AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

# Projekt bazy danych

Podstawy Baz Danych 2024/2025

Dawid Szłapa, Igor Piesik, Adam Głowacki

# Spis treści

# 0. Lokalizacja bazy

- dbmanage.lab.ii.agh.edu.pl
  - u_aglowack
    - **dbo**

# 1. Wprowadzenie

Niniejszy dokument przedstawia szczegółową dokumentację projektu bazy danych, mającego na celu zaprojektowanie i wdrożenie kompleksowego systemu bazodanowego dla firmy zajmującej się organizowaniem kursów oraz szkoleń. Projekt uwzględnia różnorodne aspekty związane z zarządzaniem procesami edukacyjnymi, w tym obsługę hybrydowego modelu świadczenia usług oraz specyficzne potrzeby związane z różnymi formami kształcenia, takimi jak webinary, kursy online czy studia stacjonarne. System został zaprojektowany z myślą o łatwej integracji z zewnętrznymi systemami płatności oraz możliwością generowania raportów wspierających analizy i decyzje operacyjne. Implementacja rozwiązania została zrealizowana w środowisku MS SQL Server.

Dokument zawiera szczegółowy opis elementów składowych bazy danych zawartych w spisie treści

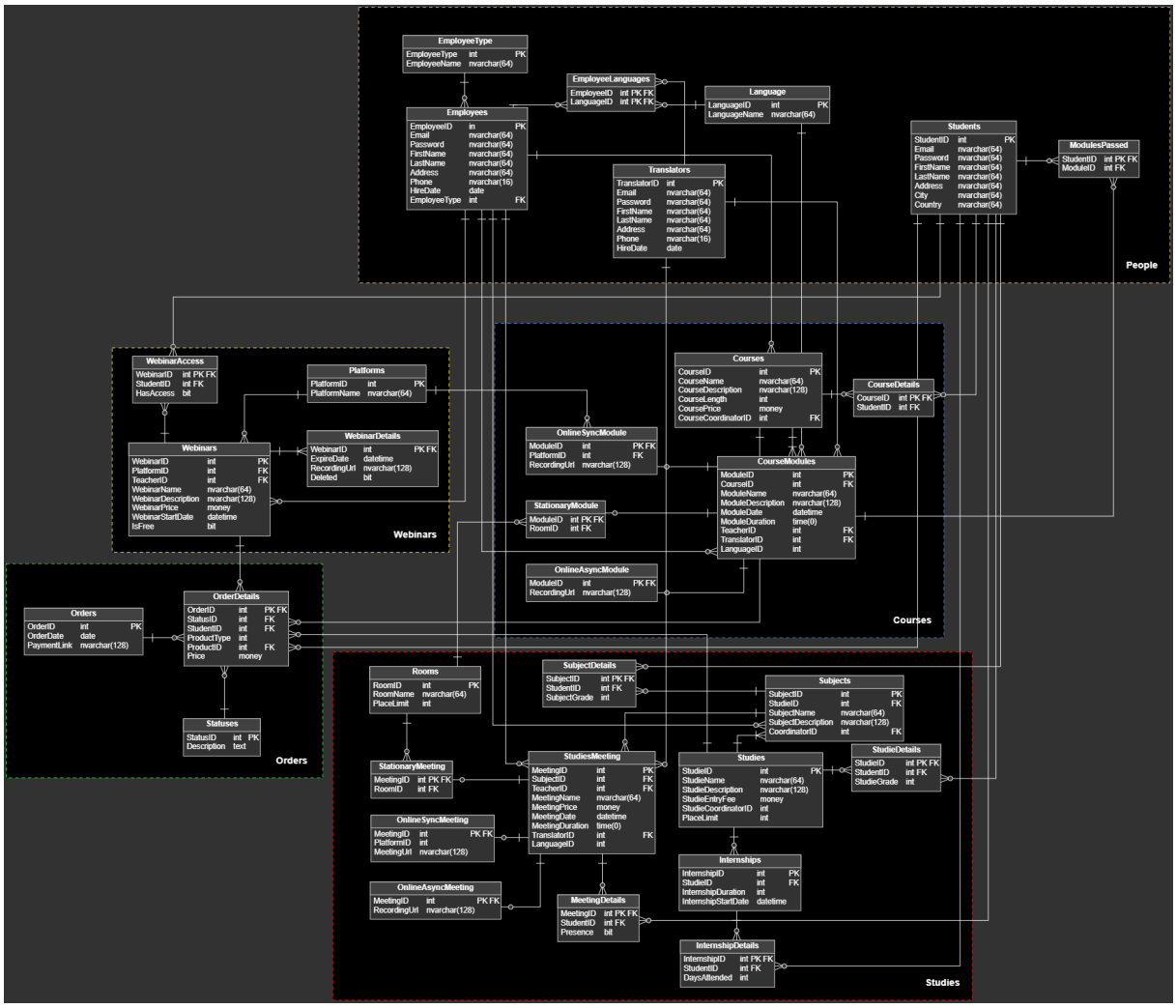
Projekt został zrealizowany przez studentów kierunku Informatyka na Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie w ramach przedmiotu Podstawy Baz Danych, autorami projektu są:

**- Dawid Szłapa**

**- Igor Piesik**

**- Adam Głowacki**

# 2. Schemat bazy danych



# 3. Opis tabel

## 3.1 Tabela CourseDetails

- Zawiera przynależność studentów do kursów.

```
CREATE TABLE CourseDetails (
   CourseID int  NOT NULL,
   StudentID int  NOT NULL,
   CONSTRAINT CourseDetails_pk PRIMARY KEY (CourseID)
);
```

## 3.2 Tabela CourseModules

- Zawiera szczegóły dotyczące modułów kursów, w tym przydzielonych prowadzących(TeacherID) oraz ewentualnego tłumacza (TranslatorID).

```
CREATE TABLE CourseModules (
   ModuleID int  NOT NULL,
   CourseID int  NOT NULL,
   ModuleName nvarchar(64)  NOT NULL CHECK (Len(ModuleName) > 0),
   ModuleDescription nvarchar(128)  NOT NULL CHECK (Len(ModuleDescription) > 0),
   ModuleDate datetime  NOT NULL CHECK (ModuleDate >= '2020-01-01' AND ModuleDate <=
'2300-01-01'),
   ModuleDuration time(0)  NOT NULL CHECK (ModuleDuration BETWEEN '00:00:00' AND
'24:00:00'),
   TeacherID int  NOT NULL,
   TranslatorID int  NULL,
   LanguageID int  NOT NULL,
   CONSTRAINT CourseModules_pk PRIMARY KEY (ModuleID)
);
```

## 3.3 Tabela Courses

- Zawiera informacje o kursach, w tym nazwę, opis, długość, cenę oraz koordynatora.

```
CREATE TABLE Courses (
   CourseID int  NOT NULL,
   CourseName nvarchar(64)  NOT NULL CHECK (Len(CourseName) > 0),
   CourseDescription nvarchar(128)  NOT NULL CHECK (Len(CourseDescription) > 0),
   CourseLength int  NOT NULL CHECK (CourseLength > 0),
   CoursePrice money  NOT NULL CHECK (CoursePrice >= 0),
   CourseCoordinatorID int  NOT NULL,
   CONSTRAINT Courses_pk PRIMARY KEY (CourseID)
);
```

## 3.4 Tabela EmployeeLanguages

- Określa języki, w których poszczególni pracownicy mają kompetencje.

```
CREATE TABLE EmployeeLanguages (
   EmployeeID int  NOT NULL CHECK (EmployeeID > 0),
   LanguageID int  NOT NULL CHECK (LanguageID > 0),
   CONSTRAINT EmployeeLanguages_pk PRIMARY KEY (EmployeeID,LanguageID)
);
```

## 3.5 Tabela Employees

- Zawiera dane o pracownikach, w tym dane kontaktowe, datę zatrudnienia oraz typ pracownika (nauczyciel lub koordynator).

```
CREATE TABLE Employees (
   EmployeeID in  NOT NULL CHECK (EmployeeID > 0),
   Email nvarchar(64)  NOT NULL CHECK (email LIKE '%_@__%.__%'),
   Password nvarchar(64)  NOT NULL,
   FirstName nvarchar(64)  NOT NULL,
   LastName nvarchar(64)  NOT NULL,
   Address nvarchar(64)  NOT NULL,
   Phone nvarchar(16)  NOT NULL,
   HireDate date  NOT NULL CHECK (HireDate <= GETDATE()),
   EmployeeType int  NOT NULL CHECK (EmployeeType BETWEEN 1 AND 2),
   CONSTRAINT Employees_pk PRIMARY KEY (EmployeeID)
);
```

## 3.6 Tabela EmployeeType

- Zawiera informację czy pracownik jest nauczycielem czy koordynatorem.

```
CREATE TABLE EmployeeType (
   EmployeeType int  NOT NULL CHECK (EmployeeType BETWEEN 1 AND 2),
   EmployeeName nvarchar(64)  NOT NULL CHECK (EmployeeName IN ('Teacher',
'Coordinator')),
   CONSTRAINT EmployeeType_pk PRIMARY KEY (EmployeeType)
);
```

## 3.7 Tabela InternshipDetails

- Zawiera dane o obecności studenta podczas praktyk.

```
CREATE TABLE InternshipDetails (
   InternshipID int  NOT NULL CHECK (InternshipID > 0),
   StudentID int  NOT NULL CHECK (StudentID > 0),
   DaysAttended int  NOT NULL CHECK (DaysAttended > 0),
   CONSTRAINT InternshipDetails_pk PRIMARY KEY (InternshipID)
);
```

## 3.8 Tabela Internships

- Zawiera podstawowe informacje o praktykach.

```
CREATE TABLE Internships (
   InternshipID int  NOT NULL CHECK (InternshipID > 0),
   StudieID int  NOT NULL CHECK (StudieID > 0),
   InternshipDuration int  NOT NULL CHECK (InternshipDuration > 0),
   InternshipStartDate datetime  NOT NULL CHECK (InternshipStartDate >= '2020-01-01'
AND InternshipStartDate <= '2100-01-01'),
   CONSTRAINT Internships_pk PRIMARY KEY (InternshipID)
);
```

## 3.9 Tabela Language

- Lista dostępnych języków w systemie.

```
CREATE TABLE Language (
   LanguageID int  NOT NULL CHECK (LanguageID > 0),
   LanguageName nvarchar(64)  NOT NULL,
   CONSTRAINT Language_pk PRIMARY KEY (LanguageID)
);
```

## 3.10 Tabela MeetingDetails

- Zawiera informacje o obecności studenta podczas zajęć na studiach.

```
CREATE TABLE MeetingDetails (
   MeetingID int  NOT NULL CHECK (MeetingID > 0),
   StudentID int  NOT NULL CHECK (StudentID > 0),
   Presence bit  NOT NULL CHECK (Presence = 0 OR Presence = 1),
   CONSTRAINT MeetingDetails_pk PRIMARY KEY (MeetingID)
);
```

## 3.11 Tabela ModulesPassed

- Zawiera zdane przez studenta moduły.

```
CREATE TABLE ModulesPassed (
   StudentID int  NOT NULL CHECK (StudentID > 0),
   ModuleID int  NOT NULL CHECK (ModuleID > 0),
   CONSTRAINT ModulesPassed_pk PRIMARY KEY (StudentID)
);
```

## 3.12 Tabela OnlineAsyncMeeting

- Zawiera link do nagrania spotkania online na studiach w trybie asynchronicznym.

```
CREATE TABLE OnlineAsyncMeeting (
   MeetingID int  NOT NULL CHECK (MeetingID > 0),
   RecordingUrl nvarchar(128)  NOT NULL CHECK (LEN(RecordingUrl) > 0 AND
LEN(RecordingUrl) <= 128),
   CONSTRAINT OnlineAsyncMeeting_pk PRIMARY KEY (MeetingID)
);
```

## 3.13 Tabela OnlineAsyncModule

- Zawiera link do nagrania modułu online w trybie asynchronicznym.

```
CREATE TABLE OnlineAsyncModule (
   ModuleID int  NOT NULL,
   RecordingUrl nvarchar(128)  NOT NULL CHECK (Len(RecordingUrl) > 0),
   CONSTRAINT OnlineAsyncModule_pk PRIMARY KEY (ModuleID)
);
```

## 3.14 Tabela OnlineSyncMeeting

- Zawiera platformę oraz link do spotkania na studiach w trybie synchronicznym.

```
CREATE TABLE OnlineSyncMeeting (
   MeetingID int  NOT NULL CHECK (MeetingID > 0),
   PlatformID int  NOT NULL CHECK (PlatformID > 0),
   MeetingUrl nvarchar(128)  NOT NULL CHECK (LEN(MeetingUrl) > 0 AND LEN(MeetingUrl)
<= 128),
   CONSTRAINT OnlineSyncMeeting_pk PRIMARY KEY (MeetingID)
);
```

## 3.15 Tabela OnlineSyncModule

- Zawiera platformę link do spotkania oraz nagrania modułu online w trybie synchronicznym.

```
CREATE TABLE OnlineSyncModule (
   ModuleID int  NOT NULL,
   PlatformID int  NOT NULL,
   RecordingUrl nvarchar(128)  NOT NULL CHECK (Len(RecordingUrl) > 0),
   CONSTRAINT OnlineSyncModule_pk PRIMARY KEY (ModuleID)
);
```

## 3.16 Tabela OrderDetails

- Zawiera szczegóły danego zamóienia.

```
CREATE TABLE OrderDetails (
   OrderID int  NOT NULL,
   StatusID int  NOT NULL,
   StudentID int  NOT NULL,
   ProductType int  NOT NULL COMMENT '''''studies''''', ''''webinar''''',
''''course''''', ''''' CHECK (ProductType IN BETWEEN 0 AND 2),
   ProductID int  NOT NULL,
   Price money  NOT NULL CHECK (Price > 0),
   CONSTRAINT OrderDetails_pk PRIMARY KEY (OrderID)
);
```

## 3.17 Tabela Orders

- Zawiera listę wszystkich zamówień.

```
CREATE TABLE Orders (
   OrderID int  NOT NULL,
   OrderDate date  NOT NULL,
   PaymentLink nvarchar(128)  NOT NULL CHECK (Len(PaymentLink) > 0),
   CONSTRAINT Orders_pk PRIMARY KEY (OrderID)
);
```

## 3.18 Tabela Platforms

- Zawiera listę platform do przeprowadzania spotkań.

```
CREATE TABLE Platforms (
   PlatformID int  NOT NULL CHECK (PlatformID > 0),
   PlatformName nvarchar(64)  NOT NULL CHECK (LEN(PlatformName) > 0 AND
LEN(PlatformName) <= 64),
   CONSTRAINT Platforms_pk PRIMARY KEY (PlatformID)
);
```

## 3.19 Tabela Rooms

- Przechowuje pomieszczenia, w których mogą odbywać się zajęcia.

```
CREATE TABLE Rooms (
   RoomID int  NOT NULL CHECK (RoomID > 0),
   RoomName nvarchar(64)  NOT NULL CHECK (LEN(RoomName) > 0 AND LEN(RoomName) <= 64),
   PlaceLimit int  NOT NULL CHECK (PlaceLimit > 0),
   CONSTRAINT Rooms_pk PRIMARY KEY (RoomID)
);
```

## 3.20 Tabela StationaryMeeting

- Zawiera informacje w jakich pomieszczeniach odbywają się zajęcia stacjonarne na studiach.

```
CREATE TABLE StationaryMeeting (
   MeetingID int  NOT NULL CHECK (MeetingID > 0),
   RoomID int  NOT NULL CHECK (RoomID > 0),
   CONSTRAINT StationaryMeeting_pk PRIMARY KEY (MeetingID)
);
```

## 3.21 Tabela StationaryModule

- Zawiera informacje w jakich pomieszczeniach odbywają się moduły stacjonarne.

```
CREATE TABLE StationaryModule (
   ModuleID int  NOT NULL,
   RoomID int  NOT NULL,
   CONSTRAINT StationaryModule_pk PRIMARY KEY (ModuleID)
);
```

## 3.22 Tabela Statuses

- Zawiera informacje o statusie zamówienia.

```
CREATE TABLE Statuses (
   StatusID int  NOT NULL,
   Description text  NOT NULL CHECK (Len(Description) > 0),
   CONSTRAINT Statuses_pk PRIMARY KEY (StatusID)
);
```

## 3.23 Tabela Students

- Zawiera dane osobowe studentów.

```
CREATE TABLE Students (
   StudentID int  NOT NULL CHECK (StudentID > 0),
   Email nvarchar(64)  NOT NULL CHECK (email LIKE '%_@__%.__%'),
   Password nvarchar(64)  NOT NULL,
   FirstName nvarchar(64)  NOT NULL,
   LastName nvarchar(64)  NOT NULL,
   Address nvarchar(64)  NOT NULL,
   City nvarchar(64)  NOT NULL,
   Country nvarchar(64)  NOT NULL,
   CONSTRAINT Students_pk PRIMARY KEY (StudentID)
);
```

## 3.24 Tabela StudieDetails

- Zawiera oceny studentów za całość studiów.

```
CREATE TABLE StudieDetails (
   StudieID int  NOT NULL CHECK (StudieID > 0),
   StudentID int  NOT NULL CHECK (StudentID > 0),
   StudieGrade int  NOT NULL CHECK (StudieGrade > 0),
   CONSTRAINT StudieDetails_pk PRIMARY KEY (StudieID)
);
```

## 3.25 Tabela Studies

- Zawiera informacje o studiach, takie jak ich koszt czy ilość miejsc.

```
CREATE TABLE Studies (
   StudieID int  NOT NULL CHECK (StudieID > 0),
   StudieName nvarchar(64)  NOT NULL CHECK (LEN(StudieName) > 0 AND LEN(StudieName)
<= 64),
   StudieDescription nvarchar(128)  NOT NULL CHECK (LEN(StudieDescription) > 0 AND
LEN(StudieDescription) <= 128),
   StudieEntryFee money  NOT NULL CHECK (StudieEntryFee > 0),
   StudieCoordinatorID int  NOT NULL CHECK (StudieCoordinatorID > 0),
   PlaceLimit int  NOT NULL CHECK (PlaceLimit > 0),
   CONSTRAINT Studies_pk PRIMARY KEY (StudieID)
);
```

## 3.26 Tabela StudiesMeeting

- Zawiera informacje o pojedynczych spotkaniach na studiach.

```
CREATE TABLE StudiesMeeting (
   MeetingID int  NOT NULL CHECK (MeetingID > 0),
   SubjectID int  NOT NULL CHECK (SubjectID > 0),
   TeacherID int  NOT NULL CHECK (TeacherID > 0),
   MeetingName nvarchar(64)  NOT NULL CHECK (LEN(MeetingName) > 0 AND
LEN(MeetingName) <= 64),
   MeetingPrice money  NOT NULL CHECK (MeetingPrice > 0),
   MeetingDate datetime  NOT NULL CHECK (MeetingDate >= '2020-01-01' AND MeetingDate
<= '2300-01-01'),
   MeetingDuration time(0)  NOT NULL CHECK (MeetingDuration BETWEEN '00:00:00' AND
'24:00:00'),
   TranslatorID int  NOT NULL CHECK (TranslatorID > 0),
   LanguageID int  NOT NULL CHECK (LanguageID > 0),
   CONSTRAINT StudiesMeeting_pk PRIMARY KEY (MeetingID)
);
```

## 3.27 Tabela SubjectDetails

- Zawiera oceny studentów za poszczególne przedmioty.

```
CREATE TABLE SubjectDetails (
   SubjectID int  NOT NULL CHECK (SubjectID > 0),
   StudentID int  NOT NULL CHECK (StudentID > 0),
   SubjectGrade int  NOT NULL CHECK (SubjectGrade > 0 AND SubjectGrade <= 5),
   CONSTRAINT SubjectDetails_pk PRIMARY KEY (SubjectID)
);
```

## 3.28 Tabela Subjects

- Zawiera informacje o przedmiotach realizowanych podczas studiów.

```
CREATE TABLE Subjects (
   SubjectID int  NOT NULL CHECK (SubjectID > 0),
   StudieID int  NOT NULL CHECK (StudieID > 0),
   SubjectName nvarchar(64)  NOT NULL CHECK (LEN(SubjectName) > 0 AND
LEN(SubjectName) <= 64),
   SubjectDescription nvarchar(128)  NOT NULL CHECK (LEN(SubjectDescription) > 0 AND
LEN(SubjectDescription) <= 128),
   CoordinatorID int  NOT NULL CHECK (CoordinatorID > 0),
   CONSTRAINT Subjects_pk PRIMARY KEY (SubjectID)
);
```

## 3.29 Tabela Translators

- Zawiera dane osobowe tłumaczów.

```
CREATE TABLE Translators (
   TranslatorID int  NOT NULL CHECK (TranslatorID > 0),
   Email nvarchar(64)  NOT NULL CHECK (email LIKE '%_@__%.__%'),
   Password nvarchar(64)  NOT NULL,
   FirstName nvarchar(64)  NOT NULL,
   LastName nvarchar(64)  NOT NULL,
   Address nvarchar(64)  NOT NULL,
   Phone nvarchar(16)  NOT NULL,
   HireDate date  NOT NULL CHECK (HireDate <= GETDATE()),
   CONSTRAINT Translators_pk PRIMARY KEY (TranslatorID)
);
```

## 3.30 Tabela WebinarAccess

- Zawiera informacje o dostępie studentów do webinarów.

```
CREATE TABLE WebinarAccess (
   WebinarID int  NOT NULL CHECK (WebinarID > 0),
   StudentID int  NOT NULL CHECK (StudentID > 0),
   HasAccess bit  NOT NULL,
   CONSTRAINT WebinarAccess_pk PRIMARY KEY (WebinarID)
);
```

## 3.31 Tabela WebinarDetails

- Zawiera szczegółowe informacje webinarów, takie jak link i kiedy zostaną usunięte.

```
CREATE TABLE WebinarDetails (
   WebinarID int  NOT NULL CHECK (WebinarID > 0),
   ExpireDate datetime  NOT NULL CHECK (ExpireDate >= '2020-01-01' AND ExpireDate <=
'2100-01-01'),
   RecordingUrl nvarchar(128)  NOT NULL CHECK (LEN(PlatformName) > 0 AND
LEN(PlatformName) <= 128),
   Deleted bit  NOT NULL,
   CONSTRAINT WebinarDetails_pk PRIMARY KEY (WebinarID)
);
```

## 3.32 Tabela Webinars

- Zawiera podstawowe informacje o webinarach, takie jak przypisanie nauczyciele i opis.

```
CREATE TABLE Webinars (
   WebinarID int  NOT NULL CHECK (WebinarID > 0),
   PlatformID int  NOT NULL CHECK (PlatformID > 0),
   TeacherID int  NOT NULL CHECK (LEN(WebinarName) > 0 AND LEN(WebinarName) <= 64),
   WebinarName nvarchar(64)  NOT NULL CHECK (LEN(WebinarName) > 0 AND
LEN(WebinarName) <= 64),
   WebinarDescription nvarchar(128)  NOT NULL CHECK (LEN(WebinarDescription) > 0 AND
LEN(WebinarDescription) <= 128),
   WebinarPrice money  NOT NULL DEFAULT WebinarPrice >= 0,
   WebinarStartDate datetime  NOT NULL DEFAULT WebinarStartDate >= '2020-01-01' AND
WebinarStartDate <= '2100-01-01',
   IsFree bit  NOT NULL CHECK (IsFree = 0 OR isFree = 1),
   CHECK (),
   CONSTRAINT Webinars_pk PRIMARY KEY (WebinarID)
);
```

# 4. Widoki

## 4.1 ALL_STUDIES_TIMETABLE

- Wyświetla informacje na temat wszystkich studiów.

```
CREATE VIEW ALL_STUDIES_TIMETABLE AS
SELECT s.StudieID, sm.MeetingID, sm.MeetingName, sm.MeetingDate, sm.MeetingDuration
FROM dbo.Studies AS s
INNER JOIN dbo.Subjects AS su
ON s.StudieID = su.StudieID
INNER JOIN dbo.StudiesMeeting AS sm
ON su.SubjectID = sm.SubjectID
```

## 4.2 ALL_COURSES_TIMETABLE

- Wyświetla informacje na temat wszystkich kursów.

```
CREATE VIEW ALL_COURSES_TIMETABLE AS
SELECT c.CourseID, cm.ModuleID, cm.ModuleName, cm.ModuleDate, cm.ModuleDuration
FROM Courses AS c
INNER JOIN CourseModules AS cm
ON c.CourseID = cm.CourseID
```

## 4.3 ALL_WEBINARS_TIMETABLE

- Wyświetla informacje na temat wszystkich webinarów.

```
CREATE VIEW ALL_WEBINARS_TIMETABLE AS
SELECT w.WebinarID, w.WebinarName, w.WebinarStartDate
FROM Webinars AS w
```

# 5. Funkcje

## 5.1 GetMaxStudyCapacity

- Sprawdza ilość miejsc na dane studia.

```
CREATE FUNCTION GetMaxStudyCapacity(@StudiesID int)
RETURNS int
AS
BEGIN
    DECLARE @MaxCapacity int;


    SELECT @MaxCapacity = MIN(Rooms.PlaceLimit)
    FROM StationaryMeeting
    JOIN StudiesMeeting ON StationaryMeeting.MeetingID = StudiesMeeting.MeetingID
    JOIN Subjects ON StudiesMeeting.SubjectID = Subjects.SubjectID
    JOIN Rooms ON StationaryMeeting.RoomID = Rooms.RoomID
    WHERE Subjects.SubjectID = @StudiesID;

    RETURN @MaxCapacity
END
```

## 5.2 HowManyStudyVacancies

- Sprawdza ilość wolnych miejsc na dany kierunek studiów.

```
CREATE FUNCTION HowManyStudyVacancies(@StudiesID INT)
RETURNS INT
AS
BEGIN
    DECLARE @MaximumCapacity INT = dbo.GetMaxStudyCapacity(@StudiesID);
    IF @MaximumCapacity IS NULL
    BEGIN
        RETURN NULL;
    END
    DECLARE @CurrentCapacity INT = (
        SELECT COUNT(*)
        FROM Students AS s
        JOIN StudieDetails AS sd
          ON s.StudentID = sd.StudentID
        WHERE sd.StudieID = @StudiesID
    );
    RETURN @MaximumCapacity - @CurrentCapacity;
END
```

## 5.3 GetWebinarAttendance

- Sprawdza liczbę osób zapisanych na dany webinar.

```
CREATE FUNCTION [dbo].[GetWebinarAttendance]
(
    @WebinarID INT
)
RETURNS INT
AS
BEGIN
    -- Zmienna do przechowywania wyniku
    DECLARE @AttendanceCount INT;

    -- Obliczenie liczby osób zapisanych na webinar
    SELECT @AttendanceCount = COUNT(*)
    FROM WebinarAccess
    WHERE WebinarID = @WebinarID
      AND HasAccess = 1;

    -- Zwrócenie wyniku
    RETURN @AttendanceCount;
END;
GO
```

## 5.4 CheckWebinarAccess

- Sprawdza datę wygaśnięcia dostępu do webinaru płatnego.

```
CREATE FUNCTION CheckWebinarAccess(@WebinarID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @IsExpired BIT;

    -- Sprawdzenie, czy webinar jest płatny i czy data wygaśnięcia nie minęła
    SELECT @IsExpired =
        CASE
            WHEN w.IsFree = 0 AND wd.ExpireDate < GETDATE() THEN 1
            ELSE 0
        END
    FROM Webinars w
    INNER JOIN WebinarDetails wd ON w.WebinarID = wd.WebinarID
    WHERE w.WebinarID = @WebinarID;

    RETURN @IsExpired;
END;
```

## 5.5 GetStudentCount

- Sprawdza liczbę zapisanych osób na dany kierunek studiów.

```
CREATE FUNCTION GetStudentCount(@StudieID INT)
RETURNS INT
AS
BEGIN
    DECLARE @StudentCount INT;

    -- Zliczanie liczby studentów zapisanych na dany kierunek studiów
    SELECT @StudentCount = COUNT(*)
    FROM StudieDetails
    WHERE StudieID = @StudieID;

    RETURN @StudentCount;
END;
```

# 5.6 StudentPassedCourse

- Sprawdza czy uczestnik zdał kurs (ma przynajmniej 80% zdanych modułów).

```
CREATE FUNCTION StudentPassedCourse(@StudentID int, @CourseID int)
RETURNS BIT
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM Students WHERE StudentID = @StudentID)
    BEGIN
        RETURN 0;  --nie znaleziono studenta
    END;

    IF NOT EXISTS (SELECT * FROM Courses WHERE CourseID = @CourseID)
    BEGIN
        RETURN 0;  --nie znaleziono kursu
    END;

    IF NOT EXISTS (SELECT * FROM Courses AS c
        JOIN CourseDetails AS cd ON cd.CourseID = c.CourseID
        WHERE StudentID = @StudentID AND cd.CourseID = @CourseID)
    BEGIN
        RETURN 0; --student nie przypisany na ten kurs
    END


    DECLARE @ModulesPassed FLOAT = (
        SELECT COUNT(*)
        FROM CourseModules cm
        JOIN ModulesPassed mp ON mp.ModuleID = cm.ModuleID
        WHERE mp.StudentID = @StudentID AND cm.CourseID = @CourseID
    );

    DECLARE @TotalModules FLOAT = (
        SELECT COUNT(*)
        FROM CourseModules cm
        WHERE cm.CourseID = @CourseID
    );

    IF @TotalModules > 0 AND @ModulesPassed / @TotalModules >= 0.8
    BEGIN
        RETURN 1; -- Student zaliczył kurs
    END;

    RETURN 0;

END;
```

# 5.7 GetSubjectAttendanceForStudent

- Zlicza frekwencję studenta 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
        RETURN 0.0;  --nie znaleziono studenta
    END

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

    IF NOT EXISTS (SELECT 1 FROM Subjects AS s
        JOIN SubjectDetails AS sd ON sd.SubjectID = s.SubjectID
        WHERE StudentID = @StudentID AND sd.SubjectID = @SubjectID)
    BEGIN
        RETURN 0.0; --student nie przypisany na ten przedmiot
    END


    DECLARE @AttendanceCount FLOAT = 0.0;
    DECLARE @MeetingsCount FLOAT = 0.0;
    SELECT @AttendanceCount = COUNT(*)
    FROM StudiesMeeting sm
    JOIN MeetingDetails md ON md.MeetingID = sm.MeetingID
    WHERE StudentID = @StudentID AND Presence = 1 AND SubjectID = @SubjectID AND
MeetingDate < GETDATE();

    SELECT @MeetingsCount = COUNT(*)
    FROM StudiesMeeting sm
    JOIN MeetingDetails md ON md.MeetingID = sm.MeetingID
    WHERE StudentID = @StudentID AND SubjectID = @SubjectID AND MeetingDate <
GETDATE();

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

## 5.8 HasCompletedYearlyInternships

- Sprawdza czy student odbył praktyki 2 razy w roku i czy miał na nich 100% frekwencji.

```
CREATE FUNCTION HasCompletedYearlyInternships(@StudentID INT, @Year INT)
RETURNS BIT
AS
BEGIN
    DECLARE @Result BIT;
    IF (SELECT COUNT(*)
        FROM InternshipDetails id
        JOIN Internships i ON i.InternshipID = id.InternshipID
        WHERE id.StudentID = @StudentID
        AND id.DaysAttended = 14
        AND YEAR(i.InternshipStartDate) = @Year) = 2
    BEGIN
        SET @Result = 1;
    END
    ELSE
    BEGIN
        SET @Result = 0;
    END

    RETURN @Result
END;
```

## 5.9 GetStudentAverageGrade

- Liczy średnią ocen studenta.

```
CREATE FUNCTION GetStudentAverageGrade(@StudentID INT, @StudieID INT)
RETURNS DECIMAL(5,2)
AS
BEGIN
    DECLARE @AverageGrade DECIMAL(5,2);

    SELECT @AverageGrade = AVG(sd.SubjectGrade)
    FROM Subjects s
    JOIN SubjectDetails sd ON sd.SubjectID = s.SubjectID
    WHERE sd.StudentID = @StudentID AND s.StudieID = @StudieID

    IF @AverageGrade IS NULL
    BEGIN
        SET @AverageGrade = 0;  -- Możesz ustawić wartość domyślną, np. 0
    END

    RETURN @AverageGrade;
END;
```

## 5.10 CheckTranslatorLanguage

- Sprawdza czy dany tłumacz zna dany język.

```
CREATE FUNCTION CheckTranslatorLanguage(@TranslatorID int, @LanguageID int)
RETURNS bit
AS
BEGIN
    DECLARE @Result bit;

    IF EXISTS(
        SELECT 1
        FROM EmployeeLanguages
        WHERE EmployeeID = @TranslatorID AND LanguageID = @LanguageID
    )
    BEGIN
        SET @Result = 1;  --tlumacz zna jezyk
    END
    ELSE
    BEGIN
        SET @Result = 0;  --tlumacz nie zna jezyka
    END

    RETURN @Result
END;
```

## 5.11 GetStudySchedule

- Generuje harmonogram spotkania na studiach używając widoku ALL_STUDIES_TIMETABLE.

```
CREATE FUNCTION GetStudySchedule(@StudiesID int)
RETURNS TABLE
AS
RETURN
    (SELECT MeetingID, MeetingName, MeetingDate, MeetingDuration
    FROM dbo.ALL_STUDIES_TIMETABLE
    WHERE StudieID = @StudiesID)
```

## 5.12 GetCourseSchedule

- Generuje harmonogram kursu używając widoku ALL_COURSES_TIMETABLE.

```
CREATE FUNCTION GetCourseSchedule(@CourseID int)
RETURNS TABLE
AS
RETURN
    (SELECT ModuleID, ModuleName, ModuleDate, ModuleDuration
    FROM dbo.ALL_COURSES_TIMETABLE
    WHERE CourseID = @CourseID)
```

# 5.13 GetStudentsSchedule

- Generuje harmonogram zajęć dla studenta.

```
CREATE FUNCTION GetStudentsSchedule(@StudentID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        sm.MeetingID AS EventID,
        'StudiesMeeting' AS EventType,
        sm.MeetingDate AS EventDate,
        sm.MeetingDuration AS Duration
    FROM
        StudiesMeeting sm
  JOIN MeetingDetails md ON md.MeetingID = sm.MeetingID
    WHERE
        md.StudentID = @StudentID

    UNION ALL

    SELECT
        w.WebinarID AS EventID,
        'Webinar' AS EventType,
        w.WebinarStartDate AS EventDate,
        NULL AS Duration
    FROM
        Webinars w
  JOIN WebinarAccess wa ON wa.WebinarID = w.WebinarID
    WHERE
        wa.StudentID = @StudentID

    UNION ALL

    SELECT
        c.CourseID AS EventID,
        'Course' AS EventType,
        c.ModuleDate AS EventDate,
        c.ModuleDuration AS Duration
    FROM
        CourseModules c
  JOIN CourseDetails cd ON cd.CourseID = c.CourseID
    WHERE
        cd.StudentID = @StudentID
)
```

# 5.14 GetTeachersSchedule

- Generuje harmonogram zajęć dla pracownika.

```
CREATE FUNCTION GetTeachersSchedule(@TeacherID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        sm.MeetingID AS EventID,
        'StudiesMeeting' AS EventType,
        sm.MeetingDate AS EventDate,
        sm.MeetingDuration AS Duration
    FROM
        StudiesMeeting sm
    WHERE
        sm.TeacherID = @TeacherID

    UNION ALL

    SELECT
        w.WebinarID AS EventID,
        'Webinar' AS EventType,
        w.WebinarStartDate AS EventDate,
        NULL AS Duration
    FROM
        Webinars w
    WHERE
        w.TeacherID = @TeacherID

    UNION ALL

    SELECT
        c.CourseID AS EventID,
        'Course' AS EventType,
        c.ModuleDate AS EventDate,
        c.ModuleDuration AS Duration
    FROM
        CourseModules c
    WHERE
        c.TeacherID = @TeacherID
)
```

## 5.15 GetTranslatorsSchedule

- Generuje harmonogram zajęć dla tłumacza.

```
CREATE FUNCTION GetTranslatorsSchedule(@TranslatorID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        m.MeetingID,
        m.TranslatorID,
        m.MeetingDate,
        m.MeetingDuration
    FROM
        StudiesMeeting m
    WHERE
        m.TranslatorID = @TranslatorID
);
```

## 5.16 GetRoomSchedule

- Generuje harmonogram danej sali.

```
CREATE FUNCTION dbo.GetRoomSchedule (@RoomID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        sm.MeetingID,
        sm.RoomID,
        stm.MeetingDate,
        stm.MeetingDuration
    FROM
        StationaryMeeting sm
    JOIN
        StudiesMeeting stm ON sm.MeetingID = stm.MeetingID
    WHERE
        sm.RoomID = @RoomID
    ORDER BY
        stm.MeetingDate
);
```

## 5.17 GetPaymentLink

- Pobiera link do płatności.

```
CREATE FUNCTION dbo.GetPaymentLink (@OrderID INT)
RETURNS NVARCHAR(128)
AS
BEGIN
    DECLARE @PaymentLink NVARCHAR(128);

    SELECT @PaymentLink = PaymentLink
    FROM Orders
    WHERE OrderID = @OrderID;

    RETURN @PaymentLink;
END;
```

## 5.18 GetCartDetails

- Wyświetla produkty w koszyku.

```
CREATE FUNCTION dbo.GetCartDetails (@OrderID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        OD.ProductID,
        OD.Price
    FROM
        OrderDetails OD
    INNER JOIN
        Orders O ON OD.OrderID = O.OrderID
    WHERE
        OD.OrderID = @OrderID
);
```

## 5.19 GetCartSummary

- Wyświetla wartość zamówienia w koszyku.

```
CREATE FUNCTION dbo.GetCartSummary (@OrderID INT)
RETURNS MONEY
AS
BEGIN
    DECLARE @TotalPrice MONEY;
    SELECT @TotalPrice = SUM(Price)
    FROM dbo.GetCartDetails(@OrderID);
    RETURN @TotalPrice;
END;
```

# 5.20 GetProductsWithStatus

- Sprawdza status zamówienia.

```
CREATE FUNCTION dbo.GetProductsWithStatus (@OrderID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        OD.ProductID,
        S.Description AS StatusDescription
    FROM
        OrderDetails OD
    INNER JOIN
        Orders O ON OD.OrderID = O.OrderID
    INNER JOIN
        Statuses S ON OD.StatusID = S.StatusID
    WHERE
        OD.OrderID = @OrderID
);
```

# 6. Procedury

## 6.1 AddWebinar

- Dodaje nowy webinar do tabeli Webinars.

```
CREATE PROCEDURE AddWebinar
    @WebinarID INT,
    @PlatformID INT,
    @TeacherID INT,
    @WebinarName NVARCHAR(64),
    @WebinarDescription varchar(128),
    @WebinarPrice DECIMAL(10, 2),
    @WebinarStartDate DATETIME,
    @IsFree BIT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia nauczyciela
        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            RAISERROR('Nie znaleziono nauczyciela.', 16, 1);
        END

        -- Wstawianie nowego webinaru
        SET IDENTITY_INSERT Webinars ON;
        INSERT INTO Webinars (WebinarID, PlatformID, TeacherID, WebinarName,
                              WebinarDescription, WebinarPrice, WebinarStartDate,
                              IsFree)
        VALUES (@WebinarID, @PlatformID, @TeacherID, @WebinarName,
                @WebinarDescription, @WebinarPrice, @WebinarStartDate,
                @IsFree);
        SET IDENTITY_INSERT Webinars OFF;

        -- Komunikat o powodzeniu
        PRINT N'Webinar został pomyślnie dodany.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

## 6.2 DeleteWebinar

- Usuwa webinar z tabeli Webinarso podanym ID.

```
CREATE PROCEDURE DeleteWebinar
    @WebinarID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia webinaru
        IF NOT EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID)
        BEGIN
            RAISERROR(N'Nie znaleziono webinaru do usunięcia.', 16, 1);
        END

        -- Usunięcie webinaru
        DELETE FROM Webinars
        WHERE WebinarID = @WebinarID;

        -- Komunikat o powodzeniu
        PRINT N'Webinar został pomyślnie usunięty.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

# 6.3 UpdateWebinar

- Pozwala edytować webinar z tabeli Webinars o danym ID.

```
CREATE PROCEDURE UpdateWebinar
    @WebinarID INT,
    @PlatformID INT = NULL,
    @TeacherID INT = NULL,
    @WebinarName NVARCHAR(64) = NULL,
    @WebinarDescription VARCHAR(128) = NULL,
    @WebinarPrice DECIMAL(10, 2) = NULL,
    @WebinarStartDate DATETIME = NULL,
    @IsFree BIT = NULL
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia webinaru
        IF NOT EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID)
        BEGIN
            RAISERROR('Nie znaleziono webinaru do edycji.', 16, 1);
        END

        -- Aktualizacja szczegółów webinaru
        UPDATE Webinars
        SET
            PlatformID = ISNULL(@PlatformID, PlatformID),
            TeacherID = ISNULL(@TeacherID, TeacherID),
            WebinarName = ISNULL(@WebinarName, WebinarName),
            WebinarDescription = ISNULL(@WebinarDescription, WebinarDescription),
            WebinarPrice = ISNULL(@WebinarPrice, WebinarPrice),
            WebinarStartDate = ISNULL(@WebinarStartDate, WebinarStartDate),
            IsFree = ISNULL(@IsFree, IsFree)
        WHERE WebinarID = @WebinarID;

        -- Komunikat o powodzeniu
        PRINT N'Szczegóły webinaru zostały pomyślnie zaktualizowane.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

# 6.4 AddCourse

- Dodaje nowy kurs do tabeli Courses.

```
CREATE PROCEDURE [dbo].[AddCourse]
    @CourseID INT,
    @CourseName NVARCHAR(64),
    @CourseDescription NVARCHAR(128),
    @CourseLength INT,
    @CoursePrice DECIMAL(10, 2),
    @CourseCoordinatorID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia koordynatora
        IF NOT EXISTS (
            SELECT 1
            FROM Employees e
            INNER JOIN dbo.EmployeeType et ON e.EmployeeType = et.EmployeeType
            WHERE e.EmployeeID = @CourseCoordinatorID
              AND et.EmployeeName LIKE 'Coordinator'
        )
        BEGIN
            RAISERROR('Koordynator o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Wstawianie nowego kursu
        SET IDENTITY_INSERT Courses ON;
        INSERT INTO Courses (CourseID, CourseName, CourseDescription, CourseLength,
CoursePrice, CourseCoordinatorID)
        VALUES (@CourseID, @CourseName, @CourseDescription, @CourseLength,
@CoursePrice, @CourseCoordinatorID);
        SET IDENTITY_INSERT Courses OFF;

        -- Komunikat o powodzeniu
        PRINT N'Kurs został pomyślnie dodany.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

## 6.5 DeleteCourse

- Usuwa kurs o podanym ID z tabeli Courses.

```
CREATE PROCEDURE [dbo].[DeleteCourse]
    @CourseID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia kursu
        IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
        BEGIN
            RAISERROR('Kurs o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Usuwanie kursu
        DELETE FROM Courses
        WHERE CourseID = @CourseID;

        -- Komunikat o powodzeniu
        PRINT N'Kurs został pomyślnie usunięty.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

# 6.6 UpdateCourse

- Pozwala edytować kurs z tabeli Courses o podanym ID.

```
CREATE PROCEDURE [dbo].[UpdateCourse]
    @CourseID INT,
    @CourseName NVARCHAR(64) = NULL,
    @CourseDescription NVARCHAR(128) = NULL,
    @CourseLength INT = NULL,
    @CoursePrice DECIMAL(10, 2) = NULL,
    @CourseCoordinatorID INT = NULL
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia kursu
        IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
        BEGIN
            RAISERROR('Kurs o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Sprawdzenie istnienia koordynatora (jeśli podano)
        IF @CourseCoordinatorID IS NOT NULL AND NOT EXISTS (
            SELECT 1
            FROM Employees e
            INNER JOIN dbo.EmployeeType et ON e.EmployeeType = et.EmployeeType
            WHERE e.EmployeeID = @CourseCoordinatorID
              AND et.EmployeeName LIKE 'Coordinator'
        )
        BEGIN
            RAISERROR('Koordynator o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Aktualizacja szczegółów kursu
        UPDATE Courses
        SET
            CourseName = ISNULL(@CourseName, CourseName),
            CourseDescription = ISNULL(@CourseDescription, CourseDescription),
            CourseLength = ISNULL(@CourseLength, CourseLength),
            CoursePrice = ISNULL(@CoursePrice, CoursePrice),
            CourseCoordinatorID = ISNULL(@CourseCoordinatorID, CourseCoordinatorID)
        WHERE CourseID = @CourseID;

        -- Komunikat o powodzeniu
        PRINT N'Szczegóły kursu zostały pomyślnie zaktualizowane.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

# 6.7 AddCourseModule

- Dodaje moduł do kursu w tabeli CourseModules.

```
CREATE PROCEDURE [dbo].[AddCourseModule]
    @CourseID INT,
    @ModuleName NVARCHAR(64),
    @ModuleDescription NVARCHAR(128),
    @ModuleDate DATETIME,
    @ModuleDuration NVARCHAR(20),
    @TeacherID INT,
    @TranslatorID INT = NULL,
    @LanguageID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia kursu
        IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
        BEGIN
            RAISERROR('Kurs o podanym ID nie istnieje.', 16, 1);
        END

        -- Sprawdzenie istnienia nauczyciela
        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @TeacherID)
        BEGIN
            RAISERROR('Nauczyciel o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Sprawdzenie istnienia tłumacza (jeśli podano)
        IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM Employees WHERE
EmployeeID = @TranslatorID)
        BEGIN
            RAISERROR(N'Tłumacz o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Sprawdzenie istnienia języka
        IF NOT EXISTS (SELECT 1 FROM Language WHERE LanguageID = @LanguageID)
        BEGIN
            RAISERROR(N'Język o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Wstawianie nowego modułu do tabeli Course Modules
        INSERT INTO CourseModules (
            CourseID, ModuleName, ModuleDescription, ModuleDate,
            ModuleDuration, TeacherID, TranslatorID, LanguageID
        )
        VALUES (
            @CourseID, @ModuleName, @ModuleDescription, @ModuleDate,
            @ModuleDuration, @TeacherID, @TranslatorID, @LanguageID
        );

        -- Komunikat o powodzeniu
        PRINT N'Moduł został pomyślnie dodany do kursu.';
```

```
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

## 6.8 AddStudie

- Dodaje nowe studia do tabeli Studies.

```
CREATE PROCEDURE [dbo].[AddStudie]
    @StudieName NVARCHAR(64),
    @StudieDescription NVARCHAR(128),
    @StudieEntryFee DECIMAL(10, 2),
    @StudieCoordinatorID INT,
    @PlaceLimit INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia koordynatora
        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID =
@StudieCoordinatorID)
        BEGIN
            RAISERROR('Koordynator o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Wstawianie nowego rekordu do tabeli Studies
        INSERT INTO Studies (StudieName, StudieDescription, StudieEntryFee,
StudieCoordinatorID, PlaceLimit)
        VALUES (@StudieName, @StudieDescription, @StudieEntryFee,
@StudieCoordinatorID, @PlaceLimit);

        PRINT N'Studia zostały pomyślnie dodane.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

## 6.9 UpdateStudie

- Pozwala edytować studia z tabeli Studies o podanym ID.

```
CREATE PROCEDURE [dbo].[UpdateStudie]
    @StudieID INT,
    @StudieName NVARCHAR(64) = NULL,
    @StudieDescription NVARCHAR(128) = NULL,
    @StudieEntryFee DECIMAL(10, 2) = NULL,
    @StudieCoordinatorID INT = NULL,
    @PlaceLimit INT = NULL
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia studiów
        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudieID = @StudieID)
        BEGIN
            RAISERROR(N'Studia o podanym ID nie istnieją.', 16, 1);
            RETURN;
        END

        -- Aktualizacja danych w tabeli Studies
        UPDATE Studies
        SET
            StudieName = ISNULL(@StudieName, StudieName),
            StudieDescription = ISNULL(@StudieDescription, StudieDescription),
            StudieEntryFee = ISNULL(@StudieEntryFee, StudieEntryFee),
            StudieCoordinatorID = ISNULL(@StudieCoordinatorID, StudieCoordinatorID),
            PlaceLimit = ISNULL(@PlaceLimit, PlaceLimit)
        WHERE StudieID = @StudieID;

        PRINT N'Studia zostały pomyślnie zaktualizowane.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

## 6.10 DeleteStudie

- Usuwa studia z tabeli Studies o podanym ID.

```
CREATE PROCEDURE [dbo].[DeleteStudie]
    @StudieID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia studiów
        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudieID = @StudieID)
        BEGIN
            RAISERROR(N'Studia o podanym ID nie istnieją.', 16, 1);
            RETURN;
        END

        -- Usuwanie rekordu z tabeli Studies
        DELETE FROM Studies
        WHERE StudieID = @StudieID;

        PRINT N'Studia zostały pomyślnie usunięte.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

## 6.11 AddSubject

- Dodaje przedmiot na studiach o podanym ID do tabeli Subjects.

```
CREATE PROCEDURE [dbo].[AddSubject]
    @StudieID INT,
    @SubjectName NVARCHAR(64),
    @SubjectDescription NVARCHAR(128),
    @CoordinatorID INT
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia studiów
        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudieID = @StudieID)
        BEGIN
            RAISERROR(N'Studia o podanym ID nie istnieją.', 16, 1);
            RETURN;
        END

        -- Sprawdzenie istnienia koordynatora
        IF NOT EXISTS (SELECT 1 FROM Employees WHERE EmployeeID = @CoordinatorID)
        BEGIN
            RAISERROR('Koordynator o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Dodanie przedmiotu do tabeli Subjects
        INSERT INTO Subjects (StudieID, SubjectName, SubjectDescription,
CoordinatorID)
        VALUES (@StudieID, @SubjectName, @SubjectDescription, @CoordinatorID);

        PRINT N'Przedmiot został pomyślnie dodany.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

# 6.12 UpdateSubjectDetails

- Pozwala edytować szczegóły przedmiotu na studiach z tabeli Subjects o podanym ID.

```
CREATE PROCEDURE [dbo].[UpdateSubjectDetails]
    @SubjectID INT,
    @StudentID INT,
    @SubjectGrade INT = NULL
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia przedmiotu
        IF NOT EXISTS (SELECT 1 FROM Subjects WHERE SubjectID = @SubjectID)
        BEGIN
            RAISERROR('Przedmiot o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Sprawdzenie istnienia studenta
        IF NOT EXISTS (SELECT 1 FROM Students WHERE StudentID = @StudentID)
        BEGIN
            RAISERROR('Student o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Aktualizacja szczegółów w tabeli SubjectDetails
        UPDATE SubjectDetails
        SET
            SubjectGrade = ISNULL(@SubjectGrade, SubjectGrade)
        WHERE SubjectID = @SubjectID AND StudentID = @StudentID;

        PRINT N'Szczegóły przedmiotu zostały pomyślnie zaktualizowane.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

## 6.13 AddInternship

- Dodaje nowy staż do tabeli Internships.

```
CREATE PROCEDURE [dbo].[AddInternship]
    @StudieID INT,
    @InternshipDuration INT,
    @InternshipStartDate DATE
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia studiów
        IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudieID = @StudieID)
        BEGIN
            RAISERROR(N'Studia o podanym ID nie istnieją.', 16, 1);
            RETURN;
        END

        -- Dodanie nowego stażu
        INSERT INTO Internships (StudieID, InternshipDuration, InternshipStartDate)
        VALUES (@StudieID, @InternshipDuration, @InternshipStartDate);

        PRINT N'Staż został pomyślnie dodany.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

# 6.14 UpdateInternshipDetails

- Pozwala edytować szczegóły stażu o podanym ID w tabeli Internships.

```
CREATE PROCEDURE [dbo].[UpdateInternshipDetails]
    @InternshipID INT,
    @StudentID INT,
    @DaysAttended INT = NULL
AS
BEGIN
    BEGIN TRY
        -- Sprawdzenie istnienia stażu
        IF NOT EXISTS (SELECT 1 FROM Internships WHERE InternshipID = @InternshipID)
        BEGIN
            RAISERROR(N'Staż o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Sprawdzenie istnienia studenta
        IF NOT EXISTS (SELECT 1 FROM Students WHERE StudentID = @StudentID)
        BEGIN
            RAISERROR('Student o podanym ID nie istnieje.', 16, 1);
            RETURN;
        END

        -- Aktualizacja szczegółów stażu
        UPDATE InternshipDetails
        SET
            DaysAttended = ISNULL(@DaysAttended, DaysAttended)
        WHERE InternshipID = @InternshipID AND StudentID = @StudentID;

        PRINT N'Szczegóły stażu zostały pomyślnie zaktualizowane.';
    END TRY
    BEGIN CATCH
        -- Obsługa błędów
        DECLARE @ErrorMessage NVARCHAR(4000) = ERROR_MESSAGE();
        DECLARE @ErrorSeverity INT = ERROR_SEVERITY();
        DECLARE @ErrorState INT = ERROR_STATE();
        RAISERROR(@ErrorMessage, @ErrorSeverity, @ErrorState);
    END CATCH
END;
GO
```

# 6.15 AddStudent

- Dodaje nowego studenta do tabeli Students.

```
CREATE PROCEDURE AddStudent
@Email nvarchar(64), @Password nvarchar(64), @FirstName nvarchar(64), @LastName
nvarchar(64), @Address nvarchar(64), @City nvarchar(64), @Country nvarchar(64)
AS
BEGIN
    DECLARE @HashedPassword VARBINARY(64);
    SET @HashedPassword = HASHBYTES('SHA2_256', @Password);

    INSERT INTO Students (Email, Password, FirstName, LastName, Address, City,
Country)
    VALUES (@Email, @HashedPassword, @FirstName, @LastName, @Address, @City,
@Country)
END;
GO
```

# 6.16 EditStudent

- Pozwala edytować dane studenta z tabeli Students o podanym ID.

```
CREATE PROCEDURE EditStudent
    @StudentId INT,
    @Email NVARCHAR(64) = NULL,
    @Password NVARCHAR(64) = NULL,
    @FirstName NVARCHAR(64) = NULL,
    @LastName NVARCHAR(64) = NULL,
    @Address NVARCHAR(64) = NULL,
    @City NVARCHAR(64) = NULL,
    @Country NVARCHAR(64) = NULL
AS
BEGIN
    DECLARE @HashedPassword VARBINARY(64);

    IF @Password IS NOT NULL
    BEGIN
        SET @HashedPassword = HASHBYTES('SHA2_256', CONVERT(NVARCHAR(MAX),
@Password));
    END

    UPDATE Students
    SET
        Email = ISNULL(@Email, Email),
        Password = CASE
                       WHEN @Password IS NOT NULL THEN @HashedPassword
                       ELSE Password
                   END,
        FirstName = ISNULL(@FirstName, FirstName),
        LastName = ISNULL(@LastName, LastName),
        Address = ISNULL(@Address, Address),
        City = ISNULL(@City, City),
        Country = ISNULL(@Country, Country)
    WHERE StudentId = @StudentId;
END;
GO
```

## 6.17 RemoveStudent

- Usuwa studenta z tabeli students o podanym ID.

```
CREATE PROCEDURE RemoveStudent
    @StudentID INT
AS
BEGIN
    DELETE FROM Students
    WHERE StudentID = @StudentID;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie znaleziono studenta o podanym ID.';
    END
    ELSE
    BEGIN
        PRINT 'Student został usunięty.';
    END
END;
GO
```

## 6.18 AddEmployee

- Dodaje nowego pracownika do tabeli Employees.

```
CREATE PROCEDURE AddEmployee
@EmployeeID int, @Email nvarchar(64), @Password nvarchar(64), @FirstName
nvarchar(64), @LastName nvarchar(64), @Phone nvarchar(64), @HireDate date,
@EmployeeType int
AS
BEGIN
    DECLARE @HashedPassword VARBINARY(64);
    SET @HashedPassword = HASHBYTES('SHA2_256', @Password);

    INSERT INTO Employees (EmployeeID, Email, Password, FirstName, LastName, Phone,
HireDate, EmployeeType)
    VALUES (@EmployeeID, @Email, @HashedPassword, @FirstName, @LastName, @Phone,
@HireDate, @EmployeeType)
END;
GO
```

# 6.19 EditEmployee

- Pozwala edytować dane pracownika z tabeli Employees o podanym ID.

```
CREATE PROCEDURE EditEmployee
    @EmployeeID INT,
    @Email NVARCHAR(64) = NULL,
    @Password NVARCHAR(64) = NULL,
    @FirstName NVARCHAR(64) = NULL,
    @LastName NVARCHAR(64) = NULL,
    @Phone NVARCHAR(64) = NULL,
    @HireDate DATE = NULL,
    @EmployeeType INT = NULL
AS
BEGIN
    DECLARE @HashedPassword VARBINARY(64);

    IF @Password IS NOT NULL
    BEGIN
        SET @HashedPassword = HASHBYTES('SHA2_256', CONVERT(NVARCHAR(MAX),
@Password));
    END

    UPDATE Employees
    SET
        Email = ISNULL(@Email, Email),
        Password = CASE
                       WHEN @Password IS NOT NULL THEN @HashedPassword
                       ELSE Password
                   END,
        FirstName = ISNULL(@FirstName, FirstName),
        LastName = ISNULL(@LastName, LastName),
        Phone = ISNULL(@Phone, Phone),
        HireDate = ISNULL(@HireDate, HireDate),
        EmployeeType = ISNULL(@EmployeeType, EmployeeType)
    WHERE EmployeeID = @EmployeeID;
END;
GO
```

## 6.20 RemoveEmployee

- Usuwa pracownika z tabeli Employees o podanym ID.

```
CREATE PROCEDURE RemoveEmployee
    @EmployeeID INT
AS
BEGIN
    DELETE FROM Employees
    WHERE EmployeeID = @EmployeeID;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie znaleziono pracownika o podanym ID.';
    END
    ELSE
    BEGIN
        PRINT 'Pracownik został pomyślnie usunięty.';
    END
END;
GO
```

## 6.21 AddTranslator

- Dodaje nowego tłumacza do tabeli Translators.

```
CREATE PROCEDURE AddTranslator
@TranslatorID int, @Email nvarchar(64), @Password nvarchar(64), @FirstName
nvarchar(64), @LastName nvarchar(64), @Address nvarchar(64), @Phone nvarchar(64),
@HireDate date
AS
BEGIN
    DECLARE @HashedPassword VARBINARY(64);
    SET @HashedPassword = HASHBYTES('SHA2_256', @Password);

    INSERT INTO Translators (TranslatorID, Email, Password, FirstName, LastName,
Address, Phone, HireDate)
    VALUES (@TranslatorID, @Email, @HashedPassword, @FirstName, @LastName, @Address,
@Phone, @HireDate)
END;
GO
```

# 6.22 EditTranslator

- Pozwala edytować dane tłumacza z tabeli Translators o podanym ID.

```
CREATE PROCEDURE EditTranslator
    @TranslatorID INT,
    @Email NVARCHAR(64) = NULL,
    @Password NVARCHAR(64) = NULL,
    @FirstName NVARCHAR(64) = NULL,
    @LastName NVARCHAR(64) = NULL,
    @Phone NVARCHAR(64) = NULL,
    @HireDate DATE = NULL
AS
BEGIN
    DECLARE @HashedPassword VARBINARY(64);

    IF @Password IS NOT NULL
    BEGIN
        SET @HashedPassword = HASHBYTES('SHA2_256', CONVERT(NVARCHAR(MAX),
@Password));
    END

    UPDATE Translators
    SET
        Email = ISNULL(@Email, Email),
        Password = CASE
                       WHEN @Password IS NOT NULL THEN @HashedPassword
                       ELSE Password
                   END,
        FirstName = ISNULL(@FirstName, FirstName),
        LastName = ISNULL(@LastName, LastName),
        Phone = ISNULL(@Phone, Phone),
        HireDate = ISNULL(@HireDate, HireDate)
    WHERE TranslatorID = @TranslatorID;
END;
GO
```

## 6.23 RemoveTranslator

- Usuwa tłumacza z tabeli Translators o podanym ID.

```
CREATE PROCEDURE RemoveTranslator
    @TranslatorID INT
AS
BEGIN
    DELETE FROM Translators
    WHERE TranslatorID = @TranslatorID;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie znaleziono tłumacza o podanym ID.';
    END
    ELSE
    BEGIN
        PRINT 'Tłumacz został pomyślnie usunięty.';
    END
END;
GO
```

## 6.24 AddProductToCart

- Dodaje produkt do koszyka

```
CREATE PROCEDURE dbo.AddProductToCart
    @OrderID INT,
    @ProductID INT,
    @ProductType INT,
    @Price MONEY
AS
BEGIN
    INSERT INTO OrderDetails (OrderID, ProductID, ProductType, Price, StatusID)
    VALUES (@OrderID, @ProductID, @ProductType, @Price, 0); -- 0 oznacza produkt w
koszyku

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie udało się dodać produktu do koszyka.';
    END
    ELSE
    BEGIN
        PRINT 'Produkt został pomyślnie dodany do koszyka.';
    END
END;
GO
```

## 6.25 RemoveProductFromCart

- Usuwa produkt z koszyka

```
CREATE PROCEDURE dbo.RemoveProductFromCart
    @OrderID INT,
    @ProductID INT,
    @ProductType INT
AS
BEGIN
    DELETE FROM OrderDetails
    WHERE OrderID = @OrderID AND ProductID = @ProductID AND ProductType =
@ProductType;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie znaleziono produktu do usunięcia z koszyka.';
    END
    ELSE
    BEGIN
        PRINT 'Produkt został pomyślnie usunięty z koszyka.';
    END
END;
GO
```

## 6.26 MakeOrder

- Składa zamówienie

```
CREATE PROCEDURE dbo.MakeOrder
    @OrderID INT
AS
BEGIN
    UPDATE OrderDetails
    SET StatusID = 5
    WHERE OrderID = @OrderID;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie znaleziono zamówienia o podanym ID w szczegółach zamówienia.';
    END
    ELSE
    BEGIN
        PRINT 'Zamówienie zostało pomyślnie złożone.';
    END
END;
GO
```

# 6.27 ManuallyAddOrder

- Dodaj zamówienie (manualnie)

```
CREATE PROCEDURE dbo.ManuallyAddOrder
    @OrderID INT,
    @OrderDate DATETIME,
    @PaymentLink NVARCHAR(MAX)
AS
BEGIN
    INSERT INTO Orders (OrderID, OrderDate, PaymentLink)
    VALUES (@OrderID, @OrderDate, @PaymentLink);

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie udało się dodać nowego zamówienia.';
    END
    ELSE
    BEGIN
        PRINT 'Nowe zamówienie zostało pomyślnie dodane.';
    END
END;
GO
```

# 6.28 ManuallyAddOrderDetails

- Dodaje szczegóły zamówienia

```
CREATE PROCEDURE dbo.ManuallyAddOrderDetails
    @OrderID INT,
    @StatusID INT,
    @StudentID INT,
    @ProductType INT,
    @ProductID INT,
    @Price MONEY
AS
BEGIN
    INSERT INTO OrderDetails (OrderID, StatusID, StudentID, ProductType, ProductID,
Price)
    VALUES (@OrderID, @StatusID, @StudentID, @ProductType, @ProductID, @Price);

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie udało się dodać szczegółów zamówienia.';
    END
    ELSE
    BEGIN
        PRINT 'Szczegóły zamówienia zostały pomyślnie dodane.';
    END
END;
GO
```

# 6.29 ManuallyUpdateOrderStatus

- Zmienia status zamówienia

```
CREATE PROCEDURE dbo.ManuallyUpdateOrderStatus
    @OrderID INT,
    @StatusID INT
AS
BEGIN
    UPDATE OrderDetails
    SET StatusID = @StatusID
    WHERE OrderID = @OrderID;

    IF @@ROWCOUNT = 0
    BEGIN
        PRINT 'Nie znaleziono zamówienia o podanym ID.';
    END
    ELSE
    BEGIN
        PRINT 'Status zamówienia został pomyślnie zaktualizowany.';
    END
END;
GO
```

# 7. Indeksy

## 7.1 CourseDetails

```
CREATE INDEX CourseDetails_CourseID ON CourseDetails (CourseID);
CREATE INDEX CourseDetails_StudentID ON CourseDetails (StudentID);
```

## 7.2 CourseModules

```
CREATE INDEX CourseModules_CourseID ON CourseModules (CourseID);
CREATE INDEX CourseModules_TeacherID ON CourseModules (TeacherID);
CREATE INDEX CourseModules_LanguageID ON CourseModules (LanguageID);
CREATE INDEX CourseModules_TranslatorID ON CourseModules (TranslatorID);
```

## 7.3 Courses

```
CREATE INDEX Courses_CourseCoordinatorID ON Courses (CourseCoordinatorID);
```

## 7.4 EmployeeLanguages

```
CREATE INDEX EmployeeLanguages_EmployeeID ON EmployeeLanguages (EmployeeID);
CREATE INDEX EmployeeLanguages_LanguageID ON EmployeeLanguages (LanguageID);
```

## 7.5 Employees

```
CREATE INDEX Employees_EmployeeType ON Employees (EmployeeType);
```

## 7.6 InternshipDetails

```
CREATE INDEX InternshipDetails_InternshipID ON InternshipDetails (InternshipID);
CREATE INDEX InternshipDetails_StudentID ON InternshipDetails (StudentID);
```

## 7.7 Internships

```
CREATE INDEX Internships_StudiesID ON Internships (StudieID);
```

## 7.8 MeetingDetails

```
CREATE INDEX MeetingDetails_StudentID ON MeetingDetails (StudentID);
CREATE INDEX MeetingDetails_MeetingID ON MeetingDetails (MeetingID);
```

## 7.9 ModulesPassed

```
CREATE INDEX ModulesPassed_ModuleID ON ModulesPassed (ModuleID);
CREATE INDEX ModulesPassed_StudentID ON ModulesPassed (StudentID);
```

## 7.10 OnlineAsyncMeeting

```
CREATE INDEX OnlineAsyncMeeting_MeetingID ON OnlineAsyncMeeting (MeetingID);
```

## 7.11 OnlineAsyncModule

```
CREATE INDEX OnlineAsyncModule_ModuleID ON OnlineAsyncModule (ModuleID);
```

## 7.12 OnlineSyncMeeting

```
CREATE INDEX OnlineSyncMeeting_MeetingID ON OnlineSyncMeeting (MeetingID);
```

## 7.13 OnlineSyncModule

```
CREATE INDEX OnlineSyncModule_ModuleID ON OnlineSyncModule (ModuleID);
CREATE INDEX OnlineSyncModule_PlatformID ON OnlineSyncModule (PlatformID);
```

### 7.14 OrderDetails

```
CREATE INDEX OrderDetails_ProductID ON OrderDetails (ProductID);
CREATE INDEX OrderDetails_StudentID ON OrderDetails (StudentID);
CREATE INDEX OrderDetails_StatusID ON OrderDetails (StatusID);
CREATE INDEX OrderDetails_OrderID ON OrderDetails (OrderID);
```

### 7.15 StationaryMeeting

```
CREATE INDEX StationaryMeeting_RoomID ON StationaryMeeting (RoomID);
CREATE INDEX StationaryMeeting_MeetingID ON StationaryMeeting (MeetingID);
```

### 7.16 StationaryModule

```
CREATE INDEX StationaryModule_RoomID ON StationaryModule (RoomID);
CREATE INDEX StationaryModule_ModuleID ON StationaryModule (ModuleID);
```

### 7.17 StudieDetails

```
CREATE INDEX StudiesDetails_StudiesID ON StudieDetails (StudieID);
CREATE INDEX StudiesDetails_StudentID ON StudieDetails (StudentID);
```

### 7.18 Studies

```
CREATE INDEX Studies_StudieCoordinatorID ON Studies (StudieCoordinatorID);
```

### 7.19 StudiesMeeting

```
CREATE INDEX StudiesMeeting_SubjectID ON StudiesMeeting (SubjectID);
CREATE INDEX StudiesMeeting_TeacherID ON StudiesMeeting (TeacherID);
CREATE INDEX StudiesMeeting_TranslatorID ON StudiesMeeting (TranslatorID);
```

### 7.20 SubjectDetails

```
CREATE INDEX SubjectDetails_SubjectID ON SubjectDetails (SubjectID);
CREATE INDEX SubjectDetails_StudentID ON SubjectDetails (StudentID);
```

### 7.21 Subjects

```
CREATE INDEX Subjects_StudieID ON Subjects (StudieID);
CREATE INDEX Subjects_CoordinatorID ON Subjects (CoordinatorID);
```

### 7.22 WebinarAccess

```
CREATE INDEX WebinarAccess_StudentID ON WebinarAccess (StudentID);
CREATE INDEX WebinarAccess_WebinarID ON WebinarAccess (WebinarID);
```

### 7.23 Webinars

```
CREATE INDEX Webinars_PlatformID ON Webinars (PlatformID);
CREATE INDEX Webinars_TeacherID ON Webinars (TeacherID);
CREATE INDEX Webinars_TranslatorID ON Webinars (TranslatorID);
CREATE INDEX Webinars_LanguageID ON Webinars (LanguageID);
```

### 7.24 TranslatorsLanguages

```
CREATE INDEX TranslatorsLanguages_TranslatorID ON TranslatorsLanguages
(TranslatorID);
CREATE INDEX TranslatorsLanguages_LanguageID ON TranslatorsLanguages (LanguageID);
```

### 7.25 WebinarDetails

```
CREATE INDEX WebinarDetails_StudentID ON WebinarDetails (StudentID);
CREATE INDEX WebinarDetails_WebinarID ON WebinarDetails (WebinarID);
```

## 7.26 Indeksy ram czasowych

```
CREATE INDEX CourseModule_Time ON CourseModules (ModuleDate, ModuleDuration);
CREATE INDEX Internships_Time ON Internships (InternshipStartDate,
InternshipDuration);
CREATE INDEX StudiesMeeting_Time ON StudiesMeeting (MeetingDate, MeetingDuration);
CREATE INDEX Webinar_Time ON Webinars (WebinarStartDate);
```

## 7.27 Indeksy cen

```
CREATE INDEX Studies_Price ON Studies (StudieEntryFee);
CREATE INDEX StudiesMeeting_Price ON StudiesMeeting (MeetingPrice);
CREATE INDEX Course_Price ON Courses (CoursePrice);
CREATE INDEX Webinar_Price ON Webinars (WebinarPrice);
```

# 8. Triggery

## 8.1 trg_after_payment

- Daje dostęp do zakupionego produktu

```
CREATE TRIGGER trg_after_payment
ON OrderDetails
AFTER UPDATE
AS
BEGIN
    -- Sprawdzamy, czy StatusID zmienił się na 4 - opłacony
    IF EXISTS (SELECT 1 FROM inserted WHERE StatusID = 4 AND StatusID <> (SELECT
StatusID FROM deleted))
    BEGIN
        DECLARE @ProductType INT, @ProductID INT, @StudentID INT;
        DECLARE order_cursor CURSOR FOR
        SELECT ProductType, ProductID, StudentID
        FROM inserted
        WHERE StatusID = 4;

        OPEN order_cursor;
        FETCH NEXT FROM order_cursor INTO @ProductType, @ProductID, @StudentID;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            IF @ProductType = 0
            BEGIN
                INSERT INTO WebinarAccess (WebinarID, StudentID, HasAccess)
                VALUES (@ProductID, @StudentID, 1);
            END
            ELSE IF @ProductType = 1
            BEGIN
                INSERT INTO CourseDetails (CourseID, StudentID)
                VALUES (@ProductID, @StudentID);
            END
            ELSE IF @ProductType = 2
            BEGIN
                INSERT INTO StudieDetails (StudieID, StudentID)
                VALUES (@ProductID, @StudentID);
            END

            FETCH NEXT FROM order_cursor INTO @ProductType, @ProductID, @StudentID;
        END

        CLOSE order_cursor;
        DEALLOCATE order_cursor;
    END
END
```

## 8.2 trg_after_delete_webinar

- Sprzątanie po usunięciu webinaru

```
CREATE TRIGGER trg_after_delete_webinar
ON Webinars
AFTER DELETE
AS
BEGIN
    DELETE FROM WebinarAccess
    WHERE WebinarID IN (SELECT WebinarID FROM deleted);

    DELETE FROM WebinarDetails
    WHERE WebinarID IN (SELECT WebinarID FROM deleted);

    DELETE FROM OrderDetails
    WHERE ProductType = 1
    AND ProductID IN (SELECT WebinarID FROM deleted)
    AND StatusID IN (0, 5); -- 0 - w koszyku, 5 - nieopłacony
END
```

## 8.3 trg_after_delete_course

- Sprzątanie po usunięciu kursu

```
CREATE TRIGGER trg_after_delete_course
ON Courses
AFTER DELETE
AS
BEGIN
    DELETE FROM CourseModules
    WHERE CourseID IN (SELECT CourseID FROM deleted);

    DELETE FROM CourseDetails
    WHERE CourseID IN (SELECT CourseID FROM deleted);

    DELETE FROM OrderDetails
    WHERE ProductType = 1
    AND ProductID IN (SELECT CourseID FROM deleted)
    AND StatusID IN (0, 5); -- koszyk, nieopłacony
END
```

# 8.4 trg_after_delete_study

- Sprzątanie po usunięciu studiów

```
CREATE TRIGGER trg_after_delete_study
ON Studies
AFTER DELETE
AS
BEGIN
    DELETE FROM Internships
    WHERE StudieID IN (SELECT StudieID FROM deleted);

    DELETE FROM Subjects
    WHERE StudieID IN (SELECT StudieID FROM deleted);

    DELETE FROM OrderDetails
    WHERE ProductType = 2
    AND ProductID IN (SELECT StudieID FROM deleted)
    AND StatusID IN (0, 5);
END
```

# 9. Uprawnienia

## 9.1 Role_Admin

- Administrator bazy danych

```
CREATE ROLE Role_Admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA :: dbo TO Role_Admin;
GRANT EXECUTE TO Role_Admin;
```

## 9.2 Role_Student

- Student w bazie danych

```
CREATE ROLE Role_Student;
GRANT SELECT, INSERT, UPDATE, DELETE ON Orders TO Role_Student;
GRANT SELECT, INSERT, UPDATE, DELETE ON OrderDetails TO Role_Student;
GRANT SELECT ON Webinars TO Role_Student;
GRANT SELECT ON WebinarDetails TO Role_Student;
GRANT SELECT ON Platforms TO Role_Student;
GRANT SELECT ON Studies TO Role_Student;
GRANT SELECT ON StudiesMeeting TO Role_Student;
GRANT SELECT ON StationaryMeeting TO Role_Student;
GRANT SELECT ON Rooms TO Role_Student;
GRANT SELECT ON OnlineSyncMeeting TO Role_Student;
GRANT SELECT ON OnlineAsyncMeeting TO Role_Student;
GRANT SELECT ON MeetingDetail TO Role_Student;
GRANT SELECT ON Internships TO Role_Student;
GRANT SELECT ON InternshipDetails TO Role_Student;
GRANT SELECT ON Subjects TO Role_Student;
GRANT SELECT ON Courses TO Role_Student;
GRANT SELECT ON CourseModules TO Role_Student;
GRANT SELECT ON OnlineSyncModule TO Role_Student;
GRANT SELECT ON StationaryModule TO Role_Student;
GRANT SELECT ON OnlineAsyncModule TO Role_Student;
GRANT SELECT ON ModulesPassed TO Role_Student;
```

## 9.3 Role_Teacher

- Nauczyciel w bazie danych

```
CREATE ROLE Role_Teacher;
GRANT SELECT ON EmployeeType TO Role_Teacher;
GRANT SELECT ON Employees TO Role_Teacher;
GRANT SELECT ON Translators TO Role_Teacher;
GRANT SELECT ON Language TO Role_Teacher;
GRANT SELECT ON Students TO Role_Teacher;
GRANT SELECT ON ModulesPassed TO Role_Teacher;
GRANT SELECT ON Courses TO Role_Teacher;
GRANT SELECT ON CourseModules TO Role_Teacher;
GRANT SELECT ON CourseDetails TO Role_Teacher;
GRANT SELECT ON OnlineSyncModule TO Role_Teacher;
GRANT SELECT ON StationaryModule TO Role_Teacher;
GRANT SELECT ON Rooms TO Role_Teacher;
GRANT SELECT ON OnlineAsyncModule TO Role_Teacher;
GRANT SELECT ON Studies TO Role_Teacher;
GRANT SELECT ON StudiesMeeting TO Role_Teacher;
GRANT SELECT ON OnlineAsyncMeeting TO Role_Teacher;
GRANT SELECT ON OnlineSyncMeeting TO Role_Teacher;
GRANT SELECT ON StationaryMeeting TO Role_Teacher;
GRANT SELECT ON SubjectDetails TO Role_Teacher;
GRANT SELECT ON Subjects TO Role_Teacher;
GRANT SELECT ON Webinars TO Role_Teacher;
GRANT SELECT ON WebinarDetails TO Role_Teacher;
GRANT SELECT ON Platforms TO Role_Teacher;
GRANT SELECT, INSERT, UPDATE ON EmployeeLanguages TO Role_Teacher;
GRANT SELECT, INSERT, UPDATE ON MeetingDetails TO Role_Teacher;
GRANT SELECT, INSERT, UPDATE ON StudieDetails TO Role_Teacher;
```

## 9.4 Role_Translator

- Tłumacz w bazie danych

```
CREATE ROLE Role_Translator;
GRANT SELECT, INSERT, UPDATE ON EmployeeLanguages TO Role_Translator;
GRANT SELECT ON Employees TO Role_Translator;
GRANT SELECT ON Translators TO Role_Translator;
GRANT SELECT ON EmployeeType TO Role_Translator;
GRANT SELECT ON StudiesMeeting TO Role_Translator;
GRANT SELECT ON StationaryMeeting TO Role_Translator;
GRANT SELECT ON Rooms TO Role_Translator;
GRANT SELECT ON OnlineSyncMeeting TO Role_Translator;
GRANT SELECT ON OnlineAsyncMeeting TO Role_Translator;
GRANT SELECT ON Subjects TO Role_Translator;
```

# 10. Kod generujący dane

Do generowania danych wykorzystano następujące narzędzia:

Python Faker – użyty do generowania fikcyjnych danych, takich jak imiona, nazwiska, adresy, numery telefonów itp. Jupyter Notebook – środowisko, w którym wykonywano skrypty do generowania danych i ich analizy. Pandas – biblioteka służąca do manipulacji danymi, tworzenia oraz zapisu do pliku CSV. Proces generowania i importowania danych:

Za pomocą biblioteki Faker wygenerowano przykładowe dane testowe. Przy użyciu Pandas dane te zostały przekształcone do formatu tabelarycznego oraz zapisane do pliku CSV. Plik CSV został ręcznie zaimportowany do bazy danych poprzez interfejs administracyjny bazy. Poniżej znajduje się nasza implementacja.

```
from faker import Faker
import pandas as pd

fake = Faker(locale='pl_PL')
fake.name()

STUDENT_COUNT = 1000
EMPLOYEE_COUNT = 300
COURSE_COUNT = 100
MODULE_COUNT = 100
SUBJECT_COUNT = 100
STUDIE_COUNT = 100
MEETING_COUNT = 1000
ORDER_COUNT = 100
INTERN_COUNT = 100
WEBINAR_COUNT = 100
ROOM_COUNT = 100
PLATFORM_COUNT = 100
LANGUAGE_COUNT = 100
STATUS_COUNT = 100
EMPLOYEE_TYPE_COUNT = 3


def create_course_details(n):
    course_details_list = []
    for i in range(1, n):
        course_details = {}
        course_details['CourseID'] = fake.random_int(1, COURSE_COUNT)
        course_details['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        course_details_list.append(course_details)
    return pd.DataFrame(course_details_list)

def create_course_modules():
    module_list = []
    module_names = [
        "Analiza Matematyczna", "Algebra Liniowa", "Fizyka",
        "Chemia", "Biologia", "Informatyka", "Statystyka",
        "Ekonomia", "Zarządzanie", "Prawo"
    ]
    module_descriptions = [
```

```
        "Zaawansowane zagadnienia analizy matematycznej, w tym granice, pochodne i
całki.",
        "Podstawy algebry liniowej, w tym macierze, wektory i przestrzenie
wektorowe.",
        "Podstawowe zasady fizyki, w tym mechanika, termodynamika i
elektromagnetyzm.",
        "Podstawy chemii, w tym struktura atomowa, reakcje chemiczne i równania
chemiczne.",
        "Podstawy biologii, w tym biologia komórkowa, genetyka i ekologia.",
        "Podstawy informatyki, obejmujące algorytmy, struktury danych i systemy
operacyjne.",
        "Podstawy statystyki, w tym analiza danych, testowanie hipotez i regresja.",
        "Podstawy ekonomii, w tym mikroekonomia, makroekonomia i teoria gier.",
        "Podstawy zarządzania, w tym planowanie, organizowanie i kontrola.",
        "Podstawy prawa, w tym prawo cywilne, karne i administracyjne."
    ]

    for name, description in zip(module_names, module_descriptions):
        module = {}
        module['CourseID'] = fake.random_int(1, COURSE_COUNT)
        module['ModuleName'] = name
        module['ModuleDescription'] = description
        module['ModuleDate'] = fake.date_this_year()
        module['ModuleDuration'] = fake.time()
        module['TeacherID'] = fake.random_int(1, EMPLOYEE_COUNT)
        module['TranslatorID'] = fake.random_int(1, EMPLOYEE_COUNT)
        module['LanguageID'] = fake.random_int(1, LANGUAGE_COUNT)
        module_list.append(module)
    return pd.DataFrame(module_list)


def create_courses():
    course_list = []
    course_names = [
        "Podstawy Programowania", "Zaawansowana Matematyka", "Wprowadzenie do
Fizyki",
        "Chemia Organiczna", "Biologia Molekularna", "Podstawy Informatyki",
        "Statystyka i Analiza Danych", "Podstawy Ekonomii", "Zarządzanie
Projektami",
        "Prawo w Biznesie"
    ]
    course_descriptions = [
        "Kurs wprowadzający do podstaw programowania, obejmujący języki takie jak
Python i Java.",
        "Zaawansowane zagadnienia matematyczne, w tym algebra, analiza i geometria.",
        "Podstawowe zasady fizyki, w tym mechanika, termodynamika i
elektromagnetyzm.",
        "Chemia organiczna, obejmująca struktury, reakcje i syntezę związków
organicznych.",
        "Biologia molekularna, w tym DNA, RNA, białka i techniki laboratoryjne.",
        "Podstawy informatyki, obejmujące algorytmy, struktury danych i systemy
operacyjne.",
        "Statystyka i analiza danych, w tym techniki analizy danych i modelowanie
statystyczne.",
        "Podstawy ekonomii, w tym mikroekonomia, makroekonomia i teoria gier.",
        "Zarządzanie projektami, w tym planowanie, realizacja i kontrola projektów.",
```

```
        "Prawo w biznesie, obejmujące prawo handlowe, umowy i regulacje."
    ]

    for name, description in zip(course_names, course_descriptions):
        course = {}
        course['CourseName'] = name
        course['CourseDescription'] = description
        course['CourseLength'] = fake.random_int(1, 52)
        course['CoursePrice'] = fake.random_int(1000, 3000, 500)
        course['CourseCoordinatorID'] = fake.random_int(1, EMPLOYEE_COUNT)
        course_list.append(course)
    return pd.DataFrame(course_list)

def create_employees(num_employees):
    employee_list = []
    for i in range(1, num_employees):
        employee = {}
        employee['id'] = i
        employee['Email'] = fake.email()
        employee['Password'] = fake.password()
        employee['FirstName'] = fake.first_name()
        employee['LastName'] = fake.last_name()
        employee['Address'] = fake.address()
        employee['Phone'] = fake.phone_number()
        employee['HireDate'] = fake.date()
        employee['EmployeeType'] = fake.random_int(min=0, max=2)
        employee_list.append(employee)
    return pd.DataFrame(employee_list)

def create_employee_languages(n):
    employee_language_list = []
    for i in range(1, n):
        employee_language = {}
        employee_language['EmployeeID'] = fake.random_int(1, EMPLOYEE_COUNT)
        employee_language['LanguageID'] = fake.random_int(1, LANGUAGE_COUNT)
        employee_language_list.append(employee_language)
    return pd.DataFrame(employee_language_list)


def create_employee_type():
    return pd.DataFrame({
        'EmployeeType': [0, 1, 2],
        'EmployeeName': ['Coordinator', 'Teacher', 'Translator']
    })


def create_internship_details(n):
    internship_details_list = []
    for i in range(1, n):
        internship_details = {}
        internship_details['InternshipID'] = fake.random_int(1, INTERN_COUNT)
        internship_details['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        internship_details['DaysAttended'] = fake.random_int(0, 30)
        internship_details_list.append(internship_details)
    return pd.DataFrame(internship_details_list)
```

```python
def create_internships(n):
    internship_list = []
    for i in range(1, n):
        internship = {}
        internship['StudieID'] = fake.random_int(1, STUDIE_COUNT)
        internship['InternshipDuration'] = fake.random_int(20, 100)
        internship['InternshipStartDate'] = fake.date_this_year()
        internship_list.append(internship)
    return pd.DataFrame(internship_list)


def create_languages(n):
    language_list = []
    for i in range(1, n):
        language = {}
        language['LanguageName'] = fake.language_name()
        language_list.append(language)
    return pd.DataFrame(language_list)

def create_meeting_details(n):
    meeting_details_list = []
    for i in range(1, n):
        meeting_details = {}
        meeting_details['MeetingID'] = fake.random_int(1, MEETING_COUNT)
        meeting_details['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        meeting_details['Presence'] = fake.random_int(min=0, max=1)
        meeting_details_list.append(meeting_details)
    return pd.DataFrame(meeting_details_list)


def create_modules_passed(n):
    modules_passed_list = []
    for i in range(1, n):
        modules_passed = {}
        modules_passed['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        modules_passed['ModuleID'] = fake.random_int(1, MODULE_COUNT)
        modules_passed_list.append(modules_passed)
    return pd.DataFrame(modules_passed_list)


def create_online_async_meeting(n):
    online_async_meeting_list = []
    for i in range(1, n):
        online_async_meeting = {}
        online_async_meeting['MeetingID'] = fake.random_int(1, MEETING_COUNT)
        online_async_meeting['RecordingUrl'] = fake.url()
        online_async_meeting_list.append(online_async_meeting)
    return pd.DataFrame(online_async_meeting_list)


def create_online_async_module(n):
    online_async_module_list = []
    for i in range(1, n):
        online_async_module = {}
        online_async_module['ModuleID'] = fake.random_int(1, MODULE_COUNT)
        online_async_module['RecordingUrl'] = fake.url()
```

```
        online_async_module_list.append(online_async_module)
    return pd.DataFrame(online_async_module_list)

def create_online_sync_meeting(n):
    online_sync_meeting_list = []
    for i in range(1, n):
        online_sync_meeting = {}
        online_sync_meeting['MeetingID'] = fake.random_int(1, MEETING_COUNT)
        online_sync_meeting['MeetingUrl'] = fake.url()
        online_sync_meeting_list.append(online_sync_meeting)
    return pd.DataFrame(online_sync_meeting_list)

def create_online_sync_module(n):
    online_sync_module_list = []
    for i in range(1, n):
        online_sync_module = {}
        online_sync_module['ModuleID'] = fake.random_int(1, MODULE_COUNT)
        online_sync_module['PlatformID'] = fake.random_int(1, PLATFORM_COUNT)
        online_sync_module['RecordingUrl'] = fake.url()
        online_sync_module_list.append(online_sync_module)
    return pd.DataFrame(online_sync_module_list)


def create_order_details(n):
    order_details_list = []
    for i in range(1, n):
        order_details = {}
        order_details['OrderID'] = fake.random_int(1, ORDER_COUNT)
        order_details['StatusID'] = fake.random_int(1, STATUS_COUNT)
        order_details['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        order_details['ProductType'] = fake.random_int(min=0, max=2) # warto
potwierdzić
        order_details['ProductID'] = fake.random_int(1, COURSE_COUNT)
        order_details['Price'] = fake.random_int(100, 1000) # warto potwierdzić
        order_details_list.append(order_details)
    return pd.DataFrame(order_details_list)

def create_orders(n):
    order_list = []
    for i in range(1, n):
        order = {}
        order['OrderDate'] = fake.date_this_year()
        order['PaymentLink'] = fake.url()
        order_list.append(order)
    return pd.DataFrame(order_list)


def create_platforms(n):
    platform_list = []
    for i in range(1, n):
        platform = {}
        platform['PlatformName'] = fake.company()
        platform_list.append(platform)
    return pd.DataFrame(platform_list)

def create_rooms(n):
```

```
    room_list = []
    for i in range(1, n):
        room = {}
        room['RoomName'] = fake.company()
        room['PlaceLimit'] = fake.random_int(30, 300)
        room_list.append(room)
    return pd.DataFrame(room_list)

def create_stationary_meeting(n):
    stationary_meeting_list = []
    for i in range(1, n):
        stationary_meeting = {}
        stationary_meeting['MeetingID'] = fake.random_int(1, MEETING_COUNT)
        stationary_meeting['RoomID'] = fake.random_int(1, ROOM_COUNT)
        stationary_meeting_list.append(stationary_meeting)
    return pd.DataFrame(stationary_meeting_list)

def create_stationary_module(n):
    stationary_module_list = []
    for i in range(1, n):
        stationary_module = {}
        stationary_module['ModuleID'] = fake.random_int(1, MODULE_COUNT)
        stationary_module['RoomID'] = fake.random_int(1, ROOM_COUNT)
        stationary_module_list.append(stationary_module)
    return pd.DataFrame(stationary_module_list)

def create_statuses(n):
    status_list = []
    for i in range(1, n):
        status = {}
        status['Description'] = fake.paragraph()
        status_list.append(status)
    return pd.DataFrame(status_list)

def create_students(n):
    students_list = []
    for i in range(1, n):
        student = {}
        student['Email'] = fake.email()
        student['Password'] = fake.password()
        student['FirstName'] = fake.first_name()
        student['LastName'] = fake.last_name()
        student['Address'] = fake.address()
        student['City'] = fake.city()
        student['Country'] = fake.country()
        students_list.append(student)

    return pd.DataFrame(students_list)

def create_studie_details(n):
    studie_details_list = []
    for i in range(1, n):
        studie_details = {}
        studie_details['StudieID'] = fake.random_int(1, STUDIE_COUNT)
        studie_details['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        studie_details['StudieGrade'] = fake.random_int(2, 5)
```

```
        studie_details_list.append(studie_details)
    return pd.DataFrame(studie_details_list)

def create_studies(n):
    studie_list = []
    for i in range(1, n):
        studie = {}
        studie['StudieName'] = fake.catch_phrase()
        studie['StudieDescription'] = fake.paragraphs()
        studie['StudieEntryFee'] = fake.random_int(1000, 3000, 500)
        studie['StudieCoordinatorID'] = fake.random_int(1, EMPLOYEE_COUNT * 3)
        studie['PlaceLimit'] = fake.random_int(30, 300)
        studie_list.append(studie)
    return pd.DataFrame(studie_list)


def create_studies_meeting(n):
    studies_meeting_list = []
    for i in range(1, n):
        studies_meeting = {}
        studies_meeting['SubjectID'] = fake.random_int(1, SUBJECT_COUNT)
        studies_meeting['TeacherID'] = fake.random_int(1, EMPLOYEE_COUNT * 3)
        studies_meeting['MeetingName'] = fake.catch_phrase()
        studies_meeting['MeetingPrice'] = fake.random_int(100, 1000, 100)
        studies_meeting['MeetingDate'] = fake.date_this_year()
        studies_meeting['MeetingDuration'] = fake.time()
        studies_meeting['MeetingDuration'] = f"{fake.random_int(0, 4)}h
{fake.random_int(20, 59)}min"
        studies_meeting['TranslatorID'] = fake.random_int(1, EMPLOYEE_COUNT * 3)
        studies_meeting['LanguageID'] = fake.random_int(1, LANGUAGE_COUNT)
        studies_meeting_list.append(studies_meeting)
    return pd.DataFrame(studies_meeting_list)

def create_subject_details(n):
    subject_details_list = []
    for i in range(1, n):
        subject_details = {}
        subject_details['SubjectID'] = fake.random_int(1, SUBJECT_COUNT)
        subject_details['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        subject_details['SubjectGrade'] = fake.random_int(2, 5)
        subject_details_list.append(subject_details)
    return pd.DataFrame(subject_details_list)


def create_subjects(n):
    subject_list = []
    for i in range(1, n):
        subject = {}
        subject['StudieID'] = fake.random_int(1, STUDIE_COUNT)
        subject['SubjectName'] = fake.catch_phrase()
        subject['SubjectDescription'] = fake.paragraphs()
        subject['CoordinatorID'] = fake.random_int(1, EMPLOYEE_COUNT * 3)
        subject_list.append(subject)
    return pd.DataFrame(subject_list)

def create_translators(n):
```

```
    translator_list = []
    for i in range(1, n):
        translator = {}
        translator['Email'] = fake.email()
        translator['Password'] = fake.password()
        translator['FirstName'] = fake.first_name()
        translator['LastName'] = fake.last_name()
        translator['Phone'] = fake.phone_number()
        translator['HireDate'] = fake.date()
        translator['Address'] = fake.address()
        translator_list.append(translator)
    return pd.DataFrame(translator_list)

def create_webinar_access(n):
    webinar_access_list = []
    for i in range(1, n):
        webinar_access = {}
        webinar_access['StudentID'] = fake.random_int(1, STUDENT_COUNT)
        webinar_access['HasAccess'] = fake.random_int(min=0, max=1)
        webinar_access_list.append(webinar_access)
    return pd.DataFrame(webinar_access_list)

def create_webinar_details(n):
    webinar_details_list = []
    for i in range(1, n):
        webinar_details = {}
        webinar_details['WebinarID'] = fake.random_int(1, WEBINAR_COUNT)
        webinar_details['ExpireDate'] = fake.date_this_year()
        webinar_details['RecordingUrl'] = fake.url()
        webinar_details['Deleted'] = fake.random_int(min=0, max=1)
        webinar_details_list.append(webinar_details)
    return pd.DataFrame(webinar_details_list)

def create_webinars(n):
    webinar_list = []
    for i in range(1, n):
        webinar = {}
        webinar['PlatformID'] = fake.random_int(1, PLATFORM_COUNT)
        webinar['TeacherID'] = fake.random_int(1, EMPLOYEE_COUNT * 3)
        webinar['WebinarName'] = fake.catch_phrase()
        webinar['WebinarDescription'] = fake.paragraphs()
        webinar['WebinarPrice'] = fake.random_int(100, 1000, 100)
        webinar['WebinarStartDate'] = fake.date_this_year()
        webinar['IsFree'] = fake.random_int(min=0, max=1)
        webinar_list.append(webinar)
    return pd.DataFrame(webinar_list)




course_details = create_course_details(100)
course_details.to_csv('course_details.csv')

course_modules = create_course_modules()
course_modules.to_csv('course_modules.csv')
```

```
courses = create_courses()
courses.to_csv('courses.csv')

employees = create_employees(EMPLOYEE_COUNT)
employees.to_csv('employees.csv')

employee_languages = create_employee_languages(EMPLOYEE_COUNT)
employee_languages.to_csv('employee_languages.csv')

employee_types = create_employee_type()
employee_types.to_csv('employee_types.csv')

internship_details = create_internship_details(INTERN_COUNT)
internship_details.to_csv('internship_details.csv')

internships = create_internships(INTERN_COUNT)
internships.to_csv('internships.csv')

languages = create_languages(LANGUAGE_COUNT)
languages.to_csv('languages.csv')

meeting_details = create_meeting_details(MEETING_COUNT)
meeting_details.to_csv('meeting_details.csv')

modules_passed = create_modules_passed(STUDENT_COUNT)
modules_passed.to_csv('modules_passed.csv')

online_async_meeting = create_online_async_meeting(MEETING_COUNT)
online_async_meeting.to_csv('online_async_meeting.csv')

online_async_module = create_online_async_module(MODULE_COUNT * 10)
online_async_module.to_csv('online_async_module.csv')

online_sync_meeting = create_online_sync_meeting(MEETING_COUNT)
online_sync_meeting.to_csv('online_sync_meeting.csv')

online_sync_module = create_online_sync_module(MODULE_COUNT)
online_sync_module.to_csv('online_sync_module.csv')

order_details = create_order_details(ORDER_COUNT * 10)
order_details.to_csv('order_details.csv')

orders = create_orders(ORDER_COUNT)
orders.to_csv('orders.csv')

platforms = create_platforms(PLATFORM_COUNT)
platforms.to_csv('platforms.csv')

rooms = create_rooms(ROOM_COUNT)
rooms.to_csv('rooms.csv')

statuses = create_statuses(STATUS_COUNT)
statuses.to_csv('statuses.csv')

students = create_students(STUDENT_COUNT)
students.to_csv('students.csv')
```

```
studie_details = create_studie_details(STUDIE_COUNT)
studie_details.to_csv('studie_details.csv')

studies = create_studies(STUDIE_COUNT)
studies.to_csv('studies.csv')

studies_meeting = create_studies_meeting(MEETING_COUNT)
studies_meeting.to_csv('studies_meeting.csv')

subject_details = create_subject_details(SUBJECT_COUNT)
subject_details.to_csv('subject_details.csv')

subjects = create_subjects(SUBJECT_COUNT)
subjects.to_csv('subjects.csv')

translators = create_translators(EMPLOYEE_COUNT)
translators.to_csv('translators.csv')

webinar_access = create_webinar_access(WEBINAR_COUNT)
webinar_access.to_csv('webinar_access.csv')

webinar_details = create_webinar_details(WEBINAR_COUNT)
webinar_details.to_csv('webinar_details.csv')

webinars = create_webinars(WEBINAR_COUNT)
webinars.to_csv('webinars.csv')

stationary_module = create_stationary_module(MODULE_COUNT)
stationary_module.to_csv('stationary_module.csv')

stationary_meeting = create_stationary_meeting(MEETING_COUNT)
stationary_meeting.to_csv('stationary_meeting.csv')
```

# 11. Podsumowanie

Przedstawiona baza danych została zaprojektowana w celu zapewnienia efektywnej obsługi firmy prowadzącej usługi edukacyjne. Główne cechy struktury bazy to:

- przejrzysta organizacja danych – dane zostały uporządkowane w logiczne tabele z jasno określonymi relacjami, co ułatwia ich interpretację i zarządzanie.

- zastosowanie kluczy głównych i obcych, wraz z triggerami i procedurami zapewnia spójność i poprawność danych.

- uwzględnienie różnorodnych form prowadzenia zajęć – kursów, webinarów czy zajęć stacjonarnych na studiach, pozwala to na kompleksowe zarządzanie informacjami o harmonogramach, frekwencji uczestników oraz powiązaniach między zajęciami a osobami odpowiedzialnymi za ich realizację.

- system ról umożliwiający przypisywanie użytkownikom różnych uprawnień – od administratorów zarządzających całą bazą, przez wykładowców odpowiedzialnych za zajęcia, aż po studentów przeglądających swoje programy i wyniki.

# 11.1 Podział pracy podczas projektu

## Adam Głowacki

Odpowiadał za:

- Schemat dotyczący pracowników i uczestników studiów.
- Funkcje i procedury związane z pracownikami i uczestnikami studiów.
- Dokumentację

Wkład czasowy: 34%.

## Igor Piesik

Odpowiadał za:

- Widoki, indeksy oraz dane dotyczące studiów, kursów i webinarów.
- Funkcje i procedury związane z powyższymi kategoriami.
- Generowanie danych do bazy.

Wkład czasowy: 33%.

## Dawid Szłapa

Odpowiadał za:

- Schemat i kategorię orders.
- Funkcje i procedury związane z zamówieniami.
- Triggery i uprawnienia.

Wkład czasowy: 33%.