Podstawy Baz Danych

Zespół nr 3 (śr. 16:45): Maciej Wiśniewski, Konrad Szymański, Kajetan Frątczak

Funkcje Systemu

- zlicza frekwencję użytkowników
- zalicza moduł prowadzony online asynchronicznie
- udostępnia nagranie użytkownikowi
    - po ukończeniu webinaru i kursu online synchronicznego udostępnia użytkownikowi nagranie na okres 30 dni
- generuje raporty i listy na żądanie uprawnionych użytkowników:
    - generowanie raportu finansowego
    - generowanie listy osób zalegających z opłatami
    - generowanie raportu o liczbie osób zapisanych na przyszłe wydarzenia
    - generowanie raportu o frekwencji
    - generowanie listy obecności
    - generowanie raportu o kolizjach użytkowników - informuje użytkownika zapisanego na dwa wydarzenia odbywające się w tym samym czasie
- przechowuje produkty w koszyku
- zapisuje wybrane produkty przez użytkownika i pozwala na płatność za wszystkie naraz
- weryfikuje terminowe dopłaty
- blokuje dostęp do kursu/studiów, jeśli użytkownik nie zapłaci 3 dni przed ich rozpoczęciem
- pilnuje limitów osób na kursach oraz studiach
- uniemożliwia dodanie użytkownika przez koordynatora przedmiotu, gdy nie ma już miejsc
- umożliwia zakładanie/edycję konta
- przechowuje dane kontaktowe
- przetrzymuje dane kursów, studiów, webinarów i użytkowników

Użytkownicy i funkcje jakie mogą realizować

Użytkownicy systemu:
- Uczestnik kursu (student, uczestnik kursu, uczestnik webinaru, uczestnik pojedynczego spotkania studyjnego)
- Administrator
- Dyrektor Szkoły
- Koordynator (studiów, przedmiotu, kursu, webinaru)
- Wykładowca
- Prowadzący praktyki
- Księgowy
- Tłumacz
- Osoba niewidniejąca w bazie (bez konta)

☐ Uczestnik kursu
- Zakłada konto
- Edytuje (dane niewrażliwe, np. adres korespondencyjny)/usuwa konto
- Zapisuje się na bezpłatne webinary
- Zapisuje się na płatne webinary/studia/kursy/spotkania studyjne
- Dodaje/usuwa zajęcia do/z koszyka
- Sprawdza swój harmonogram zajęć
- Sprawdza swoje oceny
- Generuje raport swojej obecności na zajęciach
- Ma dostęp do materiałów z zajęć prowadzonych online (na okres 30 dni)
- Może zrezygnować ze studiów

☐ Administrator
- Modyfikuje dane użytkowników
- Modyfikuje dane kursów, studiów i webinarów (dodawanie, usuwanie, zmiany)
- Zmienia role użytkowników systemu
- Sprawdza harmonogram, ogólnie i poszczególnych użytkowników
- Wprowadza zmiany do harmonogramu

☐ Dyrektor Szkoły
- Sprawdza/edytuje harmonogram
- Generuje raporty finansowe
- Generuje, przegląda, edytuje listy użytkowników
- Sprawdza frekwencję
- Generuje raporty o frekwencji
- Zarządza zaległymi płatnościami
- Ma możliwość odroczenia płatności
- Akceptuje listę uczestników płatnego kurs/webinaru/studiów

☐ Koordynator
- ❖ Studiów
  - Tworzy sylabus/program studiów
  - Wprowadza zmiany do harmonogramu
  - Może dodać/usunąć dodatkowych użytkowników do studiów
- ❖ Przedmiotu
  - Zalicza przedmioty studentom/wpisuje im oceny
  - Decyduje w sprawach odnośnie odrabiania
  - Tworzy spotkania studyjne
  - Generuje raport o liczbie osób zapisanych
  - Wybiera wykładowców i tłumaczy
- ❖ Kursu
  - Tworzy kurs
  - Generuje raport o liczbie osób zapisanych
  - Modyfikuje dane kursu
  - Wpisuje oceny z kursu

☐ Wykładowca
- Wpisuje obecności na prowadzonych zajęciach
- Generuje listy obecności i je modyfikuje
- Ma dostęp do harmonogramu prowadzonych przez niego zajęć
- Tworzy webinary i wybiera ich typ(płatny/darmowy)
- Wysyła prośby o zmianę harmonogramu (np. z powodu zdarzeń losowych)
- Wpisuje oceny z przedmiotu/kursu

☐ Prowadzący praktyki
- Zalicza praktyki studentom
- Wpisuje obecności na praktykach
- Ma dostęp do harmonogramu prowadzonych przez niego zajęć

☐ Księgowy
- Generuje raporty finansowe
- Zwraca nadpłaty
- Zbiera informacje o ilości zapisanych osób na przyszłe wydarzenia

☐ Tłumacz
- Ma dostęp do harmonogramu zajęć nie prowadzonych po polsku
- Wysyła prośby o zmianę harmonogramu (np. z powodu zdarzeń losowych)
- Przegląda zajęcia, które nie są prowadzone po polsku
- Tłumaczy zajęcia na żywo

☐ Osoba niewidniejąca w bazie
- Zakłada konto, dane zapisywane są do bazy
- Przegląda dostępną ofertę
- Ma dostęp do danych kontaktowych

Historie użytkownika dla uczestnika kursu

- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość zapisać się na zajęcia.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość dostępu do materiału z poprzednich zajęć.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość sprawdzenia swoich ocen.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość sprawdzenia harmonogramu swoich zajęć.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość generowania raportu obecności.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość rezygnacji z webinaru/kursu/studiów.

Historie użytkownika dla administratora

- Jako administrator chciałbym mieć możliwość modyfikacji danych użytkowników.
- Jako administrator chciałbym mieć możliwość dodawać/usuwać użytkowników.
- Jako administrator chciałbym mieć możliwość zmiany ról użytkowników w systemie.
- Jako administrator chciałbym mieć możliwość wprowadzenia zmian w harmonogramie.
- Jako administrator chciałbym mieć możliwość sprawdzenia harmonogramu dla poszczególnych użytkowników jak i dla ogółu.

Historie użytkownika dla Dyrektora Szkoły

- Jako Dyrektor Szkoły chciałbym mieć możliwość generowania raportów finansowych.
- Jako Dyrektor Szkoły chciałbym mieć możliwość edytowania harmonogramu zajęć.
- Jako Dyrektor Szkoły chciałbym mieć możliwość zarządzania zaległymi płatnościami (odroczenia terminu, wstrzymania blokady dostępu).
- Jako Dyrektor Szkoły chciałbym mieć możliwość przeglądania i edytowania listy użytkowników systemu.
- Jako Dyrektor Szkoły chciałbym mieć możliwość sprawdzenia raportów o frekwencji użytkowników.

Historie użytkownika dla koordynatora (studiów, przedmiotu, kursu)

- Jako koordynator chciałbym mieć możliwość tworzenia programu kursu/przedmiotu/studiów.
- Jako koordynator chciałbym mieć możliwość uprawnienia administratora dla osób pod moją koordynacją.
- Jako koordynator chciałbym mieć możliwość przypisania osób prowadzących zajęcia.
- Jako koordynator chciałbym mieć możliwość przyznania stypendium.
- Jako koordynator chciałbym mieć możliwość generować raport zapisanych osób.
- Jako koordynator chciałbym mieć możliwość wpisania ocen.

Historie użytkownika dla wykładowcy

- Jako wykładowca chciałbym mieć możliwość modyfikowania obecności na prowadzonych zajęciach.
- Jako wykładowca chciałbym mieć możliwość dostępu do harmonogramu prowadzonych zajęć.
- Jako wykładowca chciałbym mieć możliwość tworzenia webinarów i wybór jego typu.
- Jako wykładowca chciałbym mieć możliwość wysłania prośby o zmianę harmonogramu.

Historie użytkownika dla prowadzącego praktyki

- Jako prowadzący praktyki chciałbym mieć możliwość wpisywania obecności studentów na zajęciach praktycznych.
- Jako prowadzący praktyki chciałbym mieć możliwość zaliczania praktyk studentom.
- Jako prowadzący praktyki chciałbym mieć możliwość sprawdzenia harmonogramu swoich zajęć.

Historie użytkownika dla księgowego

- Jako księgowy chciałbym mieć możliwość generowania raportów finansowych.
- Jako księgowy chciałbym mieć możliwość sprawdzenia kto, a kto nie opłacił.
- Jako księgowy chciałbym mieć możliwość zwrotu nadpłat.

Historie użytkownika dla tłumacza

- Jako tłumacz chciałbym mieć możliwość przeglądania harmonogramu zajęć, które nie są prowadzone po polsku.
- Jako tłumacz chciałbym mieć możliwość wysłania prośby o zmianę harmonogramu zajęć, które mam tłumaczyć.
- Jako tłumacz chciałbym mieć możliwość otrzymywania materiałów do przygotowania się przed zajęciami.
- Jako tłumacz chciałbym mieć możliwość tłumaczenia zajęć na żywo dla uczestników.

Historie użytkownika dla osoby niewidniejącej w bazie

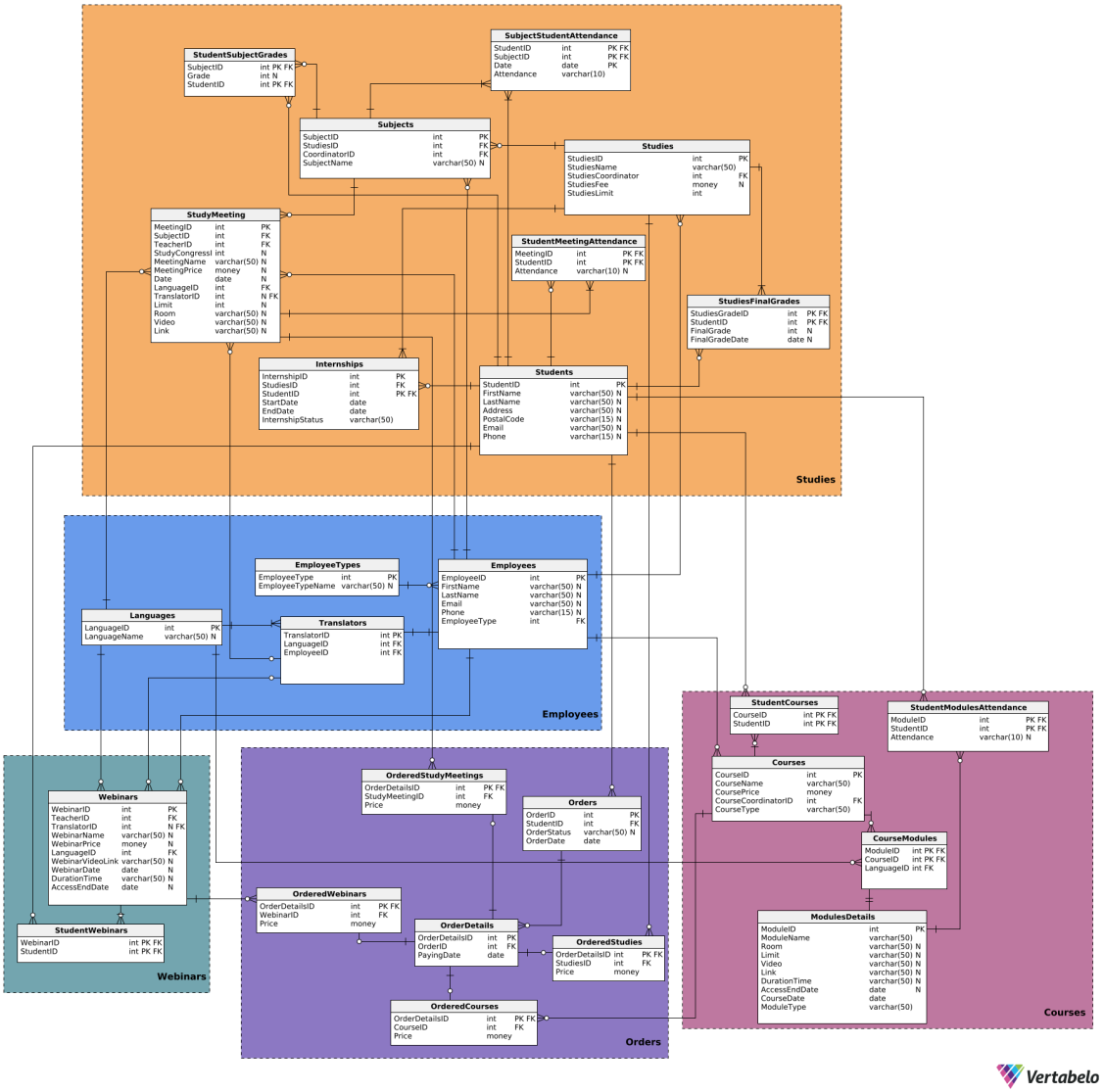
- Jako osoba niewidniejąca w bazie chciałbym mieć możliwość przeglądania dostępnej oferty kursów, webinarów i studiów.
- Jako osoba niewidniejąca w bazie chciałbym mieć możliwość przeglądania danych kontaktowych do obsługi systemu.
- Jako osoba niewidniejąca w bazie chciałbym mieć możliwość założenia konta w systemie, aby zapisać się na kurs/webinar.

Przykładowy przypadek użycia

Opłata za zajęcia.

1. Cel: Opłata za zajęcia.
2. Aktorzy systemu
    - uczestnik kursu
    - student
    - członek webinaru
3. Scenariusz główny
    - zalogowanie się na konto
    - wybór zajęć, za które chce zapłacić
    - system wpisuje użytkownika na listę
    - przejście do systemu opłat
    - opłacanie zajęć i wyświetlenie potwierdzenia
    - Dyrektor zatwierdza dołączenie użytkownika na zajęcia
4. Scenariusze alternatywne
    - odrzucenie płatności, wyświetlenie informacji o niepowodzeniu płatności
    -

Diagram bazy danych



Kod DDL

CourseModules

```
-- Table: CourseModules
CREATE TABLE dbo.CourseModules (
    ModuleID int  NOT NULL,
    CourseID int  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT PK_CourseModules PRIMARY KEY CLUSTERED (ModuleID,CourseID)
)
ON PRIMARY;

ALTER TABLE dbo.CourseModules ADD CONSTRAINT FK_CourseModules_Courses
    FOREIGN KEY (CourseID)
    REFERENCES dbo.Courses (CourseID);

ALTER TABLE dbo.CourseModules ADD CONSTRAINT FK_CourseModules_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);

ALTER TABLE dbo.CourseModules ADD CONSTRAINT FK_CourseModules_ModulesTypes
    FOREIGN KEY (ModuleID)
    REFERENCES dbo.ModulesDetails (ModuleID);
```

Courses

```
-- Table: Courses
CREATE TABLE dbo.Courses (
    CourseID int  NOT NULL,
    CourseName varchar(50)  NOT NULL,
    CoursePrice money  NOT NULL CHECK (CoursePrice > 0),
    CourseCoordinatorID int  NOT NULL,
    CourseType varchar(50)  NOT NULL,
    CONSTRAINT CHK_CoursePrice CHECK (CoursePrice > 0),
    CONSTRAINT CHK_CourseType CHECK (CourseType IN ('online-sync', 'hybrid', 'in-person', 'online-async')),
    CONSTRAINT PK_Courses PRIMARY KEY CLUSTERED (CourseID)
)
ON PRIMARY;

ALTER TABLE dbo.Courses ADD CONSTRAINT FK_Courses_Employees
    FOREIGN KEY (CourseCoordinatorID)
    REFERENCES dbo.Employees (EmployeeID);
```

Warunki integralności:

- **Cena kursu > 0**
- **Typ kursu - jeden spośród ('online-sync', 'hybrid', 'in-person', 'online-async')**

EmployeeTypes

```
-- Table: EmployeeTypes
CREATE TABLE dbo.EmployeeTypes (
    EmployeeType int  NOT NULL,
    EmployeeTypeName varchar(50)  NULL,
    CONSTRAINT PK_EmployeeTypes PRIMARY KEY CLUSTERED (EmployeeType)
)
ON PRIMARY;
```

Employees

```
-- Table: Employees
CREATE TABLE dbo.Employees (
    EmployeeID int  NOT NULL,
    FirstName varchar(50)  NULL,
    LastName varchar(50)  NULL,
    Email varchar(50)  NULL,
    Phone varchar(15)  NULL,
    EmployeeType int  NOT NULL,
    CONSTRAINT CHK_Email CHECK (Email LIKE '%_@_%._%'),
    CONSTRAINT CHK_Phone CHECK (Phone LIKE '+[0-9]%' OR Phone LIKE '[0-9]%'),
    CONSTRAINT PK_Employees PRIMARY KEY CLUSTERED (EmployeeID)
)
ON PRIMARY;


ALTER TABLE dbo.Employees ADD CONSTRAINT FK_Employees_EmployeeTypes
    FOREIGN KEY (EmployeeType)
    REFERENCES dbo.EmployeeTypes (EmployeeType);
```

Warunki integralności:

- **Email w postaci 'xxx@xxx.xxx'**
- **Numer telefonu w postaci '+[0-9]...' lub '[0-9]...'**

Internships

```
-- Table: Internships
CREATE TABLE Internships (
    InternshipID int  NOT NULL,
    StudiesID int  NOT NULL,
    StudentID int  NOT NULL,
    StartDate date  NOT NULL,
    EndDate date  NOT NULL,
    InternshipStatus varchar(50)  NOT NULL,
    CONSTRAINT CHK_StartDateEndDate CHECK (StartDate < EndDate),
    CONSTRAINT CHK_InternshipStatus CHECK (InternshipStatus IN ('in_progress', 'passed', 'failed')),
    CONSTRAINT Internships_pk PRIMARY KEY  (InternshipID,StudentID)
);

ALTER TABLE Internships ADD CONSTRAINT Students_Internships
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

ALTER TABLE Internships ADD CONSTRAINT Studies_Internships
    FOREIGN KEY (StudiesID)
    REFERENCES dbo.Studies (StudiesID);
```

Warunki integralności:

- **Data startu musi być < data zakończenia**
- **Status praktyk - jeden spośród ('in_progress', 'passed', 'failed')**

Languages

```
-- Table: Languages
CREATE TABLE dbo.Languages (
    LanguageID int  NOT NULL,
    LanguageName varchar(50)  NULL,
    CONSTRAINT UC_LanguageName UNIQUE (LanguageName),
    CONSTRAINT PK_Languages PRIMARY KEY CLUSTERED (LanguageID)
)
ON PRIMARY;
```

Warunki integralności:

- **Nazwa języka musi być unikalna**

ModuleDetails

```
-- Table: ModulesDetails
CREATE TABLE dbo.ModulesDetails (
    ModuleID int  NOT NULL,
    ModuleName varchar(50)  NOT NULL,
    Room varchar(50)  NULL,
    Limit varchar(50)  NULL,
    Video varchar(50)  NULL,
    Link varchar(50)  NULL,
    DurationTime varchar(50)  NULL,
    AccessEndDate date  NULL,
    CourseDate date  NOT NULL,
    ModuleType varchar(50)  NOT NULL,
    CONSTRAINT UC_VideoModule UNIQUE (Video),
    CONSTRAINT UC_LinkModule UNIQUE (Link),
    CONSTRAINT CHK_Limit CHECK (Limit > 0),
    CONSTRAINT PK_ModulesTypes PRIMARY KEY CLUSTERED (ModuleID)
)
ON PRIMARY;
```

Warunki integralności:

- **Limit osób > 0**
- **Link do video musi być unikalny**
- **Link do spotkania musi być unikalny**

OrderDetails

```
-- Table: OrderDetails
CREATE TABLE dbo.OrderDetails (
    OrderDetailsID int  NOT NULL,
    OrderID int  NOT NULL,
    PayingDate date  NOT NULL,
    CONSTRAINT PK_OrderDetails PRIMARY KEY CLUSTERED (OrderDetailsID)
)
ON PRIMARY;

ALTER TABLE dbo.OrderDetails ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES dbo.Orders (OrderID);
```

OrderedCourses

```
-- Table: OrderedCourses
CREATE TABLE dbo.OrderedCourses (
    OrderDetailsID int  NOT NULL,
    CourseID int  NOT NULL,
    Price money  NOT NULL CHECK (Price >= 0),
    CONSTRAINT CHK_Price CHECK (Price >= 0),
    CONSTRAINT PK_OrderedCourses PRIMARY KEY CLUSTERED (OrderDetailsID)
)
ON PRIMARY;

ALTER TABLE dbo.OrderedCourses ADD CONSTRAINT FK_OrderDetails_OrderedCourses
    FOREIGN KEY (OrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

ALTER TABLE dbo.OrderedCourses ADD CONSTRAINT FK_OrderedCourses_Courses
    FOREIGN KEY (CourseID)
    REFERENCES dbo.Courses (CourseID);
```

Warunki integralności:

- **Cena musi być >= 0**

OrderedStudies

```
-- Table: OrderedStudies
CREATE TABLE dbo.OrderedStudies (
    OrderDetailsID int  NOT NULL,
    StudiesID int  NOT NULL,
    Price money  NOT NULL CHECK (Price >= 0),
    CONSTRAINT CHK_Price CHECK (Price >= 0),
    CONSTRAINT PK_OrderedStudies PRIMARY KEY CLUSTERED (OrderDetailsID)
)

ON PRIMARY;

ALTER TABLE dbo.OrderedStudies ADD CONSTRAINT FK_OrderDetails_OrderedStudies
    FOREIGN KEY (OrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

ALTER TABLE dbo.OrderedStudies ADD CONSTRAINT OrderedStudies_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES dbo.Studies (StudiesID);
```

Warunki integralności:

- **Cena musi być >= 0**

OrderedStudyMeetings

```
-- Table: OrderedStudyMeetings
CREATE TABLE dbo.OrderedStudyMeetings (
    OrderDetailsID int  NOT NULL,
    StudyMeetingID int  NOT NULL,
    Price money  NOT NULL CHECK (Price >= 0),
    CONSTRAINT CHK_Price CHECK (Price >= 0),
    CONSTRAINT PK_OrderedStudyMeetings PRIMARY KEY CLUSTERED (OrderDetailsID)
)
ON PRIMARY;

ALTER TABLE dbo.OrderedStudyMeetings ADD CONSTRAINT FK_OrderDetails_OrderedStudyMeetings
    FOREIGN KEY (OrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

ALTER TABLE dbo.OrderedStudyMeetings ADD CONSTRAINT OrderedStudyMeetings_StudyMeeting
    FOREIGN KEY (StudyMeetingID)
    REFERENCES dbo.StudyMeeting (MeetingID);
```

Warunki integralności:

- **Cena musi być >= 0**

OrderedWebinars

```
-- Table: OrderedWebinars
CREATE TABLE dbo.OrderedWebinars (
    OrderDetailsID int  NOT NULL,
    WebinarID int  NOT NULL,
    Price money  NOT NULL CHECK (Price >= 0),
    CONSTRAINT CHK_Price CHECK (Price >= 0),
    CONSTRAINT PK_OrderedWebinars PRIMARY KEY CLUSTERED (OrderDetailsID)
)
ON PRIMARY;

ALTER TABLE dbo.OrderedWebinars ADD CONSTRAINT FK_OrderedWebinars_OrderDetails
    FOREIGN KEY (OrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

ALTER TABLE dbo.OrderedWebinars ADD CONSTRAINT OrderedWebinars_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES dbo.Webinars (WebinarID);
```

Warunki integralności:

- **Cena musi być >= 0**

Orders

```
-- Table: Orders
CREATE TABLE dbo.Orders (
    OrderID int  NOT NULL,
    StudentID int  NOT NULL,
    OrderStatus varchar(50)  NULL,
    OrderDate date  NOT NULL,
    CONSTRAINT CHK_OrderStatus CHECK (OrderStatus IN ('paid', 'unpaid', 'canceled')),
    CONSTRAINT PK_Orders PRIMARY KEY CLUSTERED (OrderID)
)
ON PRIMARY;

ALTER TABLE dbo.Orders ADD CONSTRAINT FK_Orders_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);
```

Warunki integralności:

- **Status zamówienia - jeden spośród ('paid', 'unpaid', 'canceled')**

StudentCourses

```
-- Table: StudentCourses
CREATE TABLE dbo.StudentCourses (
    CourseID int  NOT NULL,
    StudentID int  NOT NULL,
    CONSTRAINT PK_CourseDetails PRIMARY KEY CLUSTERED (CourseID,StudentID)
)
ON PRIMARY;

ALTER TABLE dbo.StudentCourses ADD CONSTRAINT FK_CourseDetails_Courses
    FOREIGN KEY (CourseID)
    REFERENCES dbo.Courses (CourseID);

ALTER TABLE dbo.StudentCourses ADD CONSTRAINT FK_CourseDetails_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);
```

StudentMeetingAttendance

```
-- Table: StudentMeetingAttendance
CREATE TABLE dbo.StudentMeetingAttendance (
    MeetingID int  NOT NULL,
    StudentID int  NOT NULL,
    Attendance varchar(10)  NULL,
    CONSTRAINT CHK_Attendance CHECK (Attendance IN ('Present', 'Absent', 'Late')),
    CONSTRAINT PK_MeetingDetails PRIMARY KEY CLUSTERED (MeetingID,StudentID)
)
ON PRIMARY;

ALTER TABLE dbo.StudentMeetingAttendance ADD CONSTRAINT FK_StudyMeeting_MeetingDetails
    FOREIGN KEY (MeetingID)
    REFERENCES dbo.StudyMeeting (MeetingID);

ALTER TABLE dbo.StudentMeetingAttendance ADD CONSTRAINT Students_MeetingDetails
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);
```

StudentModulesAttendance

```
-- Table: StudentModulesAttendance
CREATE TABLE dbo.StudentModulesAttendance (
    ModuleID int  NOT NULL,
    StudentID int  NOT NULL,
    Attendance varchar(10)  NULL,
    CONSTRAINT CHK_Attendance CHECK (Attendance IN ('Present', 'Absent', 'Late')),
    CONSTRAINT PK_ModulesDetails PRIMARY KEY CLUSTERED (ModuleID,StudentID)
)
ON PRIMARY;

ALTER TABLE dbo.StudentModulesAttendance ADD CONSTRAINT ModulesDetails_StudentModulesAttendance
    FOREIGN KEY (ModuleID)
    REFERENCES dbo.ModulesDetails (ModuleID);

ALTER TABLE dbo.StudentModulesAttendance ADD CONSTRAINT StudentModulesAttendance_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);
```

Warunki integralności:

- **Obecność - jedna spośród ('Present', 'Absent', 'Late')**

StudentSubjectGrades

```
-- Table: StudentSubjectGrades
CREATE TABLE dbo.StudentSubjectGrades (
    SubjectID int  NOT NULL,
    Grade int  NULL,
    StudentID int  NOT NULL,
    CONSTRAINT PK_SubjectDetails PRIMARY KEY CLUSTERED (SubjectID,StudentID)
)
ON PRIMARY;
ALTER TABLE dbo.StudentSubjectGrades ADD CONSTRAINT StudentSubjectGrades_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES dbo.Subjects (SubjectID);

ALTER TABLE dbo.StudentSubjectGrades ADD CONSTRAINT Students_StudentSubjectGrades
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);
```

StudentWebinars

```
-- Table: StudentWebinars
CREATE TABLE dbo.StudentWebinars (
    WebinarID int  NOT NULL,
    StudentID int  NOT NULL,
    CONSTRAINT PK_WebinarList PRIMARY KEY CLUSTERED (WebinarID,StudentID)
)
ON PRIMARY;
ALTER TABLE dbo.StudentWebinars ADD CONSTRAINT FK_WebinarList_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

ALTER TABLE dbo.StudentWebinars ADD CONSTRAINT FK_Webinars_WebinarList
    FOREIGN KEY (WebinarID)
    REFERENCES dbo.Webinars (WebinarID);
```

Students

```
-- Table: Students
CREATE TABLE dbo.Students (
    StudentID int  NOT NULL,
    FirstName varchar(50)  NULL,
    LastName varchar(50)  NULL,
    Address varchar(50)  NULL,
    PostalCode varchar(15)  NULL,
    Email varchar(50)  NULL,
    Phone varchar(15)  NULL,
    CONSTRAINT CHK_Email CHECK (Email LIKE '%_@_%._%'),
    CONSTRAINT CHK_Phone CHECK (Phone LIKE '+[0-9]%' OR Phone LIKE '[0-9]%'),
    CONSTRAINT CHK_PostalCode CHECK (PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'),
    CONSTRAINT PK_Students PRIMARY KEY CLUSTERED (StudentID)
)
ON PRIMARY;
```

Warunki integralności:

- **Email w postaci 'xxx@xxx.xxx'**
- **Numer telefonu w postaci '+[0-9]...' lub '[0-9]...'**
- **Kod pocztowy w postaci '[0-9][0-9]-[0-9][0-9][0-9]'**

Studies

```
-- Table: Studies
CREATE TABLE dbo.Studies (
    StudiesID int  NOT NULL,
    StudiesName varchar(50)  NOT NULL,
    StudiesCoordinator int  NOT NULL,
    StudiesFee money  NULL CHECK (StudiesFee > 0),
    StudiesLimit int  NOT NULL,
    CONSTRAINT CHK_StudiesFee CHECK (StudiesFee >= 0),
    CONSTRAINT CHK_StudiesLimit CHECK (StudiesLimit > 0),
    CONSTRAINT PK_Studies PRIMARY KEY CLUSTERED (StudiesID)
)
ON PRIMARY;

ALTER TABLE dbo.Studies ADD CONSTRAINT FK_Studies_Employees
    FOREIGN KEY (StudiesCoordinator)
    REFERENCES dbo.Employees (EmployeeID);
```

Warunki integralności:

- **Opłata za studia >= 0**
- **Limit studiów > 0**

StudiesFinalGrades

```
-- Table: StudiesFinalGrades
CREATE TABLE dbo.StudiesFinalGrades (
    StudiesGradeID int  NOT NULL,
    StudentID int  NOT NULL,
    FinalGrade int  NULL,
    FinalGradeDate date  NULL,
    CONSTRAINT PK_StudiesDetails PRIMARY KEY CLUSTERED (StudiesGradeID,StudentID)
)
ON PRIMARY;

ALTER TABLE dbo.StudiesFinalGrades ADD CONSTRAINT FK_StudiesDetails_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

ALTER TABLE dbo.StudiesFinalGrades ADD CONSTRAINT FK_Studies_StudiesDetails
    FOREIGN KEY (StudiesGradeID)
    REFERENCES dbo.Studies (StudiesID);
```

StudyMeeting

```
-- Table: StudyMeeting
CREATE TABLE dbo.StudyMeeting (
    MeetingID int  NOT NULL,
    SubjectID int  NOT NULL,
    TeacherID int  NOT NULL,
    StudyCongressID int  NULL,
    MeetingName varchar(50)  NULL,
    MeetingPrice money  NULL CHECK (MeetingPrice > 0),
    Date date  NULL,
    LanguageID int  NOT NULL,
    TranslatorID int  NULL,
    Limit int  NULL,
    Room varchar(50)  NULL,
    Video varchar(50)  NULL,
    Link varchar(50)  NULL,
    CONSTRAINT UC_Video UNIQUE (Video),
    CONSTRAINT UC_Link UNIQUE (Link),
    CONSTRAINT CHK_MeetingPrice CHECK (MeetingPrice >= 0),
    CONSTRAINT CHK_Limit CHECK (Limit > 0),
    CONSTRAINT PK_StudyMeeting PRIMARY KEY CLUSTERED (MeetingID)
)
ON PRIMARY;

ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT FK_StudyMeeting_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES dbo.Employees (EmployeeID);

ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT FK_StudyMeeting_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);

ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT FK_StudyMeeting_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES dbo.Subjects (SubjectID);

ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT FK_StudyMeeting_Tranlators
    FOREIGN KEY (TranslatorID)
    REFERENCES dbo.Translators (TranslatorID);
```

SubjectStudentAttendance

```
-- Table: SubjectStudentAttendance
CREATE TABLE SubjectStudentAttendance (
    StudentID int  NOT NULL,
    SubjectID int  NOT NULL,
    Date date  NOT NULL,
    Attendance varchar(10)  NOT NULL,
    CONSTRAINT CHK_Attendance CHECK (Attendance IN ('Present', 'Absent', 'Late')),
    CONSTRAINT SubjectStudentAttendance_pk PRIMARY KEY  (StudentID,SubjectID,Date)
);

ALTER TABLE SubjectStudentAttendance ADD CONSTRAINT Students_SubjectStudentAttendance
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

ALTER TABLE SubjectStudentAttendance ADD CONSTRAINT Subjects_SubjectStudentAttendance
    FOREIGN KEY (SubjectID)
    REFERENCES dbo.Subjects (SubjectID);
```

Subjects

```
-- Table: Subjects
CREATE TABLE dbo.Subjects (
    SubjectID int  NOT NULL,
    StudiesID int  NOT NULL,
    CoordinatorID int  NOT NULL,
    SubjectName varchar(50)  NULL,
    CONSTRAINT PK_Subjects PRIMARY KEY CLUSTERED (SubjectID)
)
ON PRIMARY;

ALTER TABLE dbo.Subjects ADD CONSTRAINT FK_Subjects_Employees
    FOREIGN KEY (CoordinatorID)
    REFERENCES dbo.Employees (EmployeeID);

ALTER TABLE dbo.Subjects ADD CONSTRAINT FK_Subjects_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES dbo.Studies (StudiesID);
```

Translators

```
-- Table: Translators
CREATE TABLE dbo.Translators (
    TranslatorID int  NOT NULL,
    LanguageID int  NOT NULL,
    EmployeeID int  NOT NULL,
    CONSTRAINT PK_Tranlators PRIMARY KEY CLUSTERED (TranslatorID)
)
ON PRIMARY;

ALTER TABLE dbo.Translators ADD CONSTRAINT Employees_Tranlators
    FOREIGN KEY (EmployeeID)
    REFERENCES dbo.Employees (EmployeeID);

ALTER TABLE dbo.Translators ADD CONSTRAINT FK_Tranlators_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);
```

Webinars

```
-- Table: Webinars
CREATE TABLE dbo.Webinars (
    WebinarID int  NOT NULL,
    TeacherID int  NOT NULL,
    TranslatorID int  NULL,
    WebinarName varchar(50)  NULL,
    WebinarPrice money  NULL CHECK (WebinarPrice >= 0),
    LanguageID int  NOT NULL,
    WebinarVideoLink varchar(50)  NULL,
    WebinarDate date  NULL,
    DurationTime varchar(50)  NULL,
    AccessEndDate date  NULL,
    CONSTRAINT UC_WebinarVideoLink UNIQUE (WebinarVideoLink),
    CONSTRAINT CHK_WebinarPrice CHECK (WebinarPrice >= 0),
    CONSTRAINT CHK_WebinarDate CHECK (WebinarDate <= AccessEndDate),
    CONSTRAINT PK_Webinars PRIMARY KEY CLUSTERED (WebinarID)
)
ON PRIMARY;

ALTER TABLE dbo.Webinars ADD CONSTRAINT FK_Webinars_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES dbo.Employees (EmployeeID);

ALTER TABLE dbo.Webinars ADD CONSTRAINT FK_Webinars_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);

ALTER TABLE dbo.Webinars ADD CONSTRAINT FK_Webinars_Tranlators
    FOREIGN KEY (TranslatorID)
    REFERENCES dbo.Translators (TranslatorID);
```

Widoki

1. FINANCIAL_REPORT

Zestawienie łącznych przychodów ze wszystkich źródeł: webinarów, kursów i studiów.

```
CREATE VIEW FINANCIAL_REPORT AS
-- Zestawienie przychodów z webinarów
SELECT w.WebinarID AS ID,
       w.WebinarName AS Name,
       'Webinar' AS Type,
       w.WebinarPrice *
       (SELECT COUNT(*)
        FROM OrderedWebinars ow
        JOIN OrderDetails od ON ow.WebinarsOrderDetailsID = od.OrderDetailsID
        JOIN Orders o ON od.OrderID = o.OrderID
        WHERE ow.WebinarID = w.WebinarID) AS TotalIncome
FROM Webinars w

UNION

-- Zestawienie przychodów z kursów
SELECT c.CourseID AS ID,
       c.CourseName AS Name,
       'Course' AS Type,
       c.CoursePrice *
       (SELECT COUNT(*)
        FROM OrderedCourses oc
        JOIN OrderDetails od ON oc.CoursesOrderDetailsID = od.OrderDetailsID
        JOIN Orders o ON od.OrderID = o.OrderID
        WHERE oc.CourseID = c.CourseID) AS TotalIncome
FROM Courses c

UNION

-- Zestawienie przychodów ze studiów
SELECT
    s.StudiesID AS ID,
    s.StudiesName AS Name,
    'Study' AS Type,
    COALESCE(s.StudiesFee, 0) *
    COALESCE((SELECT COUNT(*)
              FROM OrderedStudies os
              JOIN OrderDetails od ON os.OrderDetailsID = od.OrderDetailsID
              JOIN Orders o ON od.OrderID = o.OrderID
              WHERE os.StudiesID = s.StudiesID), 0) +
    COALESCE((SELECT SUM(sm.MeetingPrice)
              FROM StudyMeeting sm
              JOIN Subjects sb ON sm.SubjectID = sb.SubjectID
              WHERE sb.StudiesID = s.StudiesID), 0) AS TotalIncome
FROM
    Studies s;
```

2. WEBINARS_FINANCIAL_REPORT

Raport finansowy pokazujący przychody tylko z webinarów.

```
CREATE VIEW WEBINARS_FINANCIAL_REPORT AS
SELECT ID AS 'Webinar ID', Name, TotalIncome
FROM FINANCIAL_REPORT
WHERE Type = 'Webinar';
```

3. STUDIES_FINANCIAL_REPORT

Raport finansowy pokazujący przychody tylko ze studiów.

```
CREATE VIEW STUDIES_FINANCIAL_REPORT AS
SELECT ID AS 'Study ID', Name, TotalIncome
FROM FINANCIAL_REPORT
WHERE Type = 'Study';
```

4. COURSES_FINANCIAL_REPORT

Raport finansowy pokazujący przychody tylko z kursów.

```
CREATE VIEW COURSES_FINANCIAL_REPORT AS
SELECT ID AS 'Course ID', Name, TotalIncome
FROM FINANCIAL_REPORT
WHERE Type = 'Course';
```

5. STUDENT_DEBTORS

Widok prezentujący listę wszystkich studentów z nieuregulowanymi płatnościami wraz z kwotami.

```
CREATE VIEW STUDENT_DEBTORS AS
WITH OrderTotals AS (
    SELECT
        o.OrderID,
        o.StudentID,
        s.FirstName,
        s.LastName,
        s.Email,
        o.OrderDate,
        o.OrderStatus,
        COALESCE(SUM(c.CoursePrice), 0) AS TotalCourseCharges,
        COALESCE(SUM(st.StudiesFee), 0) AS TotalStudiesFees,
        COALESCE(SUM(w.WebinarPrice), 0) AS TotalWebinarCharges,
        COALESCE(SUM(sm.MeetingPrice), 0) AS TotalMeetingCharges
    FROM
        dbo.Orders o
    INNER JOIN
        dbo.Students s ON o.StudentID = s.StudentID
    LEFT JOIN
        dbo.OrderedCourses oc ON oc.CoursesOrderDetailsID = o.OrderID
    LEFT JOIN
        dbo.Courses c ON c.CourseID = oc.CourseID
    LEFT JOIN
        dbo.OrderedStudies os ON os.OrderDetailsID = o.OrderID

 LEFT JOIN
        dbo.Studies st ON st.StudiesID = os.StudiesID
    LEFT JOIN
        dbo.OrderedWebinars ow ON ow.WebinarsOrderDetailsID = o.OrderID

 LEFT JOIN
        dbo.Webinars w ON w.WebinarID = ow.WebinarID
    LEFT JOIN
        dbo.OrderedStudyMeetings osm ON osm.StudeyMeetingOrderDetailsID = o.OrderID
    LEFT JOIN
        dbo.StudyMeeting sm ON sm.MeetingID = osm.StudyMeetingID
    WHERE
        o.OrderStatus IS NULL OR o.OrderStatus = 'unpaid'
    GROUP BY
        o.OrderID, o.StudentID, s.FirstName, s.LastName, s.Email, o.OrderDate, o.OrderStatus
)
SELECT
    StudentID,
    FirstName,
    LastName,
    Email,
    OrderID,
    OrderDate,
    OrderStatus,
    (TotalCourseCharges + TotalStudiesFees + TotalWebinarCharges + TotalMeetingCharges) AS TotalUnpaidAmount
FROM
    OrderTotals
WHERE
    (TotalCourseCharges + TotalStudiesFees + TotalWebinarCharges + TotalMeetingCharges) > 0;
```

6. FUTURE_MEETING_STATS

Szczegółowe statystyki dotyczące przyszłych spotkań, zawierające informacje o prowadzących i liczbie dostępnych miejsc.

```
CREATE VIEW FUTURE_MEETING_STATS AS
SELECT
    sm.MeetingID,
    sm.MeetingName,
    sm.Date AS MeetingDate,
    l.LanguageName,
    e.FirstName + ' ' + e.LastName AS TeacherName,
    COUNT(DISTINCT sma.StudentID) AS RegisteredStudents,
    sm.Limit AS MaxCapacity,
    CASE
        WHEN sm.Limit IS NULL THEN NULL
        ELSE sm.Limit - COUNT(DISTINCT sma.StudentID)
    END AS RemainingSpots
FROM dbo.StudyMeeting sm
LEFT JOIN dbo.StudentMeetingAttendance sma ON sm.MeetingID = sma.MeetingID
INNER JOIN dbo.Languages l ON sm.LanguageID = l.LanguageID
INNER JOIN dbo.Employees e ON sm.TeacherID = e.EmployeeID
WHERE sm.Date > GETDATE()
GROUP BY
    sm.MeetingID,
    sm.MeetingName,
    sm.Date,
    l.LanguageName,
    e.FirstName + ' ' + e.LastName,
    sm.Limit;
```

7. FUTURE_MODULE_STATS

Statystyki przyszłych modułów kursowych wraz z informacjami o liczbie zapisanych uczestników i dostępnych miejscach.

```
CREATE VIEW FUTURE_MODULE_STATS AS
SELECT
    md.ModuleDetailsID,
    md.ModuleName,
    md.CoruseDate as ModuleDate,
    c.CourseName,
    l.LanguageName,
    COUNT(DISTINCT sc.StudentID) as RegisteredStudents,
    md.Limit as MaxCapacity,
    CASE
        WHEN md.Limit IS NULL THEN NULL
        ELSE md.Limit - COUNT(DISTINCT sc.StudentID)
    END as RemainingSpots
FROM dbo.ModulesDetails md
INNER JOIN dbo.CourseModules cm ON md.ModuleDetailsID = cm.ModuleID
INNER JOIN dbo.Courses c ON cm.CourseID = c.CourseID
INNER JOIN dbo.Languages l ON cm.LanguageID = l.LanguageID
LEFT JOIN dbo.StudentCourses sc ON c.CourseID = sc.StudentCoursesID
WHERE md.CoruseDate > GETDATE()
GROUP BY
    md.ModuleDetailsID,
    md.ModuleName,
    md.CoruseDate,
    c.CourseName,
    l.LanguageName,
    md.Limit;
```

8. FUTURE_WEBINAR_STATS

Statystyki przyszłych webinarów z informacjami o prowadzących i liczbie zarejestrowanych uczestników.

```
CREATE VIEW FUTURE_WEBINAR_STATS AS
SELECT
    w.WebinarID,
    w.WebinarName,
    w.WebinarDate,
    l.LanguageName,
    e.FirstName + ' ' + e.LastName as TeacherName,
    COUNT(DISTINCT sw.StudentID) as RegisteredStudents,
    NULL as MaxCapacity,
    NULL as RemainingSpots
FROM dbo.Webinars w
LEFT JOIN dbo.StudentWebinars sw ON w.WebinarID = sw.WebinarID
INNER JOIN dbo.Languages l ON w.LanguageID = l.LanguageID
INNER JOIN dbo.Employees e ON w.TeacherID = e.EmployeeID
WHERE w.WebinarDate > GETDATE()
GROUP BY
    w.WebinarID,
    w.WebinarName,
    w.WebinarDate,
    l.LanguageName,
    e.FirstName + ' ' + e.LastName;
```

9. FUTURE_EVENTS_STATS

Zestawienie zbiorcze informacji o wszystkich przyszłych wydarzeniach, łączące dane ze spotkań, webinarów i modułów kursowych.

```
CREATE VIEW FUTURE_EVENTS_STATS AS
SELECT
    'Meeting' as EventType,
    MeetingName as EventName,
    MeetingDate as EventDate,
    LanguageName,
    TeacherName,
    RegisteredStudents,
    MaxCapacity,
    RemainingSpots
FROM FUTURE_MEETING_STATS
UNION ALL
SELECT
    'Webinar' as EventType,
    WebinarName as EventName,
    WebinarDate as EventDate,
    LanguageName,
    TeacherName,
    RegisteredStudents,
    MaxCapacity,
    RemainingSpots
FROM FUTURE_WEBINAR_STATS
UNION ALL




SELECT
    'Course Module' as EventType,
    ModuleName as EventName,
    ModuleDate as EventDate,
    LanguageName,
    NULL as TeacherName,
    RegisteredStudents,
    MaxCapacity,
    RemainingSpots
FROM FUTURE_MODULE_STATS;
```

10. COMPLETED_EVENTS_ATTENDANCE

Zestawienie zbiorcze frekwencji na wszystkich zakończonych wydarzeniach (spotkaniach, modułach kursowych i webinarach).

```
CREATE VIEW COMPLETED_EVENTS_ATTENDANCE AS
SELECT
    'Study Meeting' AS EventType,
    sm.MeetingID AS EventID,
    sm.Date AS EventDate,
    COUNT(DISTINCT sa.StudentID) AS TotalStudents,
    SUM(CASE WHEN sa.Attendance = 'Present' THEN 1 ELSE 0 END) AS PresentStudents,
    SUM(CASE WHEN sa.Attendance = 'Late' THEN 1 ELSE 0 END) AS LateStudents,
    SUM(CASE WHEN sa.Attendance = 'Absent' THEN 1 ELSE 0 END) AS AbsentStudents,


CASE
        WHEN COUNT(DISTINCT sa.StudentID) = 0 THEN 0
        ELSE CAST(SUM(CASE WHEN sa.Attendance = 'Present' THEN 1 ELSE 0 END) * 100.0 /
                  COUNT(DISTINCT sa.StudentID) AS DECIMAL(5, 2))
    END AS AttendancePercentage
FROM StudyMeeting sm
JOIN StudentMeetingAttendance sa ON sm.MeetingID = sa.MeetingID
WHERE sm.Date < GETDATE()
GROUP BY sm.MeetingID, sm.Date

UNION ALL

SELECT
    'Course Module' AS EventType,
    md.ModuleID AS EventID,
    md.CourseDate AS EventDate,
    COUNT(DISTINCT sma.StudentID) AS TotalStudents,
    SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) AS PresentStudents,
    SUM(CASE WHEN sma.Attendance = 'Late' THEN 1 ELSE 0 END) AS LateStudents,
    SUM(CASE WHEN sma.Attendance = 'Absent' THEN 1 ELSE 0 END) AS AbsentStudents,
    CASE
        WHEN COUNT(DISTINCT sma.StudentID) = 0 THEN 0
        ELSE CAST(SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) * 100.0 /
                  COUNT(DISTINCT sma.StudentID) AS DECIMAL(5, 2))
    END AS AttendancePercentage
FROM ModulesDetails md
JOIN StudentModulesAttendance sma ON md.ModuleID = sma.ModuleID
WHERE md.CourseDate < GETDATE()
GROUP BY md.ModuleID, md.CourseDate

UNION ALL

SELECT
    'Webinar' AS EventType,
    w.WebinarID AS EventID,
    w.WebinarDate AS EventDate,
    COUNT(DISTINCT sw.StudentID) AS TotalStudents,
    COUNT(DISTINCT sw.StudentID) AS PresentStudents,
    0 AS LateStudents,
    0 AS AbsentStudents,
    100.00 AS AttendancePercentage
FROM Webinars w
JOIN StudentWebinars sw ON w.WebinarID = sw.WebinarID
WHERE w.WebinarDate < GETDATE()
GROUP BY w.WebinarID, w.WebinarDate;
```

11. CompletedModulesAttendance

Szczegółowe statystyki frekwencji dla zakończonych modułów kursowych.

```
CREATE VIEW COMPLETED_MODULES_ATTENDANCE AS
SELECT
    md.ModuleName as EventName,
    md.ModuleType as EventType,
    md.CourseDate as EventDate,
    COUNT(DISTINCT sma.StudentID) as TotalStudents,
    SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) as PresentStudents,
    SUM(CASE WHEN sma.Attendance = 'Late' THEN 1 ELSE 0 END) as LateStudents,
    SUM(CASE WHEN sma.Attendance = 'Absent' THEN 1 ELSE 0 END) as AbsentStudents,
    CASE
        WHEN COUNT(DISTINCT sma.StudentID) = 0 THEN 0
        ELSE CAST(SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) * 100.0 /
                  COUNT(DISTINCT sma.StudentID) AS DECIMAL(5,2))
    END as AttendancePercentage
FROM ModulesDetails md
JOIN StudentModulesAttendance sma ON md.ModuleID = sma.ModuleID
WHERE md.CourseDate < GETDATE()
GROUP BY md.ModuleID, md.ModuleName, md.ModuleType, md.CourseDate;
```

12. CompletedStudyMeetingsAttendance

Szczegółowe statystyki frekwencji dla zakończonych spotkań studyjnych, zawierające informacje o prowadzących i przedmiotach.

```
CREATE VIEW COMPLETED_STUDY_MEETINGS_ATTENDANCE AS
SELECT
    sm.MeetingName AS EventName,
    'Study Meeting' AS EventType,
    sm.Date AS EventDate,
    s.SubjectName AS SubjectName,
    e.FirstName + ' ' + e.LastName AS TeacherName,
    l.LanguageName AS LanguageName,
    COUNT(DISTINCT sma.StudentID) AS TotalStudents,
    SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) AS PresentStudents,
    SUM(CASE WHEN sma.Attendance = 'Late' THEN 1 ELSE 0 END) AS LateStudents,
    SUM(CASE WHEN sma.Attendance = 'Absent' THEN 1 ELSE 0 END) AS AbsentStudents,
    CASE
        WHEN COUNT(DISTINCT sma.StudentID) = 0 THEN 0
        ELSE CAST(
            SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) * 100.0 /
            COUNT(DISTINCT sma.StudentID) AS DECIMAL(5, 2)
        )
    END AS AttendancePercentage
FROM
    StudyMeeting sm
JOIN
    Subjects s ON sm.SubjectID = s.SubjectID
JOIN
    Employees e ON sm.TeacherID = e.EmployeeID
JOIN
    Languages l ON sm.LanguageID = l.LanguageID
JOIN
    StudentMeetingAttendance sma ON sm.MeetingID = sma.MeetingID
WHERE
    sm.Date < GETDATE()
GROUP BY
    sm.MeetingID, sm.MeetingName, sm.Date, s.SubjectName, e.FirstName, e.LastName, l.LanguageName;
```

13. COMPLETED_WEBINARS_ATTENDANCE

Szczegółowe statystyki uczestnictwa w zakończonych webinarach wraz z informacjami o cenach i czasie trwania.

```
CREATE VIEW COMPLETED_WEBINARS_ATTENDANCE AS
SELECT
    w.WebinarName as EventName,
    'Webinar' as EventType,
    w.WebinarDate as EventDate,
    COUNT(DISTINCT sw.StudentID) as TotalStudents,
    COUNT(DISTINCT sw.StudentID) as PresentStudents,
    0 as LateStudents,
    0 as AbsentStudents,
    100.00 as AttendancePercentage,
    w.WebinarPrice as WebinarPrice,
    w.DurationTime as DurationTime
FROM Webinars w
LEFT JOIN StudentWebinars sw ON w.WebinarID = sw.WebinarID
WHERE w.WebinarDate < GETDATE()
GROUP BY w.WebinarID, w.WebinarName, w.WebinarDate, w.WebinarPrice, w.DurationTime;
```

14. ATTENDANCE_LIST

Pełna lista obecności dla wszystkich rodzajów wydarzeń, zawierająca dane osobowe uczestników i status ich obecności.

```
CREATE VIEW ATTENDANCE_LIST AS
-- Study Meetings Attendance
SELECT
    sm.MeetingID as EventID,
    sm.MeetingName as EventName,
    sm.Date as EventDate,
    s.StudentID,
    s.FirstName,
    s.LastName,
    CASE
        WHEN ssa.Attendance = 'Present' THEN 'Obecny'
        WHEN ssa.Attendance = 'Absent' THEN 'Nieobecny'
        WHEN ssa.Attendance = 'Late' THEN 'Spóźniony'
        ELSE 'Brak informacji'
    END AS AttendanceStatus
FROM
    StudyMeeting sm

JOIN
    SubjectStudentAttendance ssa ON sm.SubjectID = ssa.SubjectID
JOIN
    Students s ON ssa.StudentID = s.StudentID

UNION ALL

-- Course Modules Attendance
SELECT
    md.ModuleID as EventID,
    md.ModuleName as EventName,
    md.CourseDate as EventDate,
    s.StudentID,
    s.FirstName,
    s.LastName,
    CASE
        WHEN sma.Attendance = 'Present' THEN 'Obecny'
        WHEN sma.Attendance = 'Absent' THEN 'Nieobecny'
        WHEN sma.Attendance = 'Late' THEN 'Spóźniony'
        ELSE 'Brak informacji'
    END AS AttendanceStatus
FROM
    ModulesDetails md
JOIN
    StudentModulesAttendance sma ON md.ModuleID = sma.ModuleID
JOIN
    Students s ON sma.StudentID = s.StudentID

UNION ALL
-- Webinars Attendance
SELECT
    w.WebinarID as EventID,
    w.WebinarName as EventName,
    w.WebinarDate as EventDate,
    s.StudentID,
    s.FirstName,
    s.LastName,
    'Obecny' AS AttendanceStatus -- Wszyscy zarejestrowani traktowani jako obecni
FROM
    Webinars w
JOIN
    StudentWebinars sw ON w.WebinarID = sw.WebinarID
JOIN
    Students s ON sw.StudentID = s.StudentID;
```

15. BILOCATION_LIST

Raport bilokacji: lista osób, które są zapisane na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo.

```
CREATE VIEW BILOCATION_LIST AS
SELECT DISTINCT
    s.StudentID,
    s.FirstName,
    s.LastName,
    e1.EventName AS FirstEvent,
    e1.EventDate AS FirstEventDate,
    e2.EventName AS SecondEvent,
    e2.EventDate AS SecondEventDate
FROM
    -- Lista przyszłych wydarzeń (zunionowana)
    (SELECT sw.StudentID,
            w.WebinarName AS EventName,
            w.WebinarDate AS EventDate
     FROM Webinars w
              JOIN StudentWebinars sw ON w.WebinarID = sw.WebinarID
     WHERE w.WebinarDate > GETDATE()

     UNION ALL

     SELECT sma.StudentID,
            md.ModuleName AS EventName,
            md.CourseDate AS EventDate
     FROM ModulesDetails md
              JOIN StudentModulesAttendance sma ON md.ModuleID = sma.ModuleID
     WHERE md.CourseDate > GETDATE()

     UNION ALL

     SELECT ssa.StudentID,
            sm.MeetingName AS EventName,
            sm.Date AS EventDate
     FROM StudyMeeting sm
              JOIN SubjectStudentAttendance ssa ON sm.SubjectID = ssa.SubjectID
     WHERE sm.Date > GETDATE()) e1

JOIN
    (SELECT sw.StudentID,
            w.WebinarName AS EventName,
            w.WebinarDate AS EventDate
     FROM Webinars w
              JOIN StudentWebinars sw ON w.WebinarID = sw.WebinarID
     WHERE w.WebinarDate > GETDATE()

     UNION ALL

     SELECT sma.StudentID,
            md.ModuleName AS EventName,
            md.CourseDate AS EventDate
     FROM ModulesDetails md
              JOIN StudentModulesAttendance sma ON md.ModuleID = sma.ModuleID
     WHERE md.CourseDate > GETDATE()

     UNION ALL

     SELECT ssa.StudentID,
            sm.MeetingName AS EventName,
            sm.Date AS EventDate
     FROM StudyMeeting sm
              JOIN SubjectStudentAttendance ssa ON sm.SubjectID = ssa.SubjectID
     WHERE sm.Date > GETDATE()) e2

ON e1.StudentID = e2.StudentID
   AND CAST(e1.EventDate AS DATE) = CAST(e2.EventDate AS DATE) -- Wydarzenia tego samego dnia
   AND e1.EventName <> e2.EventName -- Wykluczamy to samo wydarzenie
   AND e1.EventName < e2.EventName

JOIN Students s ON e1.StudentID = s.StudentID
```

Funkcje

1. Sprawdzanie, czy tłumacz może tłumaczyć w danym języku - używana w procedurze do tworzenia zajęć w innym języku

```
CREATE FUNCTION CheckTranslatorLanguage
(@TranslatorID int null, @LanguageID int null)
RETURNS bit AS
BEGIN
    IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT * FROM Translators WHERE TranslatorID = @TranslatorID)
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @LanguageID IS NOT NULL AND NOT EXISTS (SELECT * FROM Languages WHERE LanguageID = @LanguageID)
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @TranslatorID IS NULL AND @LanguageID IS NOT NULL
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @TranslatorID IS NOT NULL AND @LanguageID IS NULL
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @TranslatorID IS NOT NULL AND @LanguageID IS NOT NULL AND NOT EXISTS (SELECT * FROM Translators WHERE
TranslatorID = @TranslatorID AND LanguageID = @LanguageID)
    BEGIN
        RETURN CAST(0 AS bit)
    END

    RETURN CAST(1 AS bit)
END;
```

2. Zliczanie frekwencji na kursie

```
CREATE FUNCTION GetCourseAttendanceForStudent(@StudentID int, @CourseID int)
RETURNS REAL
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM Students WHERE StudentID = @StudentID)
    BEGIN
        --jeżeli nie znaleziono studenta
        RETURN 0.0;
    END

    IF NOT EXISTS (SELECT * FROM Courses WHERE CourseID = @CourseID)
    BEGIN
        --jeżeli nie znaleziono kursu
        RETURN 0.0;
    END

    IF NOT EXISTS (SELECT * FROM CourseModulesDetails AS cmd
        JOIN CourseModules AS cm ON cmd.ModuleID = cm.ModuleID
        WHERE StudentID = @StudentID AND CourseID = @CourseID)
    BEGIN
        --jeżeli student nie był zapisany na ten kurs
        RETURN 0.0;
    END


    DECLARE @AttendanceCount INT;
    DECLARE @ModulesCount INT;

    SELECT @AttendanceCount = COUNT(*)
    FROM CourseModulesDetails AS cmd JOIN CourseModules AS cm
    ON cmd.ModuleID = cm.ModuleID
    WHERE StudentID = @StudentID AND Presence = 1 AND CourseID = @CourseID AND Date < GETDATE();

    SELECT @AttendanceCount = COUNT(*)
    FROM CourseModulesDetails AS cmd JOIN CourseModules AS cm
    ON cmd.ModuleID = cm.ModuleID
    WHERE StudentID = @StudentID AND CourseID = @CourseID AND Date < GETDATE();

    RETURN @AttendanceCount / @ModulesCount;
END;
```

3. Zliczenie frekwencji danego uczestnika na danym przedmiocie na studiach

```
CREATE FUNCTION GetSubjectAttendanceForStudent(@StudentID int, @SubjectID int)
RETURNS REAL
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM Students WHERE StudentID = @StudentID)
    BEGIN
        --jeżeli nie znaleziono studenta
        RETURN 0.0;
    END

    IF NOT EXISTS (SELECT * FROM Subject WHERE SubjectID = @SubjectID)
    BEGIN
        --jeżeli nie znaleziono przedmiotu
        RETURN 0.0;
    END

    IF NOT EXISTS (SELECT * FROM StudyMeetingDetails AS smd
        JOIN StudyMeeting AS sm ON smd.StudyMeetingID = sm.StudyMeetingID
        WHERE StudentID = @StudentID AND SubjectID = @SubjectID)
    BEGIN
        --jeżeli student nie był zapisany na zajęcia z tego przedmiotu
        RETURN 0.0;
    END


    DECLARE @AttendanceCount INT;
    DECLARE @MeetingsCount INT;

    SELECT @AttendanceCount = COUNT(*)
    FROM StudyMeetingDetails AS smd JOIN StudyMeeting AS sm
    ON smd.StudyMeetingID = sm.StudyMeetingID
    WHERE StudentID = @StudentID AND Presence = 1 AND SubjectID = @SubjectID AND Date < GETDATE();

    SELECT @MeetingsCount = COUNT(*)
    FROM StudyMeetingDetails AS smd JOIN StudyMeeting AS sm
    ON smd.StudyMeetingID = sm.StudyMeetingID
    WHERE StudentID = @StudentID AND SubjectID = @SubjectID AND Date < GETDATE();

    RETURN @AttendanceCount / @MeetingsCount;
END;
```

4. Łączna wartość zamówienia

```
CREATE FUNCTION GetOrderValue(@OrderID int)
RETURNS money
AS
BEGIN
    DECLARE @StudiesSum money
    DECLARE @StudyMeetingsSum money
    DECLARE @CoursesSum money
    DECLARE @WebinarsSum money

    SELECT @StudiesSum = ISNULL(SUM(s.StudiesEntryFeePrice), 0)
    FROM Studies AS s
    JOIN OrderStudies AS os ON s.StudiesID = os.StudiesID
    JOIN OrderDetails AS od ON os.OrderDetailsID = od.OrderDetailsID
    WHERE od.OrderID = @OrderID

    SELECT @StudyMeetingsSum = ISNULL(SUM(sm.MeetingPrice), 0)
    FROM Studies AS s
    JOIN Subject AS su ON s.StudiesID = su.StudiesID
    JOIN StudyMeeting AS sm ON su.SubjectID = sm.SubjectID
    JOIN OrderStudies AS os ON s.StudiesID = os.StudiesID
    JOIN OrderDetails AS od ON os.OrderDetailsID = od.OrderDetailsID
    WHERE od.OrderID = @OrderID

    SELECT @CoursesSum = ISNULL(SUM(c.CoursePrice), 0)
    FROM Courses AS c
    JOIN OrderCourse AS oc ON c.CourseID = oc.CourseID
    JOIN OrderDetails AS od ON oc.OrderDetailsID = od.OrderDetailsID
    WHERE od.OrderID = @OrderID

    SELECT @WebinarsSum = ISNULL(SUM(w.WebinarPrice), 0)
    FROM Webinars AS w
    JOIN OrderWebinars AS ow ON w.WebinarID = ow.WebinarID
    JOIN OrderDetails AS od ON ow.OrderDetailsID = od.OrderDetailsID
    WHERE od.OrderID = @OrderID
    SELECT @StudyMeetingsSum =  @StudyMeetingsSum +
    ISNULL(SUM(sm.MeetingPrice * (1 + s.PriceIncrease)), 0)

    FROM StudyMeeting AS sm
    JOIN OrderStudyMeeting AS osm ON sm.StudyMeetingID = osm.StudyMeetingID
    JOIN OrderDetails AS od ON osm.OrderDetailsID = od.OrderDetailsID
    JOIN Subject AS su ON sm.SubjectID = su.SubjectID
    JOIN Studies AS s ON su.StudiesID = s.StudiesID
    WHERE od.OrderID = @OrderID

    RETURN @StudiesSum + @CoursesSum + @WebinarsSum + @StudyMeetingsSum
END;
```

Triggery

1. Dodawanie Studenta do webinaru po jego zakupieniu

```
CREATE TRIGGER [dbo].[trg_AddStudentToWebinar]
ON OrderedWebinars
AFTER INSERT
AS
BEGIN
  IF EXISTS (
    SELECT StudentID
    FROM inserted
    INNER JOIN OrderDetails
    ON inserted.OrderDetailsID = OrderDetails.OrderDetailsID
    INNER JOIN Orders
    ON OrderDetails.OrderID = Orders.OrderID
    WHERE StudentID IN (
      SELECT DISTINCT StudentID
      FROM inserted
      INNER JOIN StudentWebinars
      ON inserted.WebinarID = StudentWebinars.WebinarID
    )
  )
  BEGIN
    RAISERROR('Student o podanym ID jest już zapisany na ten webinar.', 16, 1);
  END
  ELSE
    BEGIN
        INSERT INTO StudentWebinars(StudentID, WebinarID)
        SELECT Orders.StudentID, inserted.WebinarID
        FROM inserted
      INNER JOIN OrderDetails
      ON inserted.OrderDetailsID = OrderDetails.OrderDetailsID
      INNER JOIN Orders
      ON OrderDetails.OrderID = Orders.OrderID
    END
END;
```

2. Automatyczne dodawanie studenta do studiów i odpowiednich spotkań studyjnych po jego zakupieniu

```
CREATE TRIGGER [dbo].[trg_AddStudentToStudy]
ON OrderedStudies
AFTER INSERT
AS
BEGIN
    IF EXISTS (
        SELECT StudentID
        FROM inserted
        INNER JOIN OrderDetails
        ON inserted.OrderDetailsID = OrderDetails.OrderDetailsID
        INNER JOIN Orders
        ON OrderDetails.OrderID = Orders.OrderID
        WHERE StudentID IN (
            SELECT DISTINCT StudentID
            FROM inserted
            INNER JOIN StudiesDetails
            ON inserted.StudiesID = StudiesDetails.StudiesID
        )
    )
    BEGIN
        RAISERROR('Student o podanym ID jest już zapisany na te studia.', 16, 1);
    END
    ELSE IF EXISTS (
        SELECT StudentID
        FROM inserted
        INNER JOIN OrderDetails
        ON inserted.OrderDetailsID = OrderDetails.OrderDetailsID
        INNER JOIN Orders
        ON OrderDetails.OrderID = Orders.OrderID
        WHERE dbo.IsStudentInAnyStudyMeeting(StudentID, inserted.StudiesID) = 1
    )
    BEGIN
        RAISERROR('Student o podanym ID jest zapisany na jedno ze spotkań tych studiów.', 16, 1);
    END
    ELSE
    BEGIN
        DECLARE @StudentID int;
        SELECT @StudentID = Orders.StudentID
        FROM inserted
        INNER JOIN OrderDetails
        ON inserted.OrderDetailsID = OrderDetails.OrderDetailsID
        INNER JOIN Orders
        ON OrderDetails.OrderID = Orders.OrderID;

        INSERT INTO StudiesDetails (StudiesID, StudentID, StudiesGrade)
        SELECT inserted.StudiesID, @StudentID, 2
        FROM inserted;

        INSERT INTO StudyMeetingDetails (StudyMeetingID, StudentID, Presence)
        SELECT StudyMeeting.StudyMeetingID, @StudentID, 0
        FROM inserted
        INNER JOIN Subject
        ON inserted.StudiesID = Subject.StudiesID
        INNER JOIN StudyMeeting
        ON Subject.SubjectID = StudyMeeting.SubjectID;
    END
END;
```

Procedury

1. Dodawanie nowego webinaru

```
CREATE PROCEDURE AddWebinar
@WebinarID int, @TeacherID int, @TranslatorID int null,
@WebinarName varchar(50) null, @WebinarPrice money null,
@LanguageID int ,
@WebinarVideoLink varchar(50) null, @WebinarDate date null,
@DurationTime varchar(50) null, @AccessEndDate date null

AS
BEGIN
IF NOT EXISTS (SELECT * FROM Employees WHERE EmployeeID = @TeacherID)
BEGIN
RAISERROR('Nie znaleziono nauczyciela.', 16, 1);
END
    IF dbo.CheckTranslatorLanguage(@TranslatorID, @LanguageID) = CAST(0 AS bit)
    BEGIN
        RAISERROR('Podano nieprawidłową kombinację tłumacza i języka.', 16, 1);
    END
    INSERT INTO Webinars (WebinarID, TeacherID, TranslatorID, WebinarName,
        WebinarPrice, LanguageID, WebinarVideoLink, WebinarDate,
        DurationTime,AccessEndDate)
    VALUES (@WebinarID, @TeacherID, @TranslatorID, @WebinarName,
        @WebinarPrice, @LanguageID, @WebinarVideoLink, @WebinarDate,
        @DurationTime, @AccessEndDate);

END;
```

2. Dodawanie nowego pracownika

```
CREATE PROCEDURE AddEmployee
@EmployeeID int, @FirstName varchar(50) null, @LastName varchar(50) null,
@Phone varchar(50) null,@Email varchar(15) null, @EmployeeType int
AS
BEGIN
IF NOT EXISTS (SELECT * FROM EmployeeTypes WHERE EmployeeType = @EmployeeType)
BEGIN
RAISERROR('Nieprawidłowy rodzaj pracownika.', 16, 1);
END
INSERT INTO Employees (EmployeeID, FirstName, LastName,
Phone, Email, EmployeeType)
VALUES (@EmployeeID, @FirstName, @LastName,@Phone, @Email, @EmployeeType);
END;
```

3. Dodawanie studenta

```
CREATE PROCEDURE dbo.AddStudent
@StudentID int, @FirstName varchar(50) null, @LastName varchar(50) null, @Address varchar(50) null,
@PostalCode varchar(15) null, @Email varchar(50)null,@Phone varchar(15) null
AS
BEGIN
INSERT INTO Students (StudentID, FirstName, LastName, Address,
PostalCode, Phone, Email)
VALUES (@StudentID, @FirstName, @LastName, @Address,
@PostalCode, @Phone, @Email);
END;
```

4. Dodawanie studiów

```
CREATE PROCEDURE AddStudy
@StudiesID int,
@StudiesName varchar(50),
@StudiesCoordinator int,
@StudiesFee money null,
@StudiesLimit int
AS
BEGIN
IF NOT EXISTS (SELECT * FROM Employees e
JOIN dbo.EmployeeTypes et ON e.EmployeeType = et.EmployeeType
WHERE EmployeeID = @StudiesCoordinator
AND et.EmployeeTypeName LIKE 'Koordynator')
BEGIN
RAISERROR('Koordynator o podanym ID nie istnieje.', 16, 1);
END
    IF @StudiesFee IS NULL
        BEGIN
            SET @StudiesFee = 1000
        END
    INSERT INTO Studies(StudiesID, StudiesName, StudiesCoordinator,StudiesFee, StudiesLimit)
    VALUES (@StudiesID, @StudiesName, @StudiesCoordinator,@StudiesFee ,@StudiesLimit)
END;
```

5. Dodawanie nowych kursów

```
CREATE PROCEDURE AddCourse
@CourseID int,
@CourseName varchar(50),
@CoursePrice money,
@CourseCoordinatorID int,
@CourseType varchar(50)
AS
BEGIN
-- Sprawdź, czy istnieje koordynator o podanym CourseCoordinatorID
IF NOT EXISTS (
SELECT 1
FROM Employees e
JOIN dbo.EmployeeTypes et ON e.EmployeeType = et.EmployeeType
WHERE EmployeeID = @CourseCoordinatorID
AND et.EmployeeTypeName LIKE 'Course Coordinator'
)
BEGIN
RAISERROR('Koordynator o podanym ID nie istnieje.', 16, 1);
END
    -- Wstaw nowy kurs do tabeli Courses
    INSERT INTO Courses (CourseID, CourseName,  CoursePrice, CourseCoordinatorID, CourseType)
    VALUES (@CourseID, @CourseName,  @CoursePrice, @CourseCoordinatorID,@CourseType);

    PRINT 'Kurs dodany pomyślnie.';
END;
```

6. Dodawanie do CourseModules

```
CREATE PROCEDURE AddCourseModule
    @ModuleID INT,
    @CourseID INT,
    @LanguageID INT
AS
BEGIN
    BEGIN TRY
        INSERT INTO dbo.CourseModules (ModuleID, CourseID, LanguageID)
        VALUES (@ModuleID, @CourseID, @LanguageID);
        PRINT 'Course module added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

7. Dodawanie języka do bazy

```
CREATE PROCEDURE AddLanguage
    @LanguageID INT,
    @LanguageName VARCHAR(50)
AS
BEGIN
    BEGIN TRY
        INSERT INTO dbo.Languages (LanguageID, LanguageName)
        VALUES (@LanguageID, @LanguageName);
        PRINT 'Language added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

8. Dodawanie do tabeli Module Detail

```
CREATE PROCEDURE AddModuleDetail
    @ModuleID INT,
    @ModuleName VARCHAR(50),
    @Room VARCHAR(50),
    @Limit INT,
    @Video VARCHAR(50),
    @Link VARCHAR(50),
    @DurationTime VARCHAR(50),
    @AccessEndDate DATE,
    @CourseDate DATE,
    @ModuleType VARCHAR(50)
AS
BEGIN
    BEGIN TRY
        INSERT INTO dbo.ModulesDetails (ModuleID, ModuleName, Room, Limit, Video, Link, DurationTime,
AccessEndDate, CourseDate, ModuleType)
        VALUES (@ModuleID, @ModuleName, @Room, @Limit, @Video, @Link, @DurationTime, @AccessEndDate,
@CourseDate, @ModuleType);
        PRINT 'Module detail added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

9. Dodawanie Internship

```
CREATE PROCEDURE AddInternship
    @InternshipID INT,
    @StudiesID INT,
    @StudentID INT,
    @StartDate DATE,
    @EndDate DATE,
    @InternshipStatus VARCHAR(50)
AS
BEGIN
    BEGIN TRY
        INSERT INTO Internships (InternshipID, StudiesID, StudentID, StartDate, EndDate, InternshipStatus)
        VALUES (@InternshipID, @StudiesID, @StudentID, @StartDate, @EndDate, @InternshipStatus);
        PRINT 'Internship added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

10. Dodawanie zamówienia

```
CREATE PROCEDURE AddOrder
    @OrderID INT,
    @StudentID INT,
    @OrderStatus VARCHAR(50),
    @OrderDate DATE
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy student istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Students WHERE StudentID = @StudentID)
        BEGIN
            THROW 50001, 'Student does not exist.', 1;
        END;

        -- Dodanie zamówienia
        INSERT INTO dbo.Orders (OrderID, StudentID, OrderStatus, OrderDate)
        VALUES (@OrderID, @StudentID, @OrderStatus, @OrderDate);

        PRINT 'Order added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

11. Dodawanie Student Courses

```
CREATE PROCEDURE AddStudentCourse
    @CourseID INT,
    @StudentID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy kurs istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Courses WHERE CourseID = @CourseID)
        BEGIN
            THROW 50002, 'Course does not exist.', 1;
        END;

        -- Sprawdzenie, czy student istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Students WHERE StudentID = @StudentID)
        BEGIN
            THROW 50003, 'Student does not exist.', 1;
        END;

        -- Dodanie studenta do kursu
        INSERT INTO dbo.StudentCourses (CourseID, StudentID)
        VALUES (@CourseID, @StudentID);

        PRINT 'Student added to course successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH

END;
```

12. Dodawanie do StudentMeetingAttendance

```
CREATE PROCEDURE AddStudentMeetingAttendance
    @MeetingID INT,
    @StudentID INT,
    @Attendance VARCHAR(10)
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy spotkanie istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.StudyMeeting WHERE MeetingID = @MeetingID)
        BEGIN
            THROW 50004, 'Meeting does not exist.', 1;
        END;

        -- Sprawdzenie, czy student istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Students WHERE StudentID = @StudentID)
        BEGIN
            THROW 50005, 'Student does not exist.', 1;
        END;

        -- Dodanie obecności
        INSERT INTO dbo.StudentMeetingAttendance (MeetingID, StudentID, Attendance)
        VALUES (@MeetingID, @StudentID, @Attendance);

        PRINT 'Attendance record added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

13. Dodawanie ocen ze studiów

```
CREATE PROCEDURE AddStudentSubjectGrade
    @SubjectID INT,
    @Grade INT,
    @StudentID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy przedmiot istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Subjects WHERE SubjectID = @SubjectID)
        BEGIN
            THROW 50006, 'Subject does not exist.', 1;
        END;

        -- Sprawdzenie, czy student istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Students WHERE StudentID = @StudentID)
        BEGIN
            THROW 50007, 'Student does not exist.', 1;
        END;

        -- Dodanie oceny
        INSERT INTO dbo.StudentSubjectGrades (SubjectID, Grade, StudentID)
        VALUES (@SubjectID, @Grade, @StudentID);

        PRINT 'Grade added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

14. Dodawanie studenta do webinaru

```
CREATE PROCEDURE AddStudentWebinar
    @WebinarID INT,
    @StudentID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy webinar istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.StudyMeeting WHERE MeetingID = @WebinarID)
        BEGIN
            THROW 50008, 'Webinar does not exist.', 1;
        END;

        -- Sprawdzenie, czy student istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Students WHERE StudentID = @StudentID)
        BEGIN
            THROW 50009, 'Student does not exist.', 1;
        END;

        -- Dodanie studenta do webinaru
        INSERT INTO dbo.StudentWebinars (WebinarID, StudentID)
        VALUES (@WebinarID, @StudentID);

        PRINT 'Student added to webinar successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

15. Dodawanie do StudyMeeting

```
CREATE PROCEDURE AddStudyMeeting
    @MeetingID INT,
    @SubjectID INT,
    @TeacherID INT,
    @LanguageID INT,
    @MeetingName VARCHAR(50),
    @MeetingPrice MONEY,
    @Date DATE,
    @TranslatorID INT,
    @Limit INT,
    @Room VARCHAR(50),
    @Video VARCHAR(50),
    @Link VARCHAR(50)
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy przedmiot istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Subjects WHERE SubjectID = @SubjectID)
        BEGIN
            THROW 50010, 'Subject does not exist.', 1;
        END;

        -- Sprawdzenie, czy nauczyciel istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            THROW 50011, 'Teacher does not exist.', 1;
        END;

        -- Sprawdzenie, czy język istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Languages WHERE LanguageID = @LanguageID)
        BEGIN
            THROW 50012, 'Language does not exist.', 1;
        END;

        -- Dodanie spotkania
        INSERT INTO dbo.StudyMeeting (MeetingID, SubjectID, TeacherID, LanguageID, MeetingName, MeetingPrice,
Date, TranslatorID, Limit, Room, Video, Link)
        VALUES (@MeetingID, @SubjectID, @TeacherID, @LanguageID, @MeetingName, @MeetingPrice, @Date,
@TranslatorID, @Limit, @Room, @Video, @Link);

        PRINT 'Study meeting added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

16. Raport zamówień dla studenta

```
CREATE PROCEDURE GetStudentOrdersReport
    @StudentID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy student istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Students WHERE StudentID = @StudentID)
        BEGIN
            THROW 60002, 'Student does not exist.', 1;
        END;

        -- Pobranie raportu zamówień
        SELECT
            o.OrderID,
            o.OrderDate,
            o.OrderStatus,
            c.CourseName,
            c.CoursePrice,
            od.PayingDate
        FROM dbo.Orders o
        INNER JOIN dbo.OrderedCourses oc ON o.OrderID = oc.OrderDetailsID
        INNER JOIN dbo.Courses c ON oc.CourseID = c.CourseID
        INNER JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
        WHERE o.StudentID = @StudentID;

    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

17. Usuwanie studenta z webinarów i kursów

```
CREATE PROCEDURE RemoveStudentFromCourseAndWebinars
    @StudentID INT,
    @CourseID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy student jest zapisany na kurs
        IF NOT EXISTS (SELECT 1 FROM dbo.StudentCourses WHERE StudentID = @StudentID AND CourseID = @CourseID)
        BEGIN
            THROW 60004, 'Student is not enrolled in the course.', 1;
        END;

        -- Usunięcie studenta z kursu
        DELETE FROM dbo.StudentCourses WHERE StudentID = @StudentID AND CourseID = @CourseID;

        -- Usunięcie studenta z webinarów powiązanych z kursem
        DELETE sw
        FROM dbo.StudentWebinars sw
        INNER JOIN dbo.StudyMeeting sm ON sw.WebinarID = sm.MeetingID
        WHERE sm.SubjectID IN (SELECT SubjectID FROM dbo.Subjects WHERE StudiesID IN
                               (SELECT StudiesID FROM dbo.Courses WHERE CourseID = @CourseID))
          AND sw.StudentID = @StudentID;

        PRINT 'Student removed from course and related webinars.';
    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```

18. Podsumowanie obecności na kursach i modułach

```
CREATE PROCEDURE GetStudentAttendanceSummary
    @StudentID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie, czy student istnieje
        IF NOT EXISTS (SELECT 1 FROM dbo.Students WHERE StudentID = @StudentID)
        BEGIN
            THROW 60005, 'Student does not exist.', 1;
        END;

        -- Raport o obecności
        SELECT
            c.CourseName,
            md.ModuleName,
            sma.Attendance AS ModuleAttendance,
            sm.Attendance AS MeetingAttendance
        FROM dbo.StudentModulesAttentandce sma
        INNER JOIN dbo.ModulesDetails md ON sma.ModuleID = md.ModuleID
        INNER JOIN dbo.CourseModules cm ON md.ModuleID = cm.ModuleID
        INNER JOIN dbo.Courses c ON cm.CourseID = c.CourseID
        LEFT JOIN dbo.StudentMeetingAttendance sm ON sm.MeetingID = md.ModuleID
        WHERE sma.StudentID = @StudentID;

    END TRY
    BEGIN CATCH
        PRINT 'Error occurred: ' + ERROR_MESSAGE();
    END CATCH
END;
```