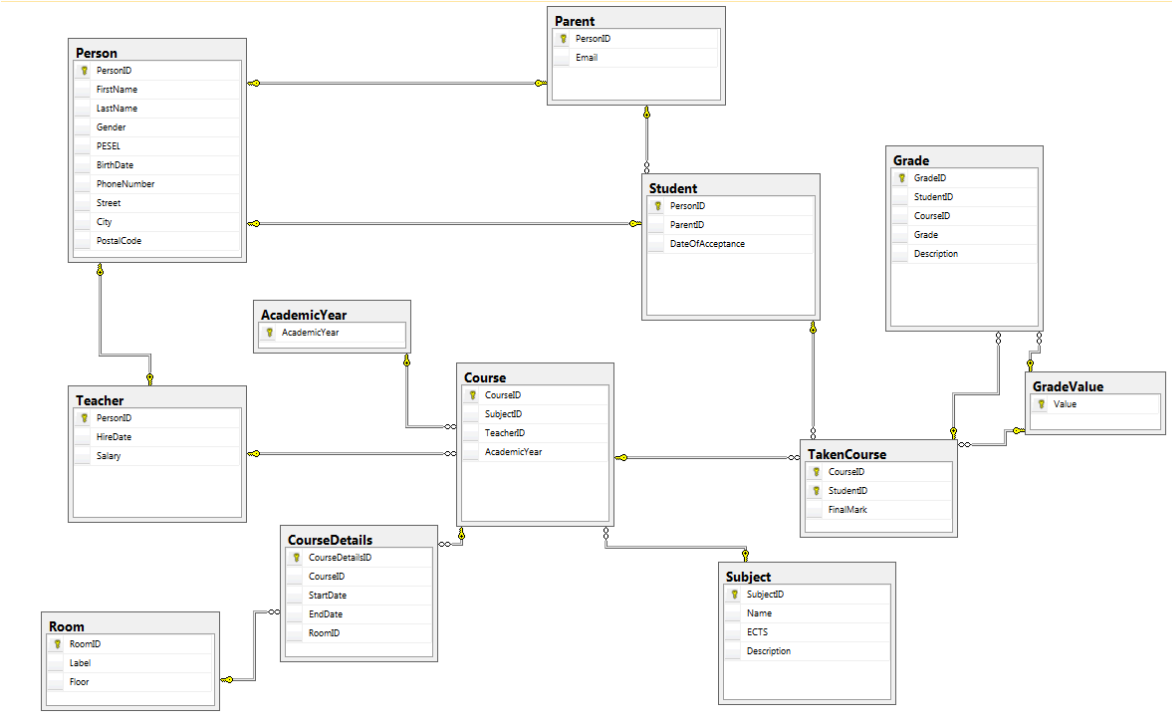
Diagram ER



Schemat bazy danych:

Tabela Person

Column Name	Data Type	Allow Nulls
PersonID	int	☐
FirstName	nvarchar(64)	☐
LastName	nvarchar(64)	☐
Gender	nchar(1)	☑
PESEL	nchar(11)	☐
BirthDate	date	☑
PhoneNumber	nvarchar(16)	☑
Street	nvarchar(32)	☑
City	nvarchar(32)	☑
PostalCode	nchar(6)	☑

Tabela Parent

Column Name	Data Type	Allow Nulls
PersonID	int	☐
Email	nchar(64)	☑

Tabela Teacher

Column Name	Data Type	Allow Nulls
PersonID	int	☐
HireDate	date	☐
Salary	money	☐

Tabela Student

Column Name	Data Type	Allow Nulls
PersonID	int	☐
ParentID	int	☐
DateOfAcceptance	date	☐

Tabela TakenCourse

Column Name	Data Type	Allow Nulls
CourseID	int	☐
StudentID	int	☐
FinalMark	float	☑

Tabela Grade

Column Name	Data Type	Allow Nulls
GradeID	int	☐
StudentID	int	☐
CourseID	int	☐
Grade	float	☐
Description	nvarchar(64)	☑

Tabela GradeValue

Column Name	Data Type	Allow Nulls
Value	float	☐

Tabela Course

Column Name	Data Type	Allow Nulls
CourseID	int	☐
SubjectID	int	☐
TeacherID	int	☐
AcademicYear	nchar(9)	☐

Tabela AcademicYear

Column Name	Data Type	Allow Nulls
AcademicYear	nchar(9)	☐

Tabela Subject

Column Name	Data Type	Allow Nulls
SubjectID	int	☐
Name	nvarchar(64)	☐
ECTS	tinyint	☐
Description	nvarchar(64)	☑

Tabela CourseDetails

Column Name	Data Type	Allow Nulls
CourseDetailsID	int	☐
CourseID	int	☐
StartDate	datetime	☐
EndDate	datetime	☐
RoomID	int	☐

Tabela Room

RoomID	int	☐
Label	nvarchar(8)	☐
Floor	tinyint	☑

Cel projektu

Stworzenie bazy danych na wzór szkoły/uczelni. W bazie danych przechowujemy informacje o nauczycielach, uczniu i jego opiekunie, przedmiotach które zaliczył/aktualnie zalicza uczeń.

Główne założenia/ograniczenia przyjęte przy projektowaniu.

- Osoba może być jednocześnie byłym studentem, aktualnym nauczycielem jak i rodzicem innego ucznia.
- Kurs to po prostu dany przedmiot, podział przedmiotów na kursy jest po to aby był podział na lata akademickie, stąd np. kurs z przedmiotu Programowanie w roku 2021/2022 może być prowadzony przez inną osobę oraz mieć lekcje w innych dniach niż kurs z przedmiotu Programowanie w roku 2022/2023
- Nie ma podziału na klasy, każdy uczeń wybiera na jakie przedmioty chce się zapisać.
- Lekcje z danego kursu nie koniecznie odbywają się w ten sam dzień i o tej samej porze, zajęcia mogą np. odbywać się raz na 2 tygodnie, albo raz w jednej sali raz w drugiej itd.
- Student może mieć wiele ocen z danego kursu, ale ma jedną ocenę końcową, jeśli nie ma oceny końcowej to oznacza, że kurs jeszcze się nie zakończył

Normalizacja, Zależności funkcyjne, Klucze potencjalne

Tabela Person

Klucze potencjalne: PersonID; PESEL
Zależności funkcyjne (nieodklucza): PostalCode -> City
(nieklucz -> nieklucz, zatem jest tylko w 2PN, zakładam, że adresy osób rzadko się powtarzają, dlatego nie normalizuje tej tabeli do BCNF, podobnie adresy są zapisane w bazie danych Northwind, więc raczej jest ok)

Tabela ParentID, Tabela TeacherID, Tabela StudentID

Klucze potencjalne: PersonID
Brak zależności funkcyjnych (nieodklucza)
Jest w BCNF.

Tabela Grade

Klucze potencjalne: GradeID; (StudentID, CourseID)
Zależności (nieodklucza): brak
Jest w BCNF

Tabela TakenCourse

Klucze potencjalne: (CourseID, StudentID)
Zależności funkcyjne (nieodklucza): Brak\
Jest w BCNF.

Tabela Course

Klucze potencjalne: CourseID, (SubjectID, AcademicYear) [może być 1 kurs w danym roku akademickim]
Zależności funkcyjne (nieodklucza): Brak
Jest w BCNF.

Tabela CourseDetails

Klucze potencjalne: CourseDetailsID, (StartDate, RoomID)
Zależności funkcyjne (nieodklucza): Brak
Jest w BCNF.

Tabela Room

Klucze potencjalne: RoomID, Label
Zależności funkcyjne (nieodklucza): Brak
Jest w BCNF.

Tabela Subject

Klucze potencjalne: SubjectID, (Name, Description)
Zależności funkcyjne (nieodklucza): Brak
Jest w BCNF.

Co można zmienić/dodać:

- Zmienić przechowywanie kiedy zaczynają i kończą się zajęcia (niepotrzebnie dla każdych zajęć przechowujemy rok bo i tak znamy rok z AcademicYear)
- Pole AcademicYear lepiej rozbic na 2 pola, zwłaszcza jeśli będzie sporo zapytań, które będzie wybierać początkowy rok i końcowy

Opis widoków, procedur, wyzwalaczy, funkcji

Widoki

```
-- Widok nr 1 - Informacje o każdym nauczycielu, i jakie przedmioty
prowadzi
CREATE VIEW TeachersInfo
AS
SELECT P.PersonID, P.FirstName, P.LastName, T.Salary, T.HireDate,
S.Name, C.AcademicYear
FROM Teacher T JOIN Person P ON T.PersonID = P.PersonID
JOIN Course C ON T.PersonID = C.TeacherID
JOIN [Subject] S ON C.SubjectID = S.SubjectID;
```

```
-- Widok nr 2 - Policz ile każdy kurs ma spotkań oraz łączną długość
spotkań
CREATE VIEW Lessons
AS
      SELECT C.AcademicYear, S.Name, COUNT(*) as NumberOfMeetings,
SUM(DATEDIFF(minute, CD.StartDate, CD.EndDate))/60 as [Length (Hours)]
      FROM Course C JOIN CourseDetails CD ON C.CourseID = CD.CourseID
      JOIN Subject S ON C.SubjectID = S.SubjectID
      GROUP BY C.AcademicYear, S.Name;
```

```
-- Widok nr 3 - Dla każdego rodzica wyświetl jego dzieci
CREATE VIEW ParentChildren
AS
      SELECT Pinfo.FirstName [Parent FirstName], Pinfo.LastName [Parent
LastName], Sinfo.FirstName [Child FirstName], Sinfo.LastName [Child
LastName]
      FROM Parent P JOIN Person Pinfo ON P.PersonID = Pinfo.PersonID
      JOIN Student S ON P.PersonID = S.ParentID
      JOIN Person Sinfo ON Sinfo.PersonID = S.PersonID;
```

```sql
-- Widok nr 4 - Dla każdej osoby kim jest (nauczycielem, studentem itp)
CREATE VIEW IdentifyPerson
AS
	SELECT P.FirstName, P.LastName,
	CASE
		WHEN (SELECT S.PersonID FROM Student S WHERE S.PersonID =
P.PersonID) IS NOT NULL THEN 'YES'
		ELSE 'NO'
	END AS [StudentCheck],
	CASE
		WHEN (SELECT S.PersonID FROM Teacher S WHERE S.PersonID =
P.PersonID) IS NOT NULL THEN 'YES'
		ELSE 'NO'
	END AS [TeacherCheck],
	CASE
		WHEN (SELECT S.PersonID FROM Parent S WHERE S.PersonID =
P.PersonID) IS NOT NULL THEN 'YES'
		ELSE 'NO'
	END AS [ParentCheck]
	FROM Person P;
```

```sql
-- Widok nr 5 - Dla każdego ucznia oblicz średnią ocen w danym kursie, na
którym zapisany był uczeń
CREATE VIEW AverageGrades
AS
	SELECT S.PersonID, P.FirstName, P.LastName, G.CourseID,
CAST(AVG(G.Grade) as decimal(10,2)) as [Average]
	FROM Student S
	JOIN Person P ON S.PersonID = P.PersonID
	JOIN TakenCourse TC ON S.PersonID = TC.StudentID
	JOIN Grade G ON TC.StudentID = G.StudentID AND TC.CourseID =
G.CourseID
	GROUP BY S.PersonID, P.FirstName, P.LastName, G.CourseID;
```

Procedury

```
-- Procedura nr 1 - Dla konkretnego ucznia wypisz jego plan lekcji w
przedziale pomiędzy zadanymi datami
CREATE PROCEDURE StudentSchedule @StudentID int, @StartDate datetime,
@EndDate datetime
AS
SET @EndDate = DATEADD(day, 1, @EndDate);--zwiekszam o jeden dzien, i
teraz tu będzie jeden dzien pozniej, ale godzina 00:00:00
SELECT * FROM Student S
JOIN TakenCourse TC ON S.PersonID = TC.StudentID
JOIN Course C ON C.CourseID = TC.CourseID
JOIN CourseDetails CD ON CD.CourseID = C.CourseID
WHERE TC.FinalMark IS NULL --jeszcze nie ma oceny końcowej więc to
oznacza, że kurs się jeszcze nie zakończył czyli powinien być w planie
lekcji
AND S.PersonID = @StudentID
AND CD.StartDate >= @StartDate AND CD.StartDate < @EndDate;

--EXEC StudentSchedule @StudentID = 7, @StartDate='2020-04-04', @EndDate
= '2022-11-18';
```

```
-- Procedura nr 2 - Dla konkretnego ucznia zwraca jakie przedmioty już
ma zakończone
CREATE PROCEDURE StudentCompleted @StudentID int
AS
SELECT S.PersonID, P.FirstName, P.LastName, C.AcademicYear, SB.Name,
TC.FinalMark FROM Student S
JOIN TakenCourse TC ON S.PersonID = TC.StudentID
JOIN Course C ON C.CourseID = TC.CourseID
JOIN [Subject] SB ON C.SubjectID = SB.SubjectID
JOIN Person P ON S.PersonID = P.PersonID
WHERE TC.FinalMark IS NOT NULL
AND S.PersonID = @StudentID

--EXEC StudentCompleted @StudentID = 7;
```

```sql
--Funkcja nr 1 (potrzebna do kolejnej procedury) - Zwraca daty z
zadanego przedziału, z odstępami o interwał
CREATE FUNCTION GenerateDatesInteval (@StartDate date, @EndDate date,
@Interval tinyint)
RETURNS TABLE
AS
RETURN
(
	WITH MY_CTE
	AS
	(SELECT @StartDate StartDate
	UNION ALL
	SELECT DATEADD(day, @Interval, StartDate)
	FROM MY_CTE
	WHERE DATEADD(day, @Interval, StartDate) <= @EndDate
	)
	SELECT * FROM MY_CTE
)
--SELECT * FROM dbo.GenerateDatesInteval('2023-08-09','2023-08-16',7);
```

```sql
-- Procedura nr 3 - Dodaje jakiś przedmiot np. w każdy wtorek o 12:00;
jeśli jest kolizja z salą to wtedy wyzwalacz zablokuje;
CREATE PROCEDURE AddCourseLesson @CourseID int, @DayOfTheWeek
nvarchar(16), @LessonStart time, @LessonEnd time, @StartDate date,
@EndDate date,
@RoomID int
AS
--Znalezienie pierwszego 'dobrego dnia'
WHILE(DATENAME(DW,@StartDate) != @DayOfTheWeek)
BEGIN
	SET @StartDate = DATEADD(day, 1, @StartDate)
END
IF @StartDate <= @EndDate
BEGIN
	INSERT INTO CourseDetails(CourseID, StartDate, EndDate, RoomID)
SELECT @CourseID, CAST(F.StartDate as datetime) + CAST(@LessonStart as
datetime),
	CAST(F.StartDate as datetime) + CAST(@LessonEnd as datetime),
@RoomID
	FROM dbo.GenerateDatesInteval(@StartDate, @EndDate, 7) F;
END
--EXEC AddCourseLesson @CourseID = 5, @DayOfTheWeek = 'Monday',
@LessonStart = '12:00:00', @LessonEnd = '13:00:00',
--@StartDate='2023-11-04', @EndDate = '2023-11-27', @RoomID = 1
```

```sql
-- Procedura nr 4 - dla podanego id studenta wypisuje informacje o nim, jego rodzicu i jego rodzeństwu
CREATE PROCEDURE StudentFamily @StudentID int
AS
	DECLARE @ParentID int;
	SET @ParentID = (SELECT S.ParentID FROM Student S WHERE S.PersonID = @StudentID);

		SELECT P.*, 'Student' as Relation FROM Student S
		JOIN Person P ON P.PersonID = S.PersonID
		WHERE S.ParentID = @ParentID
	UNION
		SELECT P.*, 'Parent' as Relation FROM Person P
		WHERE P.PersonID = @ParentID
		ORDER BY Relation;

--EXEC StudentFamily 6;
```

```sql
-- Procedura nr 5 - dla podanego id studenta i id kursu wypisuje oceny studenta, oraz średnią ocen w osobnym polu
CREATE PROCEDURE CourseGrades @StudentID int, @CourseID int
AS
	SELECT G.*, (SELECT AG.Average FROM AverageGrades AG WHERE AG.PersonID = @StudentID AND AG.CourseID = @CourseID) AS [Average]
	FROM Grade G
	WHERE G.StudentID = @StudentID AND G.CourseID = @CourseID

--EXEC CourseGrades @StudentID = 6, @CourseID = 1;
```

Wyzwalacze

```
-- Wyzwalacz nr 1 - Sprawdza czy student nie zaliczył już danego
przedmiotu w innym roku akademickim
CREATE TRIGGER SubjectAlreadyCompleted
ON TakenCourse
AFTER INSERT
AS
BEGIN
	DECLARE @counter int;
	SET @counter = (--zliczam ile razy student zapisał się na
przedmiot, jeśli jakikolwiek student zapisał się wiecej niż raz to
rollback
	SELECT MAX([counter])
		FROM (
			SELECT TC.StudentID, C.SubjectID , COUNT(*) [counter]
			FROM TakenCourse TC
			JOIN Course C ON TC.CourseID = C.CourseID
			WHERE TC.StudentID IN (SELECT StudentID FROM inserted
WHERE StudentID IS NOT NULL)--zeby sprawdzac tylko dla studentow ktorych
dotyczy zmiana
			AND (TC.FinalMark IS NULL OR TC.FinalMark != 2) --jeśli
juz zaliczał, ale dostał ocene 2 to może poprawić więc jest ok
			GROUP BY TC.StudentID, C.SubjectID
		)Subquery
	)

	IF @counter > 1
		ROLLBACK
END;
```

```
-- Wyzwalacz nr 2 - Sprawdza czy rodzic podał adres email jeśli nie podał
nr telefonu
CREATE TRIGGER ParentAnyInfo
ON Parent
AFTER INSERT
AS
	IF EXISTS(
		SELECT * FROM Person PE JOIN inserted PA ON PE.PersonID =
PA.PersonID
		WHERE PE.PhoneNumber IS NULL AND PA.Email IS NULL
	)
		ROLLBACK
;
```

```
-- Wyzwalacz nr 3 - Sprawdza czy sala nie jest już zajęta przez inne
zajęcia
CREATE TRIGGER RoomTaken
ON CourseDetails
AFTER INSERT
AS
BEGIN
IF EXISTS(
      SELECT * FROM inserted I
      CROSS APPLY(
            SELECT * FROM CourseDetails CD
            WHERE CD.CourseDetailsID != I.CourseDetailsID AND I.RoomID =
CD.RoomID AND ( (I.StartDate > CD.StartDate AND I.StartDate <
CD.EndDate) OR (I.EndDate > CD.StartDate AND I.EndDate < CD.EndDate))
OR (I.StartDate <= CD.StartDate AND I.EndDate >= CD.EndDate)
      )Subquery
)
BEGIN
      RAISERROR('Room already taken!.', 11, 2)
      ROLLBACK
END
END;
```

```
-- Wyzwalacz nr 4 - Sprawdza czy zajęcia które dodajemy dla danego kursu
rzeczywiście są w odpowiednim roku akademickim
CREATE TRIGGER AcademicYearLessons
ON CourseDetails
AFTER INSERT
AS
BEGIN
      IF EXISTS(
            SELECT * FROM inserted I
            JOIN Course C ON I.CourseID = C.CourseID
            WHERE CAST(StartDate as date) NOT BETWEEN
            CAST(LEFT(C.AcademicYear, 4) + '-09-01' as date) AND
CAST(RIGHT(C.AcademicYear, 4) + '-06-30' as date) OR
            CAST(EndDate as date) NOT BETWEEN
            CAST(LEFT(C.AcademicYear, 4) + '-09-01' as date) AND
CAST(RIGHT(C.AcademicYear, 4) + '-06-30' as date)
      )BEGIN
            RAISERROR('Wrong Academic Year !', 10, 1)
            ROLLBACK
      END
END;
```

```
--Wyzwalacz nr 5 - Sprawdzenie czy jeśli student dostaje ocene końcową z
kursu z roku akademickiego który już się skończył to nie jest to null
CREATE TRIGGER CorrectFinalMark
ON TakenCourse
AFTER INSERT
AS
BEGIN
	IF EXISTS(
		SELECT * FROM inserted I
		JOIN Course C ON I.CourseID = C.CourseID
		WHERE FinalMark IS NULL
		AND C.AcademicYear != (SELECT TOP 1 AC.AcademicYear FROM
AcademicYear AC ORDER BY LEFT(AC.AcademicYear, 4) DESC)
	)
		ROLLBACK;
END;
```

Funkcje

```
--Funkcja nr 2 - Dla konkretnego ucznia wypisuje średnią ocen końcowych
CREATE FUNCTION AverageFinalMarks
(@StudentID int)
RETURNS int
AS
BEGIN
	RETURN (SELECT AVG(TC.FinalMark) FROM TakenCourse TC WHERE
TC.StudentID = @StudentID AND TC.FinalMark IS NOT NULL GROUP BY
TC.StudentID)
END
```

```
-- Funkcja nr 5 - Dla podanej kwoty wypisuje nauczycieli zarabiających
tyle lub więcej
CREATE FUNCTION TeacherSalary
(@Salary money)
RETURNS TABLE
AS
RETURN (
	SELECT P.FirstName, P.LastName, T.HireDate, T.Salary
	FROM Teacher T JOIN Person P ON T.PersonID = P.PersonID
	WHERE T.Salary >= @Salary
)
```

```sql
--Funkcja nr 3 - Dla podanego AcademicYear wypisuje wszystkie dostępne
kursy
CREATE FUNCTION AvailableCourses
(@AcademicYear varchar(64))
RETURNS TABLE
AS
RETURN (
            SELECT C.AcademicYear, P.FirstName as [Lecturer's Name],
P.LastName [Lecturer's Surname], S.Name FROM Course C
            JOIN Subject S ON C.SubjectID = S.SubjectID
            JOIN Teacher T ON C.TeacherID = T.PersonID
            JOIN Person P ON T.PersonID = P.PersonID
            WHERE C.AcademicYear = @AcademicYear
      );
```

```sql
-- Funkcja nr 4 - Dla podanego pokoju i dnia wyświetla kiedy sala jest
zajęta
CREATE FUNCTION AvailableHours
(@RoomID int, @GivenDate date)
RETURNS @TempTable TABLE
(
      CourseID int,
      StartDate datetime,
      EndDate datetime
)
AS
BEGIN
      INSERT INTO @TempTable
            SELECT CD.CourseID, CD.StartDate, CD.EndDate FROM
CourseDetails CD
            WHERE CD.RoomID = @RoomID AND CD.StartDate >= @GivenDate AND
CD.StartDate < dateadd(day,1,@GivenDate)
      RETURN
END
```