# Baza danych szkoły

## Projekt na przedmiot Bazy Danych 2024/2025

Kinga Żmuda, Anna Gruca

# Założenia projektu

Projekt zakłada stworzenie bazy danych u podstaw zarządzania szkołą ponadpodstawową. Realizuje główne funkcjonalności dziennika internetowego (wpisywanie/wyświetlanie ocen, sprawdzanie obecności, wyświetlanie planu lekcji), automatyzuje procesy zapisu i wypisu ucznia ze szkoły/klasy wystawiania ocen końcowych i decydowania o promocji do kolejnych klas oraz przechowuje i przetwarza dane uczniów, pracowników, rodziców, organizacji i kół, kluczowe dla codziennego funkcjonowania placówki. W bazie śledzone są również dane uczniów zagrożonych niezdaniem do następnej klasy czy rankingi uczniów o najlepszej frekwencji i wynikach w nauce, celem stosownego nagrodzenia na koniec roku.

Dzięki uporządkowanemu podziałowi tabel na schematy, bazę można w przyszłości będzie łatwo zaadaptować do dedykowanej aplikacji z podziałem na użytkowników (uczniów, rodziców, nauczycieli, administratorów) o określonych uprawnieniach.

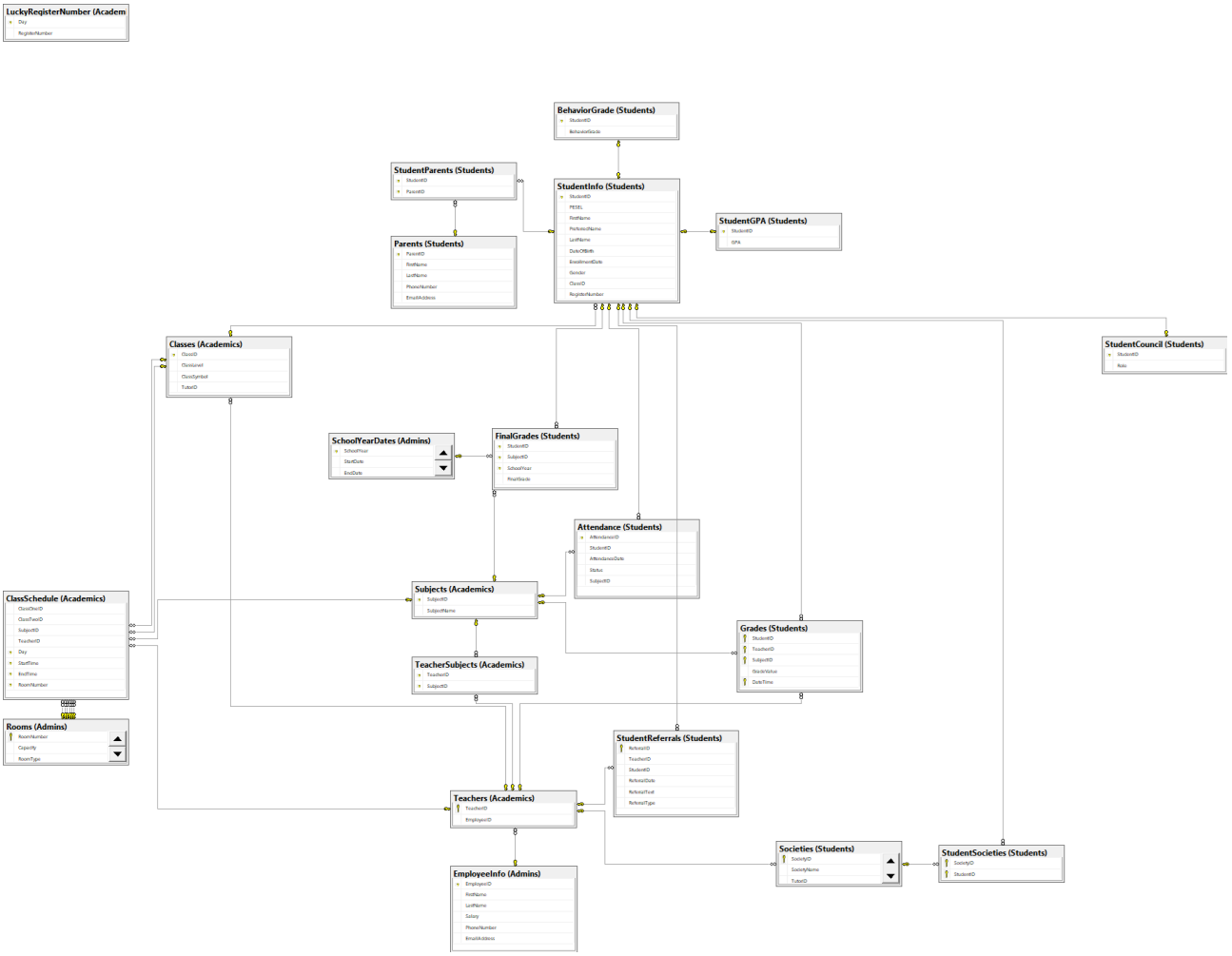
Naszym podstawowym celem przy projektowaniu i tworzeniu bazy było sięgnięcie do rzeczywistych zadań stojących przed szkołami i jak najwierniejsze (choć w sposób uproszczony) odwzorowanie tej prawdy, odpowiadając na potrzeby i wyzwania w digitalizacji systemów bez których obecnie nie mogłaby w pełni funcjonować żadna placówka edukacyjna.

Baza danych jest modelowana na polskim systemie szkolnictwa - stąd wystawianie ocen końcoworocznych bezpośrednio na podstawie cząstkowych, ocena z zachowania czy sześciostopniowa skala ocen. Niemniej do szkoły zapisany może zostać uczeń niebędący obywatelem polski, a zatem nieposiadający numeru PESEL, gdyż rejestr uczniów dopuszcza zostawienie tego pola jako puste (NULL), a każdy uczeń ma przyporządkowywany wewnątrzszkolny unikalny identyfikator ucznia.

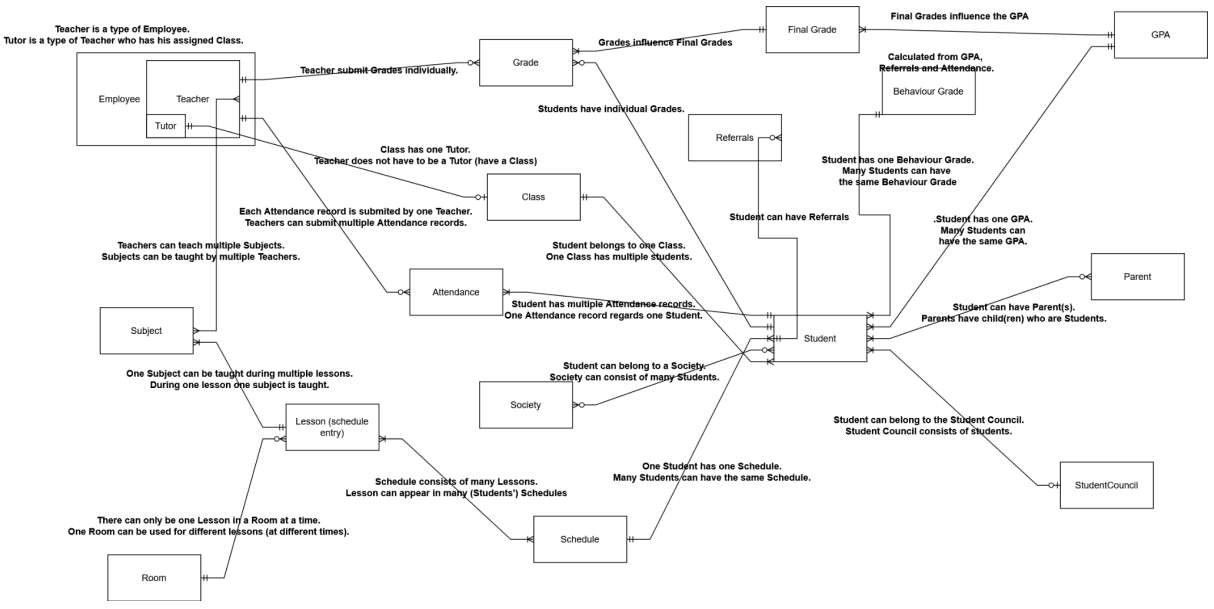
Dopuszczamy również sytuacje, w których uczeń z różnych względów posługuje się na co dzień innym imieniem niż to w dokumentach, stąd dodatkowa rubryka w rejestrze na imię preferowane. We wszystkich oficjalnych sprawozdaniach będzie musiało się pojawiać imię urzędowe, natomiast zdecydowanie otworzy to szkole możliwości na zwiększenie komfortu uczniów.

Nazwy klas składają się typowo z poziomu klasy (liczba 1 odpowiada pierwszej klasie) oraz wielkiej litery alfabetu łacińskiego, które to odróżniają klasy będące na tym samym poziomie. Maksymalna liczba uczniów w klasie wynosi mniej standardowe w publicznym szkolnictwie, lecz korzystne dla efektywnej nauki i integracji, 15 osób.

# Schemat




# Diagram ER



# Tabele

**Tabela StudentInfo** przechowuje dane osobowe oraz związane z uczęszczaniem do szkoły uczniów od momentu zapisania do placówki.

```
CREATE TABLE Students.StudentInfo (
    StudentID INT PRIMARY KEY IDENTITY(0,1),
    PESEL VARCHAR(11),
    FirstName VARCHAR(50) NOT NULL,
    PreferredName VARCHAR(50) NULL,
    LastName VARCHAR(50) NOT NULL,
    DateOfBirth DATE NOT NULL,
    EnrollmentDate DATE NOT NULL,
    Gender CHAR CHECK (Gender IN ('M', 'F', 'N')),
    ClassID INT,
    RegisterNumber INT,
    FOREIGN KEY(ClassID) REFERENCES Academics.Classes(ClassID),
    CONSTRAINT UNIQUE_PESEL UNIQUE (PESEL)
    )
```

**Tabela Parents** przechowuje dane i kontakt do rodziców uczniów uczęszczających do szkoły.

```
CREATE TABLE Students.Parents(
    ParentID INT PRIMARY KEY IDENTITY(0,1),
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    PhoneNumber VARCHAR(10),
    EmailAddress VARCHAR(50),
    )
```

**Tabela StudentParents** przechowuje pary uczeń i jego rodzic. Gdy uczeń ma dwójkę rodziców, tabela zawiera dwa rekordy z jego StudentID. Gdy rodzic ma więcej niż jedno dziecko w szkole, tabela zawiera więcej niż jeden rekord z jego ParentID.

```
CREATE TABLE Students.StudentParents(
    StudentID INT NOT NULL,
    ParentID INT NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
    FOREIGN KEY (ParentID) REFERENCES Students.Parents(ParentID),
    PRIMARY KEY (StudentID, ParentID)
)
```

**Tabela Attendance** umożliwia sprawdzanie przez nauczycieli obecności ucznia w danym dniu i podczas konkretnej lekcji. Uczeń może być obecny, nieobecny, zwolniony (nieobecność usprawiedliwiona) lub spóźniony.

```
CREATE TABLE Students.Attendance (
    AttendanceID INT PRIMARY KEY IDENTITY(0,1),
    StudentID INT NOT NULL,
    AttendanceDate DATE NOT NULL,
    Status VARCHAR(10) CHECK (Status IN ('Present', 'Absent', 'Excused', 'Late')),
    SubjectID INT NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
    FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
    CONSTRAINT UniqueRecords UNIQUE (StudentID, AttendanceDate, SubjectID)
)
```

**Tabela Subjects** zawiera spis przedmiotów nauczanych w szkole.

```
CREATE TABLE Academics.Subjects (
    SubjectID INT PRIMARY KEY IDENTITY(0,1),
    SubjectName VARCHAR(50),
    CONSTRAINT UniqueSubject UNIQUE(SubjectName)
)
```

**Tabela Grades** umożliwia wystawianie przez nauczycieli uczniom ocen cząstkowych w skali 1-6  z danych przedmiotów.

```
CREATE TABLE Students.Grades (
    StudentID INT NOT NULL,
    TeacherID INT NOT NULL,
    SubjectID INT NOT NULL,
    GradeValue INT CHECK (GradeValue >= 1 AND GradeValue <= 6) NOT NULL,
    DateTime DATETIME NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
    FOREIGN KEY (TeacherID) REFERENCES Academics.Teachers(TeacherID),
    FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
    PRIMARY KEY (StudentID, TeacherID, SubjectID, DateTime)
    )
```

**Tabela FinalGrades** umożliwia wystawianie przez nauczycieli ocen końcowych (na podstawie ocen cząstkowych) z danych przedmiotów w skali 1-6 w aktualnym roku szkolnym.

```
CREATE TABLE Students.FinalGrades (
    StudentID INT NOT NULL,
    SubjectID INT NOT NULL,
    SchoolYear INT NOT NULL,
    FinalGrade INT CHECK (FinalGrade >= 1 AND FinalGrade <= 6),
    PRIMARY KEY (StudentID, SubjectID, SchoolYear),
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
    FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
    FOREIGN KEY (SchoolYear) REFERENCES Admins.SchoolYearDates(SchoolYear)
);
```

**Tabela StudentGPA** jest związana z wyliczaniem średniej (GPA) z ocen końcowych danego ucznia.

```
CREATE TABLE Students.StudentGPA (
    StudentID INT PRIMARY KEY,
    GPA DECIMAL(3, 2) NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
    )
```

**Tabela StudentReferrals** umożliwia dokumentowanie zachowania uczniów poprzez wystawianie uwag pozytywnych i negatywnych przez nauczycieli.

```
CREATE TABLE Students.StudentReferrals (
    ReferralID INT PRIMARY KEY IDENTITY(0,1),
    TeacherID INT NOT NULL,
    StudentID INT NOT NULL,
    ReferralDate DATE DEFAULT GETDATE(),
    ReferralText VARCHAR(200) NOT NULL,
    ReferralType CHAR CHECK(ReferralType in ('+', '-')),
    FOREIGN KEY (TeacherID) REFERENCES Academics.Teachers(TeacherID),
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
    )
```

**Tabela BehaviorGrade** jest związana z wystawianiem oceny z zachowania ucznia.

```
CREATE TABLE Students.BehaviorGrade(
    StudentID INT PRIMARY KEY,
    BehaviorGrade VARCHAR(20) DEFAULT 'Good' CHECK (BehaviorGrade IN ('Inadequate', 'Good', 'Outstanding'))
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
);
```

**Tabele StudentCouncil** zawiera aktualny skład samorządu szkolnego.

```
CREATE TABLE Students.StudentCouncil (
    StudentID INT PRIMARY KEY,
    Role VARCHAR(30) CHECK (Role IN ('President', 'Vice President', 'Treasurer', 'Secretary')) NOT NULL,
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
    CONSTRAINT Unique_Council_Role UNIQUE (Role)
    )
```

**Tabela Societies** zawiera spis kół zainteresowań i stowarzyszeń działających w szkole.

```
CREATE TABLE Students.Societies (
    SocietyID INT PRIMARY KEY IDENTITY(0,1),
    SocietyName VARCHAR(50) NOT NULL,
    TutorID INT NOT NULL,
    FOREIGN KEY(TutorID) REFERENCES Academics.Teachers(TeacherID)
    )
```

**Tabela StudentSocieties** zawiera pary uczeń i koło zainteresowań, do którego należy. Uczeń może należeć do wielu kół - wtedy w tabeli znajduje się więcej niż jeden rekord z jego StudentID.

```
CREATE TABLE Students.StudentSocieties
(
    SocietyID INT NOT NULL,
    StudentID INT NOT NULL,
    PRIMARY KEY (SocietyID, StudentID),
    FOREIGN KEY (SocietyID) REFERENCES Students.Societies(SocietyID),
    FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
);
```

**Tabela EmployeeInfo** zawiera dane osobowe, kontaktowe i wysokość wypłaty miesięcznej każdego pracownika szkoły.

```
CREATE TABLE Admins.EmployeeInfo (
    EmployeeID INT PRIMARY KEY IDENTITY(0,1),
    FirstName VARCHAR(50) NOT NULL,
    LastName VARCHAR(50) NOT NULL,
    Salary FLOAT NOT NULL,
    PhoneNumber VARCHAR(12),
    EmailAddress VARCHAR(50),
    )
```

**Tabela Teachers** zawiera identyfikatory pracownicze i nauczycielskie pracowników szkoły, którzy są nauczycielami. Realizuje dziedziczenie - każdy nauczyciel jest pracownikiem, ma szystkie jego atrybuty, oraz ma dodatkowy identyfikator.

```
CREATE TABLE Academics.Teachers (
    TeacherID INT PRIMARY KEY IDENTITY(0,1),
    EmployeeID INT,
    FOREIGN KEY (EmployeeID) REFERENCES Admins.EmployeeInfo(EmployeeID)
    )
```

**Tabela TeacherSubjects** zawiera pary nauczyciel i przedmiot, którego uczy. Jeden nauczyciel może nauczać więcej niż jednego przedmiotu oraz jeden przedmiotu może być nauczany przez wielu nauczycieli.

```
CREATE TABLE Academics.TeacherSubjects (
    TeacherID INT,
    SubjectID INT,
    FOREIGN KEY (TeacherID) REFERENCES Academics.Teachers(TeacherID),
    FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
    PRIMARY KEY(TeacherID, SubjectID)
    )
```

**Tabela Rooms** zawiera spis sal lekcyjnych w budynku szkoły.

```
CREATE TABLE Admins.Rooms (
    RoomNumber INT PRIMARY KEY,
    Capacity INT NOT NULL,
    RoomType VARCHAR(12) CHECK(RoomType IN ('Classroom', 'Gym', 'Auditorium')),
)
```

**Tabela Classes** zawiera spis unikalnych klas (rozumianych tu jako grupy uczniów) w szkole.

```
CREATE TABLE Academics.Classes (
    ClassID INT PRIMARY KEY IDENTITY(0,1),
    ClassLevel INT CHECK (ClassLevel >= 1 AND ClassLevel <= 4) NOT NULL,
    ClassSymbol CHAR CHECK(ClassSymbol >= 'A' AND ClassSymbol <= 'Z'),
    TutorID INT NOT NULL,
    FOREIGN KEY(TutorID) REFERENCES Academics.Teachers(TeacherID)
    CONSTRAINT UniqueClasses UNIQUE (ClassLevel, ClassSymbol);
    )
```

**Tabela ClassSchedule** zawiera harmonogram lekcji odbywających się w szkole. ClassOneID oznacza pojedynczą klasę, która bierze w całości udział w lekcji. Lekcje mogą (ale nie muszą) być łączone między dwoma klasami, stąd drugi identyfikator klasy ClassTwoID. Gdy w lekcji bierze udział tylko jedna klasa, ClassTwoID ma wartość NULL.

```
CREATE TABLE Academics.ClassSchedule (
    ClassOneID INT NOT NULL,
    ClassTwoID INT NULL,
    SubjectID INT,
    TeacherID INT,
    Day DATE,
    StartTime TIME,
    EndTime TIME,
    RoomNumber INT,
    FOREIGN KEY (ClassOneID) REFERENCES Academics.Classes(ClassID),
    FOREIGN KEY (ClassTwoID) REFERENCES Academics.Classes(ClassID),
    FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
    FOREIGN KEY(TeacherID) REFERENCES Academics.Teachers(TeacherID),
    FOREIGN KEY(RoomNumber) REFERENCES Admins.Rooms(RoomNumber),
    PRIMARY KEY (Day, StartTime, EndTime, RoomNumber)
    )
```

**Tabela LuckyRegisterNumber** jest związana z wyliczaniem szczęśliwego numerka (jeden na dzień, stąd konieczny zapis). Szczęśliwy numerek to numer w dzienniku, który ma prawo nie brać udziału w niezapowiedzianych kartkówkach danego dnia.

```
CREATE TABLE Academics.LuckyRegisterNumber (
    Day DATE PRIMARY KEY NOT NULL,
    RegisterNumber INT NOT NULL
)
```

**Tabela SchoolYearDates** zawiera daty początku i końca danego roku szkolnego. W tabeli są lata do aktualnego włącznie. Nie ma w niej przyszłych lat szkolnych.

```
CREATE TABLE Admins.SchoolYearDates (
    SchoolYear INT PRIMARY KEY,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL
)
```

# Funkcje i widoki

## 1. Ranking średnich

**Opis**
Widok generuje ranking uczniów według średniej ocen (GPA) uporządkowanej malejąco.

**Kod**

```
CREATE VIEW Students.RankingGPA AS (
    SELECT ROW_NUMBER() OVER (ORDER BY G.GPA DESC) AS Rank, S.FirstName, S.LastName, G.GPA
    FROM Students.StudentInfo S
    JOIN Students.StudentGPA G ON G.StudentID = S.StudentID
)
```

**Typowe zapytanie**
SELECT * FROM Students.RankingGPA

## 2. Uczniowie w danej klasie z kontaktem do rodzica

**Opis**
Funkcja zwraca listę uczniów w podanej klasie, zawierającą dane osobowe oraz numery kontaktowe rodziców w formie tabeli. Uczniowie posortowani są po numerze z dziennika rosnąco.

**Kod**

```
CREATE FUNCTION Students.GetClassStudents (@ClassID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT ROW_NUMBER() OVER (ORDER BY S.RegisterNumber ASC) AS RegisterNumber,
    S.FirstName, S.LastName, S.PESEL, S.DateOfBirth, S.Gender,
    P1.PhoneNumber AS 'Parent Contact 1', P2.PhoneNumber AS 'Parent Contact 2'
    FROM Students.StudentInfo S
    LEFT JOIN Students.StudentParents SP ON S.StudentID = SP.StudentID
    LEFT JOIN Students.Parents P1 ON SP.ParentID = P1.ParentID
    LEFT JOIN Students.Parents P2 ON SP.ParentID = P2.ParentID
    WHERE S.ClassID = @ClassID

);
```

**Typowe zapytanie**
SELECT * FROM Students.GetClassStudents(1)

## 3. Oceny ucznia (cząstkowe) z danego przedmiotu i roku

**Opis**
Funkcja zwraca oceny cząstkowe podanego ucznia z danego przedmiotu i w określonym roku szkolnym w formie tabeli.

**Kod**

```
CREATE FUNCTION Students.GetStudentGradesForSubjectAndYear (
    @StudentID INT,
    @SubjectID INT,
    @SchoolYear INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        G.StudentID,
        G.SubjectID,
        G.GradeValue,
        G.DateTime
    FROM
        Students.Grades G
    JOIN
        Academics.Subjects S ON G.SubjectID = S.SubjectID
    JOIN
        Admins.SchoolYearDates SY ON SY.SchoolYear = @SchoolYear
    WHERE
        G.StudentID = @StudentID
        AND G.SubjectID = @SubjectID
        AND (G.DateTime BETWEEN SY.StartDate AND SY.EndDate)
);
```

**Typowe zapytanie**
SELECT * FROM Students.GetStudentGradesForSubjectAndYear(34,0,2024)

## 4. Oceny ucznia (finalne) z danego roku

**Opis**
Funkcja zwraca oceny końcowe ucznia z danego roku szkolnego w formie tabeli.

**Kod**

```
CREATE FUNCTION Students.GetStudentFinalGradesForYear (
    @StudentID INT,
    @SchoolYear INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        F.StudentID,
        F.SubjectID,
        F.FinalGrade
    FROM
        Students.FinalGrades F
    WHERE
        F.StudentID = @StudentID
        AND F.SchoolYear = @SchoolYear
);
```

**Przykładowe zapytanie**
SELECT * FROM Students.GetStudentFinalGradesForYear(0,2024);

## 5. Ranking frekwencji

**Opis**
Widok wyświetla ranking uczniów według liczby nieobecności w bieżącym roku szkolnym - od najmniejszej do największej.

**Kod**

```
CREATE VIEW Students.AttendanceRanking AS
SELECT
    s.StudentID,
    s.FirstName,
    s.LastName,
    c.ClassLevel,
    c.ClassSymbol,
    COUNT(a.AttendanceID) AS TotalAbsences
FROM Students.StudentInfo s
JOIN Academics.Classes c ON s.ClassID = c.ClassID
LEFT JOIN Students.Attendance a ON s.StudentID = a.StudentID AND a.Status = 'Absent'
    WHERE EXISTS (
SELECT 1
        FROM Admins.CurrentSchoolYear sy
        WHERE a.AttendanceDate BETWEEN sy.StartDate AND sy.EndDate
    )

    GROUP BY s.StudentID, s.FirstName, s.LastName, c.ClassLevel, c.ClassSymbol
    ORDER BY TotalAbsences ASC; --najmniej nieobecności na górze
```

## 6. Uczniowie mający co najmniej jedno zagrożenie

**Opis**
Widok wyświetla listę uczniów zagrożonych niezdaniem, czyli takich mających co najmniej jedną ocenę końcową niedostateczną.

**Kod**

```
CREATE VIEW Students.StudentsWithFinalGrade1 AS
    SELECT DISTINCT
        S.StudentID,
        S.FirstName,
        S.LastName
    FROM Students.StudentInfo S
    JOIN
        Students.FinalGrades F ON S.StudentID = F.StudentID
    WHERE
        F.FinalGrade = 1 AND F.SchoolYear = Admins.GetCurrentSchoolYear();
```

**Przykładowe zapytanie**
SELECT * FROM Students.StudentsWithFinalGrade1;

## 7. Plan (tygodniowy) lekcji dla ucznia

**Opis**
Funkcja zwraca w formie tabeli tygodniowy plan zajęć ucznia z uwzględnieniem przedmiotów, nauczycieli, godzin zajęć oraz sal lekcyjnych.

**Kod**

```
CREATE FUNCTION Students.GetStudentSchedule (@StudentID INT)
RETURNS TABLE
AS
    RETURN
    (
        SELECT si.StudentID,
        si.FirstName, si.LastName, cs.Day, cs.StartTime, cs.EndTime, s.SubjectName,
        e.FirstName + ' ' + e.LastName AS TeacherName, cs.RoomNumber
        FROM Academics.ClassSchedule cs
        JOIN Academics.Subjects s ON cs.SubjectID = s.SubjectID
        JOIN Academics.Teachers t ON cs.TeacherID = t.TeacherID
        JOIN Admins.EmployeeInfo e ON t.EmployeeID = e.EmployeeID
        JOIN Students.StudentInfo si ON cs.ClassOneID = si.ClassID OR si.ClassID = cs.ClassTwoID
        WHERE si.StudentID = @StudentID
    );
```

**Przykładowe zapytanie**
SELECT * FROM Students.GetStudentSchedule(0)

## 8. Oblicz kolejny wolny numerek w dzienniku (do dodawania ucznia do klasy)

**Opis**
Funkcja pomocnicza do dodawania ucznia do klasy - oblicza i zwraca pierwszy dostępny numer w dzienniku dla nowego ucznia w danej klasie.

**Kod**

```
CREATE FUNCTION Students.GetNextRegisterNumber (@ClassID INT)
RETURNS INT
AS
BEGIN
    DECLARE @NextRegisterNumber INT;

    IF NOT EXISTS (SELECT 1 FROM Academics.Classes WHERE ClassID = @ClassID)
        BEGIN
            /* The class does not exist*/
        RETURN NULL;
        END

    SELECT TOP 1 @NextRegisterNumber = RegisterNumber FROM (
        VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (11), (12), (13), (14), (15)
        )
    AS AvailableNumbers (RegisterNumber)
    WHERE RegisterNumber NOT IN (
        SELECT RegisterNumber FROM Students.StudentInfo WHERE ClassID = @ClassID AND RegisterNumber IS NOT NULL
    )

    ORDER BY RegisterNumber ASC;
    RETURN @NextRegisterNumber;

END;
```

**Przykładowe zapytanie**
SELECT Students.GetNextRegisterNumber(1)

## 9. Klasy i ich wychowawcy

**Opis**
Widok pokazuje klasy wraz z ich wychowawcami, kontaktem do wychowawcy oraz liczbą uczniów.

**Kod**

```
CREATE VIEW Academics.ClassesAndTutors AS (
    SELECT DISTINCT CONVERT(VARCHAR(1), C.ClassLevel) + ' ' +  C.ClassSymbol AS 'Class',
    E.FirstName + ' ' + E.LastName AS 'Tutor', E.PhoneNumber AS 'Tutor Contact',
    COUNT(S.StudentID) AS 'Number Of Students'
    FROM Academics.Classes C
    JOIN Academics.Teachers T ON C.TutorID = T.TeacherID
    JOIN Admins.EmployeeInfo E ON E.EmployeeID = T.EmployeeID
    JOIN Students.StudentInfo S ON C.ClassID = S.ClassID
    GROUP BY C.ClassID, C.ClassLevel, C.ClassSymbol, E.FirstName, E.LastName, E.PhoneNumber
)
```

**Przykładowe zapytanie**
SELECT * FROM Academics.ClassesAndTutors

## 10. Nauczyciele, którzy nie są wychowawcami

**Opis**
Widok pokazuje listę nauczycieli, którzy nie pełnią funkcji wychowawcy.

**Kod**

```
CREATE VIEW Academics.TeachersNotTutors AS
    SELECT T.TeacherID, E.FirstName, E.LastName, E.PhoneNumber, E.EmailAddress
    FROM Academics.Teachers T
    JOIN Admins.EmployeeInfo E ON T.EmployeeID = E.EmployeeID
    LEFT JOIN Academics.Classes C ON T.TeacherID = C.TutorID
    WHERE C.ClassID IS NULL;
```

**Przykładowe zapytanie**
SELECT * FROM Academics.TeachersNotTutors

## 11. Obecny rok szkolny

**Opis**
Funkcja pomocnicza zwraca z tabeli bieżący rok szkolny jako liczbę.

**Kod**

```
CREATE FUNCTION Admins.GetCurrentSchoolYear()
RETURNS INT
AS
BEGIN
    DECLARE @CurrentSchoolYear INT;
    SELECT @CurrentSchoolYear = MAX(SchoolYear)
    FROM SchoolYearDates;

    RETURN @CurrentSchoolYear;
END;
```

**Przykładowe zapytanie**
SELECT Admins.GetCurrentSchoolYear()

# Procedury składowane

## 1. Dodawanie nowego ucznia

### Opis

Procedura dodaje nowego ucznia po jego danych do rejestru uczniów. Na razie uczeń nie jest przypisany do żadnej klasy.

### Kod

```
CREATE PROCEDURE Students.AddNewStudent
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @PESEL VARCHAR(11),
    @DateOfBirth DATE,
    @Gender CHAR
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (SELECT 1 FROM Students.StudentInfo WHERE PESEL = @PESEL)
    BEGIN

        RAISERROR('Student with the provided PESEL already exists in the database.', 16, 1);

        RETURN;
    END

    INSERT INTO Students.StudentInfo
    (PESEL, FirstName, PreferredName, LastName, DateOfBirth, EnrollmentDate, Gender, ClassID, RegisterNumber)
    VALUES (@PESEL, @FirstName, NULL, @LastName, @DateOfBirth, GETDATE(), @Gender, NULL, NULL);

END;
```

### Przykładowe zapytanie

```
EXEC Students.AddNewStudent
     @FirstName = 'Natalie',
     @LastName = 'Monet',
     @PESEL = '03120903446',
     @DateOfBirth = '2003-12-09',
     @Gender = 'F';
```

## 2. Dodawanie ucznia do klasy

### Opis

Procedura przypisuje danego ucznia do klasy oraz przyporządkowuje mu numerek w dzienniku w obrębie tej klasy.

### Kod

```
CREATE PROCEDURE Students.AddStudentToClass
    @StudentID INT,
    @ClassID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('Student does not exist.', 16, 1);
            RETURN;
    END;
```

```
    IF NOT EXISTS (SELECT 1 FROM Academics.Classes WHERE ClassID = @ClassID)
    BEGIN
        RAISERROR('Class does not exist.', 16, 1);
        RETURN;
    END;

    IF EXISTS (SELECT 1 FROM Students.StudentInfo WHERE ClassID = @ClassID AND StudentID = @StudentID)
    BEGIN
        RAISERROR('Student already in this class', 16, 1);
        RETURN;
    END;

    DECLARE @RegisterNumber INT;
    SET @RegisterNumber = Students.GetNextRegisterNumber(@ClassID);


      UPDATE Students.StudentInfo
      SET ClassID = @ClassID, RegisterNumber = @RegisterNumber
      WHERE StudentID = @StudentID;

END;
```

**Przykładowe zapytanie**

EXEC Students.AddStudentToClass
@StudentID = 7,
@ClassID = 1

## 3. Wpisywanie oceny uczniowi

**Opis**

Procedura wpisuje daną ocenę cząstkową z przdmiotu zadanemu uczniowi.

**Kod**

```
CREATE PROCEDURE Students.AddStudentGrade
    @StudentID INT,
    @TeacherID INT,
    @GradeValue INT,
    @SubjectID INT
AS
BEGIN
    SET NOCOUNT ON;


    IF NOT EXISTS (SELECT 1 FROM Academics.Subjects WHERE SubjectID = @SubjectID)
    BEGIN
        RAISERROR('Subject does not exist', 16, 1);
            RETURN;
    END;

    IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('Student does not exist', 16, 1);
            RETURN;
    END;

    IF NOT EXISTS (SELECT 1 FROM Academics.Teachers WHERE TeacherID = @TeacherID)
    BEGIN
        RAISERROR('Teacher does not exist', 16, 1);
            RETURN;
    END;

    IF NOT EXISTS (SELECT 1 FROM Academics.TeacherSubjects WHERE TeacherID = @TeacherID AND SubjectID = @SubjectID)
    BEGIN
        RAISERROR('Teacher does not teach the subject', 16, 1);
        RETURN;
    END;

    INSERT INTO Students.Grades (StudentID, TeacherID, SubjectID, GradeValue, DateTime)
    VALUES (@StudentID, @TeacherID, @SubjectID, @GradeValue, GETDATE());

END;
```

**Przykładowe zapytanie**

EXEC Students.AddStudentGrade
@StudentID = 0,
@TeacherID = 0,
@GradeValue = 3,
@SubjectID = 0

## 4. Wpisywanie uwag uczniowi

**Opis**

Procedura dodaje pozytywną bądź negatywną uwagę do historii ucznia wraz z informacją o dacie i nauczycielu wystawiającym.

**Kod**

```
CREATE PROCEDURE Students.AddStudentReferral
    @StudentID INT,
    @TeacherID INT,
    @ReferralText VARCHAR(500),
    @ReferralType CHAR
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('Student does not exist', 16, 1);
        RETURN;
    END;

    IF NOT EXISTS (SELECT 1 FROM Academics.Teachers WHERE TeacherID = @TeacherID)
    BEGIN
        RAISERROR('Teacher does not exist', 16, 1);
            RETURN;
    END;

    INSERT INTO Students.StudentReferrals (TeacherID, StudentID, ReferralDate, ReferralText, ReferralType)
    VALUES (@TeacherID, @StudentID, GETDATE(), @ReferralText, @ReferralType);

END;
```

**Przykładowe zapytanie**

EXEC Students.AddStudentReferral
@TeacherID = 34,
@StudentID = 45;
@ReferralText = ‘uczeń nie odrobił pracy domowej po raz 10 :p’,
@ReferralType = ‘-’;

## 5. Usuwanie ucznia z klasy

**Opis**
Usuwa danego ucznia z klasy, ustawiając identyfikator klasy i numer w dzienniku na NULL oraz aktualizując numery w dzienniku oryginalnie następujące po nim na liście w celu zachowania ciągłości numeracji.

**Kod**

```
CREATE PROCEDURE Students.RemoveStudentFromClass
    @StudentID INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ClassID INT, @RegisterNumber INT;

    BEGIN TRANSACTION;

    IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('Student does not exist', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    SELECT @ClassID = ClassID, @RegisterNumber = RegisterNumber
    FROM Students.StudentInfo
    WHERE StudentID = @StudentID;

    IF @ClassID IS NULL
    BEGIN
        RAISERROR('Student is not assigned to any class.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;


     UPDATE Students.StudentInfo
     SET ClassID = NULL, RegisterNumber = NULL
     WHERE StudentID = @StudentID;

     UPDATE Students.StudentInfo
     SET RegisterNumber = RegisterNumber - 1
     WHERE ClassID = @ClassID AND RegisterNumber > @RegisterNumber;

     COMMIT TRANSACTION;

END;
```

**Przykładowe zapytanie**
EXEC Students.RemoveStudentFromClass @StudentID = 12;

## 6. Szczęśliwy numerek

**Opis**
Procedura losuje tzw. szczęśliwy numerek (objaśnienie przy tabeli LuckyRegisterNumber) z puli 1-15 (tylu uczniów może być w klasie). Zapisuje w tabeli szczęśliwy numerek z datą dzienną - jeśli już był losowany szczęśliwy numerek tego dnia, jest on tylko wyświetlany z daną datą. Jeśli nie - jest losowany po raz pierwszy.

### Kod

```
CREATE PROCEDURE Academics.GetLuckyRegisterNumber
AS
BEGIN
    DECLARE @LuckyRegisterNumber INT;

    IF NOT EXISTS (SELECT 1 FROM Academics.LuckyRegisterNumber WHERE CAST(Day AS DATE) = CAST(GETDATE() AS DATE))
        BEGIN
            SET @LuckyRegisterNumber = FLOOR(RAND() * 15) + 1;
            INSERT INTO Academics.LuckyRegisterNumber (Day, RegisterNumber) VALUES (GETDATE(), @LuckyRegisterNumber);
        END;

    SELECT RegisterNumber AS 'The Lucky Register Number for today'
    FROM Academics.LuckyRegisterNumber
    WHERE CAST(Day AS DATE) = CAST(GETDATE() AS DATE);

END;
```

### Przykładowe zapytanie

EXEC Academics.GetLuckyRegisterNumber

## 7. Usunięcie ucznia ze szkoły

### Opis

Procedura usuwa ucznia ze szkoły, a więc i wszystkich tabel, które zawierały informacje na jego temat (ktoś niebędący uczniem nie może mieć ocen czy należeć do kół zainteresowań).

### Kod

```
CREATE PROCEDURE Students.RemoveStudent
    @StudentID INT
AS
BEGIN

    IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('Student does not exist', 16, 1);
        RETURN;
    END

    BEGIN TRANSACTION;

    BEGIN TRY

        DELETE FROM Students.StudentCouncil WHERE StudentID = @StudentID;
        DELETE FROM Students.StudentSocieties WHERE StudentID = @StudentID;
        DELETE FROM Students.Grades WHERE StudentID = @StudentID;
        DELETE FROM Students.BehaviorGrade WHERE StudentID = @StudentID;
        DELETE FROM Students.StudentReferrals WHERE StudentID = @StudentID;
        DELETE FROM Students.Attendance WHERE StudentID = @StudentID;

        DECLARE @ParentID INT;
        SELECT @ParentID = ParentID FROM Students.StudentParents WHERE StudentID = @StudentID;

        DELETE FROM Students.StudentParents WHERE StudentID = @StudentID;
```

```
        IF NOT EXISTS (SELECT 1 FROM Students.StudentParents WHERE ParentID = @ParentID)
        BEGIN
            DELETE FROM Students.Parents WHERE ParentID = @ParentID;
        END

        DELETE FROM Students.StudentInfo WHERE StudentID = @StudentID;

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        ROLLBACK TRANSACTION;
    END CATCH
END;
```

**Przykładowe zapytanie**
EXEC Students.RemoveStudent

## 8. Generowanie transkryptu ucznia

**Opis**
Procedura generuje transkrypt studenta, zawierający oceny wraz z informacją o nauczycielu wystawiającym, datą i średnią ocen.

**Kod**

```
CREATE PROCEDURE Students.GenerateStudentTranscript
    @StudentID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('Student does not exist', 16, 1);
        RETURN;
    END;

    DECLARE @PESEL VARCHAR(11), @FirstName VARCHAR(50), LastName VARCHAR(50);

SELECT @FirstName = FirstName,
    @LastName = LastName,
    @PESEL = PESEL
FROM Students.StudentInfo
WHERE StudentID = @StudentID;

    SELECT
        s.SubjectName AS 'Subject',
        t.FirstName + ' ' + t.LastName AS 'Teacher',
        g.GradeValue AS 'Grade',
        g.DateTime AS 'Date'
    FROM Students.Grades g
```

```
    JOIN Academics.Subjects s ON g.SubjectID = s.SubjectID
    JOIN Academics.Teachers t ON g.TeacherID = t.TeacherID
    WHERE g.StudentID = @StudentID
    ORDER BY s.SubjectName, g.DateTime;

SELECT
    s.SubjectName AS 'Subject',
    ROUND(AVG(CAST(g.GradeValue AS FLOAT)), 2) AS 'Average grade'
FROM Students.Grades g
JOIN Academics.Subjects s ON g.SubjectID = s.SubjectID
WHERE g.StudentID = @StudentID
GROUP BY s.SubjectName;

END;
```

**Przykładowe zapytanie**
EXEC Students.GenerateTranscript @StudentID = 34;

## 9. Wpisywanie ocen końcowych

**Opis**
Procedura oblicza i zapisuje oceny końcowe ucznia na podstawie średnich ocen z poszczególnych przedmiotów w danym roku szkolnym.

**Kod**

```
CREATE PROCEDURE Students.CalculateFinalGrades
    @StudentID INT,
    @SchoolYear INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @AverageGrade DECIMAL(3, 2);
    DECLARE @FinalGrade INT;
    DECLARE @StartDate DATE;
    DECLARE @EndDate DATE;
    DECLARE @SubjectID INT;

    IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID = @StudentID)
    BEGIN
        RAISERROR('Student does not exist', 16, 1);
        RETURN;
    END

    SELECT @StartDate = StartDate, @EndDate = EndDate
    FROM Admins.SchoolYearDates
    WHERE SchoolYear = @SchoolYear;

    IF @StartDate IS NULL OR @EndDate IS NULL
    BEGIN
        RETURN;
    END
```

```
    BEGIN TRANSACTION;

    DECLARE SubjectCursor CURSOR FOR
        SELECT SubjectID
        FROM Students.Grades
        WHERE StudentID = @StudentID;

    OPEN SubjectCursor;
    FETCH NEXT FROM SubjectCursor INTO @SubjectID;

    WHILE @@FETCH_STATUS = 0
    BEGIN

        SELECT @AverageGrade = AVG(GradeValue)
        FROM Students.Grades
        WHERE StudentID = @StudentID
            AND SubjectID = @SubjectID
            AND DateTime BETWEEN @StartDate AND @EndDate;

        IF @AverageGrade IS NULL
        BEGIN

            FETCH NEXT FROM SubjectCursor INTO @SubjectID;
            CONTINUE;
        END

      SET @FinalGrade = CASE
          WHEN @AverageGrade < 1.5 THEN 1
          WHEN @AverageGrade < 2.5 THEN 2
          WHEN @AverageGrade < 3.5 THEN 3
          WHEN @AverageGrade < 4.5 THEN 4
          WHEN @AverageGrade < 5.5 THEN 5
          WHEN @AverageGrade >= 5.5 THEN 6
          ELSE NULL
      END;

      IF EXISTS (SELECT 1 FROM Students.FinalGrades WHERE StudentID = @StudentID AND SchoolYear = @SchoolYear)
      BEGIN
          UPDATE Students.FinalGrades
          SET FinalGrade = @FinalGrade
          WHERE StudentID = @StudentID AND SchoolYear = @SchoolYear AND SubjectID = @SubjectID;
      END
      ELSE
      BEGIN
          INSERT INTO Students.FinalGrades (StudentID, SchoolYear, SubjectID, FinalGrade)
          VALUES (@StudentID, @SchoolYear, @SubjectID, @FinalGrade);
      END

      FETCH NEXT FROM SubjectCursor INTO @SubjectID;
  END

   CLOSE SubjectCursor;
   DEALLOCATE SubjectCursor;
   COMMIT TRANSACTION;

END;
```

**Przykładowe zapytanie**

EXEC Students.CalculateFinalGrades @StudentID = 34, @SchoolYear = 2024;

## 10. Wpisywanie oceny z zachowania

**Opis**

Procedura dodaje oceny z zachowania - automatycznie każdy uczeń ma ocenę dobrą (‘Good’) i w zależności od uwag (pozytywnych/negatywnych), frekwencji oraz średniej ocen ocena z zachowania może zmienić się na odpowiednio bardzo dobrą, niedostateczną (outstanding lub inadequate) lub pozostać bez zmian.

**Kod**

```
CREATE PROCEDURE Students.UpdateBehaviorGrade
    @StudentID INT
AS
BEGIN
    DECLARE @GPA DECIMAL(3,2);
    DECLARE @PositiveReferrals INT;
    DECLARE @NegativeReferrals INT;
    DECLARE @Absences INT;
    DECLARE @NewBehaviorGrade VARCHAR(20) = 'Good';

    SELECT @GPA = GPA
FROM Students.StudentGPA
WHERE StudentID = @StudentID;

SELECT @PositiveReferrals = COUNT(*)
FROM Students.StudentReferrals
WHERE StudentID = @StudentID AND ReferralType = '+';

SELECT @NegativeReferrals = COUNT(*)
FROM Students.StudentReferrals
WHERE StudentID = @StudentID AND ReferralType = '-';

SELECT @Absences = COUNT(*)
FROM Students.Attendance
WHERE StudentID = @StudentID AND AttendanceStatus = 'Absent';

--logika do zmiany oceny - gpa>4.0 pozytywne uwagi >= 3 nieobecności < 10
--gpa < 3.0 negatywne uwagi >= 3 nieobecności >= 30
IF @GPA IS NOT NULL AND @GPA > 4.0 AND @PositiveReferrals >= 3 AND @Absences < 10
    SET @NewBehaviorGrade = 'Outstanding';
ELSE IF @GPA IS NOT NULL AND @GPA < 3.0 AND @NegativeReferrals >= 3 AND @Absences > 30
    SET @NewBehaviorGrade = 'Inadequate';

UPDATE Students.BehaviorGrade
SET BehaviorGrade = @NewBehaviorGrade
WHERE StudentID = @StudentID;

END;
```

**Przykładowe zapytanie**

EXEC Students.UpdateBehaviorGrade @StudentID =

## 11. Dodawanie nauczyciela

**Opis**

Procedura dodaje nauczyciela - zarówno do tabeli pracowników (EmployeeInfo), jak i nauczycieli (Teachers).

**Kod**

```
CREATE PROCEDURE Admins.AddTeacher
    @FirstName VARCHAR(50),
    @LastName VARCHAR(50),
    @Salary FLOAT,
    @PhoneNumber VARCHAR(12) = NULL,
    @EmailAddress VARCHAR(50) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewEmployeeID INT;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO Admins.EmployeeInfo (FirstName, LastName, Salary, PhoneNumber, EmailAddress)
        VALUES (@FirstName, @LastName, @Salary, @PhoneNumber, @EmailAddress);

        SET @NewEmployeeID = SCOPE_IDENTITY();

        INSERT INTO Academics.Teachers(EmployeeID)
        VALUES (@NewEmployeeID);

        COMMIT TRANSACTION;
    END TRY

    BEGIN CATCH
        ROLLBACK TRANSACTION;
        RAISERROR('Error occurred while adding a new teacher', 16, 1);
    END CATCH;
END;
```

**Przykładowe zapytanie**

EXEC Admins.AddTeacher
@FirstName = ‘Jan’,
@LastName = ‘Kowalski’,
@Salary = 5555.01,
@PhoneNumber = ‘112233445’,
@EmailAddress = ‘jan.kowalski@mail.pl’;

# Wyzwalacze

## 1. Automatyczna aktualizacja średniej ocen

**Opis**
Wyzwalacz oblicza i aktualizuje średnią ocen ucznia, gdy zostanie dodana nowa ocena końcowa.

**Kod**

```
CREATE TRIGGER Students.TrackGPA ON Students.FinalGrades
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @StudentID INT;
    DECLARE @CurrentSchoolYear INT;
    DECLARE @UpdatedGPA DECIMAL (3,2);

    SET @CurrentSchoolYear = Admins.GetCurrentSchoolYear();

    IF EXISTS (SELECT 1 FROM inserted)
        BEGIN
            SELECT @StudentID = StudentID FROM inserted;
        END

    ELSE IF EXISTS (SELECT 1 FROM deleted)
        BEGIN
            SELECT @StudentID = StudentID FROM deleted;
        END

    SELECT @UpdatedGPA = AVG(CAST(FinalGrade AS DECIMAL (3,2))) FROM Students.FinalGrades
    WHERE StudentID = @StudentID AND SchoolYear = @CurrentSchoolYear;

    IF EXISTS (SELECT 1 FROM Students.StudentGPA WHERE StudentID = @StudentID)
    BEGIN
        UPDATE Students.StudentGPA SET GPA = @UpdatedGPA WHERE StudentID = @StudentID
        RETURN;
    END

     INSERT INTO Students.StudentGPA (StudentID, GPA) VALUES (@StudentID, @UpdatedGPA)

END;
```

## 2. Zapobieganie przepełnieniu klas

**Opis**
W jednej klasie może być maksymalnie 15 osób. Jeśli dana klasa osiągnie ten limit, nie można już dopisać kolejnego ucznia.

**Kod**

```
CREATE TRIGGER Students.MaxStudentsInClass ON Students.StudentInfo
AFTER UPDATE
AS
BEGIN
    DECLARE @StudentCount INT;
    DECLARE @ClassID INT;

    SELECT @ClassID = ClassID FROM inserted;

    IF @ClassID IS NOT NULL
        BEGIN
            SELECT @StudentCount = COUNT(*) FROM Students.StudentInfo WHERE ClassID = @ClassID;

            IF @StudentCount >= 15
                BEGIN
                    RAISERROR('Class already full. Cannot add more students to this class.', 16, 1);
                    ROLLBACK;
                END
        END
END;
```

## 3. Zapobieganie usuwaniu klas z zapisanymi uczniami

**Opis**

Wyzwalacz zapobiega usunięciu instancji klasy, gdy jest do niej zapisany co najmniej jeden uczeń.

**Kod**

```
CREATE TRIGGER PreventClassDeletionWithStudents ON Academics.Classes
FOR DELETE
AS
BEGIN
    DECLARE @ClassID INT;
    SELECT @ClassID = ClassID FROM DELETED;

    IF EXISTS (SELECT 1 FROM Students.StudentInfo WHERE ClassID = @ClassID)
    BEGIN
        RAISERROR('Deleting classes with students enrolled is not allowed.', 16, 1);
        ROLLBACK;
    END
END;
```

## 4. Zapobieganie wpisywaniu przyszłej obecności

**Opis**

Wyzwalacz zapobiega wpisaniu błędnej obecności - duplikatu lub o przyszłej dacie.

**Kod**

```
CREATE TRIGGER Students.PreventFutureAttendance
ON Students.Attendance
AFTER INSERT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM inserted WHERE AttendanceDate > GETDATE())
    BEGIN
        RAISERROR('Adding attendance from the future is not allowed', 16, 1);
        ROLLBACK;
        RETURN;
    END;
END;
```

## 5. Automatyczna aktualizacja oceny z zachowania

**Opis**

Wyzwalacze zaktualizują ocenę z zachowania ucznia po dodaniu nowej oceny, uwagi lub po wpisie nieobecności.

**Kod**

```
CREATE TRIGGER Students.trg_UpdateBehaviorGrade_Referrals
ON Students.StudentReferrals
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @StudentID INT;

    DECLARE cur CURSOR FOR
    SELECT DISTINCT StudentID FROM inserted
    UNION
    SELECT DISTINCT StudentID FROM deleted;

    OPEN cur;
    FETCH NEXT FROM cur INTO @StudentID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC Students.UpdateBehaviorGrade @StudentID;
        FETCH NEXT FROM cur INTO @StudentID;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
GO
```

```
CREATE TRIGGER Students.trg_UpdateBehaviorGrade_Grades
ON Students.Grades
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @StudentID INT;

    DECLARE cur CURSOR FOR
    SELECT DISTINCT StudentID FROM inserted
    UNION
    SELECT DISTINCT StudentID FROM deleted;

    OPEN cur;
    FETCH NEXT FROM cur INTO @StudentID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC Students.UpdateBehaviorGrade @StudentID;
        FETCH NEXT FROM cur INTO @StudentID;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
GO
```

```
CREATE TRIGGER Students.trg_UpdateBehaviorGrade_Attendance
ON Students.Attendance
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
    DECLARE @StudentID INT;

    DECLARE cur CURSOR FOR
    SELECT DISTINCT StudentID FROM inserted
    UNION
    SELECT DISTINCT StudentID FROM deleted;

    OPEN cur;
    FETCH NEXT FROM cur INTO @StudentID;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC Students.UpdateBehaviorGrade @StudentID;
        FETCH NEXT FROM cur INTO @StudentID;
    END;

    CLOSE cur;
    DEALLOCATE cur;
END;
GO
```

# Strategia pielęgnacji bazy danych

W szkole, instytucji kluczowej dla dalszego rozwoju i kariery swoich wychowanków, gdzie gromadzone są ważne informacje o uczniach, nauczycielach i ocenach, kluczowe jest dbanie o bezpieczeństwo danych oraz możliwość ich szybkiego odzyskania w razie awarii. Aby zapewnić ciągłość dostępu do dziennika elektronicznego, warto stosować zarówno kopie pełne, jak i różnicowe. Szczególną uwagę należy zwrócić na częstotliwość aktualizacji danych dotyczących frekwencji czy wyników sprawdzianów, ponieważ to one ze względu na tempo zmian wymagają najczęstszych kopii zapasowych. Ponadto istotne jest określenie okresu przechowywania danych archiwalnych, zgodnie z aktualnymi przepisami dotyczącymi ochrony danych. Oprócz automatycznego tworzenia kopii zapasowych, warto regularnie testować procesy przywracania, aby mieć pewność, że w razie problemów dane zostaną skutecznie odtworzone.

Nie można przeoczyć faktu, że szkoły w dalszym ciągu są zobowiązane do trzymania ocenionych papierowych sprawdzianów czy wypracowań. To stanowi ostatnią deskę ratunku w razie, gdy powyższe plany zawiodą. W przypadku utracenia danych dotyczących ocen są one odtwarzalne z papierowych dokumentów, choć jest to zadanie czasochłonne, które należy stosować jedynie w ostateczności.

# Skrypt tworzący bazę danych

```
/* Krok 1 */
CREATE DATABASE SchoolDB
USE SchoolDB

/* Krok 2-4 */
/* tworzenie schematów - po jednym, osobno przed tabelami */
CREATE SCHEMA Students;
CREATE SCHEMA Academics;
CREATE SCHEMA Admins;

/* Krok 5  - tabele */
CREATE TABLE Admins.EmployeeInfo (
	EmployeeID INT PRIMARY KEY IDENTITY(0,1),
	FirstName VARCHAR(50) NOT NULL,
	LastName VARCHAR(50) NOT NULL,
	Salary FLOAT NOT NULL,
	PhoneNumber VARCHAR(12),
	EmailAddress VARCHAR(50),
	)

CREATE TABLE Academics.Teachers (
	TeacherID INT PRIMARY KEY IDENTITY(0,1),
	EmployeeID INT,
	FOREIGN KEY (EmployeeID) REFERENCES Admins.EmployeeInfo(EmployeeID)
	)

CREATE TABLE Academics.Subjects (
	SubjectID INT PRIMARY KEY IDENTITY(0,1),
	SubjectName  VARCHAR(50),
CONSTRAINT UniqueSubject UNIQUE(SubjectName)
	)

CREATE TABLE Academics.Classes (
	ClassID INT PRIMARY KEY IDENTITY(0,1),
	ClassLevel INT CHECK (ClassLevel >= 1 AND ClassLevel <= 4) NOT NULL,
	ClassSymbol CHAR CHECK(ClassSymbol >= 'A' AND ClassSymbol <= 'Z'),
	TutorID INT NOT NULL,
	FOREIGN KEY(TutorID) REFERENCES Academics.Teachers(TeacherID),
	CONSTRAINT UniqueClasses UNIQUE (ClassLevel, ClassSymbol)
	)

CREATE TABLE Students.StudentInfo (
	StudentID INT PRIMARY KEY IDENTITY(0,1),
	PESEL VARCHAR(11),
```

```
	FirstName VARCHAR(50) NOT NULL,
	PreferredName VARCHAR(50) NULL,
	LastName VARCHAR(50) NOT NULL,
	DateOfBirth DATE NOT NULL,
	EnrollmentDate DATE NOT NULL,
	Gender CHAR CHECK (Gender IN ('M', 'F', 'N')),
	ClassID INT,
	RegisterNumber INT,
	FOREIGN KEY(ClassID) REFERENCES Academics.Classes(ClassID),
	CONSTRAINT UNIQUE_PESEL UNIQUE (PESEL)
	)

CREATE TABLE Students.StudentGPA (
	StudentID INT PRIMARY KEY,
	GPA DECIMAL(3, 2) NOT NULL,
	FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
	)

CREATE TABLE Students.StudentReferrals (
	ReferralID INT PRIMARY KEY IDENTITY(0,1),
	TeacherID INT NOT NULL,
	StudentID INT NOT NULL,
	ReferralDate DATE DEFAULT GETDATE(),
	ReferralText VARCHAR(200) NOT NULL,
	ReferralType CHAR CHECK(ReferralType in ('+', '-')),
	FOREIGN KEY (TeacherID) REFERENCES Academics.Teachers(TeacherID),
	FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
	)


CREATE TABLE Academics.TeacherSubjects (
	TeacherID INT,
	SubjectID INT,
	FOREIGN KEY (TeacherID) REFERENCES Academics.Teachers(TeacherID),
	FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
	PRIMARY KEY(TeacherID, SubjectID)
	)

CREATE TABLE Students.Grades (
	StudentID INT NOT NULL,
	TeacherID INT NOT NULL,
	SubjectID INT NOT NULL,
	GradeValue INT CHECK (GradeValue >= 1 AND GradeValue <= 6) NOT NULL,
	DateTime DATETIME NOT NULL,
	FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
	FOREIGN KEY (TeacherID) REFERENCES Academics.Teachers(TeacherID),
	FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
	PRIMARY KEY (StudentID, TeacherID, SubjectID, DateTime)
```

```
        )

CREATE TABLE Students.Attendance (
        AttendanceID INT PRIMARY KEY IDENTITY(0,1),
        StudentID INT NOT NULL,
        AttendanceDate DATE NOT NULL,
        Status VARCHAR(10) CHECK (Status IN ('Present', 'Absent', 'Excused', 'Late')),
        SubjectID INT NOT NULL,
        FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
        FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
        CONSTRAINT UniqueRecords UNIQUE (StudentID, AttendanceDate, SubjectID)
)


CREATE TABLE Students.Parents(
        ParentID INT PRIMARY KEY IDENTITY(0,1),
        FirstName VARCHAR(50) NOT NULL,
        LastName VARCHAR(50) NOT NULL,
        PhoneNumber VARCHAR(10),
        EmailAddress VARCHAR(50),
        )

CREATE TABLE Students.StudentParents(
        StudentID INT NOT NULL,
        ParentID INT NOT NULL,
        FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
        FOREIGN KEY (ParentID) REFERENCES Students.Parents(ParentID),
        PRIMARY KEY (StudentID, ParentID)
)

CREATE TABLE Admins.Rooms (
        RoomNumber INT PRIMARY KEY,
        Capacity INT NOT NULL,
        RoomType VARCHAR(12) CHECK(RoomType IN ('Classroom', 'Gym', 'Auditorium')),
)

CREATE TABLE Students.StudentCouncil (
        StudentID INT PRIMARY KEY,
        Role VARCHAR(30) CHECK (Role IN ('President', 'Vice President', 'Treasurer',
'Secretary')) NOT NULL,
        FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
        CONSTRAINT Unique_Council_Role UNIQUE (Role)
        )

CREATE TABLE Students.Societies (
        SocietyID INT PRIMARY KEY IDENTITY(0,1),
        SocietyName VARCHAR(50) NOT NULL,
        TutorID INT NOT NULL,
```

```
	FOREIGN KEY(TutorID) REFERENCES Academics.Teachers(TeacherID)
	)

CREATE TABLE Students.StudentSocieties
(
	SocietyID INT NOT NULL,
	StudentID INT NOT NULL,
	PRIMARY KEY (SocietyID, StudentID),
	FOREIGN KEY (SocietyID) REFERENCES Students.Societies(SocietyID),
	FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
);

CREATE TABLE Academics.ClassSchedule (
	ClassOneID INT NOT NULL,
	ClassTwoID INT NULL,
	SubjectID INT,
	TeacherID INT,
	Day DATE,
	StartTime TIME,
	EndTime TIME,
	RoomNumber INT,
	FOREIGN KEY (ClassOneID) REFERENCES Academics.Classes(ClassID),
	FOREIGN KEY (ClassTwoID) REFERENCES Academics.Classes(ClassID),
	FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
	FOREIGN KEY(TeacherID) REFERENCES Academics.Teachers(TeacherID),
	FOREIGN KEY(RoomNumber) REFERENCES Admins.Rooms(RoomNumber),
	PRIMARY KEY (Day, StartTime, EndTime, RoomNumber)
	)

CREATE TABLE Academics.LuckyRegisterNumber (
	Day DATE PRIMARY KEY NOT NULL,
	RegisterNumber INT NOT NULL
)

CREATE TABLE Students.BehaviorGrade(
	StudentID INT PRIMARY KEY,
	BehaviorGrade VARCHAR(20) DEFAULT 'Good' CHECK (BehaviorGrade IN
('Inadequate', 'Good', 'Outstanding'))
	FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID)
);

CREATE TABLE Admins.SchoolYearDates (
	SchoolYear INT PRIMARY KEY,
	StartDate DATE NOT NULL,
	EndDate DATE NOT NULL
)

CREATE TABLE Students.FinalGrades (
```

```
	StudentID INT NOT NULL,
	SubjectID INT NOT NULL,
	SchoolYear INT NOT NULL,
	FinalGrade INT CHECK (FinalGrade >= 1 AND FinalGrade <= 6),
	PRIMARY KEY (StudentID, SubjectID, SchoolYear),
	FOREIGN KEY (StudentID) REFERENCES Students.StudentInfo(StudentID),
	FOREIGN KEY (SubjectID) REFERENCES Academics.Subjects(SubjectID),
	FOREIGN KEY (SchoolYear) REFERENCES Admins.SchoolYearDates(SchoolYear)
)


/* Krok 6 - widoki, funkcje, procedury, wyzwalacze*/

CREATE VIEW Students.RankingGPA AS (
	SELECT ROW_NUMBER() OVER (ORDER BY G.GPA DESC) AS Rank,
S.FirstName, S.LastName, G.GPA
	FROM Students.StudentInfo S
	JOIN Students.StudentGPA G ON G.StudentID = S.StudentID
)

CREATE FUNCTION Students.GetClassStudents (@ClassID INT)
RETURNS TABLE
AS
RETURN
(
	SELECT ROW_NUMBER() OVER (ORDER BY S.RegisterNumber ASC) AS
RegisterNumber, S.FirstName, S.LastName, S.PESEL, S.DateOfBirth, S.Gender,
P1.PhoneNumber AS 'Parent Contact 1', P2.PhoneNumber AS 'Parent Contact 2'
	FROM Students.StudentInfo S
	LEFT JOIN Students.StudentParents SP ON S.StudentID = SP.StudentID
	LEFT JOIN Students.Parents P1 ON SP.ParentID = P1.ParentID
	LEFT JOIN Students.Parents P2 ON SP.ParentID = P2.ParentID
	WHERE S.ClassID = @ClassID

);

CREATE FUNCTION Students.GetStudentGradesForSubjectAndYear (
	@StudentID INT,
	@SubjectID INT,
	@SchoolYear INT
)
RETURNS TABLE
AS
RETURN
(
	SELECT
	G.StudentID,
	G.SubjectID,
```

```
        G.GradeValue,
        G.DateTime
        FROM
        Students.Grades G
        JOIN
        Academics.Subjects S ON G.SubjectID = S.SubjectID
        JOIN
        Admins.SchoolYearDates SY ON SY.SchoolYear = @SchoolYear
        WHERE
        G.StudentID = @StudentID
        AND G.SubjectID = @SubjectID
        AND (G.DateTime BETWEEN SY.StartDate AND SY.EndDate)

);

CREATE FUNCTION Students.GetStudentFinalGradesForYear (
        @StudentID INT,
        @SchoolYear INT
)
RETURNS TABLE
AS
RETURN
(
        SELECT
        F.StudentID,
        F.SubjectID,
        F.FinalGrade
        FROM
        Students.FinalGrades F
        WHERE
        F.StudentID = @StudentID
        AND F.SchoolYear = @SchoolYear
);

CREATE VIEW Students.AttendanceRanking AS
SELECT
        s.StudentID,
        s.FirstName,
        s.LastName,
        c.ClassLevel,
        c.ClassSymbol,
        COUNT(a.AttendanceID) AS TotalAbsences
FROM Students.StudentInfo s
JOIN Academics.Classes c ON s.ClassID = c.ClassID
LEFT JOIN Students.Attendance a ON s.StudentID = a.StudentID AND a.Status = 'Absent'
        WHERE EXISTS (
SELECT 1
        FROM Admins.CurrentSchoolYear sy
```

```
        WHERE a.AttendanceDate BETWEEN sy.StartDate AND sy.EndDate
        )

        GROUP BY s.StudentID, s.FirstName, s.LastName, c.ClassLevel, c.ClassSymbol
        ORDER BY TotalAbsences ASC;

CREATE VIEW Students.StudentsWithFinalGrade1 AS
SELECT DISTINCT
        S.StudentID,
        S.FirstName,
        S.LastName
FROM
        Students.StudentInfo S
JOIN
        Students.FinalGrades F ON S.StudentID = F.StudentID
WHERE
        F.FinalGrade = 1 AND F.SchoolYear = Admins.GetCurrentSchoolYear();

CREATE FUNCTION Students.GetStudentSchedule (@StudentID INT)
RETURNS TABLE
AS
        RETURN
        (
        SELECT si.StudentID,
        si.FirstName, si.LastName, cs.Day, cs.StartTime, cs.EndTime, s.SubjectName,
e.FirstName + '' + e.LastName AS TeacherName, cs.RoomNumber
        FROM Academics.ClassSchedule cs
        JOIN Academics.Subjects s ON cs.SubjectID = s.SubjectID
        JOIN Academics.Teachers t ON cs.TeacherID = t.TeacherID
        JOIN Admins.EmployeeInfo e ON t.EmployeeID = e.EmployeeID
        JOIN Students.StudentInfo si ON cs.ClassOneID = si.ClassID OR si.ClassID =
cs.ClassTwoID
        WHERE si.StudentID = @StudentID
        );

CREATE FUNCTION Students.GetNextRegisterNumber (@ClassID INT)
RETURNS INT
AS
BEGIN
        DECLARE @NextRegisterNumber INT;

        IF NOT EXISTS (SELECT 1 FROM Academics.Classes WHERE ClassID =
@ClassID)
        BEGIN
        /* This class does not exist */
        RETURN NULL;
        END
```

```
	SELECT TOP 1 @NextRegisterNumber = RegisterNumber FROM (
	VALUES (1), (2), (3), (4), (5), (6), (7), (8), (9), (10), (11), (12), (13), (14), (15)
	)
	AS AvailableNumbers (RegisterNumber)
	WHERE RegisterNumber NOT IN (
	SELECT RegisterNumber FROM Students.StudentInfo WHERE ClassID = @ClassID
AND RegisterNumber IS NOT NULL
	)

	ORDER BY RegisterNumber ASC;
	RETURN @NextRegisterNumber;

END;

CREATE VIEW Academics.ClassesAndTutors AS (
	SELECT DISTINCT CONVERT(VARCHAR(1), C.ClassLevel) + '' +  C.ClassSymbol
AS 'Class', E.FirstName + ' ' + E.LastName AS 'Tutor', E.PhoneNumber AS 'Tutor Contact',
COUNT(S.StudentID) AS 'Number Of Students'
	FROM Academics.Classes C
	JOIN Academics.Teachers T ON C.TutorID = T.TeacherID
	JOIN Admins.EmployeeInfo E ON E.EmployeeID = T.EmployeeID
	JOIN Students.StudentInfo S ON C.ClassID = S.ClassID
	GROUP BY C.ClassID, C.ClassLevel, C.ClassSymbol, E.FirstName, E.LastName,
E.PhoneNumber
)

CREATE VIEW Academics.TeachersNotTutors AS
SELECT T.TeacherID, E.FirstName, E.LastName, E.PhoneNumber, E.EmailAddress
FROM
	Academics.Teachers T
	JOIN Admins.EmployeeInfo E ON T.EmployeeID = E.EmployeeID
	LEFT JOIN
	Academics.Classes C ON T.TeacherID = C.TutorID
WHERE
	C.ClassID IS NULL;

CREATE PROCEDURE Students.AddNewStudent
	@FirstName VARCHAR(50),
	@LastName VARCHAR(50),
	@PESEL VARCHAR(11),
	@DateOfBirth DATE,
	@Gender CHAR
AS
BEGIN
	SET NOCOUNT ON;

	IF EXISTS (SELECT 1 FROM Students.StudentInfo WHERE PESEL = @PESEL)
	BEGIN
```

```
        RAISERROR('Student with the provided PESEL already exists in the database.', 16,
1);

        RETURN;
        END

        INSERT INTO Students.StudentInfo(PESEL, FirstName, PreferredName, LastName,
DateOfBirth, EnrollmentDate, Gender, ClassID, RegisterNumber)
        VALUES (@PESEL, @FirstName, NULL, @LastName, @DateOfBirth, GETDATE(),
@Gender, NULL, NULL);

END;

CREATE PROCEDURE Students.AddStudentToClass
        @StudentID INT,
        @ClassID INT
AS
BEGIN
        SET NOCOUNT ON;


        IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID =
@StudentID)
        BEGIN
        RAISERROR('Student does not exist.', 16, 1);
        RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Academics.Classes WHERE ClassID =
@ClassID)
        BEGIN
        RAISERROR('Class does not exist.', 16, 1);
        RETURN;
        END;

        IF EXISTS (SELECT 1 FROM Students.StudentInfo WHERE ClassID = @ClassID
AND StudentID = @StudentID)
        BEGIN
        RAISERROR('Student already in this class', 16, 1);
        RETURN;
        END;

        DECLARE @RegisterNumber INT;
        SET @RegisterNumber = Students.GetNextRegisterNumber(@ClassID);

        UPDATE Students.StudentInfo
        SET ClassID = @ClassID, RegisterNumber = @RegisterNumber
```

```
        WHERE StudentID = @StudentID;

END;

CREATE PROCEDURE Students.AddStudentGrade
        @StudentID INT,
        @TeacherID INT,
        @GradeValue INT,
        @SubjectID INT
AS
BEGIN
        SET NOCOUNT ON;


        IF NOT EXISTS (SELECT 1 FROM Academics.Subjects WHERE SubjectID =
@SubjectID)
        BEGIN
        RAISERROR('Subject does not exist', 16, 1);
        RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID =
@StudentID)
        BEGIN
        RAISERROR('Student does not exist', 16, 1);
        RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Academics.Teachers WHERE TeacherID =
@TeacherID)
        BEGIN
        RAISERROR('Teacher does not exist', 16, 1);
        RETURN;
        END;

        IF NOT EXISTS (SELECT 1 FROM Academics.TeacherSubjects WHERE TeacherID
= @TeacherID AND SubjectID = @SubjectID)
        BEGIN
        RAISERROR('Teacher does not teach the subject', 16, 1);
        RETURN;
        END;

        INSERT INTO Students.Grades (StudentID, TeacherID, SubjectID, GradeValue,
DateTime)
        VALUES (@StudentID, @TeacherID, @SubjectID, @GradeValue, GETDATE());


END;
```

```
CREATE PROCEDURE Students.AddStudentReferral
	@StudentID INT,
	@TeacherID INT,
	@ReferralText VARCHAR(500),
	@ReferralType CHAR
AS
BEGIN
	SET NOCOUNT ON;

	IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID =
@StudentID)
	BEGIN
	RAISERROR('Student does not exist', 16, 1);
	RETURN;
	END;

	IF NOT EXISTS (SELECT 1 FROM Academics.Teachers WHERE TeacherID =
@TeacherID)
	BEGIN
	RAISERROR('Teacher does not exist', 16, 1);
	RETURN;
	END;

	INSERT INTO Students.StudentReferrals (TeacherID, StudentID, ReferralDate,
ReferralText, ReferralType)
	VALUES (@TeacherID, @StudentID, GETDATE(), @ReferralText, @ReferralType);

END;

CREATE PROCEDURE Students.RemoveStudentFromClass
	@StudentID INT
AS
BEGIN
	SET NOCOUNT ON;

	DECLARE @ClassID INT, @RegisterNumber INT;

	BEGIN TRANSACTION;

	IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID =
@StudentID)
	BEGIN
	RAISERROR('Student does not exist', 16, 1);
	ROLLBACK TRANSACTION;
	RETURN;
	END;
```

```
        SELECT @ClassID = ClassID, @RegisterNumber = RegisterNumber
        FROM Students.StudentInfo
        WHERE StudentID = @StudentID;

        IF @ClassID IS NULL
        BEGIN
        RAISERROR('Student is not assigned to any class.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
        END;

        UPDATE Students.StudentInfo
        SET ClassID = NULL, RegisterNumber = NULL
        WHERE StudentID = @StudentID;

        UPDATE Students.StudentInfo
        SET RegisterNumber = RegisterNumber - 1
        WHERE ClassID = @ClassID AND RegisterNumber > @RegisterNumber;

        COMMIT TRANSACTION;

END;

CREATE PROCEDURE Academics.GetLuckyRegisterNumber
AS
BEGIN
        DECLARE @LuckyRegisterNumber INT;

        IF NOT EXISTS (SELECT 1 FROM Academics.LuckyRegisterNumber WHERE
CAST(Day AS DATE) = CAST(GETDATE() AS DATE))
        BEGIN
        SET @LuckyRegisterNumber = FLOOR(RAND() * 15) + 1;
        INSERT INTO Academics.LuckyRegisterNumber (Day, RegisterNumber) VALUES
(GETDATE(), @LuckyRegisterNumber);
        END;

        SELECT RegisterNumber AS 'The Lucky Register Number for today'
        FROM Academics.LuckyRegisterNumber
        WHERE CAST(Day AS DATE) = CAST(GETDATE() AS DATE);

END;

CREATE PROCEDURE Students.RemoveStudent
        @StudentID INT
AS
BEGIN
```

```
	IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID =
@StudentID)
	BEGIN
	RAISERROR('Student does not exist', 16, 1);
	RETURN;
	END

	BEGIN TRANSACTION;

	BEGIN TRY

	DELETE FROM Students.StudentCouncil WHERE StudentID = @StudentID;
	DELETE FROM Students.StudentSocieties WHERE StudentID = @StudentID;
	DELETE FROM Students.Grades WHERE StudentID = @StudentID;
	DELETE FROM Students.BehaviorGrade WHERE StudentID = @StudentID;
	DELETE FROM Students.StudentReferrals WHERE StudentID = @StudentID;
	DELETE FROM Students.Attendance WHERE StudentID = @StudentID;

	DECLARE @ParentID INT;
	SELECT @ParentID = ParentID FROM Students.StudentParents WHERE StudentID
= @StudentID;

	DELETE FROM Students.StudentParents WHERE StudentID = @StudentID;

	IF NOT EXISTS (SELECT 1 FROM Students.StudentParents WHERE ParentID =
@ParentID)
	BEGIN
	DELETE FROM Students.Parents WHERE ParentID = @ParentID;
	END

	DELETE FROM Students.StudentInfo WHERE StudentID = @StudentID;

	COMMIT TRANSACTION;
	END TRY

	BEGIN CATCH
	ROLLBACK TRANSACTION;
	END CATCH
END;

CREATE PROCEDURE Students.GenerateStudentTranscript
	@StudentID INT
AS
BEGIN
	SET NOCOUNT ON;

	IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID =
@StudentID)
```

```
        BEGIN
        RAISERROR('Student does not exist', 16, 1);
        RETURN;
        END;

        DECLARE @PESEL VARCHAR(11), @FirstName VARCHAR(50), LastName
VARCHAR(50);

SELECT @FirstName = FirstName,
        @LastName = LastName,
        @PESEL = PESEL
FROM Students.StudentInfo
WHERE StudentID = @StudentID;

        SELECT
        s.SubjectName AS 'Subject',
        t.FirstName + ' ' + t.LastName AS 'Teacher',
        g.GradeValue AS 'Grade',
        g.DateTime AS 'Date'
        FROM Students.Grades g
        JOIN Academics.Subjects s ON g.SubjectID = s.SubjectID
        JOIN Academics.Teachers t ON g.TeacherID = t.TeacherID
        WHERE g.StudentID = @StudentID
        ORDER BY s.SubjectName, g.DateTime;

SELECT
        s.SubjectName AS 'Subject',
        ROUND(AVG(CAST(g.GradeValue AS FLOAT)), 2) AS 'Average grade'
FROM Students.Grades g
JOIN Academics.Subjects s ON g.SubjectID = s.SubjectID
WHERE g.StudentID = @StudentID
GROUP BY s.SubjectName;

END;

CREATE PROCEDURE Students.CalculateFinalGrades
        @StudentID INT,
        @SchoolYear INT
AS
BEGIN
        SET NOCOUNT ON;

        DECLARE @AverageGrade DECIMAL(3, 2);
        DECLARE @FinalGrade INT;
        DECLARE @StartDate DATE;
        DECLARE @EndDate DATE;
        DECLARE @SubjectID INT;
```

```
	IF NOT EXISTS (SELECT 1 FROM Students.StudentInfo WHERE StudentID =
@StudentID)
	BEGIN
	RAISERROR('Student does not exist', 16, 1);
	RETURN;
	END

	SELECT @StartDate = StartDate, @EndDate = EndDate
	FROM Admins.SchoolYearDates
	WHERE SchoolYear = @SchoolYear;

	IF @StartDate IS NULL OR @EndDate IS NULL
	BEGIN
	RETURN;
	END

	BEGIN TRANSACTION;

	DECLARE SubjectCursor CURSOR FOR
	SELECT SubjectID
	FROM Students.Grades
	WHERE StudentID = @StudentID;

	OPEN SubjectCursor;
	FETCH NEXT FROM SubjectCursor INTO @SubjectID;

	WHILE @@FETCH_STATUS = 0
	BEGIN

	SELECT @AverageGrade = AVG(GradeValue)
	FROM Students.Grades
	WHERE StudentID = @StudentID
	AND SubjectID = @SubjectID
	AND DateTime BETWEEN @StartDate AND @EndDate;

	IF @AverageGrade IS NULL
	BEGIN

	FETCH NEXT FROM SubjectCursor INTO @SubjectID;
	CONTINUE;
	END

	SET @FinalGrade = CASE
	WHEN @AverageGrade < 1.5 THEN 1
	WHEN @AverageGrade < 2.5 THEN 2
	WHEN @AverageGrade < 3.5 THEN 3
	WHEN @AverageGrade < 4.5 THEN 4
	WHEN @AverageGrade < 5.5 THEN 5
```

```
        WHEN @AverageGrade >= 5.5 THEN 6
        ELSE NULL
        END;

        IF EXISTS (SELECT 1 FROM Students.FinalGrades WHERE StudentID =
@StudentID AND SchoolYear = @SchoolYear)
        BEGIN
        UPDATE Students.FinalGrades
        SET FinalGrade = @FinalGrade
        WHERE StudentID = @StudentID AND SchoolYear = @SchoolYear AND SubjectID
= @SubjectID;
        END
        ELSE
        BEGIN
        INSERT INTO Students.FinalGrades (StudentID, SchoolYear, SubjectID, FinalGrade)
        VALUES (@StudentID, @SchoolYear, @SubjectID, @FinalGrade);
        END

        FETCH NEXT FROM SubjectCursor INTO @SubjectID;
        END

        CLOSE SubjectCursor;
        DEALLOCATE SubjectCursor;
        COMMIT TRANSACTION;

END;

CREATE PROCEDURE Students.UpdateBehaviorGrade
        @StudentID INT
AS
BEGIN
        DECLARE @GPA DECIMAL(3,2);
        DECLARE @PositiveReferrals INT;
        DECLARE @NegativeReferrals INT;
        DECLARE @Absences INT;
        DECLARE @NewBehaviorGrade VARCHAR(20) = 'Good';

        SELECT @GPA = GPA
FROM Students.StudentGPA
WHERE StudentID = @StudentID;

SELECT @PositiveReferrals = COUNT(*)
FROM Students.StudentReferrals
WHERE StudentID = @StudentID AND ReferralType = '+';

SELECT @NegativeReferrals = COUNT(*)
FROM Students.StudentReferrals
WHERE StudentID = @StudentID AND ReferralType = '-';
```

```
SELECT @Absences = COUNT(*)
FROM Students.Attendance
WHERE StudentID = @StudentID AND AttendanceStatus = 'Absent';

IF @GPA IS NOT NULL AND @GPA > 4.0 AND @PositiveReferrals >= 3 AND @Absences
< 10
	SET @NewBehaviorGrade = 'Outstanding';
ELSE IF @GPA IS NOT NULL AND @GPA < 3.0 AND @NegativeReferrals >= 3 AND
@Absences > 30
	SET @NewBehaviorGrade = 'Inadequate';

UPDATE Students.BehaviorGrade
SET BehaviorGrade = @NewBehaviorGrade
WHERE StudentID = @StudentID;

END;

CREATE PROCEDURE Admins.AddTeacher
	@FirstName VARCHAR(50),
	@LastName VARCHAR(50),
	@Salary FLOAT,
	@PhoneNumber VARCHAR(12) = NULL,
	@EmailAddress VARCHAR(50) = NULL
AS
BEGIN
	SET NOCOUNT ON;

	DECLARE @NewEmployeeID INT;

	BEGIN TRANSACTION;

	BEGIN TRY
	INSERT INTO Admins.EmployeeInfo (FirstName, LastName, Salary, PhoneNumber,
EmailAddress)
	VALUES (@FirstName, @LastName, @Salary, @PhoneNumber, @EmailAddress);

	SET @NewEmployeeID = SCOPE_IDENTITY();

	INSERT INTO Academics.Teachers(EmployeeID)
	VALUES (@NewEmployeeID);

	COMMIT TRANSACTION;
	END TRY
	BEGIN CATCH
	ROLLBACK TRANSACTION;
	RAISERROR('Error occurred while adding a new teacher', 16, 1);
	END CATCH;
```

```
END;

CREATE TRIGGER Students.TrackGPA ON Students.FinalGrades
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
	DECLARE @StudentID INT;
	DECLARE @CurrentSchoolYear INT;
	DECLARE @UpdatedGPA DECIMAL (3,2);

	SET @CurrentSchoolYear = Admins.GetCurrentSchoolYear();

	IF EXISTS (SELECT 1 FROM inserted)
	BEGIN
	SELECT @StudentID = StudentID FROM inserted;
	END

	ELSE IF EXISTS (SELECT 1 FROM deleted)
	BEGIN
	SELECT @StudentID = StudentID FROM deleted;
	END

	SELECT @UpdatedGPA = AVG(CAST(FinalGrade AS DECIMAL (3,2))) FROM
Students.FinalGrades
	WHERE StudentID = @StudentID AND SchoolYear = @CurrentSchoolYear;

	IF EXISTS (SELECT 1 FROM Students.StudentGPA WHERE StudentID =
@StudentID)
	BEGIN
	UPDATE Students.StudentGPA SET GPA = @UpdatedGPA WHERE StudentID =
@StudentID
	RETURN;
	END

	INSERT INTO Students.StudentGPA (StudentID, GPA) VALUES (@StudentID,
@UpdatedGPA)

END;

CREATE TRIGGER Students.MaxStudentsInClass ON Students.StudentInfo
AFTER UPDATE
AS
BEGIN
	DECLARE @StudentCount INT;
	DECLARE @ClassID INT;

	SELECT @ClassID = ClassID FROM inserted;
```

```
	IF @ClassID IS NOT NULL
	BEGIN
	SELECT @StudentCount = COUNT(*) FROM Students.StudentInfo WHERE ClassID
= @ClassID;

	IF @StudentCount >= 15
		BEGIN
		RAISERROR('Class already full. Cannot add more students to this class.', 16,
1);
		ROLLBACK;
		END
	END
END;

CREATE TRIGGER PreventClassDeletionWithStudents ON Academics.Classes
FOR DELETE
AS
BEGIN
	DECLARE @ClassID INT;
	SELECT @ClassID = ClassID FROM DELETED;

	IF EXISTS (SELECT 1 FROM Students.StudentInfo WHERE ClassID = @ClassID)
	BEGIN
	RAISERROR('Deleting classes with students enrolled is not allowed.', 16, 1);
	ROLLBACK;
	END
END;

CREATE TRIGGER Students.PreventFutureAttendance
ON Students.Attendance
AFTER INSERT
AS
BEGIN
	IF EXISTS (SELECT 1 FROM inserted WHERE AttendanceDate > GETDATE())
	BEGIN
	RAISERROR('Adding attendance from the future is not allowed', 16, 1);
	ROLLBACK;
	RETURN;
	END;
END;

CREATE TRIGGER Students.trg_UpdateBehaviorGrade_Referrals
ON Students.StudentReferrals
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
	DECLARE @StudentID INT;
```

```
	DECLARE cur CURSOR FOR
	SELECT DISTINCT StudentID FROM inserted
	UNION
	SELECT DISTINCT StudentID FROM deleted;

	OPEN cur;
	FETCH NEXT FROM cur INTO @StudentID;

	WHILE @@FETCH_STATUS = 0
	BEGIN
	EXEC Students.UpdateBehaviorGrade @StudentID;
	FETCH NEXT FROM cur INTO @StudentID;
	END;

	CLOSE cur;
	DEALLOCATE cur;
END;



CREATE TRIGGER Students.trg_UpdateBehaviorGrade_Grades
ON Students.Grades
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
	DECLARE @StudentID INT;

	DECLARE cur CURSOR FOR
	SELECT DISTINCT StudentID FROM inserted
	UNION
	SELECT DISTINCT StudentID FROM deleted;

	OPEN cur;
	FETCH NEXT FROM cur INTO @StudentID;

	WHILE @@FETCH_STATUS = 0
	BEGIN
	EXEC Students.UpdateBehaviorGrade @StudentID;
	FETCH NEXT FROM cur INTO @StudentID;
	END;

	CLOSE cur;
	DEALLOCATE cur;
END;


CREATE TRIGGER Students.trg_UpdateBehaviorGrade_Attendance
ON Students.Attendance
```

```
AFTER INSERT, UPDATE, DELETE
AS
BEGIN
	DECLARE @StudentID INT;

	DECLARE cur CURSOR FOR
	SELECT DISTINCT StudentID FROM inserted
	UNION
	SELECT DISTINCT StudentID FROM deleted;

	OPEN cur;
	FETCH NEXT FROM cur INTO @StudentID;

	WHILE @@FETCH_STATUS = 0
	BEGIN
	EXEC Students.UpdateBehaviorGrade @StudentID;
	FETCH NEXT FROM cur INTO @StudentID;
	END;

	CLOSE cur;
	DEALLOCATE cur;
END;

/* Krok 7 - uzupelnienie przykładowymi danymi */

INSERT INTO Admins.SchoolYearDates (SchoolYear, StartDate, EndDate) VALUES (2024,
'2024-09-02', '2025-06-28')

INSERT INTO Students.StudentInfo (PESEL, FirstName, PreferredName, LastName,
DateOfBirth, EnrollmentDate, Gender)
VALUES ('09120903446', 'Natalia', NULL, 'Monet', '2009-12-09', '2025-02-13', 'F'),
('06052314567', 'Marek', NULL, 'Monet', '2006-05-23', '2025-02-13', 'M'),
('09062313489', 'Damian', NULL, 'Park', '2009-06-23', '2025-02-13', 'M'),
('09080815793', 'Mateusz', NULL, 'Song', '2009-08-08', '2025-02-13', 'M'),
('06011217864', 'Jan', NULL, 'Kowalski', '2007-01-12', '2025-02-13', 'M'),
('09050612542', 'Oliwier', NULL, 'Szewczyk', '2009-05-06', '2025-02-13', 'M'),
('06091813457', 'Jakub', NULL, 'Marcinkowski', '2006-09-18', '2025-02-13', 'M'),
('09021115897', 'Wojciech', NULL, 'Kowal', '2009-02-11', '2025-02-13', 'M'),
('06060413450', 'Beniamin', NULL, 'Drozd', '2006-06-04', '2025-02-13', 'M'),
('09081717342', 'Łukasz', NULL, 'Nowak', '2009-08-17', '2025-02-13', 'M'),
('06020219563', 'Jacek', NULL, 'Nowakowski', '2006-02-02', '2025-02-13', 'M'),
('09092813712', 'Henryk', NULL, 'Miller', '2009-09-28', '2025-02-13', 'M'),
('06080712689', 'Aleksander', NULL, 'Davis', '2006-08-07', '2025-02-13', 'M'),
('09090316425', 'Jerzy', NULL, 'Malinowski', '2009-09-03', '2025-02-13', 'M'),
('06061815634', 'Tomasz', NULL, 'Hel', '2006-06-18', '2025-02-13', 'M'),
('09042314879', 'Dawid', NULL, 'García', '2009-04-23', '2025-02-13', 'M'),
('06051417384', 'Mateusz', NULL, 'Lasek', '2006-05-14', '2025-02-13', 'M'),
('09080317956', 'Donald', NULL, 'Czajkowski', '2009-08-03', '2025-02-13', 'M'),
```

('06032716792', 'Oskar', NULL, 'Smolny', '2006-03-27', '2025-02-13', 'M'),
('09051612847', 'Tomasz', NULL, 'Gwiazda', '2009-05-16', '2025-02-13', 'M'),
('06021412345', 'Natalia', NULL, 'Kowalska', '2006-02-14', '2025-02-13', 'F'),
('06032254321', 'Maria', NULL, 'Nowak', '2006-03-22', '2025-02-13', 'F'),
('06051023456', 'Katarzyna', NULL, 'Wiśniewska', '2006-05-10', '2025-02-13', 'F'),
('06070534567', 'Zofia', NULL, 'Krawczyk', '2006-07-05', '2025-02-13', 'F'),
('06081612345', 'Aleksandra', NULL, 'Jankowska', '2006-08-16', '2025-02-13', 'F'),
('06100967890', 'Olga', NULL, 'Wójcik', '2006-10-09', '2025-02-13', 'F'),
('06112513579', 'Magdalena', NULL, 'Pawlak', '2006-11-25', '2025-02-13', 'F'),
('07011856789', 'Anna', NULL, 'Szymańska', '2007-01-18', '2025-02-13', 'F'),
('07022898765', 'Elżbieta', NULL, 'Zielińska', '2007-02-28', '2025-02-13', 'F'),
('07030787654', 'Julia', NULL, 'Bąk', '2007-03-07', '2025-02-13', 'F'),
('07041965432', 'Agnieszka', NULL, 'Kaczmarek', '2007-04-19', '2025-02-13', 'F'),
('07061243210', 'Monika', NULL, 'Mazur', '2007-06-12', '2025-02-13', 'F'),
('07090323456', 'Iwona', NULL, 'Nowicki', '2007-09-03', '2025-02-13', 'F'),
('07111587654', 'Dorota', NULL, 'Sikora', '2007-11-15', '2025-02-13', 'F'),
('07122134567', 'Barbara', NULL, 'Laskowski', '2007-12-21', '2025-02-13', 'F'),
('08012223456', 'Paulina', NULL, 'Wasilewska', '2008-01-22', '2025-02-13', 'F'),
('08031087654', 'Wiktoria', NULL, 'Szczepaniak', '2008-03-10', '2025-02-13', 'F'),
('08040823456', 'Kinga', NULL, 'Górska', '2008-04-08', '2025-02-13', 'F'),
('08060134567', 'Karolina', NULL, 'Chmiel', '2008-06-01', '2025-02-13', 'F'),
('08072012345', 'Jack', NULL, 'Stolarz', '2008-07-20', '2025-02-13', 'F'),
('08091356789', 'Julia', NULL, 'Zawisza', '2008-09-13', '2025-02-13', 'F'),
('08101723456', 'Karolina', NULL, 'Wróbel', '2008-10-17', '2025-02-13', 'F'),
('09011067890', 'Agnieszka', NULL, 'Ławniczak', '2009-01-10', '2025-02-13', 'F');

INSERT INTO Admins.EmployeeInfo (FirstName, LastName, Salary, PhoneNumber, EmailAddress) VALUES
('Jan', 'Kowalski', 5000, '123456789', 'jan.kowalski@example.com'),
('Anna', 'Nowak', 5200, '987654321', 'anna.nowak@example.com'),
('Piotr', 'Zieliński', 4800, '555123789', 'piotr.zielinski@example.com'),
('Michał', 'Wójcik', 5400, '654321987', 'michal.wojcik@example.com'),
('Ewa', 'Kowalczyk', 5300, '789654123', 'ewa.kowalczyk@example.com'),
('Tomasz', 'Lewandowski', 5000, '321987654', 'tomasz.lewandowski@example.com'),
('Magdalena', 'Szymańska', 5100, '222333444', 'magdalena.szymanska@example.com'),
('Krzysztof', 'Jankowski', 4900, '555666777', 'krzysztof.jankowski@example.com'),
('Agnieszka', 'Zawisza', 4700, '888999000', 'agnieszka.zawisza@example.com'),
('Paweł', 'Piotrowski', 5200, '234567890', 'pawel.piotrowski@example.com'),
('Karolina', 'Wróbel', 4600, '101234567', 'karolina.wrobel@example.com'),
('Grzegorz', 'Kaczmarek', 5400, '555987654', 'grzegorz.kaczmarek@example.com'),
('Katarzyna', 'Adamczyk', 4800, '777123456', 'katarzyna.adamczyk@example.com'),
('Robert', 'Bąk', 5100, '444333222', 'robert.bak@example.com'),
('Olga', 'Mazur', 5000, '333222111', 'olga.mazur@example.com'),
('Andrzej', 'Nowakowski', 5200, '222555888', 'andrzej.nowakowski@example.com'),
('Patryk', 'Kubiak', 4700, '666111444', 'patryk.kubiak@example.com'),
('Izabela', 'Jabłońska', 5300, '555444333', 'izabela.jablonska@example.com'),
('Jakub', 'Bielak', 4800, '333444555', 'jakub.bielak@example.com'),

```
('Natalia', 'Rutkowska', 5100, '888777555', 'natalia.rutkowska@example.com'),
('Mateusz', 'Kwiatkowski', 5400, '999888777', 'mateusz.kwiatkowski@example.com'),
('Marta', 'Pawlak', 5200, '111222333', 'marta.pawlak@example.com'),
('Jakub', 'Nowicki', 4900, '444111999', 'jakub.nowicki@example.com'),
('Julia', 'Dąbrowska', 4700, '555666888', 'julia.dabrowska@example.com'),
('Szymon', 'Wilk', 5300, '123789456', 'szymon.wilk@example.com'),
('Wioletta', 'Lis', 5100, '999555444', 'wioletta.lis@example.com'),
('Bartłomiej', 'Ziółkowski', 5200, '234567890', 'bartlomiej.ziolkowski@example.com'),
('Paula', 'Sikora', 5000, '555444666', 'paula.sikora@example.com'),
('Maciej', 'Żuraw', 5400, '777555888', 'maciej.zuraw@example.com'),
('Karol', 'Bartosz', 4700, '888111555', 'karol.bartosz@example.com'),
('Joanna', 'Ślusarczyk', 4900, '999888666', 'joanna.slusarczyk@example.com');
DECLARE @Emp1 INT = (SELECT EmployeeID FROM Admins.EmployeeInfo WHERE
FirstName =  'Jan' AND LastName =  'Kowalski');
DECLARE @Emp2 INT = (SELECT EmployeeID FROM Admins.EmployeeInfo WHERE
FirstName = 'Maciej' AND LastName = 'Żuraw');
DECLARE @Emp3 INT = (SELECT EmployeeID FROM Admins.EmployeeInfo WHERE
FirstName = 'Karol' AND LastName = 'Bartosz');
DECLARE @Emp4 INT = (SELECT EmployeeID FROM Admins.EmployeeInfo WHERE
FirstName = 'Paula' AND LastName = 'Sikora');
DECLARE @Emp5 INT = (SELECT EmployeeID FROM Admins.EmployeeInfo WHERE
FirstName = 'Szymon' AND LastName = 'Wilk');
DECLARE @Emp6 INT = (SELECT EmployeeID FROM Admins.EmployeeInfo WHERE
FirstName = 'Robert' AND LastName = 'Bąk');
DECLARE @Emp7 INT = (SELECT EmployeeID FROM Admins.EmployeeInfo WHERE
FirstName = 'Tomasz' AND LastName = 'Lewandowski');

INSERT INTO Academics.Teachers (EmployeeID)
VALUES (@Emp1), (@Emp2), (@Emp3), (@Emp4), (@Emp5), (@Emp6), (@Emp7);


DECLARE @T1 INT = (SELECT TeacherID FROM Academics.Teachers WHERE
EmployeeID = @Emp1);
DECLARE @T2 INT = (SELECT TeacherID FROM Academics.Teachers WHERE
EmployeeID = @Emp2);
DECLARE @T3 INT = (SELECT TeacherID FROM Academics.Teachers WHERE
EmployeeID = @Emp3);
DECLARE @T4 INT = (SELECT TeacherID FROM Academics.Teachers WHERE
EmployeeID = @Emp4);
DECLARE @T5 INT = (SELECT TeacherID FROM Academics.Teachers WHERE
EmployeeID = @Emp5);
DECLARE @T6 INT = (SELECT TeacherID FROM Academics.Teachers WHERE
EmployeeID = @Emp6);
DECLARE @T7 INT = (SELECT TeacherID FROM Academics.Teachers WHERE
EmployeeID = @Emp7);


INSERT INTO Academics.Classes (ClassLevel, ClassSymbol, TutorID)
```

```
VALUES
	(1, 'A', @T1), (1, 'B', @T2), (2, 'A', @T3), (2, 'B', @T4), (3, 'A', @T5), (3, 'B', @T6),
(4, 'A', @T7), (4, 'B', @T3);

DECLARE @Student1 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09120903446');
DECLARE @Student2 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06052314567');
DECLARE @Student3 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09062313489');
DECLARE @Student4 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09080815793');
DECLARE @Student5 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06011217864');
DECLARE @Student6 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09050612542');
DECLARE @Student7 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06091813457');
DECLARE @Student8 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09021115897');
DECLARE @Student9 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06060413450');
DECLARE @Student10 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09081717342');
DECLARE @Student11 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06020219563');
DECLARE @Student12 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09092813712');
DECLARE @Student13 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06080712689');
DECLARE @Student14 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09090316425');
DECLARE @Student15 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06061815634');
DECLARE @Student16 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09042314879');
DECLARE @Student17 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06051417384');
DECLARE @Student18 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09080317956');
DECLARE @Student19 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06032716792');
DECLARE @Student20 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09051612847');
DECLARE @Student21 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06021412345');
DECLARE @Student22 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06032254321');
```

```
DECLARE @Student23 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06051023456');
DECLARE @Student24 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06070534567');
DECLARE @Student25 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06081612345');
DECLARE @Student26 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06100967890');
DECLARE @Student27 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '06112513579');
DECLARE @Student28 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07011856789');
DECLARE @Student29 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07022898765');
DECLARE @Student30 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07030787654');
DECLARE @Student31 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07041965432');
DECLARE @Student32 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07061243210');
DECLARE @Student33 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07090323456');
DECLARE @Student34 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07111587654');
DECLARE @Student35 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '07122134567');
DECLARE @Student36 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '08012223456');
DECLARE @Student37 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '08031087654');
DECLARE @Student38 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '08040823456');
DECLARE @Student39 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '08060134567');
DECLARE @Student40 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '08072012345');
DECLARE @Student41 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '08091356789');
DECLARE @Student42 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '08101723456');
DECLARE @Student43 INT = (SELECT StudentID FROM Students.StudentInfo WHERE
PESEL = '09011067890');

INSERT INTO Students.StudentCouncil (StudentID, Role) VALUES (@Student1,
'President'), (@Student2, 'Vice President'), (@Student3, 'Treasurer'), (@Student4,
'Secretary');
```

```
DECLARE @C1A INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 1 AND ClassSymbol = 'A');
DECLARE @C1B INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 1 AND ClassSymbol = 'B');
DECLARE @C2A INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 2 AND ClassSymbol = 'A');
DECLARE @C2B INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 2 AND ClassSymbol = 'B');
DECLARE @C3A INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 3 AND ClassSymbol = 'A');
DECLARE @C3B INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 3 AND ClassSymbol = 'B');
DECLARE @C4A INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 4 AND ClassSymbol = 'A');
DECLARE @C4B INT = (SELECT ClassID FROM Academics.Classes WHERE ClassLevel
= 4 AND ClassSymbol = 'B');

EXEC Students.AddStudentToClass
        @StudentID = @Student1,
        @ClassID = @C1A

EXEC Students.AddStudentToClass
        @StudentID = @Student2,
        @ClassID = @C4A


EXEC Students.AddStudentToClass
        @StudentID = @Student3,
        @ClassID = @C1A

EXEC Students.AddStudentToClass
        @StudentID = @Student4,
        @ClassID = @C1A

EXEC Students.AddStudentToClass
        @StudentID = @Student5,
        @ClassID = @C3A

EXEC Students.AddStudentToClass
        @StudentID = @Student6,
        @ClassID = @C1A

EXEC Students.AddStudentToClass
        @StudentID = @Student7,
        @ClassID = @C4A

EXEC Students.AddStudentToClass
        @StudentID = @Student8,
```

```
        @ClassID = @C1A

EXEC Students.AddStudentToClass
        @StudentID = @Student9,
        @ClassID = @C4A

EXEC Students.AddStudentToClass
        @StudentID = @Student10,
        @ClassID = @C1A

EXEC Students.AddStudentToClass
        @StudentID = @Student11,
        @ClassID = @C4A

EXEC Students.AddStudentToClass
        @StudentID = @Student12,
        @ClassID = @C1A

EXEC Students.AddStudentToClass
        @StudentID = @Student13,
        @ClassID = @C4A


EXEC Students.AddStudentToClass
        @StudentID = @Student14,
        @ClassID = @C1B

EXEC Students.AddStudentToClass
        @StudentID = @Student15,
        @ClassID = @C4A


EXEC Students.AddStudentToClass
        @StudentID = @Student16,
        @ClassID = @C1B

EXEC Students.AddStudentToClass
        @StudentID = @Student17,
        @ClassID = @C4B

EXEC Students.AddStudentToClass
        @StudentID = @Student18,
        @ClassID = @C1B

EXEC Students.AddStudentToClass
        @StudentID = @Student19,
        @ClassID = @C4B
```

```
EXEC Students.AddStudentToClass
        @StudentID = @Student20,
        @ClassID = @C1B

EXEC Students.AddStudentToClass
        @StudentID = @Student21,
        @ClassID = @C4B

EXEC Students.AddStudentToClass
        @StudentID = @Student22,
        @ClassID = @C4B

EXEC Students.AddStudentToClass
        @StudentID = @Student23,
        @ClassID = @C4B

EXEC Students.AddStudentToClass
        @StudentID = @Student24,
        @ClassID = @C4B

EXEC Students.AddStudentToClass
        @StudentID = @Student25,
        @ClassID = @C4A

EXEC Students.AddStudentToClass
        @StudentID = @Student26,
        @ClassID = @C4A

EXEC Students.AddStudentToClass
        @StudentID = @Student27,
        @ClassID = @C4B

EXEC Students.AddStudentToClass
        @StudentID = @Student28,
        @ClassID = @C4B

EXEC Students.AddStudentToClass
        @StudentID = @Student29,
        @ClassID = @C3A

EXEC Students.AddStudentToClass
        @StudentID = @Student30,
        @ClassID = @C3A

EXEC Students.AddStudentToClass
        @StudentID = @Student31,
        @ClassID = @C3A
```

```
EXEC Students.AddStudentToClass
        @StudentID = @Student32,
        @ClassID = @C3A

EXEC Students.AddStudentToClass
        @StudentID = @Student33,
        @ClassID = @C3A

EXEC Students.AddStudentToClass
        @StudentID = @Student34,
        @ClassID = @C3A

EXEC Students.AddStudentToClass
        @StudentID = @Student35,
        @ClassID = @C3A

EXEC Students.AddStudentToClass
        @StudentID = @Student36,
        @ClassID = @C2A

EXEC Students.AddStudentToClass
        @StudentID = @Student37,
        @ClassID = @C2A

EXEC Students.AddStudentToClass
        @StudentID = @Student38,
        @ClassID = @C2A

EXEC Students.AddStudentToClass
        @StudentID = @Student39,
        @ClassID = @C2A

EXEC Students.AddStudentToClass
        @StudentID = @Student40,
        @ClassID = @C2A

EXEC Students.AddStudentToClass
        @StudentID = @Student41,
        @ClassID = @C2A

EXEC Students.AddStudentToClass
        @StudentID = @Student42,
        @ClassID = @C2A

EXEC Students.AddStudentToClass
        @StudentID = @Student43,
        @ClassID = @C1B
```

```
INSERT INTO Academics.Subjects (SubjectName) VALUES ('Matematyka'), ('Język Polski'),
('Fizyka'), ('Historia'), ('Informatyka'), ('Język Angielski'), ('Chemia'), ('Biologia');

DECLARE @S1 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Matematyka');
DECLARE @S2 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Język Polski');
DECLARE @S3 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Fizyka');
DECLARE @S4 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Historia');
DECLARE @S5 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Informatyka');
DECLARE @S6 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Język Angielski');
DECLARE @S7 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Chemia');
DECLARE @S8 INT = (SELECT SubjectID FROM Academics.Subjects WHERE
SubjectName = 'Biologia');

INSERT INTO Academics.TeacherSubjects (SubjectID, TeacherID)
VALUES (@S1, @T1), (@S2, @T2), (@S3, @T1), (@S3, @T3),  (@S4, @T4), (@S5,
@T6), (@S6, @T2), (@S2, @T7), (@S7, @T7), (@S8, @T7);


INSERT INTO Students.Societies (SocietyName, TutorID) VALUES ('Debate Club', @T2),
('Chess Club', @T3);

DECLARE @Society1 INT = (SELECT SocietyID FROM Students.Societies WHERE
SocietyName = 'Debate Club');

DECLARE @Society2 INT = (SELECT SocietyID FROM Students.Societies WHERE
SocietyName = 'Chess Club');


INSERT INTO Students.StudentSocieties (StudentID, SocietyID) VALUES (@Student1,
@Society1),  (@Student2, @Society1),  (@Student3, @Society1), (@Student3, @Society2),
(@Student32, @Society1), (@Student33, @Society2), (@Student43, @Society2);


INSERT INTO Students.Parents (FirstName, LastName, PhoneNumber, EmailAddress)
VALUES ('Jan', 'Monet', '111111111', 'j.m@example.pl'),
('Karina', 'Kowalska-Monet', '214561781', 'monet.k@example.pl'),
('Anna', 'Park', '987805799', 'anna.park@example.pl'),
('Anna', 'Song', '987105799', 'anna.s@example.pl'),
('Marcin', 'Kowalski', '187805799', 'm.kowalski@example.pl'),
('Julia', 'Kowalska', '187805599', 'j.kowalska@example.pl'),
('Piotr', 'Szewczyk', '498567320', 'piotr.szewczyk@example.pl'),
```

('Katarzyna', 'Szewczyk', '498167320', 'k.szewczyk@example.pl'),
('Marta', 'Marcinkowska', '507123456', 'marta.marcinkowska@example.pl'),
('Tomasz', 'Marcinkowski', '508654321', 'tomasz.marcinkowski@example.pl'),
('Andrzej', 'Kowal', '503432876', 'andrzej.kowal@example.pl'),
('Maria', 'Kowal', '522432876', 'm.kowal@example.pl'),
('Zofia', 'Drozd', '509876543', 'zofia.drozd@example.pl'),
('Artur', 'Drozd', '111876543', 'artur.drozd@example.pl'),
('Katarzyna', 'Nowak', '506789321', 'katarzyna.nowak@example.pl'),
('Gabriel', 'Nowak', '506000321', 'g.nowak@example.pl'),
('Piotr', 'Nowakowski', '502341987', 'piotr.nowakowski@example.pl'),
('Klara', 'Nowakowska', '302341937', 'klara.n@example.pl'),
('Marek', 'Miller', '510876324', 'marek.miller@example.pl'),
('Anna', 'Miller', '566676324', 'anna.miller@example.pl'),
('Agnieszka', 'Davis', '503456789', 'agnieszka.davis@example.pl'),
('Thomas', 'Davis', '503776789', 'th.davis@example.pl'),
('Tadeusz', 'Malinowski', '505123456', 'tadeusz.malinowski@example.pl'),
('Renata', 'Malinowska', '805123496', 'r.malinowska@example.pl'),
('Renata', 'Hel', '506234678', 'renata.hel@example.pl'),
('Katarzyna', 'Hel', '506234688', 'k.hel@example.pl'),
('Artur', 'García', '507345890', 'a.garcia@example.pl'),
('Maria', 'García', '506902890', 'm.garcia@example.pl'),
('Kamil', 'Lasek', '509456312', 'kamil.lasek@example.pl'),
('Kamila', 'Lasek', '719456312', 'kamila.lasek@example.pl'),
('Zuzanna', 'Czajkowski', '666876543', 'zuzanna.czajkowski@example.pl'),
('Aleksander', 'Czajkowski', '501876543', 'czajkowski.a@example.pl'),
('Tadeusz', 'Smolny', '507234567', 'tadeusz.smolny@example.pl'),
('Aneta', 'Smolny', '500234507', 'a.smolny@example.pl'),
('Anna', 'Gwiazda', '504789231', 'anna.gwiazda@example.pl'),
('Magdalena', 'Nowak', '502876543', 'magdalena.nowak@example.pl'),
('Jan', 'Wiśniewski', '505678901', 'jan.wisniewski@example.pl'),
('Michał', 'Krawczyk', '508901234', 'michal.krawczyk@example.pl'),
('Bożena', 'Jankowska', '507345678', 'bozena.jankowska@example.pl'),
('Jarosław', 'Wójcik', '502123456', 'jaroslaw.wojcik@example.pl'),
('Jolanta', 'Pawlak', '508234567', 'jolanta.pawlak@example.pl'),
('Paweł', 'Szymańska', '509876543', 'pawel.szymanska@example.pl'),
('Tomasz', 'Zieliński', '507654321', 'tomasz.zielinski@example.pl'),
('Katarzyna', 'Bąk', '509543210', 'katarzyna.bak@example.pl'),
('Wojciech', 'Kaczmarek', '506876543', 'wojciech.kaczmarek@example.pl'),
('Ewa', 'Mazur', '504567890', 'ewa.mazur@example.pl'),
('Julian', 'Mazur', '504567880', 'ewa.mazur@example.pl'),
('Marek', 'Nowicki', '503234567', 'marek.nowicki@example.pl'),
('Barbara', 'Sikora', '502987654', 'barbara.sikora@example.pl'),
('Arkadiusz', 'Sikora', '577907654', 'a.sikora@example.pl'),
('Andrzej', 'Laskowski', '507123789', 'andrzej.laskowski@example.pl'),
('Agnieszka', 'Wasilewska', '509876543', 'agnieszka.wasilewska@example.pl'),
('Piotr', 'Szczepaniak', '505432198', 'piotr.szczepaniak@example.pl'),
('Wojciech', 'Górska', '504567890', 'wojciech.gorska@example.pl'),
('Maria', 'Chmiel', '508987654', 'maria.chmiel@example.pl'),

```
('Michał', 'Stolarz', '506234876', 'michal.stolarz@example.pl'),
('Anna', 'Stolarz', '534981110', 'a.stolarz@example.pl'),
('Alicja', 'Zawisza', '509876543', 'alicja.zawisza@example.pl'),
('Michalina', 'Wróbel', '555123456', 'mm.wrobel@example.pl'),
('Paweł', 'Wróbel', '504123456', 'pawel.wrobel@example.pl'),
('Tadeusz', 'Ławniczak', '507654321', 'tadeusz.lawniczak@example.pl');

DECLARE @P1 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Jan' AND LastName = 'Monet');
DECLARE @P2 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Karina' AND LastName = 'Kowalska-Monet');
DECLARE @P3 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Anna' AND LastName = 'Park');
DECLARE @P4 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Anna' AND LastName = 'Song');
DECLARE @P5 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Marcin' AND LastName = 'Kowalski');
DECLARE @P6 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Julia' AND LastName = 'Kowalska');
DECLARE @P7 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Piotr' AND LastName = 'Szewczyk');
DECLARE @P8 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Katarzyna' AND LastName = 'Szewczyk');
DECLARE @P9 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Marta' AND LastName = 'Marcinkowska');
DECLARE @P10 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Tomasz' AND LastName = 'Marcinkowski');
DECLARE @P11 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Andrzej' AND LastName = 'Kowal');
DECLARE @P12 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Maria' AND LastName = 'Kowal');
DECLARE @P13 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Zofia' AND LastName = 'Drozd');
DECLARE @P14 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Artur' AND LastName = 'Drozd');
DECLARE @P15 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Katarzyna' AND LastName = 'Nowak');
DECLARE @P16 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Gabriel' AND LastName = 'Nowak');
DECLARE @P17 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Piotr' AND LastName = 'Nowakowski');
DECLARE @P18 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Klara' AND LastName = 'Nowakowska');
DECLARE @P19 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Marek' AND LastName = 'Miller');
DECLARE @P20 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Anna' AND LastName = 'Miller');
```

```
DECLARE @P21 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Agnieszka' AND LastName = 'Davis');
DECLARE @P22 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Thomas' AND LastName = 'Davis');
DECLARE @P23 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Tadeusz' AND LastName = 'Malinowski');
DECLARE @P24 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Renata' AND LastName = 'Malinowska');
DECLARE @P25 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Renata' AND LastName = 'Hel');
DECLARE @P26 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Katarzyna' AND LastName = 'Hel');
DECLARE @P27 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Artur' AND LastName = 'García');
DECLARE @P28 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Maria' AND LastName = 'García');
DECLARE @P29 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Kamil' AND LastName = 'Lasek');
DECLARE @P30 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Kamila' AND LastName = 'Lasek');
DECLARE @P31 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Zuzanna' AND LastName = 'Czajkowski');
DECLARE @P32 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Aleksander' AND LastName = 'Czajkowski');
DECLARE @P33 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Tadeusz' AND LastName = 'Smolny');
DECLARE @P34 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Aneta' AND LastName = 'Smolny');
DECLARE @P35 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Anna' AND LastName = 'Gwiazda');
DECLARE @P36 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Magdalena' AND LastName = 'Nowak');
DECLARE @P37 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Jan' AND LastName = 'Wiśniewski');
DECLARE @P38 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Michał' AND LastName = 'Krawczyk');
DECLARE @P39 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Bożena' AND LastName = 'Jankowska');
DECLARE @P40 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Jarosław' AND LastName = 'Wójcik');
DECLARE @P41 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Jolanta' AND LastName = 'Pawlak');
DECLARE @P42 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Paweł' AND LastName = 'Szymańska');
DECLARE @P43 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Tomasz' AND LastName = 'Zieliński');
DECLARE @P44 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Katarzyna' AND LastName = 'Bąk');
```

```
DECLARE @P45 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Wojciech' AND LastName = 'Kaczmarek');
DECLARE @P46 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Ewa' AND LastName = 'Mazur');
DECLARE @P47 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Julian' AND LastName = 'Mazur');
DECLARE @P48 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Marek' AND LastName = 'Nowicki');
DECLARE @P49 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Barbara' AND LastName = 'Sikora');
DECLARE @P50 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Arkadiusz' AND LastName = 'Sikora');
DECLARE @P51 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Andrzej' AND LastName = 'Laskowski');
DECLARE @P52 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Agnieszka' AND LastName = 'Wasilewska');
DECLARE @P53 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Piotr' AND LastName = 'Szczepaniak');
DECLARE @P54 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Wojciech' AND LastName = 'Górska');
DECLARE @P55 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Maria' AND LastName = 'Chmiel');
DECLARE @P56 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Michał' AND LastName = 'Stolarz');
DECLARE @P57 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Anna' AND LastName = 'Stolarz');
DECLARE @P58 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Alicja' AND LastName = 'Zawisza');
DECLARE @P59 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Michalina' AND LastName = 'Wróbel');
DECLARE @P60 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Paweł' AND LastName = 'Wróbel');
DECLARE @P61 INT = (SELECT ParentID FROM Students.Parents WHERE FirstName =
'Tadeusz' AND LastName = 'Ławniczak');

INSERT INTO Students.StudentParents (StudentID, ParentID) VALUES (@Student1, @P1),
(@Student1, @P2), (@Student2, @P1), (@Student2, @P2),
(@Student3, @P3),
(@Student4, @P4),
(@Student5, @P5), (@Student5, @P6),
(@Student6, @P7), (@Student6, @P8),
(@Student7, @P9), (@Student7, @P10),
(@Student8, @P11), (@Student8, @P12),
(@Student9, @P13), (@Student9, @P14),
(@Student10, @P15), (@Student10, @P16),
(@Student11, @P17), (@Student11, @P18),
(@Student12, @P19), (@Student12, @P20),
(@Student13, @P21), (@Student13, @P22),
```

```
(@Student14, @P23), (@Student14, @P24),
(@Student15, @P25), (@Student15, @P26),
(@Student16, @P27), (@Student16, @P28),
(@Student17, @P29), (@Student17, @P30),
(@Student18, @P31), (@Student18, @P32),
(@Student19, @P33), (@Student19, @P34),

(@Student20, @P35),
(@Student21, @P36),
(@Student22, @P37),
(@Student23, @P38),
(@Student24, @P39),
(@Student25, @P40),
(@Student26, @P41),
(@Student27, @P42),
(@Student28, @P43),
(@Student29, @P44),
(@Student30, @P45),
(@Student31, @P46), (@Student31, @P47),
(@Student32, @P48),
(@Student33, @P49), (@Student33, @P50),

(@Student34, @P51),
(@Student35, @P52),
(@Student36, @P53),
(@Student37, @P54),
(@Student38, @P55),
(@Student39, @P56), (@Student39, @P57),
(@Student40, @P58),
(@Student41, @P59),
(@Student42, @P60),
(@Student43, @P61);

EXEC Students.AddStudentGrade
        @StudentID = @Student30,
        @TeacherID = @T1,
        @GradeValue = 4,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student29,
        @TeacherID = @T1,
        @GradeValue = 5,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student31,
        @TeacherID = @T1,
```

```
        @GradeValue = 3,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student32,
        @TeacherID = @T1,
        @GradeValue = 2,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student33,
        @TeacherID = @T1,
        @GradeValue = 3,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student34,
        @TeacherID = @T1,
        @GradeValue = 6,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student35,
        @TeacherID = @T1,
        @GradeValue = 6,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student36,
        @TeacherID = @T1,
        @GradeValue = 1,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student37,
        @TeacherID = @T1,
        @GradeValue = 4,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student38,
        @TeacherID = @T1,
        @GradeValue = 4,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student39,
        @TeacherID = @T1,
```

```
        @GradeValue = 5,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student40,
        @TeacherID = @T1,
        @GradeValue = 2,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student41,
        @TeacherID = @T1,
        @GradeValue = 5,
        @SubjectID = @S1

EXEC Students.AddStudentGrade
        @StudentID = @Student42,
        @TeacherID = @T1,
        @GradeValue = 4,
        @SubjectID = @S1
```