Dokumentacja

Oktawiusz Doroszuk Wojciech Kaźmierczak
Bartłomiej Stylski

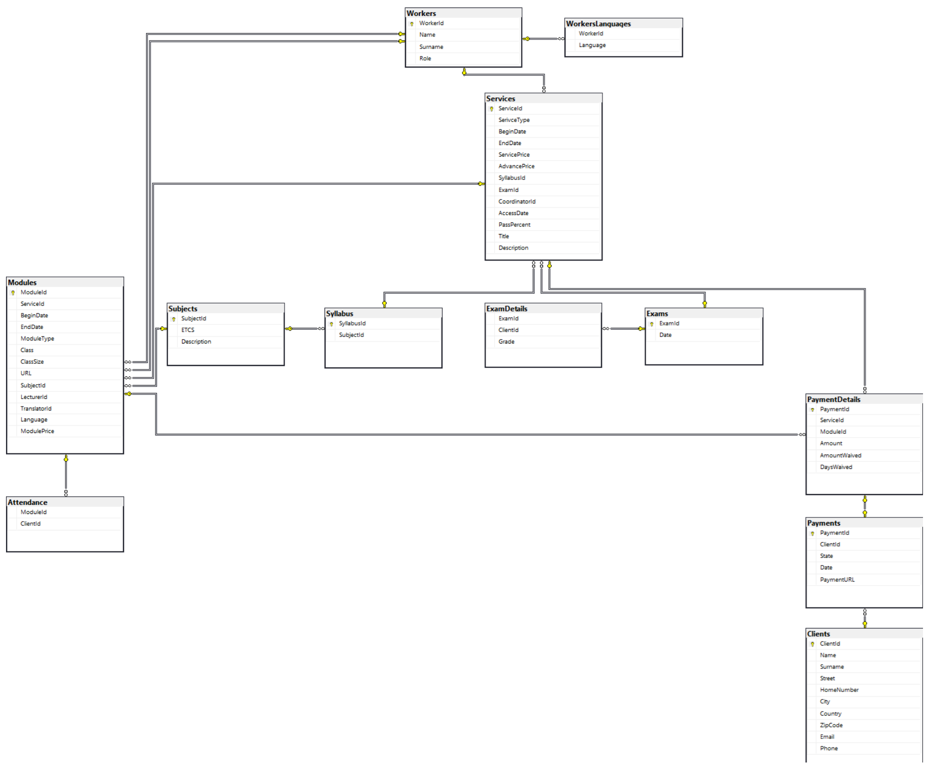
styczeń 2024

Spis treści

1 Rozkład pracy

- Oktawiusz Doroszuk
  1. Tabele - 2 %
  2. Procedury - 7 %
  3. Dokumentacja - 9 %
  4. Generowanie danych 15 %
- Wojciech Kaźmierczak
  1. Funkcje - 20 %
  2. Indeksy - 6 %
  3. Użytkownicy i uprawnienia - 7 %
- Bartłomiej Stylski
  1. Warunki integralności - 9%
  2. Widoki - 14 %
  3. Triggery - 10 %

2 Schemat bazy danych



3 Opis tabeli i warunki integralnościowe

3.1 Clients

Clients	
ClientId	Primary key Auto increment Unique Int > 0
Name	VARCHAR(50)
Surname	VARCHAR(50)
Street	Allow nulls VARCHAR(50)
HomeNumber	VARCHAR(50)
City	VARCHAR(50)
Country	VARCHAR(50)
ZipCode	VARCHAR(6) - wartość musi mieć dokładnie 6 znaków
Email	VARCHAR(50)
Phone	VARCHAR(11) - wartość nie może mieć mniej niż 9 znaków

```sql
CREATE TABLE [dbo].[Clients] (
    [ClientId]   INT          NOT NULL,
    [Name]       VARCHAR (50) NOT NULL,
    [Surname]    VARCHAR (50) NOT NULL,
    [Street]     VARCHAR (50) NULL,
    [HomeNumber] VARCHAR (50) NOT NULL,
    [City]       VARCHAR (50) NOT NULL,
    [Country]    VARCHAR (50) NOT NULL,
    [ZipCode]    VARCHAR (6)  NOT NULL,
    [Email]      VARCHAR (50) NOT NULL,
    [Phone]      VARCHAR (11) NOT NULL,
    CONSTRAINT [PK_Clients] PRIMARY KEY CLUSTERED ([ClientId] ASC),
    CONSTRAINT [Phone] CHECK (len([Phone])>=(9)),
    CONSTRAINT [ZipCode] CHECK (len([ZipCode])=(6))
);
```

3.2 Payments

Payments	
PaymentId	Primary key Auto Increment Unique Int > 0
ClientId	Int > 0 Foreign key do tabeli Clients (ClientId)
State	Tinyint - wartość może być: 0 - płatność w trakcie 1 - płatność zakończona sukcesem 2 - płatność zakończona niepowodzeniem
Date	DateTime - data i czas rozpoczęcia transakcji
PaymentURL	VARCHAR(MAX) - link do płatności

```sql
CREATE TABLE [dbo].[Payments] (
    [PaymentId]  INT           NOT NULL,
    [ClientId]   INT           NOT NULL,
    [State]      TINYINT       NOT NULL,
    [Date]       DATETIME      NOT NULL,
    [PaymentURL] VARCHAR (MAX) NOT NULL,
    CONSTRAINT [PK_Payments] PRIMARY KEY CLUSTERED ([PaymentId] ASC
    ),
    CONSTRAINT [State] CHECK ([State] IN (0, 1, 2)),
    CONSTRAINT [FK_Payments_Clients] FOREIGN KEY ([ClientId])
    REFERENCES [dbo].[Clients] ([ClientId]),
    CONSTRAINT [FK_Payments_PaymentDetails] FOREIGN KEY ([PaymentId
    ]) REFERENCES [dbo].[PaymentDetails] ([PaymentId])
);
```

3.3 PaymentDetails

PaymentDetails	
PaymentId	Primary key Unique Int > 0 Foreign key do tabeli Payments(PaymentId)
ServiceId	Int > 0 Allow nulls Foreign key do tabeli Services (ServiceId)
ModuleId	Int > 0 Allow nulls Foreign key do tabeli Modules (ModuleId)
Amount	Money > 0
AmountWaived	Allow nulls Money > 0
DaysWaived	Allow nulls Int > 0

```
CREATE TABLE [dbo].[PaymentDetails] (
    [PaymentId]    INT   NOT NULL,
    [ServiceId]    INT   NULL,
    [ModuleId]     INT   NULL,
    [Amount]       MONEY NOT NULL,
    [AmountWaived] MONEY NULL,
    [DaysWaived]   INT   NULL,
    CONSTRAINT [PK_PaymentDetails] PRIMARY KEY CLUSTERED ([PaymentId] ASC),
    CONSTRAINT [Amount] CHECK (Amount > 0),
    CONSTRAINT [AmountWaived] CHECK (AmountWaived > 0),
    CONSTRAINT [DaysWaived] CHECK (DaysWaived > 0),
    CONSTRAINT [FK_PaymentDetails_Modules] FOREIGN KEY ([ModuleId]) REFERENCES [dbo].[Modules] ([ModuleId]),
    CONSTRAINT [FK_PaymentDetails_Services] FOREIGN KEY ([ServiceId]) REFERENCES [dbo].[Services] ([ServiceId])
);
```

3.4 Services

Services	
ServiceId	Primary key Auto increment Unique Int $>$ 0
ServiceType	Tinyint ze zbioru {0, 1, 2} kolejno oznaczająca webinar, kurs, studia
BeginDate	Date
EndDate	Date $>$ BeginDate
ServicePrice	Money $>$ 0
AdvancePrice	ServicePrice $>=$ Money $>=$ 0 - zaliczka
SyllabusId	Int $>$ 0
ExamId	Int $>$ 0 Foreign key do tabeli Exams(ExamId)
CoordinatorId	Int $>$ 0 Foreign key do tabeli Workers(WorkerId)
AccessDate	Date $>$ EndDate jeśli ServiceType jest webinarem, w przeciwnym przypadku Null
PassPercent	100 $>=$ Tinyint $>=$ 0
Title	VARCHAR(MAX)
Description	VARCHAR(MAX)

```
CREATE TABLE [dbo].[Services] (
    [ServiceId]     INT          NOT NULL,
    [ServiceType]   TINYINT      NOT NULL,
    [BeginDate]     DATE         NOT NULL,
    [EndDate]       DATE         NOT NULL,
    [ServicePrice]  MONEY        NOT NULL,
    [AdvancePrice]  MONEY        NOT NULL,
    [SyllabusId]    INT          NULL,
    [ExamId]         INT           NULL,
    [CoordinatorId]  INT           NULL,
    [AccessDate]     DATE          NULL,
    [PassPercent]    TINYINT       NOT NULL,
    [Title]          VARCHAR (50)  NOT NULL,
    [Description]    VARCHAR (50)  NOT NULL,
    CONSTRAINT [PK_Services] PRIMARY KEY CLUSTERED ([ServiceId] ASC
    ),
    CONSTRAINT [AccessDate] CHECK ([AccessDate] IS NULL OR [
    ServiceType]=(0) AND [AccessDate]>[EndDate]),
    CONSTRAINT [AdvancePrice] CHECK ([AdvancePrice]>=(0) AND [
    AdvancePrice]<=[ServicePrice]),
    CONSTRAINT [EndDate] CHECK ([EndDate]>=[BeginDate]),
    CONSTRAINT [PassPercent] CHECK ([PassPercent]>=(0) AND [
    PassPercent]<=(100)),
    CONSTRAINT [ServicePrice] CHECK ([ServicePrice]>(0)),
    CONSTRAINT [ServiceType] CHECK ([ServiceType]=(2) OR [
    ServiceType]=(1) OR [ServiceType]=(0)),
    CONSTRAINT [FK_Services_Exams] FOREIGN KEY ([ExamId])
    REFERENCES [dbo].[Exams] ([ExamId]),
    CONSTRAINT [FK_Services_Workers] FOREIGN KEY ([CoordinatorId])
    REFERENCES [dbo].[Workers] ([WorkerId])
);
```

3.5 Modules

Modules	
ModuleId	Primary key Auto Increment Int > 0
ServiceId	Int > 0 Foreign key do tabeli Services(ServiceId)
ModuleBeginDate	DateTime
ModuleEndDate	DateTime $>$ BeginDate
ModuleType	Tinyint - wartość może być 0 - stacjonarne 1 - zdalnie
Class	VARCHAR(50)
ClassSize	Tinyint > 0
URL	VARCHAR(MAX)
SubjectId	Int > 0 Foreign key do tabeli Subjects(SubjectId)
LecturerId	Int > 0 Foreign key do tabeli Workers(WorkerId)
TranslatorId	Int > 0 Foreign key do tabeliu Workers(WorkerId)
Language	VARCHAR(2) - wartość ma dokładnie 2 znaki Przetrzymuje kod języka Może mieć wartość tylko z puli dostępnych języków
ModulePrice	Money $>= 0$

```
CREATE TABLE [dbo].[Modules] (
    [ModuleId]        INT           NOT NULL,
    [ServiceId]       INT           NOT NULL,
    [ModuleBeginDate] DATETIME      NOT NULL,
    [ModuleEndDate]   DATETIME      NOT NULL,
    [ModuleType]      TINYINT       NOT NULL,
    [Class]           VARCHAR (50)  NOT NULL,
    [ClassSize]       TINYINT       NOT NULL,
    [URL]             VARCHAR (MAX) NULL,
    [SubjectId]       INT           NOT NULL,
    [LecturerId]      INT           NOT NULL,
    [TranslatorId]    INT           NULL,
    [Language]        VARCHAR (2)   NULL,
    [ModulePrice]     MONEY         NOT NULL,
    CONSTRAINT [PK_Modules] PRIMARY KEY CLUSTERED ([ModuleId] ASC),
    CONSTRAINT [ClassSize] CHECK ([ClassSize]>(0)),
    CONSTRAINT [Language] CHECK ([Language] IS NULL OR len([
    Language])=(2)),
    CONSTRAINT [ModuleEndDate] CHECK ([ModuleEndDate]>=[
    ModuleBeginDate]),
    CONSTRAINT [ModulePrice] CHECK ([ModulePrice]>=(0)),
    CONSTRAINT [ModuleType] CHECK ([ModuleType]=(1) OR [ModuleType
    ]=(0)),
    CONSTRAINT [FK_Modules_Services] FOREIGN KEY ([ServiceId])
    REFERENCES [dbo].[Services] ([ServiceId]),
    CONSTRAINT [FK_Modules_Subjects] FOREIGN KEY ([SubjectId])
    REFERENCES [dbo].[Subjects] ([SubjectId]),
    CONSTRAINT [FK_Modules_Workers_Lecturers] FOREIGN KEY ([
    LecturerId]) REFERENCES [dbo].[Workers] ([WorkerId]),
    CONSTRAINT [FK_Modules_Workers_Translators] FOREIGN KEY ([
    TranslatorId]) REFERENCES [dbo].[Workers] ([WorkerId])
);
```

3.6 Exams

Exams	
ExamId	Primary key Auto Increment Unique Int > 0
Date	DateTime

```
CREATE TABLE [dbo].[Exams] (
    [ExamId] INT      NOT NULL,
    [Date]   DATETIME NULL,
    CONSTRAINT [PK_Exams] PRIMARY KEY CLUSTERED ([ExamId] ASC)
);
```

3.7 ExamDetails

ExamDetails	
ExamId	Int > 0 Foreign key do tabeli Exams(ExamId)
ClientId	Int > 0
Grade	Float wartości - {2.0, 3.0, 3.5, 4.0, 4.5, 5.0}

```sql
CREATE TABLE [dbo].[ExamDetails] (
    [ExamId]   INT        NOT NULL,
    [ClientId] INT        NOT NULL,
    [Grade]    FLOAT (53) NOT NULL,
    CONSTRAINT [Grade] CHECK ([Grade] IN (2.0, 3.0, 3.5, 4.0, 4.5,
    5.0)),
    CONSTRAINT [FK_ExamDetails_Exams] FOREIGN KEY ([ExamId])
    REFERENCES [dbo].[Exams] ([ExamId])
);
```

3.8 Workers

Workers	
WorkerId	Primary key Auto Increment Unique Int > 0
Name	VARCHAR(50)
Surname	VARCHAR(50)
Role	Tinyint - wartość może być: 0 - Lecturer 1 - Translator 2 - Coordinator

```sql
CREATE TABLE [dbo].[Workers] (
    [WorkerId] INT          NOT NULL,
    [Name]     VARCHAR (50) NOT NULL,
    [Surname]  VARCHAR (50) NOT NULL,
    [Role]     TINYINT      NOT NULL,
    CONSTRAINT [PK_Workers] PRIMARY KEY CLUSTERED ([WorkerId] ASC),
    CONSTRAINT [Role] CHECK ([Role] IN (0, 1, 2))
);
```

3.9 Subjects

Subjects	
SubjectId	Primary key
	Auto Increment
	Unique
	Int $>$ 0
ECTS	20 $>=$ Tinyint $>=$ 0
Description	VARCHAR(50)

```
CREATE TABLE [dbo].[Subjects] (
    [SubjectId]   INT          NOT NULL,
    [ECTS]        TINYINT      NOT NULL,
    [Description] VARCHAR (50) NOT NULL,
    CONSTRAINT [PK_Subjects] PRIMARY KEY CLUSTERED ([SubjectId] ASC
    ),
    CONSTRAINT [ECTS] CHECK ([ECTS] >= 0 AND [ECTS] <= 20)
);
```

3.10 Syllabus

Syllabus	
SyllabusId	Int $>$ 0
SubjectId	Int $>$ 0

```
CREATE TABLE [dbo].[Syllabus] (
    [SyllabusId] INT NOT NULL,
    [SubjectId]  INT NOT NULL,
);
```

3.11 WorkersLanguages

WorkersLanguages	
WorkerId	Int $>$ 0
	Foreign key do tabeli Workers(WorkerId)
WorkerLanguage	VARCHAR(2) - wartość ma dokładnie 2 znaki
	Przetrzymuje kod języka
	Może mieć wartość tylko z puli dostępnych języków

```
CREATE TABLE [dbo].[WorkersLanguages] (
    [WorkerId]       INT         NOT NULL,
    [WorkerLanguage] VARCHAR (2) NOT NULL,
    CONSTRAINT [WorkerLanguage] CHECK (len([WorkerLanguage])=(2)),
    CONSTRAINT [FK_WorkersLanguages_Workers] FOREIGN KEY ([WorkerId
    ]) REFERENCES [dbo].[Workers] ([WorkerId])
);
```

3.12 Attendance

Attendance	
ModuleId	Int > 0 Foreign key do tabeli Modules(ModuleId)
ClientId	Int > 0

```sql
CREATE TABLE [dbo].[Attendance] (
    [ModuleId] INT NOT NULL,
    [ClientId] INT NOT NULL,
    CONSTRAINT [FK_Attendance_Modules] FOREIGN KEY ([ModuleId])
    REFERENCES [dbo].[Modules] ([ModuleId])
);
```

4 Generowanie danych

Dane zostały wygenerowane losowo za pomocą prostego skryptu w pythonie.

5 Widoki

5.1 Widok wszystkich przyszłych modułów

```sql
SELECT Services.Title,Modules.ModuleBeginDate, Modules.
    ModuleEndDate,Modules.ModulePrice
FROM Modules
JOIN Services ON Modules.ServiceId = Services.ServiceId
WHERE Modules.ModuleEndDate >= CURRENT_TIMESTAMP;
```

5.2 Widok wszystkich obecnie dostępnych usług

```sql
CREATE VIEW [dbo].[AvailableServicesView] AS
SELECT DISTINCT Services.Title,Services.Description, Services.
    ServiceType,Services.BeginDate,Services.EndDate,Services.
    ServicePrice
FROM Services
JOIN Modules ON Services.ServiceId = Modules.ServiceId
WHERE Modules.ModuleEndDate >= CURRENT_TIMESTAMP;
```

5.3 Widok wszystkich modułow, przy których pracuje dany pracownik

```sql
CREATE VIEW [dbo].[WorkersModulesView] AS
SELECT Workers.WorkerId, Workers.Name, Workers.Surname, Modules.
    ModuleId
FROM Workers
LEFT JOIN Modules ON Workers.WorkerId = Modules.LecturerId OR
    Workers.WorkerId = Modules.TranslatorId;
```

5.4 Widok wszystkich uczestników studiów, którzy zdali

```sql
CREATE VIEW [dbo].[PassedStudentsView] AS
SELECT Clients.ClientId, Clients.Name,Clients.Surname,ExamDetails.
    Grade
FROM Clients
JOIN ExamDetails ON Clients.ClientId = ExamDetails.ClientId
JOIN Exams ON ExamDetails.ExamId = Exams.ExamId
WHERE ExamDetails.Grade > 2 ;
```

5.5 Widok wszystkich uczestników studiów, którzy nie zdali

```sql
CREATE VIEW [dbo].[FailedStudentsView] AS
SELECT Clients.ClientId, Clients.Name,Clients.Surname,ExamDetails.
    Grade
FROM Clients
JOIN ExamDetails ON Clients.ClientId = ExamDetails.ClientId
JOIN Exams ON ExamDetails.ExamId = Exams.ExamId
WHERE ExamDetails.Grade = 2 ;
```

5.6 Widok wszystkich studiów

```sql
CREATE VIEW [dbo].[AllStudiesView] AS
SELECT Services.ServiceId,Services.BeginDate, Services.EndDate,
    Services.ServicePrice, Workers.Name AS CoordinatorName, Workers
    .Surname AS CoordinatorSurname
FROM Services
JOIN Workers ON Services.CoordinatorId = Workers.WorkerId
WHERE Services.ServiceType = 2;
```

5.7 Widok wszystkich webinarów

```sql
CREATE VIEW [dbo].[AllWebinarsView] AS
SELECT Services.ServiceId,Services.BeginDate,
Services.EndDate,
Services.ServicePrice
FROM Services
WHERE Services.ServiceType = 0;
```

5.8 Widok wszystkich kursów

```sql
CREATE VIEW [dbo].[AllCoursesView] AS
SELECT Services.ServiceId,Services.BeginDate,
Services.EndDate,
Services.ServicePrice
FROM Services
WHERE Services.ServiceType = 1;
```

5.9 Widok dochodów z każdego serwisu

```sql
CREATE VIEW [dbo].[ServiceRevenueView] AS
SELECT Services.ServiceId, Services.Title, COALESCE(SUM(
    PaymentDetails.Amount), 0) AS TotalRevenue
FROM Services
LEFT JOIN PaymentDetails ON Services.ServiceId = PaymentDetails.
    ServiceId
GROUP BY Services.ServiceId, Services.Title;
```

5.10 Widok wszystkich przychodów z danego typu modułu

```sql
CREATE VIEW [dbo].[ModuleTypeRevenueView] AS
SELECT Modules.ModuleType, COALESCE(SUM(PaymentDetails.Amount), 0)
    AS TotalRevenue
FROM Modules
LEFT JOIN PaymentDetails ON Modules.ModuleId = PaymentDetails.
    ModuleId
GROUP BY Modules.ModuleType;
```

5.11 Widok niedokończonych płatności

```sql
CREATE VIEW [dbo].[UnpaidClientsView] AS
SELECT DISTINCT c.ClientId, c.Name, c.Surname, SUM(Amount - ISNULL(
    AmountWaived, 0)) AS 'Zaleglosci'
FROM Payments p
INNER JOIN Clients c ON c.ClientId = p.PaymentId
INNER JOIN PaymentDetails pd ON p.PaymentId = pd.PaymentId
WHERE p.[State] = 0
GROUP BY c.ClientId, c.Name, c.Surname
```

6 Procedury

6.1 Dodanie modułu

```sql
CREATE PROCEDURE [dbo].[AddModule]
    @ServiceId int,
    @BeginDate datetime,
    @EndDate datetime,
    @ModuleType tinyint,
    @Class VARCHAR(50),
    @ClassSize tinyint,
    @SubjectId int,
    @LecturerId int,
    @ModulePrice money,
    @TranslatorId int = NULL,
    @Language VARCHAR(50) = NULL
AS
BEGIN
```

```
    INSERT INTO Modules (ServiceId, ModuleBeginDate, ModuleEndDate,
     ModuleType, Class, ClassSize, SubjectId, LecturerId,
    ModulePrice, TranslatorId, [Language])
    VALUES (@ServiceId, @BeginDate, @EndDate, @ModuleType, @Class,
    @ClassSize, @SubjectId, @LecturerId, @ModulePrice,
    @TranslatorId, @Language)
END
```

6.2 Dodanie kursu

```
CREATE PROCEDURE [dbo].[AddService]
    @ServiceType tinyint,
    @BeginDate date,
    @EndDate date,
    @ServicePrice money,
    @AdvancePrice money,
    @SyllabusId int = NULL,
    @ExamId int = NULL,
    @CoordinatorId int = NULL,
    @PassPercent tinyint,
    @Title VARCHAR(50),
    @Description VARCHAR(50)
AS
BEGIN
    INSERT INTO Services (ServiceType, BeginDate, EndDate,
    ServicePrice, AdvancePrice, SyllabusId, ExamId, CoordinatorId,
    PassPercent, Title, [Description])
    VALUES (@ServiceType, @BeginDate, @EndDate, @ServicePrice,
    @AdvancePrice, @SyllabusId, @ExamId, @CoordinatorId,
    @PassPercent, @Title, @Description)
END
```

6.3 Dodanie webinaru

```
CREATE PROCEDURE [dbo].[AddWebinar]
    @ModuleBeginDate datetime,
    @ModuleEndDate datetime,
    @AccessDate datetime = NULL,
    @ServicePrice money,
    @AdvancePrice money,
    @WebinarSize tinyint,
    @SubjectId int,
    @LecturerId int,
    @TranslatorId int = NULL,
    @Language VARCHAR(50) = NULL,
    @Title VARCHAR(50),
    @Description VARCHAR(50)
AS
BEGIN
    DECLARE @ServiceType tinyint = 0
    DECLARE @ModuleType tinyint = 1
    DECLARE @ServiceBeginDate date = CAST(@ModuleBeginDate AS date)
    DECLARE @ServiceEndDate date = CAST(@ModuleEndDate AS date)
    DECLARE @Output TABLE (ServiceId int)

    INSERT INTO Services (ServiceType, BeginDate, EndDate,
    ServicePrice, AdvancePrice, Title, [Description])
    OUTPUT INSERTED.ServiceId INTO @Output(ServiceId)
    VALUES (@ServiceType, @ServiceBeginDate, @ServiceEndDate,
    @ServicePrice, @AdvancePrice, @Title, @Description)

    DECLARE @ServiceId int = (SELECT ServiceId FROM @Output)

    INSERT INTO Modules (ServiceId, ModuleBeginDate, ModuleEndDate,
     ModuleType, Class, ClassSize, SubjectId, LecturerId,
    TranslatorId, [Language], ModulePrice)
    VALUES (@ServiceId, @ModuleBeginDate, @ModuleEndDate,
    @ModuleType, 'online', @WebinarSize, @SubjectId, @LecturerId,
    @TranslatorId, @Language, @ServicePrice)
END
```

6.4 Dodanie klienta

```
CREATE PROCEDURE [dbo].[AddClient]
    @Name VARCHAR(50),
    @Surname VARCHAR(50),
    @Street VARCHAR(50) = NULL,
    @HomeNumber VARCHAR(50),
    @City VARCHAR(50),
    @Country VARCHAR(50),
    @ZipCode VARCHAR(50),
    @Email VARCHAR(50),
    @Phone VARCHAR(50)
AS
BEGIN
    INSERT INTO Clients ([Name], Surname, Street, HomeNumber, City,
     Country, ZipCode, Email, Phone)
    VALUES (@Name, @Surname, @Street, @HomeNumber, @City, @Country,
     @ZipCode, @Email, @Phone)
END
```

6.5 Dodanie pracownika

```
CREATE PROCEDURE [dbo].[AddWorker]
    @Name VARCHAR(50),
    @Surname VARCHAR(50),
    @Role tinyint
AS
BEGIN
    INSERT INTO Workers ([Name], Surname, [Role])
    VALUES (@Name, @Surname, @Role)
END
```

6.6 Dodanie języka dla tłumacza

```sql
CREATE PROCEDURE [dbo].[AddLanguageToTranslator]
    @WorkerId int,
    @Language VARCHAR(50)
AS
BEGIN
    INSERT INTO WorkersLanguages
    VALUES (@WorkerId, @Language)
END
```

6.7 Dodanie przedmiotu

```sql
CREATE PROCEDURE [dbo].[AddSubject]
    @ECTS tinyint,
    @Description VARCHAR(50)
AS
BEGIN
    INSERT INTO Subjects (ECTS, [Description])
    VALUES (@ECTS, @Description)
END
```

6.8 Dodanie ulgi od płatności dla klienta

```sql
CREATE PROCEDURE [dbo].[AddWaive]
    @PaymentId int,
    @AmountWaived money = NULL,
    @DaysWaived int = NULL
AS
BEGIN
    UPDATE PaymentDetails
    SET AmountWaived = @AmountWaived, DaysWaived = @DaysWaived
    WHERE PaymentId = @PaymentId
END
```

7 Funkcje

7.1 Całkowity dochód firmy

```sql
CREATE FUNCTION TotalIncome()
RETURNS MONEY
AS
BEGIN
        RETURN (SELECT sum(Amount) From PaymentDetails
        Group by PaymentId)
END
GO
```

7.2 Średni miesięczny dochów firmy

```sql
CREATE FUNCTION AvgIncome()
RETURNS MONEY
AS
BEGIN
        RETURN (select avg(total_amount) from
        (SELECT sum(amount) as total_amount
        from Payments as p
        INNER JOIN PaymentDetails as pd
        ON p.PaymentId=pd.PaymentId
        Group by YEAR(Date), MONTH(Date)) as sums)
END
GO
```

7.3 Dochód z danego kursu

```sql
CREATE FUNCTION IncomeFromCourse( @courseid Int
)
RETURNS INT
AS
BEGIN
        RETURN (SELECT sum(ModulePrice) From Modules
        Where ServiceId = @courseid
        Group by ServiceId)
END
GO
```

7.4 Czy dana osoba zaliczyła kurs

```sql
CREATE FUNCTION DidClientPass(@StudentId int, @ServiceId int)
RETURNS INT
AS
BEGIN

    DECLARE @StudentAttendance tinyint
    SET @StudentAttendance = [dbo].StudentAttendance(@StudentId,
    @ServiceId)

        DECLARE @AttendancePass INT
    SET @AttendancePass = [dbo].IsAttendanceEnough(
    @StudentAttendance, @ServiceId)
        IF @AttendancePass = 0
        BEGIN
                RETURN 0
        END

        DECLARE @ExamId int
    SET @ExamId = (SELECT ExamId FROM Services WHERE ServiceId =
    @ServiceId)

        IF @ExamId IS NULL
        BEGIN
                RETURN 1
        END

        DECLARE @ExamGrade FLOAT
    SET @ExamGrade = (
                SELECT Grade
                FROM ExamDetails exd
                INNER JOIN Exams ex ON ex.ExamId = exd.ExamId
                INNER JOIN Services ser On ser.ExamId = ex.ExamId
                WHERE ClientId = @StudentId AND ser.ServiceId = @ServiceId
                )

        IF @ExamGrade > 2.0
        BEGIN
                RETURN 1
        END
        RETURN 0
END
GO
```

7.5 Jaki dług ma dany student

```
CREATE FUNCTION StudentRemainingPayments(@StudentId int)
RETURNS MONEY
AS
BEGIN
        RETURN (
                SELECT SUM(s.ServicePrice) + SUM(m.ModulePrice) - SUM(ISNULL(pd.AmountWaived, 0))
                FROM Payments p
                INNER JOIN PaymentDetails pd ON p.PaymentId = pd.PaymentId
                INNER JOIN Services s ON s.ServiceId = pd.ServiceId
                INNER JOIN Modules m ON m.ModuleId = pd.ModuleId
                WHERE p.ClientId = @StudentId AND p.State = 0
                )
END
GO
```

7.6 Deficyt ECTS danego studenta

```
CREATE FUNCTION StudentECTSLoss(@StudentId int)
RETURNS int
AS
BEGIN
        RETURN (
                SELECT SUM(sub.ECTS)
                FROM Exams ex
                INNER JOIN ExamDetails exd ON ex.ExamId = exd.ExamId
                INNER JOIN Services ser ON ex.ExamId = ser.ExamId
                INNER JOIN Syllabus syl ON ser.SyllabusId = syl.SyllabusId
                INNER JOIN Subjects sub ON sub.SubjectId = syl.SubjectId
                WHERE exd.ClientId = @StudentId AND exd.Grade = 2.0
        )
END
GO
```

7.7 Liczba pracowników

```
CREATE FUNCTION NumOfWorkers(
)
RETURNS INT
AS
BEGIN
        RETURN (SELECT count(*) From Workers
        Group by WorkerId)
END
GO
```

7.8 Liczba klientów

```
CREATE FUNCTION NumOfClients(
)
RETURNS INT
AS
BEGIN
        RETURN (SELECT count(*) From Clients
        Group by ClientId)
END
GO
```

7.9 Frekwencja danego studenta

```
CREATE FUNCTION StudentAttendance(@StudentId int, @ServiceId int)
RETURNS TINYINT
AS
BEGIN
    DECLARE @ModulesCount INT
        SET @ModulesCount = (
                SELECT COUNT(*)
                FROM Modules
                WHERE ServiceId = @ServiceId
        )

    DECLARE @ModulesAttended INT
        SET @ModulesAttended = (
                SELECT COUNT(*)
                FROM Modules mod
                INNER JOIN Attendance att ON mod.ModuleId = att.
    ModuleId
                WHERE att.ClientId = @StudentId
        )

        RETURN @ModulesAttended / @ModulesCount
END
GO
```

7.10 Czy student ma wystarczającą frekwencję do zaliczenia

```sql
CREATE FUNCTION IsAttendanceEnough(@att Tinyint, @service Int)
RETURNS INT
AS
BEGIN
        DECLARE @pass_per TINYINT

        SET @pass_per = (Select PassPercent from Services
        where ServiceId = @service)

        IF @pass_per <= @att
                BEGIN
                        RETURN 1
                END
        RETURN 0
END
GO
```

7.11 Liczba trwających kursów

```sql
CREATE FUNCTION NumOFCouresInProgress(@curr_date DATE)
RETURNS INT
AS
BEGIN
        return( select count(ServiceId) from Services
        where BeginDate < @curr_date AND @curr_date < EndDate
)
END
GO
```

7.12 Wszystkie trwające kursy

```sql
CREATE FUNCTION CouresInProgress(@curr_date DATE)
RETURNS TABLE
AS
return( select ServiceId from Services
where BeginDate < @curr_date AND @curr_date < EndDate
)
GO
```

7.13 Języki, w jakich firma oferuje zajęcia

```sql
CREATE FUNCTION AvailableLanguages()
RETURNS TABLE
AS
RETURN
(
select Distinct(WorkerLanguage) from WorkersLanguages as wl
        Group by WorkerLanguage
)
GO
```

7.14 Wszyscy tłumacze, którzy znają dany język

```
CREATE FUNCTION TranslatorsThatCanSpeak(@given_language VARCHAR)
RETURNS TABLE
AS
RETURN
(
select Name, Surname from WorkersLanguages as wl
        Inner join Workers as w on w.WorkerId = wl.WorkerId
        where WorkerLanguage = @given_language
        Group by WorkerLanguage, Name, Surname
)
```

7.15 Czy dany student ma kolizję modułów

7.16 Średnia ocen z danego egzaminu

```
CREATE FUNCTION AvgGradeFromExam(@given_exam_id INT)
RETURNS INT
AS
BEGIN
        return( select avg(grade) from ExamDetails
        where ExamId = @given_exam_id
        Group by ExamId)
END
GO
```

7.17 Rozkład ocen z danego egzaminu

```
CREATE FUNCTION DistributionExamsGrades(@given_exam_id INT)
RETURNS TABLE
AS
RETURN
(
select Grade, COUNT(*) as num_of_grades from ExamDetails
        where ExamId = @given_exam_id
        Group by ExamId, Grade
)
```

7.18 Ilość wolnych miejsc dla danego kursu

```sql
CREATE FUNCTION ServiceFreePlaces(@ServiceId INT)
RETURNS INT
AS
BEGIN
        DECLARE @TotalPlaces INT
    SET @TotalPlaces = (
                SELECT MIN(ClassSize)
                FROM Modules
                WHERE ServiceId = @ServiceId
        )

        DECLARE @AdvancePrice MONEY
    SET @AdvancePrice = (SELECT AdvancePrice FROM Services WHERE ServiceId = @ServiceId)

        DECLARE @TakenPlaces INT
    SET @TakenPlaces = (
                SELECT COUNT(*)
                FROM Payments p
                INNER JOIN PaymentDetails pd ON p.PaymentId = pd.PaymentId
                WHERE p.State = 1 AND pd.ServiceId = @ServiceId
                GROUP BY p.ClientId
                HAVING SUM(pd.Amount - ISNULL(pd.AmountWaived, 0)) >= @AdvancePrice
        )

        RETURN @TotalPlaces - @TakenPlaces
END
GO
```

8 Triggery

8.1 Zniżka 10% dla stałych klientów (takich którzy wydali co najmniej 1000 zł)

```sql
CREATE TRIGGER trg_apply_discount
ON Payments
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @total_amount DECIMAL(10, 2);

    SELECT @total_amount = SUM(Amount)
    FROM Payments
    WHERE ClientId IN (SELECT ClientId FROM INSERTED);

    IF @total_amount > 1000
    BEGIN
        DECLARE @discount_amount DECIMAL(10, 2);
        SET @discount_amount = @total_amount * 0.1;

        UPDATE Payments
        SET Discount = @discount_amount
```

```sql
            WHERE PaymentId IN (SELECT PaymentId FROM INSERTED);
    END;
END;
```

8.2 Ustawienie daty końca dostępu do webinaru na 30 dni po jego rozpoczęciu

```sql
CREATE TRIGGER trg_set_webinar_end_date
ON Services
AFTER INSERT, UPDATE
AS
BEGIN
    UPDATE Services
    SET EndDate = DATEADD(DAY, 30, BeginDate)
    WHERE ServiceType = 0
    AND ServiceId IN (SELECT ServiceId FROM INSERTED)
    AND EndDate IS NULL;
END;
```

9 Indeksy

9.1 Module id

```sql
CREATE UNIQUE INDEX Modules_idx
ON Modules (ModuleId)
```

9.2 Payments id

```sql
CREATE UNIQUE INDEX Payments_idx
ON Payments (PaymentId)
```

9.3 Client id

```sql
CREATE UNIQUE INDEX Clients_idx
ON Clients (ClientId)
```

9.4 Worker id

```sql
CREATE UNIQUE INDEX Workers_idx
ON Workers (WorkerId)
```

9.5 Service id

```sql
CREATE UNIQUE INDEX Services_idx
ON Services (ServiceId)
```

9.6 Exam id

```sql
CREATE UNIQUE INDEX Exams_idx
ON Exams (ExamId)
```

9.7 ExamDetails id

```sql
CREATE UNIQUE INDEX ExamsDetails_idx
ON ExamDetails (ExamId)
```

9.8 Syllabus id

```sql
CREATE UNIQUE INDEX Syllabus_idx
ON Syllabus (SyllabusId)
```

9.9 Subjects id

```sql
CREATE UNIQUE INDEX Subjects_idx
ON Subjects (SubjectId)
```

9.10 WorkersLanguages id

```sql
CREATE INDEX WorkersLanguages_idx
ON WorkersLanguages (WorkerId)
```

9.11 PaymentDetails id

```sql
CREATE INDEX PaymentDetails_idx
ON PaymentDetails (PaymentId, ServiceId)
```

9.12 Attendance id

```sql
CREATE INDEX Attendance_idx
ON Attendance (ModuleId, ClientId)
```

10 Użytkownicy i uprawnienia

10.1 Niezalogowany użytkownik

```sql
CREATE ROLE not_logged_in_user

GRANT SELECT ON Services TO not_logged_in_user
GRANT SELECT (ServiceId, BeginDate, EndDate, Language, ModulePrice)
     ON Modules TO not_logged_in_user
GRANT SELECT ON Syllabus TO not_logged_in_user
GRANT SELECT ON Subjects TO not_logged_in_user
```

10.2 Zalogowany użytkownik

```sql
CREATE ROLE logged_in_user

GRANT SELECT ON Services TO logged_in_user
GRANT SELECT (
ServiceId, ModuleBeginDate, ModuleEndDate, Language, ModulePrice,
    LecturerId, TranslatorId, ClassSize
)
ON Modules TO logged_in_user
GRANT SELECT ON Syllabus TO logged_in_user
GRANT SELECT ON Subjects TO logged_in_user
GRANT SELECT, INSERT ON Payments TO logged_in_user
GRANT SELECT ON Attendance TO logged_in_user
GRANT SELECT ON Exams TO logged_in_user
```

10.3 Prowadzący

```sql
CREATE ROLE tutor

GRANT SELECT, INSERT, UPDATE, DELETE ON Attendance TO tutor
GRANT SELECT ON Exams TO tutor
GRANT SELECT ON Modules TO tutor
```

10.4 Koordynator kursu

```sql
CREATE ROLE course_coordinator

GRANT SELECT, INSERT, UPDATE, DELETE ON Attendance TO
    course_coordinator
GRANT SELECT ON Exams TO course_coordinator
GRANT SELECT ON Modules TO course_coordinator
GRANT SELECT, INSERT, UPDATE, DELETE ON Syllabus TO
    course_coordinator
GRANT SELECT, INSERT, UPDATE, DELETE ON Services TO
    course_coordinator
GRANT SELECT, INSERT, UPDATE, DELETE ON Exams TO course_coordinator
GRANT SELECT, INSERT, UPDATE, DELETE ON Modules TO
    course_coordinator
```

10.5 Dyrektor szkoły

```sql
CREATE ROLE headmaster

GRANT SELECT ON SCHEMA :: [dbo] TO headmaster
GRANT SELECT, INSERT, UPDATE, DELETE ON Workers TO headmaster
GRANT SELECT, INSERT, UPDATE, DELETE ON Services TO headmaster
GRANT SELECT, INSERT, UPDATE, DELETE ON Modules TO headmaster
```

10.6 Admin

```sql
CREATE ROLE admin

GRANT SELECT, INSERT, UPDATE, DELETE ON SCHEMA::[dbo] to admin
```