,
  @TeacherID int,
  @SubjectID int,
  @StartTime datetime,
  @EndTime datetime,
  @ClassPrice money = NULL,
  @ModuleID int,
  @PractiseID int = NULL

AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      DECLARE @StudiesMaxParticipants int;
      DECLARE @NewClassID int;

      SELECT @StudiesMaxParticipants = (
              SELECT MaxParticipants from Studies
              where StudiesID = @StudiesID)
      IF (@StudiesMaxParticipants > @MaxParticipants)
      BEGIN
          THROW 50000, 'Amount of each ClassesMaxParticipants should be equal or greater than
StudiesMaxParticipants.', 1;
      END;

      EXEC AddOfflineClass @RoomNumber, @MaxParticipants, @TeacherID, @SubjectID, @StartTime, @EndTime,
@ClassPrice, @ModuleID, @PractiseID, @NewClassID OUTPUT

      COMMIT TRAN
      PRINT 'StudiesOfflineClasses added successfully.';

  END TRY
  BEGIN CATCH
      ROLLBACK TRANSACTION
      DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
      THROW 52000, @msg, 1;
  END CATCH
END;
```

18. Procedura, pozwalająca na aktualizację oceny studenta z egzaminu z konkretnych studiów. W przypadku oceny, która mieści się w zakresie 50%-100% ustawia się zaliczenie danego programu w RegisteredPrograms za pomocą triggera #2.

```
CREATE PROCEDURE SetExamMark(
   @StudentID int,
   @StudiesID int,
   @Mark int)
   AS
   BEGIN
      BEGIN TRY
          IF NOT @MARK BETWEEN 0 AND 100
          THROW 52313, N'Mark should be positive value between 0 and 100', 1;

          IF NOT EXISTS (SELECT 1 FROM Students WHERE StudentID = @StudentID)
          THROW 52313, N'StudentID does not exist in the Students table.', 1;

          IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudiesID = @StudiesID)
          THROW 52313, N'StudiesID does not exist in the Studies table.', 1;

          IF NOT EXISTS (select ss.RegisteredProgramID
              from StudentStudies(@StudentID) as ss
```

```
            where ss.StudiesID = @StudiesID)
        THROW 52313, N'Student is not registered for this studies.', 1;

        IF EXISTS (select 1 FROM Exams where StudentID = @StudentID and StudiesID = @StudiesID)
        BEGIN
            UPDATE EXAMS
            SET Mark = @Mark
            WHERE StudentID = @StudentID and StudiesID = @StudiesID
            PRINT 'Exam updated successfully.';
        END

        ELSE
        BEGIN
            INSERT INTO Exams (StudiesID, StudentID, Mark) VALUES (@StudiesID, @StudentID, @Mark)
            PRINT 'Exam added successfully.';
        END
    END TRY

    BEGIN CATCH
         DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
         THROW 52000, @msg, 1;
    END CATCH
  END
```

19. "Jawna" zmiana statusu dostępu do programu edukacyjnego, którą jawnie ustawiać może wyłącznie dyrektor szkoły.

```
CREATE PROCEDURE SetProgramAccess(
 @RegisteredProgramID int,
 @Status int)
AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY

      IF NOT EXISTS (SELECT 1 FROM RegisteredPrograms WHERE RegisteredProgramID = @RegisteredProgramID)
      BEGIN
        THROW 50000, 'RegisteredProgramID does not exist in the Students table.', 1;
      END;

      IF @Status != 0 AND @Status != 1
      BEGIN;
          throw 52000, N'Status should be equal to "1" to confirm access or "0" to cancel access', 1
      END

      BEGIN
          UPDATE RegisteredPrograms SET Access = @Status WHERE RegisteredProgramID = @RegisteredProgramID
      END

  END TRY
  BEGIN CATCH
    DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
    THROW 52000, @msg, 1;
  END CATCH
END
```

20. "Jawna" zmiana statusu dostępu do pojedynczych zajęć, którą jawnie ustawiać może wyłącznie dyrektor szkoły.

```
CREATE PROCEDURE SetClassAccess(
 @RegisteredClassID int,
 @Status int)
AS
BEGIN
  SET NOCOUNT ON;
  BEGIN TRY
      IF NOT EXISTS (SELECT 1 FROM RegisteredClasses WHERE RegisteredClassID = @RegisteredClassID)
      BEGIN
```

```
          THROW 50000, 'RegisteredClassID does not exist in the RegisteredClasses table.', 1;
      END;

      IF @Status != 0 AND @Status != 1
      BEGIN;
          throw 52000, N'Status should be equal to "1" to confirm access or "0" to cancel access', 1
      END
      BEGIN
          UPDATE RegisteredClasses SET Access = @Status WHERE RegisteredClassID = @RegisteredClassID
      END
  END TRY
  BEGIN CATCH
    DECLARE @msg NVARCHAR(2048) = N'ERROR: ' + ERROR_MESSAGE();
    THROW 52000, @msg, 1;
  END CATCH
END
```

4. **Funkcje**

1. Obliczanie średniej ocen dla studenta

```
CREATE FUNCTION CalculateAverageGradeForStudent
(
  @StudentID int,
  @StudiesID int
)
RETURNS DECIMAL(5, 2)
AS
BEGIN
  DECLARE @AverageGrade DECIMAL(5, 2);

  SELECT @AverageGrade = AVG(Mark)
  FROM Exams
  WHERE StudentID = @StudentID and StudiesID = @StudiesID;

  RETURN ISNULL(@AverageGrade, 0); -- Jeśli nie ma ocen, zwraca 0
END;
```

2. Liczba studentów obecnych na zajęciach

```
CREATE FUNCTION GetClassAttendanceCount
(
   @ClassID INT
)
RETURNS INT
AS
BEGIN
   IF EXISTS (SELECT 1 FROM Attendance WHERE ClassId = @ClassID)
       BEGIN
       DECLARE @AttendanceCount INT;


       SELECT @AttendanceCount = COUNT(*)
       FROM Attendance
       WHERE ClassID = @ClassID AND Present = 1


       RETURN @AttendanceCount;
   END
   RETURN 0
END;
```

3. Obliczanie ilości dni pozostałych do zakończenia programu edukacyjnego

```
CREATE FUNCTION DaysRemainingInProgram
(
   @ProgramID INT
)
RETURNS INT
AS
BEGIN
   DECLARE @DaysRemaining INT;

   SELECT @DaysRemaining = DATEDIFF(DAY, GETDATE(), ProgramEnd)
   FROM EducationalPrograms
   WHERE ProgramID = @ProgramID;

   -- Jeżeli program już się zakończył, zwróć 0
   IF @DaysRemaining < 0
       SET @DaysRemaining = 0;

   RETURN @DaysRemaining;
END;
```

4. Obliczanie sumy pełnych kwot za wszystkie programy na danym zamówieniu

```
CREATE FUNCTION CalculateFullPriceForOrder
(
    @OrderID int
)
RETURNS MONEY
AS
BEGIN
    DECLARE @fullprice MONEY;
    SELECT @fullprice =
    SUM(ISNULL(EP.ProgramPrice,0)) + SUM(ISNULL(C.ClassPrice,0))
    FROM Orders O
    LEFT JOIN RegisteredPrograms RP ON RP.OrderID = O.OrderID
    LEFT JOIN RegisteredClasses RC ON O.OrderID = RC.OrderID
    LEFT JOIN Classes C ON RC.ClassID = C.ClassID
    LEFT JOIN EducationalPrograms EP ON EP.ProgramID = RP.ProgramID
    GROUP BY O.OrderID
    HAVING O.OrderId = @OrderID

    RETURN ISNULL(@fullprice, 0)

END;
```

5. Obliczanie sum cen wpisowych na programy na danym zamówieniu

```
CREATE FUNCTION CalculateEntryPriceForOrder
(
    @OrderID int
)
RETURNS MONEY
AS
BEGIN
    DECLARE @entryprice MONEY;
    SELECT @entryprice =
    SUM(ISNULL(S.EntryFee,0)) + SUM(ISNULL(CS.Advance,0)) + SUM(ISNULL(C.ClassPrice,0))
    FROM Orders O
    LEFT JOIN RegisteredPrograms RP ON RP.OrderID = O.OrderID
    LEFT JOIN RegisteredClasses RC ON O.OrderID = RC.OrderID
    LEFT JOIN Classes C ON RC.ClassID = C.ClassID
    LEFT JOIN EducationalPrograms EP ON EP.ProgramID = RP.ProgramID
    LEFT JOIN Studies S ON EP.StudiesID = S.StudiesID
    LEFT JOIN Courses CS ON CS.CourseID = EP.CourseID
    LEFT JOIN Webinars W ON W.WebinarID = EP.WebinarID
    GROUP BY O.OrderID
```

```
    HAVING O.OrderID = @OrderID

    RETURN ISNULL(@entryprice, 0)

END;
```

6. Obliczanie łącznej kwoty wydanej przez danego studenta na Programy edukacyjne za konkretny okres czasowy

```
CREATE FUNCTION CalculateTotalPaymentsForStudent
(
  @StudentID int,
  @startDate date,
  @endDate date
)
RETURNS MONEY
AS
BEGIN
  DECLARE @TotalPayments MONEY;

  SELECT @TotalPayments = SUM(Amount)
  FROM Payments
  WHERE OrderID IN (SELECT OrderID FROM Orders WHERE StudentID = @StudentID) AND status = 1 AND date between
@startDate and @endDate;

  RETURN ISNULL(@TotalPayments, 0);
END;
```

7. Wyświetlanie harmonogramu zajęć na konkretny dzień dla konkretnego studenta

```
CREATE FUNCTION ScheduleForStudent(@StudentID int, @day DATE)
   RETURNS TABLE
       AS
       RETURN
       SELECT Student, ModuleName, SubjectName, Teacher, StartTime, EndTime, RoomNumber
       FROM OfflineParticipantsList as ofp
       WHERE @StudentID = ofp.StudentID and @day = cast(StartTime as date)
```

8. Obliczenie ilości zamówionych przez studentów programów w danym roku

```
CREATE FUNCTION OrdersProgramsAmount(@year int)
   RETURNS INT
   AS
   BEGIN
       DECLARE @Amount INT;
       SET @Amount = (
       select count(*) from orders
       inner join RegisteredPrograms as rp
           on orders.OrderID = rp.OrderID
           where year(cast (OrderDate as DATE))  = @year)
       RETURN @Amount
   END
```

9. Wyświetlenie trwających w tej chwili synchronicznych zajęć niestacjonarnych

```
CREATE FUNCTION LiveOnlineSynchClasses()
   RETURNS TABLE
       AS
       RETURN
       SELECT c.StartTime, c.EndTime, oc.Link from OnlineClasses as oc
```

```
            INNER JOIN Classes as c on oc.ClassID = c.ClassID
            WHERE oc.Synch = 'true' AND GETDATE() between c.StartTime AND c.EndTime
```

10. Obliczenie średniej oceny na pojedynczych zajęciach (tylko w przypadku, gdy ocenę wystawiono każdemu uczestnikowi zajęć)

```
CREATE FUNCTION AverageMarkOnClass(@ClassID int)
   RETURNS INT
   BEGIN
       DECLARE @Average int
       IF NOT EXISTS (SELECT MARK FROM Attendance WHERE MARK is null and @ClassID = Attendance.ClassID)
           SET @Average = (select AVG(MARK) from Attendance where @ClassID = Attendance.ClassID)
       ELSE set @Average = null
       RETURN @Average
   END
```

11. Funkcja sprawdzająca minimalnie możliwej liczby uczęstników zajęć w ramach modułu studiów (jeśli danę zajęcia są dodawane do studiów)

```
CREATE FUNCTION CalculateMinClassParticipantsForStudies(@ModuleID int)
RETURNS INT
AS
BEGIN
  DECLARE @MinParticipants INT;
  SET @MinParticipants = COALESCE((select s.MaxParticipants
   from Modules as m
       inner join EducationalPrograms as ep
           on m.ProgramID = ep.ProgramID
       inner join Studies as s
           on ep.StudiesID = s.StudiesID
       where ModuleID = @ModuleID), 0)
  RETURN @MinParticipants;
END;
```

12. Wyświetlenie listy osób zapisanych na dane offline-wydarzenie w ramach studiów

```
CREATE FUNCTION AllClassParticipants(@ClassID int)
    RETURNS TABLE
        AS
        RETURN
        with t as (
        select c.ClassID, s.StudiesID
        from Classes as c
            inner join Modules as m
                on m.ModuleID = c.ModuleID
            inner join EducationalPrograms as ep
                on m.ProgramID = ep.ProgramID
            inner join Studies as s
                on ep.StudiesID = s.StudiesID
        )

        select StudentID, Student, StudiesID
            from OfflineParticipantsList
                inner join t
                    on t.ClassID = OfflineParticipantsList.ClassID and t.ClassID = @ClassID
        UNION
        select StudentID, Student, StudiesID
            from StudentsOuterClasses
                inner join t
                    on t.ClassID = StudentsOuterClasses.ClassID and t.ClassID = @ClassID
```

13. Wyświetlenie listy osób zapisanych na dany program

```
CREATE FUNCTION AllProgramParticipants(@ProgramID int)
   RETURNS TABLE
       AS
       RETURN
       select distinct StudentID, Student
       from StudentsPrograms as sp
           inner join EducationalPrograms as ep
               on ep.ProgramID = sp.ProgramID
       where ep.ProgramID = @ProgramID
```

14. Wyświetlenie listy studiów, na które dany student jest zapisany i ma dostęp. Funkcja wykorzystywana jest w procedurze #18 oraz triggerze #2

```
CREATE FUNCTION StudentStudies(@StudentID int)
    RETURNS TABLE
        AS
        RETURN
        select StudentID, Student, RegisteredProgramID, StudiesID, sp.ProgramName, sp.ProgramStart,
sp.ProgramEnd, sp.Passed
        from StudentsPrograms as sp
            inner join EducationalPrograms as ep
                on ep.ProgramID = sp.ProgramID
        where @StudentID = sp.StudentID and StudiesID is not null and access = 'true'
```

15. Wyświetlenie listy webinarów, na które dany student jest zapisany i ma dostęp

```
CREATE FUNCTION StudentWebinars(@StudentID int)
    RETURNS TABLE
        AS
        RETURN
        select StudentID, Student, RegisteredProgramID, WebinarID, sp.ProgramName, sp.ProgramStart,
sp.ProgramEnd, sp.Passed
        from StudentsPrograms as sp
            inner join EducationalPrograms as ep
                on ep.ProgramID = sp.ProgramID
        where @StudentID = sp.StudentID and WebinarID is not null and access = 'true'
```

16. Wyświetlenie listy kursów, na które dany student jest zapisany i ma dostęp

```
CREATE FUNCTION StudentCourses(@StudentID int)
    RETURNS TABLE
        AS
        RETURN
        select StudentID, Student, RegisteredProgramID, CourseID, sp.ProgramName, sp.ProgramStart, sp.ProgramEnd,
sp.Passed
        from StudentsPrograms as sp
            inner join EducationalPrograms as ep
                on ep.ProgramID = sp.ProgramID
        where @StudentID = sp.StudentID and CourseID is not null and access = 'true'
```

5. **Triggery**

1. Aktualiacja stanu zapłaty zamówienia po udanej transakcji w tabeli Payments.

```
CREATE TRIGGER trg_UpdateOrderStatus
ON Payments
AFTER UPDATE
AS
BEGIN
```

```
    SET NOCOUNT ON;
    IF UPDATE(Status)
    BEGIN
        IF (SELECT Status FROM INSERTED) = 1
        BEGIN
            DECLARE @OrderID INT;
            DECLARE @PaymentAmount MONEY;


            SELECT @OrderID = i.OrderID
            FROM INSERTED i;


            SELECT @PaymentAmount = i.Amount
            FROM INSERTED i;


            IF(@PaymentAmount = dbo.CalculateFullPriceForOrder(@OrderID))
            BEGIN
                UPDATE Orders
                SET OrderStatus = 'FULL PAID'
                WHERE OrderID = @OrderID
            END


            ELSE
            BEGIN
                IF(@PaymentAmount = dbo.CalculateFullPriceForOrder(@OrderID))
                BEGIN
                    UPDATE Orders
                    SET OrderStatus = 'ENTRY PAID'
                    WHERE OrderID = @OrderID
                END
                ELSE
                BEGIN
                    UPDATE Orders
                    SET OrderStatus = 'NOT PAID'
                    WHERE OrderID = @OrderID
                END
            END
        END
    END
END;
```

2. Ustawienie statusu zaliczenia konkretnych studiów konkretnego studenta w przypadku aktualizacji/dodawaniu jego oceny za egzamin, która musi mieścić się w zakresie od 50% do 100%.

```
CREATE TRIGGER PassStudies
ON Exams
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON
    IF (select Mark from inserted) >= 50
    BEGIN
        DECLARE @StudiesID INT
        DECLARE @StudentID INT
        SELECT @StudiesID = StudiesID FROM INSERTED
        SELECT @StudentID = StudentID FROM INSERTED
        UPDATE RegisteredPrograms set Passed = 'true'
        WHERE RegisteredPrograms.RegisteredProgramID in
              (select ss.RegisteredProgramID
               from StudentStudies(@StudentID) as ss
                where ss.StudiesID = @StudiesID)
    END
END
```

3. Trigger, który uniemożliwia zapisywanie się na zajęcia, na których już nie ma miejsc.

```
CREATE TRIGGER ClassesParticipantsAmountCheck
ON RegisteredClasses
AFTER INSERT
AS
BEGIN
   SET NOCOUNT ON
   BEGIN
       DECLARE @ClassID INT
       DECLARE @MaxParticipants INT
       SELECT @ClassID = ClassID FROM INSERTED
       SELECT @MaxParticipants = MaxParticipants from OfflineClasses where ClassID = @ClassID
       IF (SELECT COUNT(*) FROM AllClassParticipants(@ClassID)) > @MaxParticipants
       BEGIN
           THROW 52313, N'Participants number over MaxParticipants number for this Classes', 1;
       END
   END
END
```

4. Trigger, który uniemożliwia rejestrowanie na dany program edukacyjny jeśli on jest studiami, w przypadku przekroczenia maksymalnie możliwej liczby uczęstników.

```
CREATE TRIGGER StudiesParticipantsAmountCheck
ON RegisteredPrograms
AFTER INSERT
AS
BEGIN
   SET NOCOUNT ON
   BEGIN
       DECLARE @ProgramID INT
       DECLARE @MaxParticipants INT
       SELECT @ProgramID = ProgramID FROM INSERTED
       IF (SELECT StudiesID from EducationalPrograms where ProgramID = @ProgramID) IS NOT NULL
           SELECT @MaxParticipants = MaxParticipants
                           from EducationalPrograms as ep
                               inner join Studies as st
                                   on ep.StudiesID = st.StudiesID
                                   where ProgramID = @ProgramID
           IF (SELECT COUNT(*) FROM AllProgramParticipants(@ProgramID)) > @MaxParticipants
           BEGIN
               THROW 52313, N'Participants number over MaxParticipants number for this Studies', 1;
           END
   END
END
```

6. **Indeksy**

```
CREATE INDEX idx_student_firstname_lastname
ON Students (FirstName, LastName);

CREATE INDEX idx_teacher_firstname_lastname
ON Teachers (FirstName, LastName);

CREATE INDEX idx_translator_firstname_lastname
ON Translators (FirstName, LastName);

CREATE INDEX idx_orders_studentID
ON Orders (StudentID);

CREATE INDEX idx_registeredPrograms_orderID
ON RegisteredPrograms (OrderID);

CREATE INDEX idx_registeredPrograms_programID
ON RegisteredPrograms (ProgramID);
```

```
CREATE INDEX idx_registeredClasses_orderID
ON RegisteredClasses (OrderID);

CREATE INDEX idx_registeredClasses_classID
ON RegisteredClasses (ClassID);

CREATE INDEX idx_program_dates
ON EducationalPrograms (ProgramStart, ProgramEnd);

CREATE INDEX idx_subjects_categoryID
ON Subjects (CategoryID);

CREATE INDEX idx_classes_time
ON Classes (StartTime, EndTime);

CREATE INDEX idx_OnlineClasses_ClassID
ON OnlineClasses (ClassID)

CREATE INDEX idx_OfflineClasses_ClassID
ON OfflineClasses (ClassID)

CREATE INDEX idx_exam_studentID
ON Exams (StudentID);

CREATE INDEX idx_modules_programID
ON Modules (ProgramID);

CREATE INDEX idx_practises_studiesID
ON Practises (StudiesID);

CREATE INDEX idx_webinars_classID
ON Webinars (ClassID);

CREATE INDEX idx_classes_teacherID
ON Classes (TeacherID);

CREATE INDEX idx_classes_subjectID
ON Classes (SubjectID);

CREATE INDEX idx_attendance_classID
ON Attendance (ClassID);

CREATE INDEX idx_attendance_participantID
ON Attendance (ParticipantID);

CREATE INDEX idx_Classes_ModuleID
ON Classes (ModuleID)

CREATE INDEX idx_Payments_OrderID
ON Payments (OrderID)

CREATE INDEX idx_exam_studentID
ON Exams (StudentID);
```

7. **Uprawnienia**

Administrator

```
CREATE ROLE admin
GRANT ALL PRIVILEGES ON u_smyka.dbo to admin
```

Administrator

```
CREATE ROLE admin
GRANT ALL PRIVILEGES ON u_smyka.dbo to admin
```

Dyrektor szkoły

```
CREATE ROLE director

GRANT SELECT ON WebinarsRevenue to director
GRANT SELECT ON CoursesRevenue to director
GRANT SELECT ON StudiesRevenue to director
GRANT SELECT ON Debtors to director
GRANT SELECT ON NumOfInterestedInFutureClasses to director
GRANT SELECT ON NumOfInterestedInFutureEducationalPrograms to director
GRANT SELECT ON OfflineParticipantsList to director
GRANT SELECT ON AttendanceAllClasses to director
GRANT SELECT ON BilocationsList to director
GRANT SELECT ON NumberOfParticipations to director
GRANT SELECT ON ExamDetails to director
GRANT SELECT ON StudiesSubjectsInfo to director
GRANT SELECT ON CoursesSubjectsInfo to director
GRANT SELECT ON WebinarsInfo to director
GRANT SELECT ON StudentsPrograms to director
GRANT SELECT ON StudentsOuterClasses to director


GRANT EXECUTE ON AddStudent to director
GRANT EXECUTE ON DeleteStudent to director
GRANT EXECUTE ON AddCourse to director
GRANT EXECUTE ON ChangeStudentData to director
GRANT EXECUTE ON AddTeacher to director
GRANT EXECUTE ON AddOnlineClass to director
GRANT EXECUTE ON AddOfflineClass to director
GRANT EXECUTE ON RedoAttendance to director
GRANT EXECUTE ON AddStudies to director
GRANT EXECUTE ON UpdateEducationalProgram to director
GRANT EXECUTE ON AddStudiesOfflineClasses to director
GRANT EXECUTE ON SetProgramAccess to director
GRANT EXECUTE ON SetClassesAccess to director
GRANT EXECUTE ON SetExamMark to director


GRANT EXECUTE ON CalculateAverageGradeForStudent to director
GRANT EXECUTE ON GetClassAttendanceCount to director
GRANT EXECUTE ON DaysRemainingInProgram to director
GRANT EXECUTE ON CalculateFullPriceForOrder to director
GRANT EXECUTE ON CalculateEntryPriceForOrder to director
GRANT EXECUTE ON OrdersProgramsAmount to director
GRANT EXECUTE ON AverageMarkOnClass to director
GRANT EXECUTE ON CalculateMinClassParticipantsForStudies to director


GRANT SELECT ON ScheduleForStudent to director
GRANT SELECT ON LiveOnlineSynchClasses to director
GRANT SELECT ON StudentStudies to director
GRANT SELECT ON StudentWebinars to director
GRANT SELECT ON StudentCourses to director
GRANT SELECT ON AllClassParticipants to director
GRANT SELECT ON AllProgramParticipants to director
```

Pracownik systemow (moderator)

```
CREATE ROLE moderator

GRANT SELECT ON WebinarsRevenue to moderator
GRANT SELECT ON CoursesRevenue to moderator
GRANT SELECT ON StudiesRevenue to moderator
GRANT SELECT ON Debtors to moderator
GRANT SELECT ON NumOfInterestedInFutureClasses to moderator
GRANT SELECT ON NumOfInterestedInFutureEducationalPrograms to moderator
GRANT SELECT ON OfflineParticipantsList to moderator
GRANT SELECT ON AttendanceAllClasses to moderator
```

```
GRANT SELECT ON BilocationsList to moderator
GRANT SELECT ON NumberOfParticipations to moderator
GRANT SELECT ON ExamDetails to moderator
GRANT SELECT ON StudiesSubjectsInfo to moderator
GRANT SELECT ON CoursesSubjectsInfo to moderator
GRANT SELECT ON WebinarsInfo to moderator
GRANT SELECT ON StudiesSubjectsInfo to moderator
GRANT SELECT ON StudentsPrograms to moderator
GRANT SELECT ON StudentsOuterClasses to moderator


GRANT EXECUTE ON AddStudent to moderator
GRANT EXECUTE ON DeleteStudent to moderator
GRANT EXECUTE ON AddCourse to moderator
GRANT EXECUTE ON ChangeStudentData to moderator
GRANT EXECUTE ON AddTeacher to moderator
GRANT EXECUTE ON AddOnlineClass to moderator
GRANT EXECUTE ON AddOfflineClass to moderator
GRANT EXECUTE ON AddWebinar to moderator
GRANT EXECUTE ON RedoAttendance to moderator
GRANT EXECUTE ON AddStudies to moderator
GRANT EXECUTE ON UpdateEducationalProgram to moderator
GRANT EXECUTE ON AddStudiesOfflineClasses to moderator


GRANT EXECUTE ON CalculateAverageGradeForStudent to moderator
GRANT EXECUTE ON GetClassAttendanceCount to moderator
GRANT EXECUTE ON DaysRemainingInProgram to moderator
GRANT EXECUTE ON CalculateFullPriceForOrder to moderator
GRANT EXECUTE ON CalculateEntryPriceForOrder to moderator
GRANT EXECUTE ON OrdersProgramsAmount to moderator
GRANT EXECUTE ON AverageMarkOnClass to moderator
GRANT EXECUTE ON CalculateMinClassParticipantsForStudies to moderator


GRANT SELECT ON ScheduleForStudent to moderator
GRANT SELECT ON LiveOnlineSynchClasses to moderator
GRANT SELECT ON StudentStudies to moderator
GRANT SELECT ON StudentWebinars to moderator
GRANT SELECT ON StudentCourses to moderator
GRANT SELECT ON AllClassParticipants to moderator
GRANT SELECT ON AllProgramParticipants to moderator
```

Pracownik naukowy (teacher & translator)

```
CREATE ROLE educator


GRANT SELECT ON OfflineParticipantsList to educator
GRANT SELECT ON AttendanceAllClasses to educator
GRANT SELECT ON NumberOfParticipations to educator
GRANT SELECT ON ExamDetails to educator
GRANT SELECT ON StudiesSubjectsInfo to educator
GRANT SELECT ON CoursesSubjectsInfo to educator
GRANT SELECT ON WebinarsInfo to educator
GRANT SELECT ON StudentsOuterClasses to educator


GRANT EXECUTE ON RedoAttendance to educator


GRANT EXECUTE ON CalculateAverageGradeForStudent to educator
GRANT EXECUTE ON GetClassAttendanceCount to educator
GRANT EXECUTE ON DaysRemainingInProgram to educator
GRANT EXECUTE ON AverageMarkOnClass to educator


GRANT SELECT ON LiveOnlineSynchClasses to educator
GRANT SELECT ON AllProgramParticipants to educator
```

Student

```
CREATE ROLE student

GRANT SELECT ON OfflineParticipantsList to student
GRANT SELECT ON BilocationsList to student
GRANT SELECT ON ExamDetails to student
GRANT SELECT ON StudiesSubjectsInfo to student
GRANT SELECT ON CoursesSubjectsInfo to student
GRANT SELECT ON WebinarsInfo to student
GRANT SELECT ON StudentsPrograms to student


GRANT EXECUTE ON AddOrder to student
GRANT EXECUTE ON RegisterClass to student
GRANT EXECUTE ON RegisterProgram to student
GRANT EXECUTE ON AddPayment to student


GRANT EXECUTE ON CalculateAverageGradeForStudent to student
GRANT EXECUTE ON DaysRemainingInProgram to student
GRANT EXECUTE ON AverageMarkOnClass to student


GRANT SELECT ON ScheduleForStudent to student
GRANT SELECT ON StudentStudies to student
GRANT SELECT ON StudentWebinars to student
GRANT SELECT ON StudentCourses to student
GRANT SELECT ON AllClassParticipants to student
GRANT SELECT ON AllProgramParticipants to student
```

8. **Generator danych**

Do napisania generatora danych posłużyliśmy się językiem Python. Do komunikacji z bazą danych wykorzystana została biblioteka pyodbc, a do generowania losowych wartości biblioteka Faker.