Akademia Górniczo Hutnicza
im. Stanisława Staszica
w Krakowie

PODSTAWY BAZ DANYCH

PROJEKT:
KURSY I SZKOLENIA

AUTORZY:

JANECZKO TOMASZ

SMYDA TOMASZ

ZIELIŃSKI PRZEMYSŁAW

PROWADZĄCY:

DR INŻ. ROBERT MARCJAN

18 GRUDNIA 2023

Spis treści

1 Wprowadzenie

Celem projektu było zaplanowanie systemu bazodanowego dla firmy oferującej różnego rodzaju kursy i szkolenia. Początkowo oferowane usługi były świadczone wyłącznie stacjonarnie, ale ze względu na pandemię COVID-19 usługi zostały w różnym stopniu zdigitalizowane. Obecnie model świadczenia usług jest hybrydowy, ale bardzo niejednolity dla różnych usług. Oferowane usługi dzielą się na webinary, kursy oraz szkolenia.

2 Funkcje realizowane przez system

2.1 Funkcje użytkowników

2.1.1 Administrator

- Dodawanie pracowników
- Obsługa platformy chmurowej (w tym usuwanie nagrań webinariów i kursów)

2.1.2 Dyrektor

- Wydanie zgody na płatność odroczoną w czasie

2.1.3 Menadżer

- Generowanie i przetwarzanie informacji o płatnościach
- Wprowadzenie cen dla płatnych webinariów, kursów oraz studiów
- Tworzenie programu studiów
- Wprowadzanie informacji o kolejnych dniach webinarium
- Wprowadzanie informacji o salach (kursy oraz studia stacjonarne)
- Tworzenie oraz zmienianie harmonogramu zajęć studiów
- Dostęp do raportów bilokacji – listy osób zapisanych na kolidujące się zajęcia
- Dodawanie nauczycieli
- Generowanie raportów finansowych – informacje o płatnościach klientów, zestawienie przychodów dla każdej oferowanej formy szkolenia

2.1.4 Nauczyciel

- Prowadzenie webinariów, kursów oraz studiów
- Dostęp do listy klientów i ich obecności na poszczególnych modułach zajęć

2.1.5 Klient

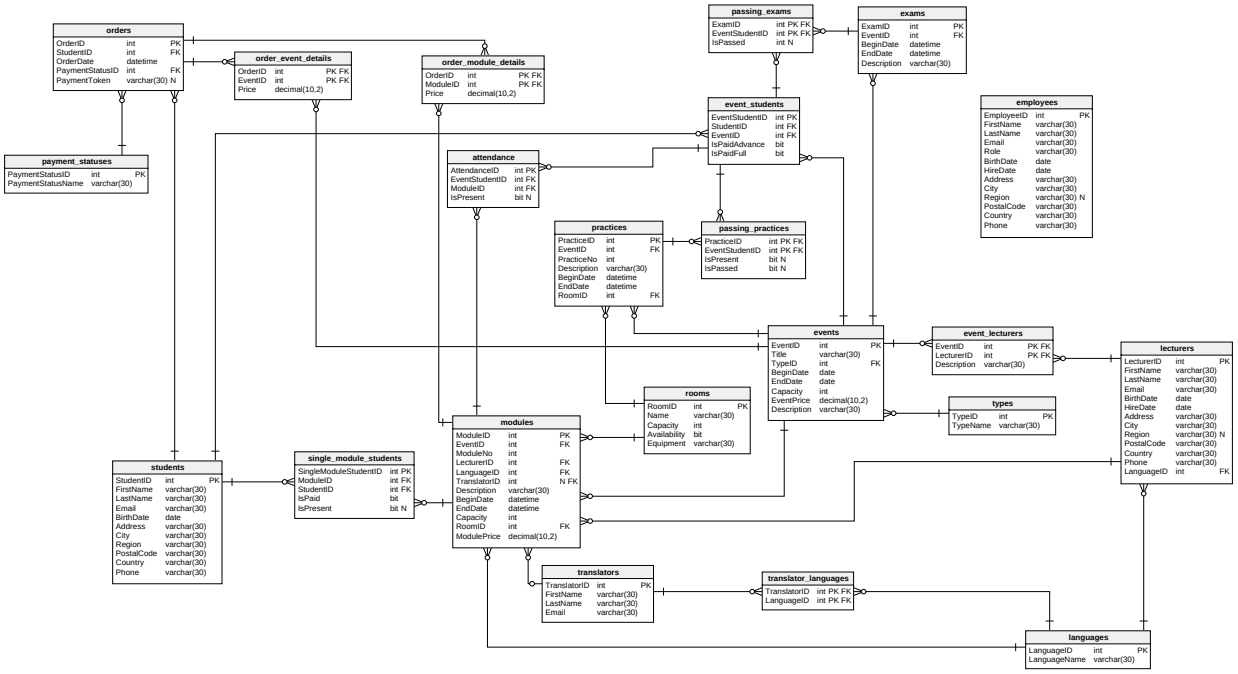
- Dostęp do darmowego webinarium
- Wykupienie dostępu do płatnego webinarium lub kursu
- Sprawdzanie przypisanych sal (kursy oraz studia stacjonarne)
- Sprawdzenie dotychczasowej frekwencji własnej na studiach
- Dostęp do nagrań (kursy online asynchroniczne)

- Dostęp do nagrań „na żywo” oraz do archiwalnych nagrań przez 30 dni (kursy online synchroniczne)
- Zapisanie się na zajęcia lub kurs w celu odrobienia nieobecności
- Wyszukanie i zapisanie się na pojedyncze spotkanie studyjne

2.1.6 Niezarejestrowany użytkownik

- Założenie konta

3 Diagram



4 Tabele

4.1 Tabela employees

Przechowuje informacje na temat zatrudnionych pracowników.

EmployeeID - Identyfikator pracownika

FirstName -Imię pracownika

LastName - Nazwisko pracownika

Email - Adres email pracownika

Role - Rola pracownika

BirthDate - Data urodzenia pracownika

HireDate - Data zatrudnienia pracownika

Address - Adres pracownika

City - Miasto zamieszkania pracownika

Region - Region zamieszkania pracownika

PostalCode - Kod pocztowy pracownika

Country - Kraj zamieszkania pracownika

Phone - Numer telefonu pracownika

```
01 | CREATE TABLE employees (
02 |     EmployeeID int  NOT NULL,
03 |     FirstName varchar(30)  NOT NULL,
04 |     LastName varchar(30)  NOT NULL,
05 |     Email varchar(30)  NOT NULL,
06 |     Role varchar(30)  NOT NULL,
07 |     BirthDate date  NOT NULL,
08 |     HireDate date  NOT NULL,
09 |     Address varchar(30)  NOT NULL,
10 |     City varchar(30)  NOT NULL,
11 |     Region varchar(30)  NULL,
12 |     PostalCode varchar(30)  NOT NULL,
13 |     Country varchar(30)  NOT NULL,
14 |     Phone varchar(30)  NOT NULL,
15 |     CONSTRAINT employees_pk PRIMARY KEY  (EmployeeID)
16 | );
```

4.2 Tabela attendance

Przechowuje informacje o obecnościach studentów w danym module. Dzięki tej tabeli jesteśmy w stanie dowiedzieć się czy dany student był obecny na danym module zajęć.

AttendanceID - Identyfikator obecności na module

EventStudentID - Identyfikator przypisania studenta do wydarzenia

ModuleID - Identyfikator modułu

IsPresent - Informacja czy student był obecny na zajęciach

```
CREATE TABLE attendance (
    AttendanceID int  NOT NULL,
    EventStudentID int  NOT NULL,
    ModuleID int  NOT NULL,
    IsPresent bit  NULL,
    CONSTRAINT attendance_pk PRIMARY KEY  (AttendanceID)
);

ALTER TABLE attendance ADD CONSTRAINT attendance_event_students
    FOREIGN KEY (EventStudentID)
    REFERENCES event_students (EventStudentID);

ALTER TABLE attendance ADD CONSTRAINT attendance_modules
    FOREIGN KEY (ModuleID)
    REFERENCES modules (ModuleID);
```

4.3 Tabela event_lecturers

Łączy dane wydarzenie z prowadzącym je wykładowcą.

EventID - Identyfikator wydarzenia

LecturerID - Identyfikator wykładowcy

Description - Opis obowiązków prowadzącego

```
CREATE TABLE event_lecturers (
    EventID int  NOT NULL,
    LecturerID int  NOT NULL,
    Description varchar(30)  NOT NULL,
    CONSTRAINT event_lecturers_pk PRIMARY KEY  (EventID,LecturerID)
);

ALTER TABLE event_lecturers ADD CONSTRAINT event_lecturers_events
    FOREIGN KEY (EventID)
    REFERENCES events (EventID);

ALTER TABLE event_lecturers ADD CONSTRAINT event_lecturers_lecturers
    FOREIGN KEY (LecturerID)
    REFERENCES lecturers (LecturerID);
```

4.4 Tabela event_students

Łączy dane wydarzenie z uczestniczącym w nim studentem.

EventStudentID - Identyfikator przypisania studenta do wydarzenia

StudentID - Identyfikator studenta

EventID - Identyfikator wydarzenia

IsPaidAdvance - Informacja czy zaliczka została wpłacona

IsPaidFull - Informacja czy została zapłacona całkowita kwota

```
CREATE TABLE event_students (
    EventStudentID int  NOT NULL,
    StudentID int  NOT NULL,
    EventID int  NOT NULL,
    IsPaidAdvance bit  NOT NULL,
    IsPaidFull bit  NOT NULL,
    CONSTRAINT event_students_pk PRIMARY KEY  (EventStudentID)
```

```
);

ALTER TABLE event_students ADD CONSTRAINT event_students_events
    FOREIGN KEY (EventID)
    REFERENCES events (EventID);

ALTER TABLE event_students ADD CONSTRAINT event_students_students
    FOREIGN KEY (StudentID)
    REFERENCES students (StudentID);
```

4.5 Tabela events

Zawiera informacje na temat wydarzeń.

EventID - Identyfikator wydarzenia

Title - Tytuł wydarzenia

TypeID - Rodzaj wydarzenia

BeginDate - Data początkowa wydarzenia

EndDate - Data końcowa wydarzenia

Capacity - Limit osób, które mogą uczestniczyć w wydarzeniu

EventPrice - Cena wydarzenia

Description - Opis wydarzenia

```
CREATE TABLE events (
    EventID int  NOT NULL,
    Title varchar(30)  NOT NULL,
    TypeID int  NOT NULL,
    BeginDate date  NOT NULL,
    EndDate date  NOT NULL,
    Capacity int  NOT NULL,
    EventPrice decimal(10,2)  NOT NULL,
    Description varchar(30)  NOT NULL,
    CONSTRAINT events_pk PRIMARY KEY  (EventID)
);

ALTER TABLE events ADD CONSTRAINT events_types
    FOREIGN KEY (TypeID)
    REFERENCES types (TypeID);
```

4.6 Tabela exams

Zawiera informacje na temat egzaminów.

ExamID - Identyfikator egzaminu

EventID - Identyfikator wydarzenia, pod które podlega egzamin

BeginDate - Data początkowa egzaminu

EndDate - Data końcowa egzaminu

Description - Opis

```
CREATE TABLE exams (
    ExamID int  NOT NULL,
    EventID int  NOT NULL,
    BeginDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    Description varchar(30)  NOT NULL,
    CONSTRAINT exams_pk PRIMARY KEY  (ExamID)
);

ALTER TABLE exams ADD CONSTRAINT exams_events
    FOREIGN KEY (EventID)
    REFERENCES events (EventID);
```

4.7 Tabela languages

Zawiera informacje na temat języków, w którym mogą być prowadzone zajęcia.

LanguageID - Identyfikator języka

LanguageName - Nazwa języka

```
CREATE TABLE languages (
    LanguageID int  NOT NULL,
    LanguageName varchar(30)  NOT NULL,
    CONSTRAINT languages_pk PRIMARY KEY  (LanguageID)
);
```

4.8 Tabela lecturers

Zawiera informacje na temat wykładowców.

LecturerID - Identyfikator wykładowcy

FirstName - Imię wykładowcy

LastName - Nazwisko wykładowcy

Email - Adres email wykładowcy

BirthDate - Data urodzenia wykładowcy

HireDate - Data zatrudnienia wykładowcy

Address - Adres zamieszkania wykładowcy

City - Miasto zamieszkania wykładowcy

Region - Region zamieszkania wykładowcy

PostalCode - Kod pocztowy wykładowcy

Country - Kraj zamieszkania wykładowcy

Phone - Numer telefonu wykładowcy

LanguageID - ID języka wykładowcy, w którym prowadzi wykład

```
CREATE TABLE lecturers (
    LecturerID int  NOT NULL,
    FirstName varchar(30)  NOT NULL,
    LastName varchar(30)  NOT NULL,
    Email varchar(30)  NOT NULL,
    BirthDate date  NOT NULL,
    HireDate date  NOT NULL,
    Address varchar(30)  NOT NULL,
    City varchar(30)  NOT NULL,
    Region varchar(30)  NULL,
    PostalCode varchar(30)  NOT NULL,
    Country varchar(30)  NOT NULL,
    Phone varchar(30)  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT lecturers_pk PRIMARY KEY  (LecturerID)
);

ALTER TABLE lecturers ADD CONSTRAINT lecturers_languages
    FOREIGN KEY (LanguageID)
    REFERENCES languages (LanguageID);
```

4.9 Tabela modules

Zawiera informacje na temat pojedycznego modułu.

ModuleID - Identyfikator modułu

EventID - Identyfikator wydarzenia, pod który podlega moduł

ModuleNo - Numer modułu

LecturerID - ID wykładowcy prowadzącego moduł

LanguageID - ID języka, w którym moduł jest prowadzony

TranslatorID - ID tłumacza

Description - Opis

BeginDate - Data początkowa modułu

EndDate - Data końcowa modułu

Capacity - Maksymalna liczba osób mogących uczestniczyć w module

RoomID - ID sali, w której odbywa się moduł

ModulePrice - Cena zakupu modułu

```
CREATE TABLE modules (
    ModuleID int  NOT NULL,
    EventID int  NOT NULL,
    ModuleNo int  NOT NULL,
    LecturerID int  NOT NULL,
    LanguageID int  NOT NULL,
    TranslatorID int  NULL,
    Description varchar(30)  NOT NULL,
    BeginDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    Capacity int  NOT NULL,
    RoomID int  NOT NULL,
    ModulePrice decimal(10,2)  NOT NULL,
    CONSTRAINT modules_pk PRIMARY KEY  (ModuleID)
```

```
);

ALTER TABLE modules ADD CONSTRAINT modules_events
    FOREIGN KEY (EventID)
    REFERENCES events (EventID);

ALTER TABLE modules ADD CONSTRAINT modules_languages
    FOREIGN KEY (LanguageID)
    REFERENCES languages (LanguageID);

ALTER TABLE modules ADD CONSTRAINT modules_lecturers
    FOREIGN KEY (LecturerID)
    REFERENCES lecturers (LecturerID);

ALTER TABLE modules ADD CONSTRAINT modules_rooms
    FOREIGN KEY (RoomID)
    REFERENCES rooms (RoomID);

ALTER TABLE modules ADD CONSTRAINT modules_translators
    FOREIGN KEY (TranslatorID)
    REFERENCES translators (TranslatorID);
```

4.10 Tabela order_event_details

Zawiera cenę oraz identyfikator wydarzenia, dla wydarzeń zakupionych w danym zamówieniu.

OrderID - Identyfikator zamówienia

EventID - Identyfikator wydarzenia

Price - Cena zakupu wydarzenia

```
CREATE TABLE order_event_details (
    OrderID int  NOT NULL,
    EventID int  NOT NULL,
    Price decimal(10,2)  NOT NULL,
    CONSTRAINT order_event_details_pk PRIMARY KEY  (OrderID,EventID)
);

ALTER TABLE order_event_details ADD CONSTRAINT order_event_details_orders
    FOREIGN KEY (OrderID)
    REFERENCES orders (OrderID);

ALTER TABLE order_event_details ADD CONSTRAINT order_module_details_events
    FOREIGN KEY (EventID)
    REFERENCES events (EventID);
```

4.11 Tabela order_module_details

Zawiera cenę oraz identyfikator modułu, dla modułów zakupionych w danym zamówieniu.

OrderID - Identyfikator zamówienia

ModuleID - Identyfikator modułu

Price - Cena zakupu modułu

```
CREATE TABLE order_module_details (
    OrderID int  NOT NULL,
    ModuleID int  NOT NULL,
    Price decimal(10,2)  NOT NULL,
    CONSTRAINT order_module_details_pk PRIMARY KEY  (OrderID,ModuleID)
);
```

```
07 |
08 | ALTER TABLE order_module_details ADD CONSTRAINT order_module_details_modules
09 |     FOREIGN KEY (ModuleID)
10 |     REFERENCES modules (ModuleID);
11 |
12 | ALTER TABLE order_module_details ADD CONSTRAINT order_module_details_orders
13 |     FOREIGN KEY (OrderID)
14 |     REFERENCES orders (OrderID);
```

4.12 Tabela orders

Zawiera informacje na temat dokonanych zamówień.

OrderID - Identyfikator zamówienia

StudentID - Identyfikator studenta, który dokonał zamówienia

OrderDate - Data złożenia zamówienia

PaymentStatusID - Identyfikator statusu płatności

PaymentToken - Unikalny token płatności

```
01 | CREATE TABLE orders (
02 |     OrderID int  NOT NULL,
03 |     StudentID int  NOT NULL,
04 |     OrderDate datetime  NOT NULL,
05 |     PaymentStatusID int  NOT NULL,
06 |     PaymentToken varchar(30)  NULL,
07 |     CONSTRAINT orders_pk PRIMARY KEY  (OrderID)
08 | );
09 |
10 | ALTER TABLE orders ADD CONSTRAINT orders_payment_statuses
11 |     FOREIGN KEY (PaymentStatusID)
12 |     REFERENCES payment_statuses (PaymentStatusID);
13 |
14 | ALTER TABLE orders ADD CONSTRAINT orders_students
15 |     FOREIGN KEY (StudentID)
16 |     REFERENCES students (StudentID);
```

4.13 Tabela passing_exams

Przechowuje informacje o zaliczeniach egzaminów przez studentów.

ExamID - Identyfikator egzaminu

EventStudentID - Identyfikator przypisania studenta do danego wydarzenia

IsPassed - Informacja czy student zdał egzamin

```
01 | CREATE TABLE passing_exams (
02 |     ExamID int  NOT NULL,
03 |     EventStudentID int  NOT NULL,
04 |     IsPassed int  NULL,
05 |     CONSTRAINT passing_exams_pk PRIMARY KEY  (ExamID,EventStudentID)
06 | );
07 |
08 | ALTER TABLE passing_exams ADD CONSTRAINT passing_exams_event_students
09 |     FOREIGN KEY (EventStudentID)
10 |     REFERENCES event_students (EventStudentID);
11 |
12 | ALTER TABLE passing_exams ADD CONSTRAINT passing_exams_exams
13 |     FOREIGN KEY (ExamID)
14 |     REFERENCES exams (ExamID);
```

4.14 Tabela passing_practices

Przechowuje informacje o zaliczeniach praktyk oraz obecnościach na nich przez studentów.

PracticeID - Identyfikator praktyk

EventStudentID - Identyfikator przypisania studenta do danego wydarzenia

IsPresent - Informacja czy student był na praktykach

IsPassed - Informacja czy student zaliczył praktyki

```
01 | CREATE TABLE passing_practices (
02 |     PracticeID int  NOT NULL,
03 |     EventStudentID int  NOT NULL,
04 |     IsPresent bit  NULL,
05 |     IsPassed bit  NULL,
06 |     CONSTRAINT passing_practices_pk PRIMARY KEY  (PracticeID,EventStudentID)
07 | );
08 |
09 | ALTER TABLE passing_practices ADD CONSTRAINT passing_practices_event_students
10 |     FOREIGN KEY (EventStudentID)
11 |     REFERENCES event_students (EventStudentID);
12 |
13 | ALTER TABLE passing_practices ADD CONSTRAINT passing_practices_practices
14 |     FOREIGN KEY (PracticeID)
15 |     REFERENCES practices (PracticeID);
```

4.15 Tabela payment_statuses

PaymentStatusID - Identyfikator statusu płatności

PaymentStatusName - Nazwa statusu płatności

```
01 | CREATE TABLE payment_statuses (
02 |     PaymentStatusID int  NOT NULL,
03 |     PaymentStatusName varchar(30)  NOT NULL,
04 |     CONSTRAINT payment_statuses_pk PRIMARY KEY  (PaymentStatusID)
05 | );
```

4.16 Tabela practices

Tabela zawiera informacje na temat praktyk przeprowadzanych w ramach zaliczenia studiów.

PracticeID - Identyfikator praktyk

EventID - Identyfikator wydarzenia, pod które podlegają praktyki

PracticeNo - Numer praktyk

Description - Opis

BeginDate - Data rozpoczęcia praktyk

EndDate - Data zakończenia praktyk

RoomID - Identyfikator sali, w której odbywają się praktyki

```
CREATE TABLE practices (
    PracticeID int  NOT NULL,
    EventID int  NOT NULL,
    PracticeNo int  NOT NULL,
    Description varchar(30)  NOT NULL,
    BeginDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    RoomID int  NOT NULL,
    CONSTRAINT practices_pk PRIMARY KEY  (PracticeID)
);

ALTER TABLE practices ADD CONSTRAINT practices_events
    FOREIGN KEY (EventID)
    REFERENCES events (EventID);

ALTER TABLE practices ADD CONSTRAINT practices_rooms
    FOREIGN KEY (RoomID)
    REFERENCES rooms (RoomID);
```

4.17 Tabela rooms

Tabela zawiera informacje na temat sal, w których mogą odbywać się wydarzenia, praktyki, egzaminy itd.

RoomID - Identyfikator sali

Name - Nazwa sali

Capacity - Maksymalna liczba osób mogących przebywać w sali

Availability - Dostępność sali

Equipment - Opis wyposażenia sali

```
CREATE TABLE rooms (
    RoomID int  NOT NULL,
    Name varchar(30)  NOT NULL,
    Capacity int  NOT NULL,
    Availability bit  NOT NULL,
    Equipment varchar(30)  NOT NULL,
    CONSTRAINT rooms_pk PRIMARY KEY  (RoomID)
);
```

4.18 Tabela single_module_students

Tabela zawiera przypisania studentów do pojedynczych modułów.

SingleModuleStudentID - Identyfikator przypisania studenta do pojedynczego modułu

ModuleID - Identyfikator modułu

StudentID - Identyfikator studenta

IsPaid - Informacja, czy student opłacił moduł

IsPresent - Informacja, czy student był na module

```
CREATE TABLE single_module_students (
    SingleModuleStudentID int  NOT NULL,
    ModuleID int  NOT NULL,
    StudentID int  NOT NULL,
    IsPaid bit  NOT NULL,
    IsPresent bit  NULL,
    CONSTRAINT single_module_students_pk PRIMARY KEY  (SingleModuleStudentID)
);

ALTER TABLE single_module_students ADD CONSTRAINT module_students_modules
    FOREIGN KEY (ModuleID)
    REFERENCES modules (ModuleID);

ALTER TABLE single_module_students ADD CONSTRAINT module_students_students
    FOREIGN KEY (StudentID)
    REFERENCES students (StudentID);
```

4.19 Tabela students

Tabela zawiera informacje na temat zarejestrowanych studentów.

StudentID - Identyfikator studenta

FirstName - Imię studenta

LastName - Nazwisko studenta

Email - Adres email studenta

BirthDate - Data urodzenia studenta

Address - Adres zamieszkania studenta

City - Miasto zamieszkania studenta

Region - Region zamieszkania studenta

PostalCode - Kod pocztowy studenta

Country - Kraj zamieszkania studenta

Phone - Numer telefonu studenta

```
CREATE TABLE students (
    StudentID int  NOT NULL,
    FirstName varchar(30)  NOT NULL,
    LastName varchar(30)  NOT NULL,
    Email varchar(30)  NOT NULL,
    BirthDate date  NOT NULL,
    Address varchar(30)  NOT NULL,
    City varchar(30)  NOT NULL,
    Region varchar(30)  NOT NULL,
    PostalCode varchar(30)  NOT NULL,
    Country varchar(30)  NOT NULL,
    Phone varchar(30)  NOT NULL,
    CONSTRAINT students_pk PRIMARY KEY  (StudentID)
);
```

4.20 Tabela translator_languages

Tabela łączy identyfikator tłumacza z identyfikatorem języka, którym tłumacz potrafi się posługiwać.

TranslatorID - Identyfikator tłumacza

LanguageID - Identyfikator języka

```
01 | CREATE TABLE translator_languages (
02 |     TranslatorID int  NOT NULL,
03 |     LanguageID int  NOT NULL,
04 |     CONSTRAINT translator_languages_pk PRIMARY KEY  (TranslatorID,LanguageID)
05 | );
06 |
07 | ALTER TABLE translator_languages ADD CONSTRAINT translator_languages_languages
08 |     FOREIGN KEY (LanguageID)
09 |     REFERENCES languages (LanguageID);
10 |
11 | ALTER TABLE translator_languages ADD CONSTRAINT translator_languages_translators
12 |     FOREIGN KEY (TranslatorID)
13 |     REFERENCES translators (TranslatorID);
```

4.21 Tabela translators

Tabela przechowuje informacje na temat zatrudnionych tłumaczy.

TranslatorID - Identyfikator tłumacza

FirstName - Imię tłumacza

LastName - Nazwisko tłumacza

Email - Adres email tłumacza

```
01 | CREATE TABLE translators (
02 |     TranslatorID int  NOT NULL,
03 |     FirstName varchar(30)  NOT NULL,
04 |     LastName varchar(30)  NOT NULL,
05 |     Email varchar(30)  NOT NULL,
06 |     CONSTRAINT translators_pk PRIMARY KEY  (TranslatorID)
07 | );
```

4.22 Tabela types

Tabela zawiera informacje na temat typów wydarzeń, jakie są w ofercie firmy.

TypeID - Identyfikator typu

TypeName - Nazwa typu

```
01 | CREATE TABLE types (
02 |     TypeID int  NOT NULL,
03 |     TypeName varchar(30)  NOT NULL,
04 |     CONSTRAINT types_pk PRIMARY KEY  (TypeID)
05 | );
```

5 Procedury

5.1 RegisterStudent

Rejestruje nowego studenta do bazy danych.

```
CREATE PROCEDURE RegisterStudent
@FirstName varchar(30),
@LastName varchar(30),
@Email varchar(30),
@BirthDate date,
@Address varchar(30),
@City varchar(30),
@Region varchar(30),
@PostalCode varchar(30),
@Country varchar(30),
@Phone varchar(30)
AS
BEGIN
    INSERT INTO students (FirstName, LastName, Email, BirthDate, Address, City, Region, PostalCode, Country, Phone)
    VALUES (@FirstName, @LastName, @Email, @BirthDate, @Address, @City, @Region, @PostalCode, @Country, @Phone)
END
```

5.2 RegisterLecturer

Rejestruje nowego wykładowcę do bazy danych

```
CREATE PROCEDURE [dbo].[RegisterLecturer]
@FirstName varchar(30),
@LastName varchar(30),
@Email varchar(30),
@BirthDate date,
@HireDate date = NULL,
@Address varchar(30),
@City varchar(30),
@Region varchar(30),
@PostalCode varchar(30),
@Country varchar(30),
@Phone varchar(30),
@Language varchar(30)
AS
BEGIN
    IF @HireDate IS NULL
    SET @HireDate = GETDATE();

    DECLARE @LanguageID INT;
    SELECT @LanguageID = LanguageID FROM languages WHERE LanguageName = @Language;

    INSERT INTO lecturers (FirstName, LastName, Email, BirthDate, HireDate, Address, City, Region, PostalCode, Country, Phone, LanguageID)
    VALUES (@FirstName, @LastName, @Email, @BirthDate, @HireDate, @Address, @City, @Region, @PostalCode, @Country, @Phone, @LanguageID)
END
```

5.3 AddEvent

Dodaje nowe wydarzenie.

```
CREATE PROCEDURE AddEvent
@Title varchar(30),
@Type varchar(30),
@BeginDate DATE,
```

```
@EndDate DATE,
@Capacity INT,
@EventPrice DECIMAL(10,2),
@Description varchar(30)

AS
BEGIN
    DECLARE @TypeID INT;
    SELECT @TypeID = TypeID FROM types WHERE TypeName = @Type

    INSERT INTO events (Title, TypeID, BeginDate, EndDate, Capacity, EventPrice,
    Description)
    VALUES (@Title, @TypeID, @BeginDate, @EndDate, @Capacity, @EventPrice,
    @Description)
END
```

5.4 AddModule

Dodaje nowy moduł.

```
CREATE PROCEDURE AddModule
@EventID INT,
@ModuleNo INT,
@LecturerID INT,
@TranslatorID INT = NULL,
@Description VARCHAR(30) = '',
@BeginDate DATETIME,
@EndDate DATETIME,
@Capacity INT = NULL,
@RoomID INT,
@ModulePrice INT
AS
BEGIN
    DECLARE @LanguageID INT
    IF @TranslatorID IS NULL
        SELECT @LanguageID = LanguageID FROM lecturers WHERE LecturerID = @LecturerID
    ELSE
        SELECT @LanguageID = LanguageID FROM languages WHERE LanguageName = 'polish'

    IF @Capacity IS NULL
    BEGIN

        DECLARE @RoomCapacity INT
        SELECT @RoomCapacity = Capacity FROM rooms WHERE RoomID = @RoomID
        SET @Capacity = @RoomCapacity
    END

    DECLARE @EventCapacity INT
    SELECT @EventCapacity = Capacity FROM events WHERE EventID = @EventID
    IF @Capacity < @EventCapacity
        THROW 50000, 'Capacity is less than EventCapacity', 1

    DECLARE @TakenRooms TABLE (RoomNumber INT)
    INSERT INTO @TakenRooms (RoomNumber)
    EXEC RoomsTaken @BeginDate, @EndDate

    IF EXISTS (
        SELECT 1
        FROM @TakenRooms as sub
        WHERE sub.RoomNumber = @RoomID
    )
        THROW 500001, 'Room is not available then', 1

    INSERT INTO modules (EventID, ModuleNo, LecturerID, LanguageID, TranslatorID,
    Description, BeginDate, EndDate, Capacity, RoomID, ModulePrice)
```

```
    VALUES (@EventID, @ModuleNo, @LecturerID, @LanguageID, @TranslatorID,
    @Description, @BeginDate, @EndDate, @Capacity, @RoomID, @ModulePrice)
END
```

5.5 AddWebinar

Dodaje nowy webinar.

```
CREATE PROCEDURE AddWebinar
@Title varchar(30),
@BeginDate DATETIME,
@EndDate DATETIME,
@Capacity INT = NULL,
@EventPrice DECIMAL(10,2),
@Description varchar(30),
@LecturerID INT,
@TranslatorID INT = NULL


AS
BEGIN
    DECLARE @TypeID INT;
    SELECT @TypeID = TypeID FROM types WHERE 'webinar' = TypeName

    DECLARE @LanguageID INT
    IF @TranslatorID IS NULL
        SELECT @LanguageID = LanguageID FROM lecturers WHERE LecturerID = @LecturerID
    ELSE
        SELECT @LanguageID = LanguageID FROM languages WHERE LanguageName = 'polish'

    IF @Capacity IS NULL
    BEGIN
        SET @Capacity = 1000
    END


    INSERT INTO events (Title, TypeID, BeginDate, EndDate, Capacity, EventPrice,
    Description)
    VALUES (@Title, @TypeID, CAST(@BeginDate AS DATE), CAST(@EndDate AS DATE),
    @Capacity, @EventPrice, @Description)

    DECLARE @LastIndex INT
    SET @LastIndex = SCOPE_IDENTITY();

    INSERT INTO modules(EventID, ModuleNo, LecturerID, LanguageID, TranslatorID, [
    Description], BeginDate, EndDate, Capacity, RoomID, ModulePrice)
    VALUES (@LastIndex, 1, @LecturerID, @LanguageID, @TranslatorID, @Description,
    @BeginDate, @EndDate, @Capacity, 1, @EventPrice)
END
```

5.6 RoomsFree

Wyświetla listę wolnych sal w zadanym przedziale czasowym.

```
CREATE PROCEDURE RoomsFree
@BeginDate datetime,
@EndDate datetime
AS
BEGIN
    SELECT R.RoomID FROM modules M
    RIGHT JOIN rooms R ON R.RoomID = M.RoomID
    WHERE (M.EndDate < @BeginDate OR M.BeginDate > @EndDate) OR M.ModuleID IS NULL
END
```

5.7 RoomsTaken

Wyświetla listę zajętych sal w zadanym przedziale czasowym.

```
CREATE PROCEDURE RoomsTaken
@BeginDate datetime,
@EndDate datetime
AS
BEGIN
    SELECT R.RoomID FROM modules M
    JOIN rooms R ON R.RoomID = M.RoomID
    WHERE (M.BeginDate BETWEEN @BeginDate AND @EndDate) OR (M.EndDate BETWEEN
    @BeginDate AND @EndDate)
END
```

5.8 EnrollStudent2Event

Zapisuje danego studenta do danego wydarzenia.

```
CREATE PROCEDURE EnrollStudent2Event
@StudentID int,
@EventID int
AS
BEGIN
    INSERT INTO event_students (StudentID, EventID, IsPaidAdvance, IsPaidFull)
    VALUES (@StudentID, @EventID, 1, 1)
    DECLARE @LastIdentity INT
    SET @LastIdentity = SCOPE_IDENTITY();

    DECLARE @EventModules TABLE (ModuleID INT)
    INSERT INTO @EventModules (ModuleID)
    SELECT ModuleID FROM modules M WHERE M.EventID = @EventID

    DECLARE @RowCount INT = 1
    DECLARE @TotalRows INT;

    SELECT @TotalRows = COUNT(*) FROM @EventModules

    WHILE @RowCount <= @TotalRows
    BEGIN
        DECLARE @TempModuleID INT
        SELECT @TempModuleID = Sub.ModuleID FROM (
            SELECT EM.ModuleID, ROW_NUMBER() OVER (ORDER BY EM.ModuleID) AS ROW_NUM
    FROM @EventModules AS EM
        ) as Sub
        WHERE ROW_NUM = @RowCount
        INSERT INTO attendance (EventStudentID, ModuleID, IsPresent)
        VALUES (@LastIdentity, @TempModuleID, NULL)
        SET @RowCount = @RowCount+1;
    END
END
```