Dokumentacja projekt bazy danych 2024/2025

Zespół

- Błażej Naziemiec - 33%
- Dawid Żak - 33%
- Szymon Żuk - 33%

Grupa 10, wtorek 18:30

Role

W bazie danych utworzone są następujące role:

- Administrator
- Lecturer
- Translator
- Student
- SchoolDirector
- Accountant

Uprawnienia

Administrator

Rola `Administrator` ma następujące uprawnienia:

- Może dodawać, zmieniać i usuwać dane dla: użytkowników
- Może dodawać, zmieniać i usuwać dane dla: pracowników
- Może dodawać, zmieniać i usuwać dane dla: webinarów
- Może dodawać, zmieniać i usuwać dane dla: kursów
- Może dodawać, zmieniać i usuwać dane dla: studiów
- Może dodawać, zmieniać i usuwać dane dla: zamówień
- Może dodawać, zmieniać i usuwać dane dla: szczegółów zamówień
- Może przeglądać dane dla: zamówień
- Może przeglądać dane dla: szczegółów zamówień

```
GRANT INSERT, UPDATE, DELETE ON Users TO Administrator;
GRANT INSERT, UPDATE, DELETE ON Employees TO Administrator;
GRANT INSERT, UPDATE, DELETE ON Webinars TO Administrator;
GRANT INSERT, UPDATE, DELETE ON Courses TO Administrator;
GRANT INSERT, UPDATE, DELETE ON Studies TO Administrator;
GRANT INSERT, UPDATE, DELETE ON Orders TO Administrator;
GRANT INSERT, UPDATE, DELETE ON OrderDetails TO Administrator;
GRANT SELECT ON Orders TO Administrator;
GRANT SELECT ON OrderDetails TO Administrator;
```

Lecturer

Rola `Lecturer` ma następujące uprawnienia:

- Może przeglądać dane: studentów
- Może przeglądać dane: zakupionych webinarów przez studentów
- Może przeglądać dane: zaliczonych modułów kursów
- Może przeglądać dane: zaliczonych lekcji kursów
- Może przeglądać dane: zaliczonych webinarów
- Może przeglądać dane: zaliczonych lekcji studiów

```
GRANT SELECT ON Students TO Lecturer;
GRANT SELECT ON StudentBoughtWebinars TO Lecturer;
GRANT SELECT ON CourseModulesPassed TO Lecturer;
GRANT SELECT ON CourseLessonsPassed TO Lecturer;
GRANT SELECT ON WebinarsPassed TO Lecturer;
GRANT SELECT ON StudiesLessonPassed TO Lecturer;
```

Translator

Rola `Translator` ma następujące uprawnienia:

- Może przeglądać dane: webinarów
- Może przeglądać dane: kursów
- Może przeglądać dane: studiów

```
GRANT SELECT ON Webinars TO Translator;
GRANT SELECT ON Courses TO Translator;
GRANT SELECT ON Studies TO Translator;
```

Student

Rola `Student` ma następujące uprawnienia:

- Może przeglądać dane dla: webinarów
- Może przeglądać dane dla: kursów
- Może przeglądać dane dla: studiów
- Może przeglądać dane dla: zakupionych webinarów przez studentów
- Może przeglądać dane dla: zaliczonych modułów kursów
- Może przeglądać dane dla: zaliczonych lekcji kursów
- Może przeglądać dane dla: zaliczonych webinarów
- Może przeglądać dane dla: zaliczonych lekcji studiów

```
GRANT SELECT ON Webinars TO Student;
GRANT SELECT ON Courses TO Student;
GRANT SELECT ON Studies TO Student;
GRANT SELECT ON StudentBoughtWebinars TO Student;
GRANT SELECT ON CourseModulesPassed TO Student;
GRANT SELECT ON CourseLessonsPassed TO Student;
GRANT SELECT ON WebinarsPassed TO Student;
GRANT SELECT ON StudiesLessonPassed TO Student;
```

SchoolDirector

Rola `SchoolDirector` ma następujące uprawnienia:

- Ma pełne uprawnienia (dodawanie, zmienianie, usuwanie, przeglądanie) do danych dla: użytkowników
- Ma pełne uprawnienia (dodawanie, zmienianie, usuwanie, przeglądanie) do danych dla: pracowników
- Ma pełne uprawnienia (dodawanie, zmienianie, usuwanie, przeglądanie) do danych dla: webinarów
- Ma pełne uprawnienia (dodawanie, zmienianie, usuwanie, przeglądanie) do danych dla: kursów
- Ma pełne uprawnienia (dodawanie, zmienianie, usuwanie, przeglądanie) do danych dla: studiów
- Ma pełne uprawnienia (dodawanie, zmienianie, usuwanie, przeglądanie) do danych dla: zamówień
- Ma pełne uprawnienia (dodawanie, zmienianie, usuwanie, przeglądanie) do danych dla: szczegółów zamówień

```
GRANT ALL ON Users TO SchoolDirector;
GRANT ALL ON Employees TO SchoolDirector;
GRANT ALL ON Webinars TO SchoolDirector;
GRANT ALL ON Courses TO SchoolDirector;
GRANT ALL ON Studies TO SchoolDirector;
GRANT ALL ON Orders TO SchoolDirector;
GRANT ALL ON OrderDetails TO SchoolDirector;
```

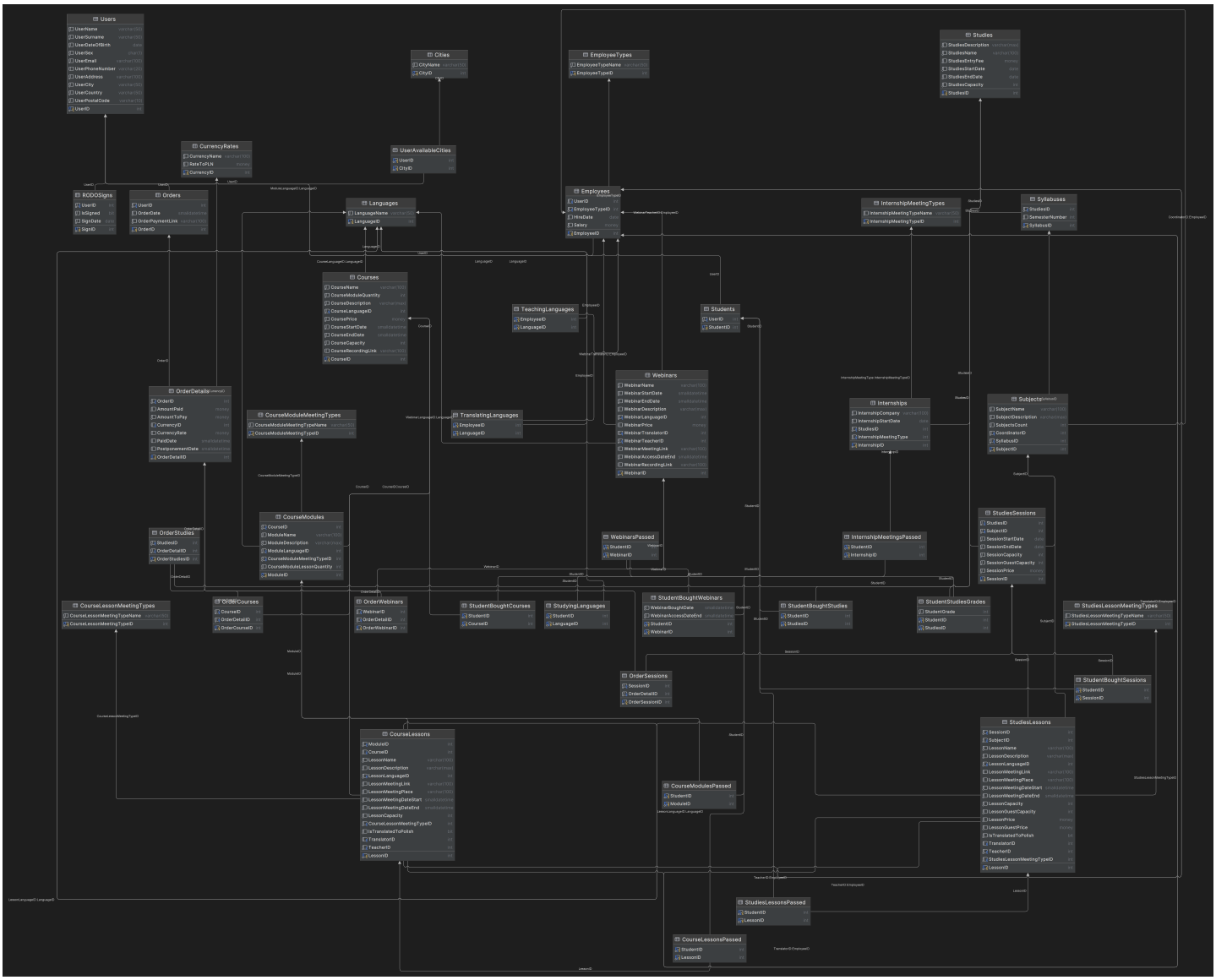
Accountant

Rola `Accountant` ma następujące uprawnienia:

- Może przeglądać dane dla: zamówień
- Może przeglądać dane dla: szczegółów zamówień

```
GRANT SELECT ON Orders TO Accountant;
GRANT SELECT ON OrderDetails TO Accountant;
```

Diagram bazy danych



Opis tabel

- **Users**: Przechowuje informacje o użytkownikach systemu.
    - **Klucz podstawowy**: UserID
    - **Pola tabeli**:
        - UserID: ID użytkownika
        - UserName: Imię użytkownika
        - UserSurname: Nazwisko użytkownika
        - UserDateOfBirth: Data urodzenia użytkownika
        - UserSex: Płeć użytkownika
        - UserEmail: Adres email użytkownika
        - UserPhoneNumber: Numer telefonu użytkownika
        - UserAddress: Adres użytkownika
        - UserCity: Miasto użytkownika
        - UserCountry: Kraj użytkownika
        - UserPostalCode: Kod pocztowy użytkownika

```
CREATE TABLE Users
(
    UserID          INT PRIMARY KEY IDENTITY(1,1),
    UserName        VARCHAR(50)  NOT NULL,
    UserSurname     VARCHAR(50)  NOT NULL,
    UserDateOfBirth DATE         NOT NULL,
    UserSex         CHAR(1)      NOT NULL,
    UserEmail       VARCHAR(100) NOT NULL,
    UserPhoneNumber VARCHAR(20),
    UserAddress     VARCHAR(100) NOT NULL,
    UserCity        VARCHAR(50) NOT NULL,
    UserCountry     VARCHAR(50) NOT NULL,
    UserPostalCode  VARCHAR(10) NOT NULL
)
```

- **Languages**: Przechowuje informacje o językach.
    - **Klucz podstawowy**: LanguageID
    - **Pola tabeli**:
        - LanguageID: ID języka
        - LanguageName: Nazwa języka

```
CREATE TABLE Languages
(
    LanguageID   INT PRIMARY KEY IDENTITY(1,1),
    LanguageName VARCHAR(50) NOT NULL
)
```

- **CurrencyRates**: Przechowuje kursy walut.
    - **Klucz podstawowy**: CurrencyID
    - **Pola tabeli**:
        - CurrencyID: ID waluty
        - CurrencyName: Nazwa waluty
        - RateToPLN: Kurs waluty do PLN

```
CREATE TABLE CurrencyRates(
    CurrencyID INT PRIMARY KEY IDENTITY(1,1),
    CurrencyName VARCHAR(100) NOT NULL,
    RateToPLN MONEY NOT NULL
)
```

- **EmployeeTypes**: Przechowuje typy pracowników.
    - **Klucz podstawowy**: EmployeeTypeID
    - **Pola tabeli**:
        - EmployeeTypeID: ID typu pracownika
        - EmployeeTypeName: Nazwa typu pracownika

```
CREATE TABLE EmployeeTypes
(
    EmployeeTypeID   INT PRIMARY KEY IDENTITY(1,1),
    EmployeeTypeName VARCHAR(50) NOT NULL
)
```

- **Employees**: Przechowuje informacje o pracownikach.
    - **Klucz podstawowy**: EmployeeID
    - **Klucze obce**: UserID (z tabeli Users), EmployeeTypeID (z tabeli EmployeeTypes)
    - **Pola tabeli**:
        - EmployeeID: ID pracownika
        - UserID: ID użytkownika
        - EmployeeTypeID: ID typu pracownika
        - HireDate: Data zatrudnienia
        - Salary: Wynagrodzenie

```
CREATE TABLE Employees
(
    EmployeeID     INT PRIMARY KEY IDENTITY(1,1),
    UserID         INT NOT NULL,
    EmployeeTypeID INT NOT NULL,
    HireDate       DATE NOT NULL,
    Salary         MONEY NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users (UserID),
    FOREIGN KEY (EmployeeTypeID) REFERENCES EmployeeTypes (EmployeeTypeID)
)
```

- **TeachingLanguages**: Przechowuje informacje o językach, w których uczą pracownicy.
    - **Klucze podstawowe**: EmployeeID, LanguageID
    - **Klucze obce**: EmployeeID (z tabeli Employees), LanguageID (z tabeli Languages)
    - **Pola tabeli**:
        - EmployeeID: ID pracownika
        - LanguageID: ID języka

```
CREATE TABLE TeachingLanguages(
    EmployeeID INT,
    LanguageID INT,
    PRIMARY KEY (EmployeeID, LanguageID),
    FOREIGN KEY (EmployeeID) REFERENCES Employees (EmployeeID),
    FOREIGN KEY (LanguageID) REFERENCES Languages (LanguageID)
)
```

- **Students**: Przechowuje informacje o studentach.
    - **Klucz podstawowy**: StudentID
    - **Klucz obcy**: UserID (z tabeli Users)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - UserID: ID użytkownika

```
CREATE TABLE Students
(
    StudentID  INT PRIMARY KEY IDENTITY(1,1),
    UserID     INT NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users (UserID)
)
```

- **StudyingLanguages**: Przechowuje informacje o językach, w których chce uczyć się student.
    - **Klucze podstawowe**: StudentID, LanguageID
    - **Klucze obce**: StudentID (z tabeli Students), LanguageID (z tabeli Languages)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - LanguageID: ID języka

```
CREATE TABLE StudyingLanguages(
    StudentID INT,
    LanguageID INT,
    PRIMARY KEY (StudentID, LanguageID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID),
    FOREIGN KEY (LanguageID) REFERENCES Languages (LanguageID)
)
```

- **TranslatingLanguages**: Przechowuje informacje o języku, jaki może tłumaczyć tłumacz.
    - **Klucze podstawowe**: EmployeeID, LanguageID
    - **Klucze obce**: EmployeeID (z tabeli Employees), LanguageID (z tabeli Languages)
    - **Pola tabeli**:
        - EmployeeID: ID pracownika
        - LanguageID: ID języka

```
CREATE TABLE TranslatingLanguages(
   EmployeeID INT,
   LanguageID INT,
   PRIMARY KEY (EmployeeID, LanguageID),
   FOREIGN KEY (EmployeeID) REFERENCES Employees (EmployeeID),
   FOREIGN KEY (LanguageID) REFERENCES Languages (LanguageID)
)
```

- **Webinars**: Przechowuje informacje o webinarach.
    - **Klucz podstawowy**: WebinarID
    - **Klucze obce**: WebinarLanguageID (z tabeli Languages), WebinarTranslatorID (z tabeli Employees), WebinarTeacherID (z tabeli Employees)
    - **Pola tabeli**:
        - WebinarID: ID webinaru
        - WebinarName: Nazwa webinaru
        - WebinarStartDate: Data rozpoczęcia webinaru
        - WebinarEndDate: Data zakończenia webinaru
        - WebinarDescription: Opis webinaru
        - WebinarLanguageID: ID języka webinaru
        - WebinarPrice: Cena webinaru
        - WebinarTranslatorID: ID tłumacza webinaru
        - WebinarTeacherID: ID nauczyciela webinaru
        - WebinarMeetingLink: Link do spotkania webinaru
        - WebinarAccessDateEnd: Data zakończenia dostępu do webinaru
        - WebinarRecordingLink: Link do nagrania webinaru

```
CREATE TABLE Webinars(
   WebinarID INT PRIMARY KEY IDENTITY(1,1),
   WebinarName VARCHAR(100) NOT NULL,
   WebinarStartDate SMALLDATETIME NOT NULL,
   WebinarEndDate SMALLDATETIME NOT NULL,
   WebinarDescription VARCHAR(MAX) NOT NULL,
   WebinarLanguageID INT NOT NULL,
   WebinarPrice MONEY,
   WebinarTranslatorID INT,
   WebinarTeacherID INT NOT NULL,
   WebinarMeetingLink VARCHAR(100),
   WebinarAccessDateEnd SMALLDATETIME NOT NULL,
   WebinarRecordingLink VARCHAR(100)
   FOREIGN KEY (WebinarLanguageID) REFERENCES Languages (LanguageID),
   FOREIGN KEY (WebinarTranslatorID) REFERENCES Employees (EmployeeID),
   FOREIGN KEY (WebinarTeacherID) REFERENCES Employees (EmployeeID)
)
```

- **StudentBoughtWebinars**: Przechowuje informacje o zakupionych webinarach przez studentów.
    - **Klucze podstawowe**: StudentID, WebinarID
    - **Klucze obce**: StudentID (z tabeli Students), WebinarID (z tabeli Webinars)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - WebinarID: ID webinaru
        - WebinarBoughtDate: Data zakupu webinaru
        - WebinarAccessDateEnd: Data zakończenia dostępu do webinaru

```
CREATE Table StudentBoughtWebinars(
   StudentID INT,
   WebinarID INT,
   WebinarBoughtDate SMALLDATETIME NOT NULL,
   WebinarAccessDateEnd SMALLDATETIME NOT NULL,
   PRIMARY KEY (StudentID, WebinarID),
   FOREIGN KEY (StudentID) REFERENCES Students (StudentID),
   FOREIGN KEY (WebinarID) REFERENCES Webinars (WebinarID)
)
```

- **Cities**: Przechowuje informacje o miastach.
    - **Klucz podstawowy**: CityID
    - **Pola tabeli**:
        - CityID: ID miasta
        - CityName: Nazwa miasta

```
CREATE TABLE Cities(
   CityID INT PRIMARY KEY IDENTITY(1,1),
   CityName VARCHAR(50) NOT NULL
)
```

- **UserAvailableCities**: Przechowuje informacje o dostępnych miastach dla użytkowników.
    - **Klucze podstawowe**: UserID, CityID
    - **Klucze obce**: UserID (z tabeli Users), CityID (z tabeli Cities)
    - **Pola tabeli**:
        - UserID: ID użytkownika
        - CityID: ID miasta

```
CREATE TABLE UserAvailableCities(
   UserID INT,
   CityID INT,
   PRIMARY KEY (UserID, CityID),
   FOREIGN KEY (UserID) REFERENCES Users (UserID),
   FOREIGN KEY (CityID) REFERENCES Cities (CityID)
)
```

- **Courses**: Przechowuje informacje o kursach.
    - **Klucz podstawowy**: CourseID
    - **Klucz obcy**: CourseLanguageID (z tabeli Languages)
    - **Pola tabeli**:
        - CourseID: ID kursu
        - CourseName: Nazwa kursu
        - CourseModuleQuantity: Ilość modułów kursu
        - CourseDescription: Opis kursu
        - CourseLanguageID: ID języka kursu
        - CoursePrice: Cena kursu
        - CourseStartDate: Data rozpoczęcia kursu
        - CourseEndDate: Data zakończenia kursu
        - CourseCapacity: Maksymalna ilość osób w kursie
        - CourseRecordingLink: Link do nagrania kursu

```
CREATE TABLE Courses(
   CourseID INT PRIMARY KEY IDENTITY(1,1),
   CourseName VARCHAR(100) NOT NULL,
   CourseModuleQuantity INT NOT NULL,
   CourseDescription VARCHAR(MAX) NOT NULL,
   CourseLanguageID INT NOT NULL,
   CoursePrice MONEY NOT NULL,
   CourseStartDate SMALLDATETIME NOT NULL,
   CourseEndDate SMALLDATETIME NOT NULL,
   CourseCapacity INT NOT NULL,
   CourseRecordingLink VARCHAR(100)
   FOREIGN KEY (CourseLanguageID) REFERENCES Languages (LanguageID),
)
```

- **StudentBoughtCourses**: Przechowuje powiązanie studenta z zakupionymi kursami.
    - **Klucze podstawowe**: StudentID, CourseID
    - **Klucze obce**: StudentID (z tabeli Students), CourseID (z tabeli Courses)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - CourseID: ID kursu

```
CREATE Table StudentBoughtCourses(
    StudentID INT,
    CourseID INT,
    PRIMARY KEY (StudentID, CourseID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID),
    FOREIGN KEY (CourseID) REFERENCES Courses (CourseID)
)
```

- **CourseModuleMeetingTypes**: Przechowuje typy spotkań modułów kursów.
    - **Klucz podstawowy**: CourseModuleMeetingTypeID
    - **Pola tabeli**:
        - CourseModuleMeetingTypeID: ID typu spotkania modułu
        - CourseModuleMeetingTypeName: Nazwa typu spotkania

```
CREATE TABLE CourseModuleMeetingTypes(
   CourseModuleMeetingTypeID INT PRIMARY KEY IDENTITY(1,1),
   CourseModuleMeetingTypeName VARCHAR(50) NOT NULL
)
```

- **CourseModules**: Przechowuje informacje o modułach kursów.
    - **Klucz podstawowy**: CourseModuleID
    - **Klucze obce**: CourseID (z tabeli Course), ModuleLanguageID (z tabeli Languages), CourseModuleMeetingTypeID (z tabeli CourseModuleMeetingTypes)
    - **Pola tabeli**:
        - ModuleID: ID modułu
        - CourseID: ID kursu
        - ModuleName: Nazwa modułu
        - ModuleDescription: Opis modułu
        - ModuleLanguageID: ID języka modułu
        - CourseModuleMeetingTypeID: ID typu spotkania modułu
        - CourseModuleLessonQuantity: Ilość lekcji modułu

```
CREATE TABLE CourseModules(
   ModuleID INT PRIMARY KEY IDENTITY(1,1),
   CourseID INT NOT NULL,
   ModuleName VARCHAR(100) NOT NULL,
   ModuleDescription VARCHAR(MAX) NOT NULL,
   ModuleLanguageID INT NOT NULL,
   CourseModuleMeetingTypeID INT NOT NULL,
   CourseModuleLessonQuantity INT NOT NULL,
   FOREIGN KEY (CourseID) REFERENCES Courses (CourseID),
   FOREIGN KEY (ModuleLanguageID) REFERENCES Languages (LanguageID),
   FOREIGN KEY (CourseModuleMeetingTypeID) REFERENCES CourseModuleMeetingTypes
(CourseModuleMeetingTypeID)
)
```

- **CourseLessonMeetingTypes**: Przechowuje typy spotkań lekcji kursów.
    - **Klucz podstawowy**: CourseLessonMeetingTypeID
    - **Pola tabeli**:
        - CourseLessonMeetingTypeID: ID typu spotkania lekcji
        - CourseLessonMeetingTypeName: Nazwa typu spotkania

```
CREATE TABLE CourseLessonMeetingTypes(
   CourseLessonMeetingTypeID INT PRIMARY KEY IDENTITY(1,1),
   CourseLessonMeetingTypeName VARCHAR(50) NOT NULL
)
```

- **CourseLessons**: Przechowuje informacje o lekcjach kursów.
    - **Klucz podstawowy**: CourseLessonID
    - **Klucze obce**: ModuleID (z tabeli CourseModules), CourseID (z tabeli Courses), LessonLanguageID (z tabeli Languages), CourseLessonMeetingTypeID (z tabeli CourseLessonMeetingTypes) TranslatorID (z tabeli Employees), TeacherID (z tabeli Employees)
    - **Pola tabeli**:
        - LessonID: ID lekcji
        - ModuleID: ID modułu
        - CourseID: ID kursu
        - LessonName: Nazwa lekcji
        - LessonDescription: Opis lekcji
        - LessonLanguageID: ID języka lekcji
        - LessonMeetingLink: Link do spotkania lekcji
        - LessonMeetingPlace: Miejsce spotkania lekcji
        - LessonMeetingDateStart: Data rozpoczęcia spotkania lekcji
        - LessonMeetingDateEnd: Data zakończenia spotkania lekcji
        - LessonCapacity: Maksymalna ilość osób na lekcji
        - CourseLessonMeetingTypeID: ID typu spotkania lekcji
        - IsTranslatedToPolish: Czy lekcja jest tłumaczona na polski
        - TranslatorID: ID tłumacza
        - TeacherID: ID nauczyciela

```
CREATE TABLE CourseLessons(
    LessonID INT PRIMARY KEY IDENTITY(1,1),
    ModuleID INT NOT NULL,
    CourseID INT NOT NULL,
    LessonName VARCHAR(100) NOT NULL,
    LessonDescription VARCHAR(MAX) NOT NULL,
    LessonLanguageID INT NOT NULL,
    LessonMeetingLink VARCHAR(100) NOT NULL,
    LessonMeetingPlace VARCHAR(100) NOT NULL,
    LessonMeetingDateStart SMALLDATETIME NOT NULL,
    LessonMeetingDateEnd SMALLDATETIME NOT NULL,
    LessonCapacity INT NOT NULL,
    CourseLessonMeetingTypeID INT NOT NULL,
    IsTranslatedToPolish BIT NOT NULL,
    TranslatorID INT,
    TeacherID INT NOT NULL,
    FOREIGN KEY (ModuleID) REFERENCES CourseModules (ModuleID),
    FOREIGN KEY (CourseID) REFERENCES Courses (CourseID),
    FOREIGN KEY (LessonLanguageID) REFERENCES Languages (LanguageID),
    FOREIGN KEY (CourseLessonMeetingTypeID) REFERENCES CourseLessonMeetingTypes
(CourseLessonMeetingTypeID),
    FOREIGN KEY (TranslatorID) REFERENCES Employees (EmployeeID),
    FOREIGN KEY (TeacherID) REFERENCES Employees (EmployeeID),
)
```

- **CourseModulesPassed**: Przechowuje informacje o zaliczonych modułach kursów przez studentów.
    - **Klucze podstawowe**: StudentID, CourseID
    - **Klucze obce**: StudentID (z tabeli Students), ModuleID (z tabeli CourseModules)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - ModuleID: ID modułu

```
CREATE TABLE CourseModulesPassed(
    StudentID INT,
    ModuleID INT,
    PRIMARY KEY (StudentID, ModuleID),
    FOREIGN KEY (ModuleID) REFERENCES CourseModules (ModuleID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
)
```

- **CourseLessonsPassed**: Przechowuje informacje o zaliczonych lekcjach kursów przez studentów.
    - **Klucze podstawowe**: StudentID, LessonID
    - **Klucze obce**: StudentID (z tabeli Students), LessonID (z tabeli CourseLessons)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - LessonID: ID lekcji

```
CREATE TABLE CourseLessonsPassed(
    StudentID INT,
    LessonID INT,
    PRIMARY KEY (StudentID, LessonID),
    FOREIGN KEY (LessonID) REFERENCES CourseLessons (LessonID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
)
```

- **WebinarsPassed**: Przechowuje informacje o zaliczonych webinarach przez studentów.
    - **Klucze podstawowe**: StudentID, WebinarID
    - **Klucze obce**: StudentID (z tabeli Students), WebinarID (z tabeli Webinars)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - WebinarID: ID webinaru

```
CREATE TABLE WebinarsPassed(
    StudentID INT,
    WebinarID INT,
    PRIMARY KEY (StudentID, WebinarID),
    FOREIGN KEY (WebinarID) REFERENCES Webinars (WebinarID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
)
```

- **Orders**: Przechowuje informacje o zamówieniach.
    - **Klucz podstawowy**: OrderID
    - **Klucz obcy**: UserID (z tabeli Users)
    - **Pola tabeli**:
        - OrderID: ID zamówienia
        - UserID: ID użytkownika
        - OrderDate: Data zamówienia
        - OrderPaymentLink: Link do płatności zamówienia

```
CREATE TABLE Orders(
    OrderID INT PRIMARY KEY IDENTITY(1,1),
    UserID INT NOT NULL,
    OrderDate SMALLDATETIME NOT NULL,
    OrderPaymentLink VARCHAR(100) NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users (UserID)
)
```

- **OrderDetails**: Przechowuje szczegóły zamówień.
    - **Klucz podstawowy**: OrderDetailID
    - **Klucze obce**: OrderID (z tabeli Orders), CurrencyID (z tabeli CurrencyRates)
    - **Pola tabeli**:
        - OrderDetailID: ID szczegółu zamówienia
        - OrderID: ID zamówienia
        - AmountPaid: Kwota zapłacona
        - AmountToPay: Kwota do zapłaty
        - CurrencyID: ID waluty
        - CurrencyRate: Kurs waluty
        - PaidDate: Data płatności
        - PostponementDate: Data odroczenia płatności

```
CREATE TABLE OrderDetails(
    OrderDetailID INT PRIMARY KEY IDENTITY(1,1),
    OrderID INT NOT NULL,
    AmountPaid MONEY NOT NULL,
    AmountToPay MONEY NOT NULL,
    CurrencyID INT,
    CurrencyRate MONEY,
    PaidDate SMALLDATETIME,
    PostponementDate SMALLDATETIME,
    FOREIGN KEY (OrderID) REFERENCES Orders (OrderID),
    FOREIGN KEY (CurrencyID) REFERENCES CurrencyRates (CurrencyID)
)
```

- **OrderWebinars**: Przechowuje informacje o webinarach w zamówieniach.
    - **Klucz podstawowy**: OrderWebinarID
    - **Klucze obce**: OrderDetailID (z tabeli OrderDetails), WebinarID (z tabeli Webinars)
    - **Pola tabeli**:
        - OrderWebinarID: ID webinaru w zamówieniu
        - WebinarID: ID webinaru
        - OrderDetailID: ID szczegółu zamówienia

```
CREATE TABLE OrderWebinars(
    OrderWebinarID INT PRIMARY KEY IDENTITY(1,1),
    WebinarID INT NOT NULL,
    OrderDetailID INT NOT NULL,
    FOREIGN KEY (OrderDetailID) REFERENCES OrderDetails (OrderDetailID),
    FOREIGN KEY (WebinarID) REFERENCES Webinars (WebinarID)
)
```

- **OrderCourses**: Przechowuje informacje o zamówionych kursach.
    - **Klucz podstawowy**: OrderCourseID
    - **Klucze obce**: OrderDetailID (z tabeli OrderDetails), CourseID (z tabeli Courses)
    - **Pola tabeli**:
        - OrderCourseID: ID kursu w zamówieniu
        - CourseID: ID kursu
        - OrderDetailID: ID szczegółu zamówienia

```
CREATE TABLE OrderCourses(
    OrderCourseID INT PRIMARY KEY IDENTITY(1,1),
    CourseID INT NOT NULL,
    OrderDetailID INT NOT NULL,
    FOREIGN KEY (OrderDetailID) REFERENCES OrderDetails (OrderDetailID),
    FOREIGN KEY (CourseID) REFERENCES Courses (CourseID)
)
```

- **Studies**: Przechowuje informacje o studiach.
    - **Klucz podstawowy**: StudiesID
    - **Pola tabeli**:
        - StudiesID: ID studiów
        - StudiesDescription: Opis studiów
        - StudiesName: Nazwa studiów
        - StudiesEntryFee: Opłata za studia
        - StudiesStartDate: Data rozpoczęcia studiów
        - StudiesEndDate: Data zakończenia studiów
        - StudiesCapacity: Maksymalna ilość osób na studiach

```
CREATE TABLE Studies(
    StudiesID INT PRIMARY KEY IDENTITY(1,1),
    StudiesDescription VARCHAR(MAX) NOT NULL,
    StudiesName VARCHAR(100) NOT NULL,
    StudiesEntryFee MONEY NOT NULL,
    StudiesStartDate DATE NOT NULL,
    StudiesEndDate DATE NOT NULL,
    StudiesCapacity INT NOT NULL
)
```

- **Syllabuses**: Przechowuje informacje o sylabusach.
    - **Klucz podstawowy**: SyllabusID
    - **Klucz obcy**: StudiesID (z tabeli Studies)
    - **Pola tabeli**:
        - SyllabusID: ID sylabusa
        - StudiesID: ID studiów
        - SemesterNumber: Numer semestru

```
CREATE TABLE Syllabuses(
    SyllabusID INT PRIMARY KEY IDENTITY(1,1),
    StudiesID INT NOT NULL,
    SemesterNumber INT NOT NULL,
    FOREIGN KEY (StudiesID) REFERENCES Studies (StudiesID)
)
```

- **Subjects**: Przechowuje informacje o przedmiotach.
    - **Klucz podstawowy**: SubjectID
    - **Klucze obce**: CoordinatorID (z tabeli Employees), SyllabusID (z tabeli Syllabuses)
    - **Pola tabeli**:
        - SubjectID: ID przedmiotu
        - SubjectName: Nazwa przedmiotu
        - SubjectDescription: Opis przedmiotu
        - SubjectsCount: Liczba zajęć na przedmiocie
        - CoordinatorID: ID koordynatora
        - SyllabusID: ID sylabusa

```
CREATE TABLE Subjects(
    SubjectID INT PRIMARY KEY IDENTITY(1,1),
    SubjectName VARCHAR(100) NOT NULL,
    SubjectDescription VARCHAR(MAX) NOT NULL,
    SubjectsCount INT NOT NULL,
    CoordinatorID INT NOT NULL,
    SyllabusID INT NOT NULL,
    FOREIGN KEY (CoordinatorID) REFERENCES Employees (EmployeeID),
    FOREIGN KEY (SyllabusID) REFERENCES Syllabuses (SyllabusID)
)
```

- **StudiesLessonMeetingTypes**: Przechowuje typy spotkań lekcji studiów.
    - **Klucz podstawowy**: StudiesLessonMeetingTypeID
    - **Pola tabeli**:
        - StudiesLessonMeetingTypeID: ID typu spotkania lekcji
        - StudiesLessonMeetingTypeName: Nazwa typu spotkania

```
CREATE TABLE StudiesLessonMeetingTypes(
    StudiesLessonMeetingTypeID INT PRIMARY KEY IDENTITY(1,1),
    StudiesLessonMeetingTypeName VARCHAR(50) NOT NULL
)
```

- **StudiesSessions**: Przechowuje informacje o semestrach na studiach.
    - **Klucz podstawowy**: SessionID
    - **Klucze obce**: StudiesID (z tabeli Studies), SubjectID (z tabeli Subjects)
    - **Pola tabeli**:
        - SessionID: ID semestru
        - StudiesID: ID studiów
        - SubjectID: ID przedmiotu
        - SessionStartDate: Data rozpoczęcia semestru
        - SessionEndDate: Data zakończenia semestru
        - SessionCapacity: Ilość osób na semestrze
        - SessionGuestCapacity: Ilość gości na semestrze
        - SessionPrice: Cena za semestr

```
CREATE TABLE StudiesSessions(
    SessionID INT PRIMARY KEY IDENTITY(1,1),
    StudiesID INT NOT NULL,
    SubjectID INT NOT NULL,
    SessionStartDate DATE NOT NULL,
    SessionEndDate DATE NOT NULL,
    SessionCapacity INT NOT NULL,
    SessionGuestCapacity INT NOT NULL,
    SessionPrice MONEY NOT NULL,
    FOREIGN KEY (StudiesID) REFERENCES Studies (StudiesID),
    FOREIGN KEY (SubjectID) REFERENCES Subjects (SubjectID)
)
```

- **StudiesLessons**: Przechowuje informacje o lekcjach studiów.
    - **Klucz podstawowy**: LessonID
    - **Klucze obce**: SessionID (z tabeli StudiesSessions), SubjectID (z tabeli Subjects), LessonLanguageID (z tabeli Languages), TranslatorID (z tabeli Employees), TeacherID (z tabeli Employees), StudiesLessonMeetingTypeID (z tabeli StudiesLessonMeetingTypes)
    - **Pola tabeli**:
        - LessonID: ID lekcji
        - SessionID: ID semestru
        - SubjectID: ID przedmiotu
        - LessonName: Nazwa lekcji
        - LessonDescription: Opis lekcji
        - LessonLanguageID: ID języka lekcji
        - LessonMeetingLink: Link do spotkania lekcji
        - LessonMeetingPlace: Miejsce spotkania lekcji
        - LessonMeetingDateStart: Data rozpoczęcia spotkania lekcji
        - LessonMeetingDateEnd: Data zakończenia spotkania lekcji
        - LessonCapacity: Ilość osób na lekcji
        - LessonGuestCapacity: Ilość gości na lekcji
        - LessonPrice: Cena lekcji
        - LessonGuestPrice: Cena dla gości
        - IsTranslatedToPolish: Czy lekcja jest tłumaczona na polski
        - TranslatorID: ID tłumacza
        - TeacherID: ID nauczyciela
        - StudiesLessonMeetingTypeID: ID typu spotkania lekcji

```
CREATE TABLE StudiesLessons(
    LessonID INT PRIMARY KEY IDENTITY(1,1),
    SessionID INT NOT NULL,
    SubjectID INT NOT NULL,
    LessonName VARCHAR(100) NOT NULL,
    LessonDescription VARCHAR(MAX) NOT NULL,
    LessonLanguageID INT NOT NULL,
    LessonMeetingLink VARCHAR(100),
    LessonMeetingPlace VARCHAR(100),
    LessonMeetingDateStart SMALLDATETIME NOT NULL,
    LessonMeetingDateEnd SMALLDATETIME NOT NULL,
    LessonCapacity INT NOT NULL,
    LessonGuestCapacity INT NOT NULL,
    LessonPrice MONEY NOT NULL,
    LessonGuestPrice MONEY NOT NULL,
    IsTranslatedToPolish BIT NOT NULL,
    TranslatorID INT,
    TeacherID INT NOT NULL,
    StudiesLessonMeetingTypeID INT NOT NULL,
    FOREIGN KEY (SessionID) REFERENCES StudiesSessions (SessionID),
    FOREIGN KEY (SubjectID) REFERENCES Subjects (SubjectID),
    FOREIGN KEY (LessonLanguageID) REFERENCES Languages (LanguageID),
    FOREIGN KEY (TranslatorID) REFERENCES Employees (EmployeeID),
    FOREIGN KEY (TeacherID) REFERENCES Employees (EmployeeID),
    FOREIGN KEY (StudiesLessonMeetingTypeID) REFERENCES StudiesLessonMeetingTypes
(StudiesLessonMeetingTypeID),
)
```

- **StudiesLessonsPassed**: Przechowuje informacje o zaliczonych lekcjach studiów przez studentów.
    - **Klucze podstawowe**: StudentID, LessonID
    - **Klucze obce**: StudentID (z tabeli Students), LessonID (z tabeli StudiesLessons)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - LessonID: ID lekcji

```
CREATE TABLE StudiesLessonsPassed(
    StudentID INT,
    LessonID INT,
    PRIMARY KEY (StudentID, LessonID),
    FOREIGN KEY (LessonID) REFERENCES StudiesLessons (LessonID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
)
```

- **StudentStudiesGrades**: Przechowuje oceny studentów za studia.
    - **Klucze podstawowe**: StudentID, StudiesID
    - **Klucze obce**: StudentID (z tabeli Students), StudiesID (z tabeli Studies)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - StudiesID: ID studiów
        - StudentGrade: Ocena studenta

```
CREATE TABLE StudentStudiesGrades(
    StudentID INT,
    StudiesID INT,
    StudentGrade INT NOT NULL,
    PRIMARY KEY (StudentID, StudiesID),
    FOREIGN KEY (StudiesID) REFERENCES Studies (StudiesID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
)
```

- **StudentBoughtStudies**: Przechowuje informacje o zakupionych studiach przez studentów.
    - **Klucze podstawowe**: StudentID, StudiesID
    - **Klucze obce**: StudentID (z tabeli Students), StudiesID (z tabeli Studies)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - StudiesID: ID studiów

```
CREATE TABLE StudentBoughtStudies(
    StudentID INT,
    StudiesID INT,
    PRIMARY KEY (StudentID, StudiesID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID),
    FOREIGN KEY (StudiesID) REFERENCES Studies (StudiesID)
)
```

- **StudentBoughtSessions**: Przechowuje informacje o zakupionych sesjach przez studentów.
    - **Klucze podstawowe**: StudentID, SessionID
    - **Klucze obce**: StudentID (z tabeli Students), SessionID (z tabeli StudiesSessions)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - SessionID: ID sesji

```
CREATE TABLE StudentBoughtSessions(
    StudentID INT,
    SessionID INT,
    PRIMARY KEY (StudentID, SessionID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID),
    FOREIGN KEY (SessionID) REFERENCES StudiesSessions (SessionID)
)
```

- **InternshipMeetingTypes**: Przechowuje typy spotkań praktyk.
    - **Klucz podstawowy**: InternshipMeetingTypeID
    - **Pola tabeli**:
        - InternshipMeetingTypeID: ID typu spotkania
        - InternshipMeetingTypeName: Nazwa typu spotkania

```
CREATE TABLE InternshipMeetingTypes(
    InternshipMeetingTypeID INT PRIMARY KEY IDENTITY(1,1),
    InternshipMeetingTypeName VARCHAR(50) NOT NULL
)
```

- **Internships**: Przechowuje informacje o praktykach.
    - **Klucz podstawowy**: InternshipID
    - **Klucze obce**: StudiesID (z tabeli Studies), InternshipMeetingType (z tabeli InternshipMeetingTypes)
    - **Pola tabeli**:
        - InternshipID: ID praktyk
        - InternshipCompany: Firma praktyk
        - InternshipStartDate: Data rozpoczęcia praktyk
        - StudiesID: ID studiów
        - InternshipMeetingType: ID typu spotkania

```
CREATE TABLE Internships(
    InternshipID INT PRIMARY KEY IDENTITY(1,1),
    InternshipCompany VARCHAR(100) NOT NULL,
    InternshipStartDate DATE NOT NULL,
    StudiesID INT NOT NULL,
    InternshipMeetingType INT NOT NULL,
    FOREIGN KEY (StudiesID) REFERENCES Studies (StudiesID),
    FOREIGN KEY (InternshipMeetingType) REFERENCES InternshipMeetingTypes (InternshipMeetingTypeID)
)
```

- **InternshipMeetingsPassed**: Przechowuje informacje o zaliczonych praktykach przez studentów.
    - **Klucze podstawowe**: StudentID, InternshipID
    - **Klucze obce**: StudentID (z tabeli Students), InternshipID (z tabeli Internships)
    - **Pola tabeli**:
        - StudentID: ID studenta
        - InternshipID: ID praktyk

```
CREATE TABLE InternshipMeetingsPassed(
    StudentID INT,
    InternshipID INT,
    PRIMARY KEY (StudentID, InternshipID),
    FOREIGN KEY (InternshipID) REFERENCES Internships (InternshipID),
    FOREIGN KEY (StudentID) REFERENCES Students (StudentID)
)
```

- **OrderStudies**: Przechowuje informacje o studiach w zamówieniach.
    - **Klucz podstawowy**: OrderStudiesID
    - **Klucze obce**: OrderDetailID (z tabeli OrderDetails), StudiesID (z tabeli Studies)
    - **Pola tabeli**:
        - OrderStudiesID: ID studiów w zamówieniu
        - StudiesID: ID studiów
        - OrderDetailID: ID szczegółu zamówienia

```
CREATE TABLE OrderStudies(
    OrderStudiesID INT PRIMARY KEY IDENTITY(1,1),
    StudiesID INT NOT NULL,
    OrderDetailID INT NOT NULL,
    FOREIGN KEY (OrderDetailID) REFERENCES OrderDetails (OrderDetailID),
    FOREIGN KEY (StudiesID) REFERENCES Studies (StudiesID)
)
```

- **OrderSessions**: Przechowuje informacje o sesjach w zamówieniach.
    - **Klucz podstawowy**: OrderSessionID
    - **Klucze obce**: OrderDetailID (z tabeli OrderDetails), SessionID (z tabeli StudiesSessions)
    - **Pola tabeli**:
        - OrderSessionID: ID sesji w zamówieniu
        - SessionID: ID sesji
        - OrderDetailID: ID szczegółu zamówienia

```
CREATE TABLE OrderSessions(
    OrderSessionID INT PRIMARY KEY IDENTITY(1,1),
    SessionID INT NOT NULL,
    OrderDetailID INT NOT NULL,
    FOREIGN KEY (OrderDetailID) REFERENCES OrderDetails (OrderDetailID),
    FOREIGN KEY (SessionID) REFERENCES StudiesSessions (SessionID)
)
```

- **RODOSigns**: Przechowuje informacje o zgodach RODO użytkowników.
    - **Klucz podstawowy**: SignID
    - **Klucz obcy**: UserID (z tabeli Users)
    - **Pola tabeli**:
        - SignID: ID zgody
        - UserID: ID użytkownika
        - IsSigned: Czy zgoda została wyrażona
        - SignDate: Data wyrażenia zgody

```
CREATE TABLE RODOSigns (
    SignID INT PRIMARY KEY IDENTITY(1,1),
    UserID INT NOT NULL,
    IsSigned BIT NOT NULL,
    SignDate DATE NOT NULL,
    FOREIGN KEY (UserID) REFERENCES Users (UserID)
)
```

Widoki

DebtorsList

Ten widok zapewnia listę dłużników, pokazując kwotę należną dla każdego użytkownika.

```
CREATE VIEW DebtorsList AS
SELECT
    Users.UserID,
    Users.UserName,
    Users.UserSurname,
    Orders.OrderID,
    Orders.OrderDate,
    OrderDetails.AmountToPay,
    OrderDetails.AmountPaid,
    (OrderDetails.AmountToPay - OrderDetails.AmountPaid) AS AmountDue
FROM
    Users
INNER JOIN
    Orders ON Users.UserID = Orders.UserID
INNER JOIN
    OrderDetails ON Orders.OrderID = OrderDetails.OrderID
WHERE
    OrderDetails.AmountPaid < OrderDetails.AmountToPay;
```

PaymentStatus

Ten widok zapewnia status płatności zamówień, wskazując, czy są opłacone(`Paid`), czy oczekujące(`Pending`).

```
CREATE VIEW PaymentStatus AS
SELECT
    Orders.OrderID,
    Orders.UserID,
    Users.UserName,
    Users.UserSurname,
    OrderDetails.AmountPaid,
    OrderDetails.AmountToPay,
    CASE
        WHEN OrderDetails.AmountPaid >= OrderDetails.AmountToPay THEN 'Paid'
        ELSE 'Pending'
    END AS PaymentStatus
FROM
    Orders
INNER JOIN
    OrderDetails ON Orders.OrderID = OrderDetails.OrderID
INNER JOIN
    Users ON Orders.UserID = Users.UserID;
```

RegisteredParticipants

Ten widok zapewnia listę zarejestrowanych uczestników wydarzeń wykładowcy.

```
CREATE VIEW RegisteredParticipants AS
SELECT
    Webinars.WebinarID AS EventID,
    Webinars.WebinarName AS EventName,
    Users.UserID,
    Users.UserName,
    Users.UserSurname
FROM
    Webinars
INNER JOIN
    StudentBoughtWebinars ON Webinars.WebinarID = StudentBoughtWebinars.WebinarID
INNER JOIN
    Users ON StudentBoughtWebinars.StudentID = Users.UserID
WHERE
    Webinars.WebinarTeacherID = CURRENT_USER
UNION
SELECT
    Courses.CourseID AS EventID,
    Courses.CourseName AS EventName,
    Users.UserID,
    Users.UserName,
    Users.UserSurname
FROM
    Courses
INNER JOIN
    CourseModulesPassed ON Courses.CourseID = CourseModulesPassed.ModuleID
INNER JOIN
    Users ON CourseModulesPassed.StudentID = Users.UserID
WHERE
    Courses.CourseID IN (SELECT CourseID FROM CourseModules WHERE ModuleID = CourseModulesPassed.ModuleID)
UNION
SELECT
    StudiesLessons.LessonID AS EventID,
    StudiesLessons.LessonName AS EventName,
    Users.UserID,
    Users.UserName,
    Users.UserSurname
FROM
    StudiesLessons
INNER JOIN
    StudiesLessonPassed ON StudiesLessons.LessonID = StudiesLessonPassed.LessonID
INNER JOIN
    Users ON StudiesLessonPassed.StudentID = Users.UserID
WHERE
    StudiesLessons.TeacherID = CURRENT_USER;
```

EventDetails

Ten widok zapewnia szczegóły wydarzeń dla wykładowcy.

```
CREATE VIEW EventDetails AS
SELECT
    Webinars.WebinarID AS EventID,
    Webinars.WebinarName AS EventName,
    Webinars.WebinarStartDate AS EventDate,
    Webinars.WebinarDescription AS EventDescription,
    Webinars.WebinarTeacherID AS LecturerID
FROM
    Webinars
WHERE
    Webinars.WebinarTeacherID = CURRENT_USER
UNION
SELECT
    Courses.CourseID AS EventID,
    Courses.CourseName AS EventName,
    NULL AS EventDate,
    Courses.CourseDescription AS EventDescription,
    NULL AS LecturerID
FROM
    Courses
UNION
SELECT
    StudiesLessons.LessonID AS EventID,
    StudiesLessons.LessonName AS EventName,
    StudiesLessons.LessonMeetingDateStart AS EventDate,
    StudiesLessons.LessonDescription AS EventDescription,
    StudiesLessons.TeacherID AS LecturerID
FROM
    StudiesLessons
WHERE
    StudiesLessons.TeacherID = CURRENT_USER;
```

AttendanceReports

Ten widok zapewnia raporty frekwencji dla wydarzeń wykładowcy.

```
CREATE VIEW AttendanceReports AS
SELECT
    Webinars.WebinarID AS EventID,
    Webinars.WebinarName AS EventName,
    Users.UserID,
    Users.UserName,
    Users.UserSurname,
    WebinarsPassed.PassedDate AS AttendanceDate
FROM
    Webinars
INNER JOIN
    WebinarsPassed ON Webinars.WebinarID = WebinarsPassed.WebinarID
INNER JOIN
    Users ON WebinarsPassed.StudentID = Users.UserID
WHERE
    Webinars.WebinarTeacherID = CURRENT_USER
UNION
SELECT
    Courses.CourseID AS EventID,
    Courses.CourseName AS EventName,
    Users.UserID,
    Users.UserName,
    Users.UserSurname,
    CourseLessonsPassed.PassedDate AS AttendanceDate
FROM
    Courses
INNER JOIN
    CourseLessonsPassed ON Courses.CourseID = CourseLessonsPassed.LessonID
INNER JOIN
    Users ON CourseLessonsPassed.StudentID = Users.UserID
WHERE
    Courses.CourseID IN (SELECT CourseID FROM CourseModules WHERE ModuleID = CourseLessonsPassed.LessonID)
UNION
SELECT
    StudiesLessons.LessonID AS EventID,
    StudiesLessons.LessonName AS EventName,
    Users.UserID,
    Users.UserName,
    Users.UserSurname,
    StudiesLessonPassed.PassedDate AS AttendanceDate
FROM
    StudiesLessons
INNER JOIN
    StudiesLessonPassed ON StudiesLessons.LessonID = StudiesLessonPassed.LessonID
INNER JOIN
    Users ON StudiesLessonPassed.StudentID = Users.UserID
WHERE
    StudiesLessons.TeacherID = CURRENT_USER;
```

TranslatedEvents

Ten widok zapewnia listę wydarzeń, do których przypisany jest tłumacz.

```
CREATE VIEW TranslatedEvents AS
SELECT
    Webinars.WebinarID AS EventID,
    Webinars.WebinarName AS EventName,
    Webinars.WebinarStartDate AS EventDate,
    Webinars.WebinarDescription AS EventDescription,
    Webinars.WebinarTranslatorID AS TranslatorID
FROM
    Webinars
WHERE
    Webinars.WebinarTranslatorID = CURRENT_USER
UNION
SELECT
    Courses.CourseID AS EventID,
    Courses.CourseName AS EventName,
    NULL AS EventDate,
    Courses.CourseDescription AS EventDescription,
    NULL AS TranslatorID
FROM
    Courses
UNION
SELECT
    StudiesLessons.LessonID AS EventID,
    StudiesLessons.LessonName AS EventName,
    StudiesLessons.LessonMeetingDateStart AS EventDate,
    StudiesLessons.LessonDescription AS EventDescription,
    StudiesLessons.TranslatorID AS TranslatorID
FROM
    StudiesLessons
WHERE
    StudiesLessons.TranslatorID = CURRENT_USER;
```

TranslatorEventDetails

Ten widok zapewnia szczegóły wydarzeń dla tłumacza.

```
CREATE VIEW TranslatorEventDetails AS
SELECT
    Webinars.WebinarID AS EventID,
    Webinars.WebinarName AS EventName,
    Webinars.WebinarStartDate AS EventDate,
    Webinars.WebinarDescription AS EventDescription,
    Webinars.WebinarTranslatorID AS TranslatorID
FROM
    Webinars
WHERE
    Webinars.WebinarTranslatorID = CURRENT_USER
UNION
SELECT
    Courses.CourseID AS EventID,
    Courses.CourseName AS EventName,
    NULL AS EventDate,
    Courses.CourseDescription AS EventDescription,
    NULL AS TranslatorID
FROM
    Courses
UNION
SELECT
    StudiesLessons.LessonID AS EventID,
    StudiesLessons.LessonName AS EventName,
    StudiesLessons.LessonMeetingDateStart AS EventDate,
    StudiesLessons.LessonDescription AS EventDescription,
    StudiesLessons.TranslatorID AS TranslatorID
FROM
    StudiesLessons
WHERE
    StudiesLessons.TranslatorID = CURRENT_USER;
```

AvailableEvents

Ten widok zapewnia listę dostępnych wydarzeń dla studentów.

```
CREATE VIEW AvailableEvents AS
SELECT
    Webinars.WebinarID AS EventID,
    Webinars.WebinarName AS EventName,
    Webinars.WebinarStartDate AS EventDate,
    Webinars.WebinarDescription AS EventDescription
FROM
    Webinars
WHERE
    Webinars.WebinarStartDate > CURRENT_DATE
UNION
SELECT
    Courses.CourseID AS EventID,
    Courses.CourseName AS EventName,
    NULL AS EventDate,
    Courses.CourseDescription AS EventDescription
FROM
    Courses
UNION
SELECT
    StudiesLessons.LessonID AS EventID,
    StudiesLessons.LessonName AS EventName,
    StudiesLessons.LessonMeetingDateStart AS EventDate,
    StudiesLessons.LessonDescription AS EventDescription
FROM
    StudiesLessons
WHERE
    StudiesLessons.LessonMeetingDateStart > CURRENT_DATE;
```

RegisteredEventDetails

Ten widok zapewnia szczegóły zarejestrowanych wydarzeń dla studentów.

```
CREATE VIEW
    Webinars.WebinarID AS EventID,
    Webinars.WebinarName AS EventName,
    Webinars.WebinarStartDate AS EventDate,
    Webinars.WebinarDescription AS EventDescription,
    StudentBoughtWebinars.StudentID
FROM
    Webinars
INNER JOIN
    StudentBoughtWebinars ON Webinars.WebinarID = StudentBoughtWebinars.WebinarID
WHERE
    StudentBoughtWebinars.StudentID = CURRENT_USER
UNION
SELECT
    Courses.CourseID AS EventID,
    Courses.CourseName AS EventName,
    NULL AS EventDate,
    Courses.CourseDescription AS EventDescription,
    CourseModulesPassed.StudentID
FROM
    Courses
INNER JOIN
    CourseModulesPassed ON Courses.CourseID = CourseModulesPassed.ModuleID
WHERE
    CourseModulesPassed.StudentID = CURRENT_USER
UNION
SELECT
    StudiesLessons.LessonID AS EventID,
    StudiesLessons.LessonName AS EventName,
    StudiesLessons.LessonMeetingDateStart AS EventDate,
    StudiesLessons.LessonDescription AS EventDescription,
    StudiesLessonPassed.StudentID
FROM
    StudiesLessons
INNER JOIN
    StudiesLessonPassed ON StudiesLessons.LessonID = StudiesLessonPassed.LessonID
WHERE
    StudiesLessonPassed.StudentID = CURRENT_USER;
```

CompletionStatus

Ten widok zapewnia status ukończenia dla studentów.

```
CREATE VIEW CompletionStatus AS
SELECT
    CourseModulesPassed.StudentID,
    CourseModulesPassed.ModuleID AS ItemID,
    'CourseModule' AS ItemType,
    'Passed' AS Status
FROM
    CourseModulesPassed
UNION
SELECT
    CourseLessonsPassed.StudentID,
    CourseLessonsPassed.LessonID AS ItemID,
    'CourseLesson' AS ItemType,
    'Passed' AS Status
FROM
    CourseLessonsPassed
UNION
SELECT
    WebinarsPassed.StudentID,
    WebinarsPassed.WebinarID AS ItemID,
    'Webinar' AS ItemType,
    'Passed' AS Status
FROM
    WebinarsPassed
UNION
SELECT
    StudiesLessonPassed.StudentID,
    StudiesLessonPassed.LessonID AS ItemID,
    'StudiesLesson' AS ItemType,
    'Passed' AS Status
FROM
    StudiesLessonPassed;
```

StudentPaymentStatus

Ten widok zapewnia status płatności dla studentów.

```
CREATE VIEW StudentPaymentStatus AS
SELECT
    Orders.OrderID,
    Orders.UserID,
    Users.UserName,
    Users.UserSurname,
    OrderDetails.AmountPaid,
    OrderDetails.AmountToPay,
    CASE
        WHEN OrderDetails.AmountPaid >= OrderDetails.AmountToPay THEN 'Paid'
        ELSE 'Pending'
    END AS PaymentStatus
FROM
    Orders
INNER JOIN
    OrderDetails ON Orders.OrderID = OrderDetails.OrderID
INNER JOIN
    Users ON Orders.UserID = Users.UserID
WHERE
    Orders.UserID = CURRENT_USER;
```

Funkcje

GetAvailableEvents

Przechowywanie listy dostępnych webinarów, kursów i studiów

```
CREATE FUNCTION GetAvailableEvents()
RETURNS TABLE
AS
RETURN
(
    SELECT
        'Webinar' AS EventType,
        WebinarID AS EventID,
        WebinarName AS EventName,
        WebinarDate AS EventDate,
        WebinarLanguageID AS LanguageID
    FROM Webinars
    UNION ALL
    SELECT
        'Course' AS EventType,
        CourseID AS EventID,
        CourseName AS EventName,
        CourseStartDate AS EventDate,
        CourseLanguageID AS LanguageID
    FROM Courses
    UNION ALL
    SELECT
        'Study' AS EventType,
        StudiesID AS EventID,
        StudiesName AS EventName,
        StudiesStartDate AS EventDate,
        StudiesLanguageID AS LanguageID
    FROM Studies
);
```

GetSyllabusesAndSchedules

Przechowywanie sylabusów i harmonogramów

```
CREATE FUNCTION GetSyllabusesAndSchedules()
RETURNS TABLE
AS
RETURN
(
    SELECT
        s.StudiesID,
        s.StudiesName,
        sy.SyllabusID,
        sy.SyllabusContent,
        sl.ScheduleID,
        sl.ScheduleContent
    FROM Studies s
    JOIN Syllabuses sy ON s.StudiesID = sy.StudiesID
    JOIN Schedules sl ON s.StudiesID = sl.StudiesID
);
```

GetEventDetails

Przechowywanie szczegółów wydarzenia (np. daty, miejsca, linków do spotkań online)

```
CREATE FUNCTION GetEventDetails()
RETURNS TABLE
AS
RETURN
(
    SELECT
        'Webinar' AS EventType,
        WebinarID AS EventID,
        WebinarName AS EventName,
        WebinarDate AS EventDate,
        WebinarLocation AS EventLocation,
        WebinarLink AS EventLink
    FROM Webinars
    UNION ALL
    SELECT
        'Course' AS EventType,
        CourseID AS EventID,
        CourseName AS EventName,
        CourseStartDate AS EventDate,
        CourseLocation AS EventLocation,
        CourseLink AS EventLink
    FROM Courses
    UNION ALL
    SELECT
        'Study' AS EventType,
        StudiesID AS EventID,
        StudiesName AS EventName,
        StudiesStartDate AS EventDate,
        StudiesLocation AS EventLocation,
        StudiesLink AS EventLink
    FROM Studies
);
```

GetCompletionStatus

Przechowywanie statusu zaliczeń (moduły kursowe, frekwencja na studiach)

```
CREATE FUNCTION GetCompletionStatus()
RETURNS TABLE
AS
RETURN
(
    SELECT
        'CourseModule' AS CompletionType,
        cm.ModuleID AS CompletionID,
        cm.ModuleName AS CompletionName,
        cmp.StudentID,
        cmp.CompletionDate
    FROM CourseModules cm
    JOIN CourseModulesPassed cmp ON cm.ModuleID = cmp.ModuleID
    UNION ALL
    SELECT
        'StudyAttendance' AS CompletionType,
        sl.LessonID AS CompletionID,
        sl.LessonName AS CompletionName,
        slp.StudentID,
        slp.AttendanceDate AS CompletionDate
    FROM StudiesLessons sl
    JOIN StudiesLessonsPassed slp ON sl.LessonID = slp.LessonID
);
```

GetPaymentStatus

Sprawdzanie statusu płatności

```
CREATE FUNCTION GetPaymentStatus()
RETURNS TABLE
AS
RETURN
(
    SELECT
        o.OrderID,
        o.UserID,
        u.UserName,
        u.UserSurname,
        od.AmountPaid,
        od.AmountToPay,
        CASE
            WHEN od.AmountPaid >= od.AmountToPay THEN 'Paid'
            ELSE 'Pending'
        END AS PaymentStatus
    FROM
        Orders o
    JOIN
        OrderDetails od ON o.OrderID = od.OrderID
    JOIN
        Users u ON o.UserID = u.UserID
);
```

GetFinancialReports

Generowanie raportów finansowych (przychody dla każdego typu wydarzeń)

```
CREATE FUNCTION GetFinancialReports()
RETURNS TABLE
AS
RETURN
(
    SELECT
        'Webinar' AS EventType,
        w.WebinarID AS EventID,
        w.WebinarName AS EventName,
        SUM(od.AmountPaid) AS TotalRevenue
    FROM Webinars w
    JOIN OrderDetails od ON w.WebinarID = od.EventID
    GROUP BY w.WebinarID, w.WebinarName
    UNION ALL
    SELECT
        'Course' AS EventType,
        c.CourseID AS EventID,
        c.CourseName AS EventName,
        SUM(od.AmountPaid) AS TotalRevenue
    FROM Courses c
    JOIN OrderDetails od ON c.CourseID = od.EventID
    GROUP BY c.CourseID, c.CourseName
    UNION ALL
    SELECT
        'Study' AS EventType,
        s.StudiesID AS EventID,
        s.StudiesName AS EventName,
        SUM(od.AmountPaid) AS TotalRevenue
    FROM Studies s
    JOIN OrderDetails od ON s.StudiesID = od.EventID
    GROUP BY s.StudiesID, s.StudiesName
);
```

GetEnrollmentReports

Wyświetlanie raportu liczby zapisanych osób

```
CREATE FUNCTION GetEnrollmentReports()
RETURNS TABLE
AS
RETURN
(
    SELECT
        'Webinar' AS EventType,
        w.WebinarID AS EventID,
        w.WebinarName AS EventName,
        COUNT(sbw.StudentID) AS EnrollmentCount
    FROM Webinars w
    JOIN StudentBoughtWebinars sbw ON w.WebinarID = sbw.WebinarID
    GROUP BY w.WebinarID, w.WebinarName
    UNION ALL
    SELECT
        'Course' AS EventType,
        c.CourseID AS EventID,
        c.CourseName AS EventName,
        COUNT(sbc.StudentID) AS EnrollmentCount
    FROM Courses c
    JOIN StudentBoughtCourses sbc ON c.CourseID = sbc.CourseID
    GROUP BY c.CourseID, c.CourseName
    UNION ALL
    SELECT
        'Study' AS EventType,
        s.StudiesID AS EventID,
        s.StudiesName AS EventName,
        COUNT(sbs.StudentID) AS EnrollmentCount
    FROM Studies s
    JOIN StudentBoughtStudies sbs ON s.StudiesID = sbs.StudiesID
    GROUP BY s.StudiesID, s.StudiesName
);
```

GetAttendanceReports

Tworzenie list obecności oraz raportów dotyczących frekwencji

```
CREATE FUNCTION GetAttendanceReports()
RETURNS TABLE
AS
RETURN
(
    SELECT
        'CourseModule' AS EventType,
        cm.ModuleID AS EventID,
        cm.ModuleName AS EventName,
        COUNT(cmp.StudentID) AS AttendanceCount
    FROM CourseModules cm
    JOIN CourseModulesPassed cmp ON cm.ModuleID = cmp.ModuleID
    GROUP BY cm.ModuleID, cm.ModuleName
    UNION ALL
    SELECT
        'StudyLesson' AS EventType,
        sl.LessonID AS EventID,
        sl.LessonName AS EventName,
        COUNT(slp.StudentID) AS AttendanceCount
    FROM StudiesLessons sl
    JOIN StudiesLessonsPassed slp ON sl.LessonID = slp.LessonID
    GROUP BY sl.LessonID, sl.LessonName
);
```

FilteredEvents

Filtrowanie po rodzaju (webinary, kursy, studia), formie (stacjonarne, online, hybrydowe), języku, dostępności miejsc, cenie

```
CREATE FUNCTION FilteredEvents()
RETURNS TABLE
AS
RETURN
(
    SELECT WebinarID AS EventID, WebinarName AS EventName, 'Webinar' AS EventType, WebinarLanguageID AS
LanguageID, WebinarPrice AS Price, NULL AS Capacity
    FROM Webinars
    UNION
    SELECT CourseID AS EventID, CourseName AS EventName, 'Course' AS EventType, CourseLanguageID AS
LanguageID, CoursePrice AS Price, NULL AS Capacity
    FROM Courses
    UNION
    SELECT StudiesID AS EventID, StudiesName AS EventName, 'Studies' AS EventType, NULL AS LanguageID, NULL AS
Price, NULL AS Capacity
    FROM Studies
);
```

Procedury

AddUser

Dodaje nowego użytkownika do systemu

```
CREATE PROCEDURE AddUser
    @UserName VARCHAR(50),
    @UserSurname VARCHAR(50),
    @UserDateOfBirth DATE,
    @UserSex CHAR(1),
    @UserEmail VARCHAR(100),
    @UserPhoneNumber VARCHAR(20),
    @UserAddress VARCHAR(100),
    @UserCity VARCHAR(50),
    @UserCountry VARCHAR(50),
    @UserPostalCode VARCHAR(10)
AS
BEGIN
    INSERT INTO Users (UserName, UserSurname, UserDateOfBirth, UserSex, UserEmail, UserPhoneNumber,
UserAddress, UserCity, UserCountry, UserPostalCode)
    VALUES (@UserName, @UserSurname, @UserDateOfBirth, @UserSex, @UserEmail, @UserPhoneNumber, @UserAddress,
@UserCity, @UserCountry, @UserPostalCode);
END;
```

UpdateUserProfile

Aktualizuje dane profilu użytkownika

```
CREATE PROCEDURE UpdateUserProfile
    @UserID INT,
    @UserName VARCHAR(50),
    @UserSurname VARCHAR(50),
    @UserDateOfBirth DATE,
    @UserSex CHAR(1),
    @UserEmail VARCHAR(100),
    @UserPhoneNumber VARCHAR(20),
    @UserAddress VARCHAR(100),
    @UserCity VARCHAR(50),
    @UserCountry VARCHAR(50),
    @UserPostalCode VARCHAR(10)
AS
BEGIN
    UPDATE Users
    SET UserName = @UserName,
        UserSurname = @UserSurname,
        UserDateOfBirth = @UserDateOfBirth,
        UserSex = @UserSex,
        UserEmail = @UserEmail,
        UserPhoneNumber = @UserPhoneNumber,
        UserAddress = @UserAddress,
        UserCity = @UserCity,
        UserCountry = @UserCountry,
        UserPostalCode = @UserPostalCode
    WHERE UserID = @UserID;
END;
```

DeleteUser

Usuwa użytkownika z systemu

```
CREATE PROCEDURE DeleteUser
    @UserID INT
AS
BEGIN
    DELETE FROM Users
    WHERE UserID = @UserID;
END;
```

AddRole

Dodawanie użytkownikowi nowej roli

```
CREATE PROCEDURE AddRole
    @UserID INT,
    @RoleID INT
AS
BEGIN
    INSERT INTO UserRoles (UserID, RoleID)
    VALUES (@UserID, @RoleID);
END;
```

RemoveRole

Usuwanie użytkownikowi roli

```
CREATE PROCEDURE RemoveRole
    @UserID INT,
    @RoleID INT
AS
BEGIN
    DELETE FROM UserRoles
    WHERE UserID = @UserID AND RoleID = @RoleID;
END;
```

AddEvent

Dodaje nowe wydarzenie (webinar, kurs, studium)

```
CREATE PROCEDURE AddEvent
    @EventName VARCHAR(100),
    @EventType VARCHAR(50),
    @EventStartDate DATE,
    @EventEndDate DATE,
    @EventDescription VARCHAR(MAX),
    @EventLanguageID INT,
    @EventPrice MONEY,
    @EventTranslatorID INT,
    @EventTeacherID INT,
    @EventMeetingLink VARCHAR(100)
AS
BEGIN
    IF @EventType = 'Webinar'
    BEGIN
        INSERT INTO Webinars (WebinarName, WebinarStartDate, WebinarEndDate, WebinarDescription,
WebinarLanguageID, WebinarPrice, WebinarTranslatorID, WebinarTeacherID, WebinarMeetingLink)
        VALUES (@EventName, @EventStartDate, @EventEndDate, @EventDescription, @EventLanguageID, @EventPrice,
@EventTranslatorID, @EventTeacherID, @EventMeetingLink);
    END
    ELSE IF @EventType = 'Course'
    BEGIN
        INSERT INTO Courses (CourseName, CourseDescription, CourseLanguageID, CoursePrice)
        VALUES (@EventName, @EventDescription, @EventLanguageID, @EventPrice);
    END
    ELSE IF @EventType = 'Studies'
    BEGIN
        INSERT INTO Studies (StudiesName, StudiesDescription, StudiesStartDate, StudiesEndDate)
        VALUES (@EventName, @EventDescription, @EventStartDate, @EventEndDate);
    END
END;
```

UpdateEventDetails

Aktualizuje szczegóły wydarzenia

```
CREATE PROCEDURE UpdateEventDetails
    @EventID INT,
    @EventType VARCHAR(50),
    @EventName VARCHAR(100),
    @EventStartDate DATE,
    @EventEndDate DATE,
    @EventDescription VARCHAR(MAX),
    @EventLanguageID INT,
    @EventPrice MONEY,
    @EventTranslatorID INT,
    @EventTeacherID INT,
    @EventMeetingLink VARCHAR(100)
AS
BEGIN
    IF @EventType = 'Webinar'
    BEGIN
        UPDATE Webinars
        SET WebinarName = @EventName,
            WebinarStartDate = @EventStartDate,
            WebinarEndDate = @EventEndDate,
            WebinarDescription = @EventDescription,
            WebinarLanguageID = @EventLanguageID,
            WebinarPrice = @EventPrice,
            WebinarTranslatorID = @EventTranslatorID,
            WebinarTeacherID = @EventTeacherID,
            WebinarMeetingLink = @EventMeetingLink
        WHERE WebinarID = @EventID;
    END
    ELSE IF @EventType = 'Course'
    BEGIN
        UPDATE Courses
        SET CourseName = @EventName,
            CourseDescription = @EventDescription,
            CourseLanguageID = @EventLanguageID,
            CoursePrice = @EventPrice
        WHERE CourseID = @EventID;
    END
    ELSE IF @EventType = 'Studies'
    BEGIN
        UPDATE Studies
        SET StudiesName = @EventName,
            StudiesDescription = @EventDescription,
            StudiesStartDate = @EventStartDate,
            StudiesEndDate = @EventEndDate
        WHERE StudiesID = @EventID;
    END
END;
```

DeleteEvent

Usuwa wydarzenie z systemu

```
CREATE PROCEDURE DeleteEvent
    @EventID INT,
    @EventType VARCHAR(50)
AS
BEGIN
    IF @EventType = 'Webinar'
    BEGIN
        DELETE FROM Webinars
        WHERE WebinarID = @EventID;
    END
    ELSE IF @EventType = 'Course'
    BEGIN
        DELETE FROM Courses
        WHERE CourseID = @EventID;
    END
    ELSE IF @EventType = 'Studies'
    BEGIN
        DELETE FROM Studies
        WHERE StudiesID = @EventID;
    END
END;
```

CreatePaymentLink

Generuje link do płatności

```
CREATE PROCEDURE CreatePaymentLink
    @OrderID INT,
    @PaymentLink VARCHAR(100)
AS
BEGIN
    UPDATE Orders
    SET OrderPaymentLink = @PaymentLink
    WHERE OrderID = @OrderID;
END;
```

ProcessPaymentResult

Przetwarza wynik płatności (udana/nieudana)

```
CREATE PROCEDURE ProcessPaymentResult
    @OrderDetailID INT,
    @AmountPaid MONEY,
    @PaidDate DATE
AS
BEGIN
    UPDATE OrderDetails
    SET AmountPaid = @AmountPaid,
        PaidDate = @PaidDate
    WHERE OrderDetailID = @OrderDetailID;
END;
```

ApplyPaymentException

Zapisuje wyjątek płatniczy (np. odroczenie)

```
CREATE PROCEDURE ApplyPaymentException
    @OrderDetailID INT,
    @PostponementDate DATE
AS
BEGIN
    UPDATE OrderDetails
    SET PostponementDate = @PostponementDate
    WHERE OrderDetailID = @OrderDetailID;
END;
```

RegisterForEvent

Rejestruje użytkownika na wydarzenie

```
CREATE PROCEDURE RegisterForEvent
    @UserID INT,
    @EventID INT,
    @EventType VARCHAR(50)
AS
BEGIN
    IF @EventType = 'Webinar'
    BEGIN
        INSERT INTO StudentBoughtWebinars (StudentID, WebinarID, WebinarBoughtDate, WebinarAccessDateEnd)
        VALUES (@UserID, @EventID, GETDATE(), DATEADD(DAY, 30, GETDATE()));
    END
    ELSE IF @EventType = 'Course'
    BEGIN
        INSERT INTO CourseModulesPassed (StudentID, ModuleID)
        VALUES (@UserID, @EventID);
    END
    ELSE IF @EventType = 'Studies'
    BEGIN
        INSERT INTO StudentStudies (StudentID, StudiesID)
        VALUES (@UserID, @EventID);
    END
END;
```

CancelRegistration

Anuluje rejestrację na wydarzenie

```
CREATE PROCEDURE CancelRegistration
    @UserID INT,
    @EventID INT,
    @EventType VARCHAR(50)
AS
BEGIN
    IF @EventType = 'Webinar'
    BEGIN
        DELETE FROM StudentBoughtWebinars
        WHERE StudentID = @UserID AND WebinarID = @EventID;
    END
    ELSE IF @EventType = 'Course'
    BEGIN
        DELETE FROM CourseModulesPassed
        WHERE StudentID = @UserID AND ModuleID = @EventID;
    END
    ELSE IF @EventType = 'Studies'
    BEGIN
        DELETE FROM StudentStudies
        WHERE StudentID = @UserID AND StudiesID = @EventID;
    END
END;
```

Triggery

HandleWebinarPayment

Ten trigger obsługuje płatności za webinary i dodaje studenta do listy zakupionych webinarów, jeśli płatność jest pełna lub odroczona.

```
CREATE TRIGGER HandleWebinarPayment
ON OrderDetails
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT Orders.UserID, Webinars.WebinarID
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderWebinars
        ON OrderWebinars.OrderDetailID = inserted.OrderDetailID
        INNER JOIN Webinars
        ON Webinars.WebinarID = OrderWebinars.WebinarID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND WebinarStartDate > GETDATE()
        AND Orders.UserID IN (
            SELECT DISTINCT StudentID FROM StudentBoughtWebinars
            )
       )
    BEGIN
        RAISERROR('Student already bought that webinar', 1, 1)
    END
    ELSE
    BEGIN
        INSERT INTO StudentBoughtWebinars(StudentID, WebinarID, WebinarBoughtDate, WebinarAccessDateEnd)
        (SELECT Orders.UserID as StudentID, Webinars.WebinarID, GETDATE() as WebinarBoughtDate,
                DATEADD(DAY, 30, GETDATE()) as WebinarAccessDateEnd
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderWebinars
        ON OrderWebinars.OrderDetailID = inserted.OrderDetailID
        INNER JOIN Webinars
        ON Webinars.WebinarID = OrderWebinars.WebinarID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND WebinarStartDate > GETDATE())
    END
END;
```

HandleCoursePayment

Ten trigger obsługuje płatności za kursy i dodaje studenta do listy zakupionych kursów, jeśli płatność jest pełna lub odroczona.

```
CREATE TRIGGER HandleCoursePayment
ON OrderDetails
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT Orders.UserID, Courses.CourseID
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderCourses
        ON OrderCourses.OrderDetailID = inserted.OrderDetailID
        INNER JOIN Courses
        ON Courses.CourseID = OrderCourses.CourseID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND DATEADD(DAY, 3, GETDATE()) < Courses.CourseStartDate
        AND Orders.UserID IN (
            SELECT DISTINCT StudentID FROM StudentBoughtCourses
            )
       )
    BEGIN
        RAISERROR('Student already bought that course', 1, 1)
    END
    ELSE
    BEGIN
        INSERT INTO StudentBoughtCourses(StudentID, CourseID)
        (SELECT Orders.UserID as StudentID, Courses.CourseID
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderCourses
        ON OrderCourses.OrderDetailID = inserted.OrderDetailID
        INNER JOIN Courses
        ON Courses.CourseID = OrderCourses.CourseID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND DATEADD(DAY, 3, GETDATE()) < Courses.CourseStartDate)
    END
END
```

HandleSessionPayment

Ten trigger obsługuje płatności za sesje studiów i dodaje studenta do listy zakupionych sesji, jeśli płatność jest pełna lub odroczona.

```
CREATE TRIGGER HandleSessionPayment
ON OrderDetails
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT Orders.UserID, StudiesSessions.SessionID
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderSessions
        ON OrderSessions.OrderDetailID = inserted.OrderDetailID
        INNER JOIN Courses
        ON Courses.CourseID = OrderSessions.SessionID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND DATEADD(DAY, 3, GETDATE()) < StudiesSessions.SessionStartDay
        AND Orders.UserID IN (
            SELECT DISTINCT StudentID FROM StudentBoughtSessions
            )
       )
    BEGIN
        RAISERROR('Student already bought that session', 1, 1)
    END
    ELSE
    BEGIN
        INSERT INTO StudentBoughtSessions(StudentID, SessionID)
        (SELECT Orders.UserID as StudentID, Courses.CourseID
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderSessions
        ON OrderSessions.OrderDetailID = inserted.OrderDetailID
        INNER JOIN StudiesSessions
        ON StudiesSessions.SessionID = OrderSessions.SessionID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND DATEADD(DAY, 3, GETDATE()) < StudiesSessions.SessionStartDate)
    END
END;
```

HandleStudiesPayment

Ten trigger obsługuje płatności za studia i dodaje studenta do listy zakupionych studiów, jeśli płatność jest pełna lub odroczona.

```
CREATE TRIGGER HandleStudiesPayment
ON OrderDetails
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT Orders.UserID, Studies.StudiesID
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderStudies
        ON OrderStudies.OrderDetailID = inserted.OrderDetailID
        INNER JOIN Studies
        ON Studies.StudiesID = OrderStudies.StudiesID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND DATEADD(DAY, 3, Studies.StudiesStartDate) < GETDATE()
        AND Orders.UserID IN (
            SELECT DISTINCT StudentID FROM StudentBoughtSessions
            )
       )
    BEGIN
        RAISERROR('Student already bought that studies', 1, 1)
    END
    ELSE
    BEGIN
        INSERT INTO StudentBoughtSessions(StudentID, StudiesID)
        (SELECT Orders.UserID as StudentID, Studies.StudiesID
        FROM inserted
        INNER JOIN Orders
        ON Orders.OrderID = inserted.OrderID
        INNER JOIN OrderStudies
        ON OrderStudies.OrderDetailID = inserted.OrderDetailID
        INNER JOIN Studies
        ON Studies.StudiesID = OrderStudies.StudiesID
        WHERE inserted.AmountPaid = inserted.AmountToPay OR inserted.PostponementDate > GETDATE()
        AND StudiesStartDate > GETDATE())
    END
END;
```

VerifyOnlineParticipation

Ten trigger automatyczne weryfikowanie udziału w wydarzeniach online

```
CREATE TRIGGER VerifyOnlineParticipation
ON StudentBoughtWebinars
AFTER INSERT
AS
BEGIN
    UPDATE StudentBoughtWebinars
    SET WebinarAccessDateEnd = DATEADD(DAY, 30, WebinarBoughtDate)
    WHERE WebinarAccessDateEnd IS NULL;
END;
```

Indeksy

Users

Indeksuje po kolumnach `UserEmail` i `UserCity`.

```
CREATE NONCLUSTERED INDEX idx_users_email ON Users(UserEmail);
CREATE NONCLUSTERED INDEX idx_users_city ON Users(UserCity);
```

Languages

Indeksuje po kolumnie `LanguageName`.

```
CREATE NONCLUSTERED INDEX idx_languages_name ON Languages(LanguageName);
```

CurrencyRates

Indeksuje po kolumnie `CurrencyName`.

```
CREATE NONCLUSTERED INDEX idx_currencyrates_name ON CurrencyRates(CurrencyName);
```

EmployeeTypes

Indeksuje po kolumnie `EmployeeTypeName`.

```
CREATE NONCLUSTERED INDEX idx_employeetypes_name ON EmployeeTypes(EmployeeTypeName);
```

Employees

Indeksuje po kolumnach `UserID` i `EmployeeTypeID`.

```
CREATE NONCLUSTERED INDEX idx_employees_userid ON Employees(UserID);
CREATE NONCLUSTERED INDEX idx_employees_employeetypeid ON Employees(EmployeeTypeID);
```

TeachingLanguages

Indeksuje po kolumnie `EmployeeID`.

```
CREATE NONCLUSTERED INDEX idx_teachinglanguages_employeeid ON TeachingLanguages(EmployeeID);
```

Students

Indeksuje po kolumnie `UserID`.

```
CREATE NONCLUSTERED INDEX idx_students_userid ON Students(UserID);
```

StudyingLanguages

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studyinglanguages_studentid ON StudyingLanguages(StudentID);
```

TranslatingLanguages

Indeksuje po kolumnie `EmployeeID`.

```
CREATE NONCLUSTERED INDEX idx_translatinglanguages_employeeid ON TranslatingLanguages(EmployeeID);
```

Webinars

Indeksuje po kolumnie `WebinarLanguageID`.

```
CREATE NONCLUSTERED INDEX idx_webinars_languageid ON Webinars(WebinarLanguageID);
```

StudentBoughtWebinars

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studentboughtwebinars_studentid ON StudentBoughtWebinars(StudentID);
```

Cities

Indeksuje po kolumnie `CityName`.

```
CREATE NONCLUSTERED INDEX idx_cities_cityname ON Cities(CityName);
```

UserAvailableCities

Indeksuje po kolumnie `UserID`.

```
CREATE NONCLUSTERED INDEX idx_useravailablecities_userid ON UserAvailableCities(UserID);
```

Courses

Indeksuje po kolumnie `CourseLanguageID`.

```
CREATE NONCLUSTERED INDEX idx_courses_languageid ON Courses(CourseLanguageID);
```

StudentBoughtCourses

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studentboughtcourses_studentid ON StudentBoughtCourses(StudentID);
```

CourseModuleMeetingTypes

Indeksuje po kolumnie `MeetingTypeName`.

```
CREATE NONCLUSTERED INDEX idx_coursemodulemeetingtypes_name ON CourseModuleMeetingTypes(MeetingTypeName);
```

CourseModules

Indeksuje po kolumnie `CourseID`.

```
CREATE NONCLUSTERED INDEX idx_coursemodules_courseid ON CourseModules(CourseID);
```

CourseLessonMeetingTypes

Indeksuje po kolumnie `MeetingTypeName`.

```
CREATE NONCLUSTERED INDEX idx_courselessonmeetingtypes_name ON CourseLessonMeetingTypes(MeetingTypeName);
```

CourseLessons

Indeksuje po kolumnie `ModuleID`.

```
CREATE NONCLUSTERED INDEX idx_courselessons_moduleid ON CourseLessons(ModuleID);
```

CourseModulesPassed

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_coursemodulespassed_studentid ON CourseModulesPassed(StudentID);
```

CourseLessonsPassed

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_courselessonspassed_studentid ON CourseLessonsPassed(StudentID);
```

WebinarsPassed

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_webinarspassed_studentid ON WebinarsPassed(StudentID);
```

Orders

Indeksuje po kolumnie `UserID`.

```
CREATE CLUSTERED INDEX idx_orders_userid ON Orders(UserID);
```

OrderDetails

Indeksuje po kolumnie `OrderID`.

```
CREATE NONCLUSTERED INDEX idx_orderdetails_orderid ON OrderDetails(OrderID);
```

OrderWebinars

Indeksuje po kolumnie `OrderDetailID`.

```
CREATE NONCLUSTERED INDEX idx_orderwebinars_orderdetailid ON OrderWebinars(OrderDetailID);
```

OrderCourses

Indeksuje po kolumnie `OrderDetailID`.

```
CREATE NONCLUSTERED INDEX idx_ordercourses_orderdetailid ON OrderCourses(OrderDetailID);
```

Studies

Indeksuje po kolumnie `StudiesName`.

```
CREATE NONCLUSTERED INDEX idx_studies_name ON Studies(StudiesName);
```

Syllabuses

Indeksuje po kolumnie `StudiesID`.

```
CREATE NONCLUSTERED INDEX idx_syllabuses_studiesid ON Syllabuses(StudiesID);
```

Subjects

Indeksuje po kolumnie `CoordinatorID`.

```
CREATE NONCLUSTERED INDEX idx_subjects_coordinatorid ON Subjects(CoordinatorID);
```

StudiesLessonMeetingTypes

Indeksuje po kolumnie `MeetingTypeName`.

```
CREATE NONCLUSTERED INDEX idx_studieslessonmeetingtypes_name ON StudiesLessonMeetingTypes(MeetingTypeName);
```

StudiesSessions

Indeksuje po kolumnie `StudiesID`.

```
CREATE NONCLUSTERED INDEX idx_studiessessions_studiesid ON StudiesSessions(StudiesID);
```

StudentLessons

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studentlessons_studentid ON StudentLessons(StudentID);
```

StudiesLessonsPassed

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studieslessonspassed_studentid ON StudiesLessonsPassed(StudentID);
```

StudentStudiesGrades

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studentstudiesgrades_studentid ON StudentStudiesGrades(StudentID);
```

StudentBoughtStudies

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studentboughtstudies_studentid ON StudentBoughtStudies(StudentID);
```

StudentBoughtSessions

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_studentboughtsessions_studentid ON StudentBoughtSessions(StudentID);
```

InternshipMeetingTypes

Indeksuje po kolumnie `InternshipMeetingTypeName`.

```
CREATE NONCLUSTERED INDEX idx_internshipmeetingtypes_name ON
InternshipMeetingTypes(InternshipMeetingTypeName);
```

Internships

Indeksuje po kolumnie `StudiesID`.

```
CREATE NONCLUSTERED INDEX idx_internships_studiesid ON Internships(StudiesID);
```

InternshipMeetingsPassed

Indeksuje po kolumnie `StudentID`.

```
CREATE NONCLUSTERED INDEX idx_internshipmeetingspassed_studentid ON InternshipMeetingsPassed(StudentID);
```

OrderStudies

Indeksuje po kolumnie `OrderDetailID`.

```
CREATE NONCLUSTERED INDEX idx_orderstudies_orderdetailid ON OrderStudies(OrderDetailID);
```

OrderSessions

Indeksuje po kolumnie `OrderDetailID`.

```
CREATE NONCLUSTERED INDEX idx_ordersessions_orderdetailid ON OrderSessions(OrderDetailID);
```

RODOSigns

Indeksuje po kolumnie `UserID`.

```
CREATE NONCLUSTERED INDEX idx_rodosigns_userid ON RODOSigns(UserID);
```