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

21 STYCZNIA 2024

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

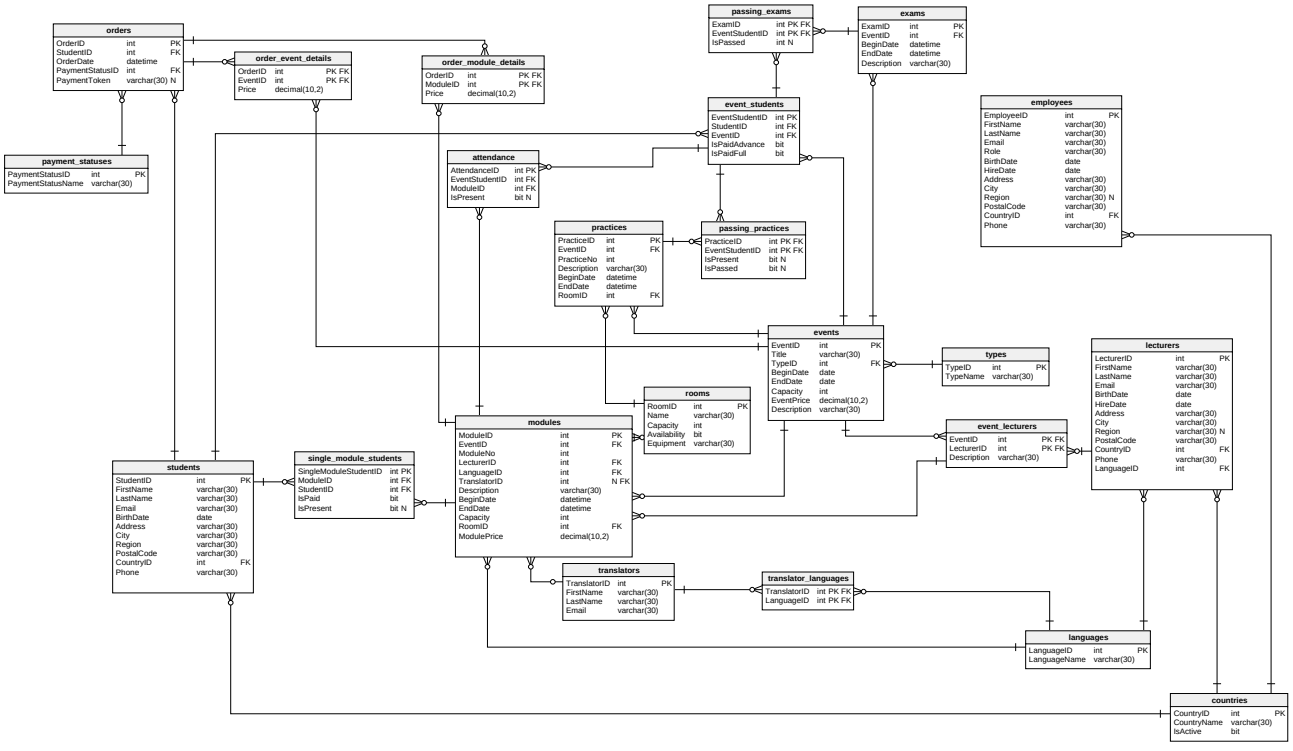
- Dostęp do darmowego webinarium
- Wykupienie dostępu do płatnego webinarium lub kursu
- Sprawdzanie przypisanych sal (kursy oraz studia stacjonarne)
- Sprawdzenie dotychczasowej frekwencji własnej na studiach
- Dostęp do nagrań (kursy online asynchroniczne)

- 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

CountryID - Identyfikator kraju zamieszkania pracownika

Phone - Numer telefonu pracownika

```
01 | CREATE TABLE employees (
02 |     EmployeeID int  NOT NULL IDENTITY,
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 |     CountryID int  NOT NULL,
14 |     Phone varchar(30)  NOT NULL,
15 |     CONSTRAINT employees_pk PRIMARY KEY  (EmployeeID),
16 |     CHECK (BirthDate > '01-01-1910'),
17 |     CHECK (HireDate < getdate())
18 | );
19 |
20 | ALTER TABLE employees ADD CONSTRAINT employees_countries
21 |     FOREIGN KEY (CountryID)
22 |     REFERENCES countries (CountryID);
```

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 IDENTITY,
    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

```sql
CREATE TABLE event_students (
    EventStudentID int  NOT NULL IDENTITY,
    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 (studiów, kursów i webinarów).

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

```sql
CREATE TABLE events (
    EventID int  NOT NULL IDENTITY,
    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),
    CHECK (EndDate > BeginDate),
    CHECK (EventPrice >= 0)
);

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 IDENTITY,
    EventID int  NOT NULL,
    BeginDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    Description varchar(30)  NOT NULL,
    CONSTRAINT exams_pk PRIMARY KEY  (ExamID),
    CHECK (EndDate > BeginDate)
);

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 IDENTITY,
    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

CountryID - Identyfikator kraju zamieszkania wykładowcy

Phone - Numer telefonu wykładowcy

LanguageID - ID języka wykładowcy, w którym prowadzi wykład

```
CREATE TABLE lecturers (
    LecturerID int  NOT NULL IDENTITY,
    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,
    CountryID int  NOT NULL,
    Phone varchar(30)  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT lecturers_pk PRIMARY KEY  (LecturerID),
    CHECK (BirthDate > '01-01-1910'),
    CHECK (HireDate < getdate())
);

ALTER TABLE lecturers ADD CONSTRAINT lecturers_countries
    FOREIGN KEY (CountryID)
    REFERENCES countries (CountryID);

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 IDENTITY,
    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),
    CHECK (EndDate > BeginDate),
    CHECK (ModulePrice >= 0)
);

ALTER TABLE modules ADD CONSTRAINT modules_event_lecturers
    FOREIGN KEY (EventID,LecturerID)
    REFERENCES event_lecturers (EventID,LecturerID);

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_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),
    CHECK (Price >= 0)
);

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),
    CHECK (Price >= 0)
);

ALTER TABLE order_module_details ADD CONSTRAINT order_module_details_modules
    FOREIGN KEY (ModuleID)
    REFERENCES modules (ModuleID);

ALTER TABLE order_module_details ADD CONSTRAINT order_module_details_orders
    FOREIGN KEY (OrderID)
    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

```
CREATE TABLE orders (
    OrderID int  NOT NULL IDENTITY,
    StudentID int  NOT NULL,
    OrderDate datetime  NOT NULL,
    PaymentStatusID int  NOT NULL,
    PaymentToken varchar(30)  NULL,
    CONSTRAINT orders_pk PRIMARY KEY  (OrderID)
);

ALTER TABLE orders ADD CONSTRAINT orders_payment_statuses
    FOREIGN KEY (PaymentStatusID)
    REFERENCES payment_statuses (PaymentStatusID);

ALTER TABLE orders ADD CONSTRAINT orders_students
    FOREIGN KEY (StudentID)
    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

```
CREATE TABLE passing_exams (
    ExamID int  NOT NULL,
    EventStudentID int  NOT NULL,
    IsPassed int  NULL,
    CONSTRAINT passing_exams_pk PRIMARY KEY  (ExamID,EventStudentID)
);

ALTER TABLE passing_exams ADD CONSTRAINT passing_exams_event_students
    FOREIGN KEY (EventStudentID)
    REFERENCES event_students (EventStudentID);

ALTER TABLE passing_exams ADD CONSTRAINT passing_exams_exams
    FOREIGN KEY (ExamID)
    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

```
CREATE TABLE passing_practices (
    PracticeID int  NOT NULL,
    EventStudentID int  NOT NULL,
    IsPresent bit  NULL,
    IsPassed bit  NULL,
    CONSTRAINT passing_practices_pk PRIMARY KEY  (PracticeID,EventStudentID),
    CHECK (IsPresent IS NULL OR IsPresent = 1 OR (IsPresent = 0 AND IsPassed = 0))
);

ALTER TABLE passing_practices ADD CONSTRAINT passing_practices_event_students
    FOREIGN KEY (EventStudentID)
    REFERENCES event_students (EventStudentID);

ALTER TABLE passing_practices ADD CONSTRAINT passing_practices_practices
    FOREIGN KEY (PracticeID)
    REFERENCES practices (PracticeID);
```

4.15 Tabela payment_statuses

PaymentStatusID - Identyfikator statusu płatności

PaymentStatusName - Nazwa statusu płatności

```
CREATE TABLE payment_statuses (
    PaymentStatusID int  NOT NULL IDENTITY,
    PaymentStatusName varchar(30)  NOT NULL,
    CONSTRAINT payment_statuses_pk PRIMARY KEY  (PaymentStatusID)
);
```

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

```
01 | CREATE TABLE practices (
02 |     PracticeID int  NOT NULL IDENTITY,
03 |     EventID int  NOT NULL,
04 |     PracticeNo int  NOT NULL,
05 |     Description varchar(30)  NOT NULL,
06 |     BeginDate datetime  NOT NULL,
07 |     EndDate datetime  NOT NULL,
08 |     RoomID int  NOT NULL,
09 |     CONSTRAINT practices_pk PRIMARY KEY  (PracticeID),
10 |     CHECK (EndDate > BeginDate)
11 | );
12 |
13 | ALTER TABLE practices ADD CONSTRAINT practices_events
14 |     FOREIGN KEY (EventID)
15 |     REFERENCES events (EventID);
16 |
17 | ALTER TABLE practices ADD CONSTRAINT practices_rooms
18 |     FOREIGN KEY (RoomID)
19 |     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

```
01 | CREATE TABLE rooms (
02 |     RoomID int  NOT NULL IDENTITY,
03 |     Name varchar(30)  NOT NULL,
04 |     Capacity int  NOT NULL,
05 |     Availability bit  NOT NULL,
06 |     Equipment varchar(30)  NOT NULL,
07 |     CONSTRAINT rooms_pk PRIMARY KEY  (RoomID)
08 | );
```

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

```
01 | CREATE TABLE single_module_students (
02 |     SingleModuleStudentID int  NOT NULL IDENTITY,
03 |     ModuleID int  NOT NULL,
04 |     StudentID int  NOT NULL,
05 |     IsPaid bit  NOT NULL,
06 |     IsPresent bit  NULL,
07 |     CONSTRAINT single_module_students_pk PRIMARY KEY  (SingleModuleStudentID)
08 | );
09 |
10 | ALTER TABLE single_module_students ADD CONSTRAINT module_students_modules
11 |     FOREIGN KEY (ModuleID)
12 |     REFERENCES modules (ModuleID);
13 |
14 | ALTER TABLE single_module_students ADD CONSTRAINT module_students_students
15 |     FOREIGN KEY (StudentID)
16 |     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

CountryID - Identyfikator kraju zamieszkania studenta

Phone - Numer telefonu studenta

```
01 | CREATE TABLE students (
02 |     StudentID int  NOT NULL IDENTITY,
03 |     FirstName varchar(30)  NOT NULL,
04 |     LastName varchar(30)  NOT NULL,
05 |     Email varchar(30)  NOT NULL,
06 |     BirthDate date  NOT NULL,
07 |     Address varchar(30)  NOT NULL,
```

```
    City varchar(30)  NOT NULL,
    Region varchar(30)  NOT NULL,
    PostalCode varchar(30)  NOT NULL,
    CountryID int  NOT NULL,
    Phone varchar(30)  NOT NULL,
    CONSTRAINT students_pk PRIMARY KEY  (StudentID),
    CHECK (BirthDate > '01-01-1910')
);

ALTER TABLE students ADD CONSTRAINT students_countries
    FOREIGN KEY (CountryID)
    REFERENCES countries (CountryID);
```

4.20 Tabela translator_languages

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

TranslatorID - Identyfikator tłumacza

LanguageID - Identyfikator języka

```
CREATE TABLE translator_languages (
    TranslatorID int  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT translator_languages_pk PRIMARY KEY  (TranslatorID,LanguageID)
);

ALTER TABLE translator_languages ADD CONSTRAINT translator_languages_languages
    FOREIGN KEY (LanguageID)
    REFERENCES languages (LanguageID);

ALTER TABLE translator_languages ADD CONSTRAINT translator_languages_translators
    FOREIGN KEY (TranslatorID)
    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

```
CREATE TABLE translators (
    TranslatorID int  NOT NULL IDENTITY,
    FirstName varchar(30)  NOT NULL,
    LastName varchar(30)  NOT NULL,
    Email varchar(30)  NOT NULL,
    CONSTRAINT translators_pk PRIMARY KEY  (TranslatorID)
);
```

4.22 Tabela types

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

TypeID - Identyfikator typu

TypeName - Nazwa typu

```
CREATE TABLE types (
    TypeID int  NOT NULL IDENTITY,
    TypeName varchar(30)  NOT NULL,
    CONSTRAINT types_pk PRIMARY KEY  (TypeID)
);
```

4.23 Tabela countries

Tabela jest słownikiem państw.

CountryID - Identyfikator państwa

CountryName - Nazwa państwa

isActive - Informacja czy państwo nadal istnieje

```
CREATE TABLE countries (
    CountryID int  NOT NULL IDENTITY,
    CountryName varchar(30)  NOT NULL,
    IsActive bit  NOT NULL,
    CONSTRAINT countries_pk PRIMARY KEY  (CountryID)
);
```

5 Widoki

5.1 UpcomingEvents

```
01 | CREATE VIEW UpcomingEvents AS
02 | SELECT * FROM events
03 | WHERE BeginDate>GETDATE()
```

5.2 ShowBilocations

Widok wyświetla informację, którzy studenci mają zajęcia kolidujące ze sobą.

```
01 | CREATE VIEW [dbo].[ShowBilocations] AS
02 |     WITH modulesWithDetails AS (
03 |             SELECT S.StudentID, S.FirstName, S.LastName,  E.Title, E.EventID, M.
   |     BeginDate, M.EndDate, M.ModuleID
04 |             FROM students S
05 |             JOIN event_students ES ON ES.StudentID = S.StudentID
06 |             JOIN events E ON E.EventID = ES.EventID
07 |             JOIN modules M ON M.EventID = E.EventID
08 |         )
09 |     SELECT
10 |         m1.StudentID as StudentID,
11 |         m1.FirstName as FirstName,
12 |         m1.LastName as LastName,
13 |         m1.Title AS Title1,
14 |         m1.EventID AS EventID1,
15 |         m1.ModuleID AS ModuleID1,
16 |         m1.BeginDate AS BeginDate1,
17 |         m1.EndDate AS EndDate1,
18 |         m2.Title AS Title2,
19 |         m2.EventID AS EventID2,
20 |         m2.ModuleID AS ModuleID2,
21 |         m2.BeginDate AS BeginDate2,
22 |         m2.EndDate AS EndDate2
23 |     FROM modulesWithDetails m1
24 |     JOIN modulesWithDetails m2 ON m1.ModuleID > m2.ModuleID
25 |     WHERE m1.BeginDate <= m2.EndDate
26 |     AND m1.EndDate >= m2.BeginDate
27 |     AND m1.StudentID = m2.StudentID
```

6 Procedury

6.1 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),
@CountryID varchar(30),
@Phone varchar(30)
AS
BEGIN
    INSERT INTO students (FirstName, LastName, Email, BirthDate, Address, City, Region, PostalCode, CountryID, Phone)
    VALUES (@FirstName, @LastName, @Email, @BirthDate, @Address, @City, @Region, @PostalCode, @CountryID, @Phone)
END
```

6.2 RegisterLecturer

Rejestruje nowego wykładowcę do bazy danych

```
CREATE PROCEDURE 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),
@CountryID varchar(30),
@Phone varchar(30),
@LanguageID INT
AS
BEGIN
    IF @HireDate IS NULL
    SET @HireDate = GETDATE();

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

6.3 AddEvent

Dodaje nowe studia lub kurs.

```
CREATE PROCEDURE AddEvent
@Title varchar(30),
@TypeID INT,
@BeginDate DATE,
@EndDate DATE,
@Capacity INT,
@EventPrice DECIMAL(10,2),
```

```
@Description varchar(30)

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

6.4 AddModule

Dodaje nowy moduł dla kursu lub dla studiów.

```
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

    IF EXISTS (
        SELECT 1
        FROM dbo.RoomsTaken(@BeginDate, @EndDate) as sub
        WHERE sub.RoomID = @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
```

6.5 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
```

6.6 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 (
```

```
24 |             SELECT EM.ModuleID, ROW_NUMBER() OVER (ORDER BY EM.ModuleID) AS ROW_NUM
   FROM @EventModules AS EM
25 |         ) as Sub
26 |         WHERE ROW_NUM = @RowCount
27 |         INSERT INTO attendance (EventStudentID, ModuleID, IsPresent)
28 |         VALUES (@LastIdentity, @TempModuleID, NULL)
29 |         SET @RowCount = @RowCount+1;
30 |     END
31 | END
```

7 Funkcje

7.1 RoomsTaken

Funkcja zwraca tabelę z zajętymi pomieszczeniami w zadanym przedziale czasowym. Przyjmuje argumenty:

@BeginDate - Data początkowa sprawdzanego zakresu

@EndDate - Data końcowa sprawdzanego zakresu

```
01 | CREATE FUNCTION dbo.RoomsTaken
02 | (
03 |     @BeginDate datetime,
04 |     @EndDate datetime
05 | )
06 | RETURNS TABLE
07 | AS
08 | RETURN (
09 |     SELECT R.RoomID
10 |     FROM modules M
11 |     JOIN rooms R ON R.RoomID = M.RoomID
12 |     WHERE ((M.BeginDate BETWEEN @BeginDate AND @EndDate) OR (M.EndDate BETWEEN
       @BeginDate AND @EndDate)) AND R.Name<>'ONLINE'
13 | );
```

7.2 Bilocation

Funkcja zwraca tabelę z zajęciami, które nachodzą na siebie dla podanego studenta. Przyjmuje argumenty:

@StudentID - ID sprawdzanego studenta

```
01 | CREATE FUNCTION Bilocation
02 | (
03 |     @StudentID int
04 | )
05 | RETURNS TABLE
06 | AS
07 | RETURN (
08 |     WITH modulesWithDetails AS (
09 |         SELECT E.EventID, M.BeginDate, M.EndDate, M.ModuleID
10 |         FROM students S
11 |         JOIN event_students ES ON ES.StudentID = S.StudentID
12 |         JOIN events E ON E.EventID = ES.EventID
13 |         JOIN modules M ON M.EventID = E.EventID
14 |         WHERE S.StudentID = @StudentID
15 |     )
16 |     SELECT
17 |         m1.EventID AS EventID1,
18 |         m1.ModuleID AS ModuleID1,
19 |         m1.BeginDate AS BeginDate1,
20 |         m1.EndDate AS EndDate1,
21 |         m2.EventID AS EventID2,
22 |         m2.ModuleID AS ModuleID2,
23 |         m2.BeginDate AS BeginDate2,
24 |         m2.EndDate AS EndDate2
25 |     FROM modulesWithDetails m1
26 |     JOIN modulesWithDetails m2 ON m1.ModuleID > m2.ModuleID
27 |     WHERE m1.BeginDate <= m2.EndDate
28 |     AND m1.EndDate >= m2.BeginDate
29 | );
```

7.3 ShowModuleParticipants

Funkcja zwraca tabelę z informacją o liście studnetów zapisanych na dany moduł. Przyjmuje argumenty:

@ModuleID - Identyfikator moudłu

```
CREATE FUNCTION [dbo].[ShowModuleParticipants]
(
    @ModuleID int
)
RETURNS TABLE
AS
RETURN (
    SELECT S.StudentID, S.FirstName, S.LastName FROM event_students ES
    JOIN students S ON S.StudentID = ES.StudentID
    JOIN modules M ON M.EventID = ES.EventID
    WHERE M.ModuleID = @ModuleID
    UNION ALL
    SELECT SMS.StudentID, S.FirstName, S.LastName FROM single_module_students SMS
    JOIN students S ON S.StudentID = SMS.StudentID
    WHERE SMS.ModuleID = @ModuleID
);
```

7.4 ShowEventParticipants

Funkcja zwraca tabelę z informacją o liście studnetów zapisanych na dane wydarzenie. Przyjmuje argumenty:

@EventID - Identyfikator wydarzenia

```
CREATE FUNCTION [dbo].[ShowEventParticipants]
(
    @EventID int
)
RETURNS TABLE
AS
RETURN (
    SELECT S.StudentID, S.FirstName, S.LastName
    FROM students S
    JOIN event_students ES ON ES.StudentID = S.StudentID
    WHERE ES.EventID = @EventID
);
```

7.5 ShowMostPopularEvents

Pokazuje listę 10 najpopularniejszych wydarzeń, które rozpoczynają się oraz kończą w zadanym przedziale czasowym.

```
CREATE FUNCTION [dbo].[ShowMostPopularEvents]
(
    @BeginDate DATE,
    @EndDate DATE
)
RETURNS TABLE
AS
RETURN (
    SELECT TOP 10 E.EventID, E.Title, COUNT(*) as Count FROM event_students ES
    JOIN events E ON E.EventID = ES.EventID
    WHERE E.BeginDate >= @BeginDate AND E.EndDate<=@EndDate
    GROUP BY E.EventID, E.Title
    ORDER BY COUNT(*) DESC
);
```

7.6 ShowMostPopularSingleModules

Pokazuje listę 10 najpopularniejszych pojedynczych modułów w zadanym przedziale czasowym, na które zapisywali się studenci.

```
01 | CREATE FUNCTION [dbo].[ShowMostPopularSingleModules]
02 | (
03 |     @BeginDate DATE,
04 |     @EndDate DATE
05 | )
06 | RETURNS TABLE
07 | AS
08 | RETURN (
09 |     SELECT TOP 10 M.ModuleID, COUNT(*) as Count  FROM single_module_students SMS
10 |     JOIN modules M ON SMS.ModuleID = M.ModuleID
11 |     WHERE M.BeginDate >= @BeginDate AND M.EndDate <= @EndDate
12 |     GROUP BY M.ModuleID
13 |     ORDER BY Count DESC
14 | );
```

8 Triggery

8.1 InsertToModules

Trigger w momencie dodawania modułu, dodaje wykładowce z eventem do tabeli event_lecturers, tak aby wszystko było spójne ze sobą.

```
01 | CREATE TRIGGER InsertToModules
02 | ON modules
03 | INSTEAD OF INSERT
04 | AS
05 | BEGIN
06 |
07 |     INSERT INTO event_lecturers (EventID, LecturerID, Description)
08 |     SELECT I.EventID, I.LecturerID, I.Description
09 |     FROM INSERTED I
10 |     LEFT JOIN event_lecturers EL ON I.EventID = EL.EventID AND I.LecturerID = EL.
    LecturerID
11 |     WHERE EL.EventID IS NULL AND EL.LecturerID IS NULL;
12 |
13 |     INSERT INTO modules (EventID, ModuleNo, LecturerID, LanguageID, TranslatorID,
    Description, BeginDate, EndDate, Capacity, RoomID, ModulePrice)
14 |     SELECT I.EventID, I.ModuleNo, I.LecturerID, I.LanguageID, I.TranslatorID, I.
    Description, I.BeginDate, I.EndDate, I.Capacity, I.RoomID, I.ModulePrice
15 |     FROM INSERTED I
16 | END;
```

8.2 TriggerAddStudentAfterCompletedPayment

Trigger w momencie zmiany statusu płatności na zakończoną automatycznie zapisuje studenta na zakupione wydarzenie.

```
01 | CREATE TRIGGER TriggerAddStudentAfterCompletedPayment
02 | ON orders
03 | AFTER UPDATE
04 | AS
05 | BEGIN
06 |     SET NOCOUNT ON;
07 |     IF UPDATE(PaymentStatusID)
08 |     BEGIN
09 |         DECLARE @NewStudentID INT, @ToEventID INT;
10 |         SELECT @NewStudentID = O.StudentID, @ToEventID = OES.EventID
11 |         FROM
12 |             inserted O
13 |             JOIN order_event_details OES ON OES.OrderID = O.OrderID
14 |         WHERE O.PaymentStatusID = 3
15 |         IF @NewStudentID IS NOT NULL
16 |         BEGIN
17 |             EXEC EnrollStudent2Event @StudentID = @NewStudentID, @EventID =
    @ToEventID
18 |         END
19 |     END
20 | END;
```