



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

22 STYCZNIA 2024

# Spis treści

<b>1</b>	<b>Wprowadzenie</b>	<b>3</b>
<b>2</b>	<b>Funkcje realizowane przez system</b>	<b>3</b>
2.1	Funkcje użytkowników . . . . .	3
2.1.1	Administrator . . . . .	3
2.1.2	Dyrektor . . . . .	3
2.1.3	Menadżer . . . . .	3
2.1.4	Nauczyciel . . . . .	3
2.1.5	Klient . . . . .	3
2.1.6	Niezarejestrowany użytkownik . . . . .	4
<b>3</b>	<b>Diagram</b>	<b>4</b>
<b>4</b>	<b>Tabele</b>	<b>5</b>
4.1	Tabela employees . . . . .	5
4.2	Tabela attendance . . . . .	5
4.3	Tabela event_lecturers . . . . .	6
4.4	Tabela event_students . . . . .	6
4.5	Tabela events . . . . .	7
4.6	Tabela exams . . . . .	7
4.7	Tabela languages . . . . .	8
4.8	Tabela lecturers . . . . .	8
4.9	Tabela modules . . . . .	9
4.10	Tabela order_event_details . . . . .	10
4.11	Tabela order_module_details . . . . .	10
4.12	Tabela orders . . . . .	11
4.13	Tabela passing_exams . . . . .	11
4.14	Tabela passing_practices . . . . .	12
4.15	Tabela payment_statuses . . . . .	12
4.16	Tabela practices . . . . .	13
4.17	Tabela rooms . . . . .	13
4.18	Tabela single_module_students . . . . .	14
4.19	Tabela students . . . . .	14
4.20	Tabela translator_languages . . . . .	15
4.21	Tabela translators . . . . .	15
4.22	Tabela types . . . . .	15
4.23	Tabela countries . . . . .	16
<b>5</b>	<b>Widoki</b>	<b>17</b>
5.1	UpcomingEvents . . . . .	17
5.2	ShowBilocations . . . . .	17
5.3	ShowDebtors . . . . .	17
<b>6</b>	<b>Procedury</b>	<b>18</b>
6.1	RegisterStudent . . . . .	18
6.2	RegisterLecturer . . . . .	18
6.3	AddEvent . . . . .	18
6.4	AddModule . . . . .	19
6.5	AddWebinar . . . . .	19
6.6	EnrollStudent2Event . . . . .	20

<b>7</b>	<b>Funkcje</b>	<b>22</b>
7.1	RoomsTaken . . . . .	22
7.2	Bilocation . . . . .	22
7.3	ShowModuleParticipants . . . . .	22
7.4	ShowEventParticipants . . . . .	23
7.5	ShowMostPopularEvents . . . . .	23
7.6	ShowMostPopularSingleModules . . . . .	24
7.7	StudentsCountInEvent . . . . .	24
7.8	ShowAttendance . . . . .	24
7.9	ShowIncomeReport . . . . .	25
7.10	ShowMostActiveStudents . . . . .	25
7.11	GetPassedExamsStudents . . . . .	25
7.12	GetPassedPracticesStudents . . . . .	26
7.13	GetGraduatedStudentsAddresses . . . . .	26
<b>8</b>	<b>Triggery</b>	<b>27</b>
8.1	InsertToModules . . . . .	27
8.2	TriggerAddStudentAfterCompletedPayment . . . . .	27
<b>9</b>	<b>Indeksy</b>	<b>28</b>
9.1	Tabela employees . . . . .	28
9.2	Tabela attendance . . . . .	28
9.3	Tabela event_lecturers . . . . .	28
9.4	Tabela event_students . . . . .	28
9.5	Tabela events . . . . .	28
9.6	Tabela exams . . . . .	28
9.7	Tabela languages . . . . .	28
9.8	Tabela lecturers . . . . .	29
9.9	Tabela modules . . . . .	29
9.10	Tabela order_event_details . . . . .	29
9.11	Tabela order_module_details . . . . .	29
9.12	Tabela orders . . . . .	29
9.13	Tabela passing_exams . . . . .	29
9.14	Tabela passing_practices . . . . .	30
9.15	Tabela payment_statuses . . . . .	30
9.16	Tabela practices . . . . .	30
9.17	Tabela rooms . . . . .	30
9.18	Tabela single_module_students . . . . .	30
9.19	Tabela students . . . . .	30
9.20	Tabela translator_languages . . . . .	30
9.21	Tabela translators . . . . .	31
9.22	Tabela types . . . . .	31
9.23	Tabela countries . . . . .	31
<b>10</b>	<b>Uprawnienia</b>	<b>32</b>
10.1	Administrator . . . . .	32
10.2	Dyrektor . . . . .	32
10.3	Menadżer . . . . .	32
10.4	Nauczyciel . . . . .	33
10.5	Klient . . . . .	33
10.6	Niezarejestrowany użytkownik . . . . .	34

# 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 niejednorodny 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 webinarów 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 webinarów, 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

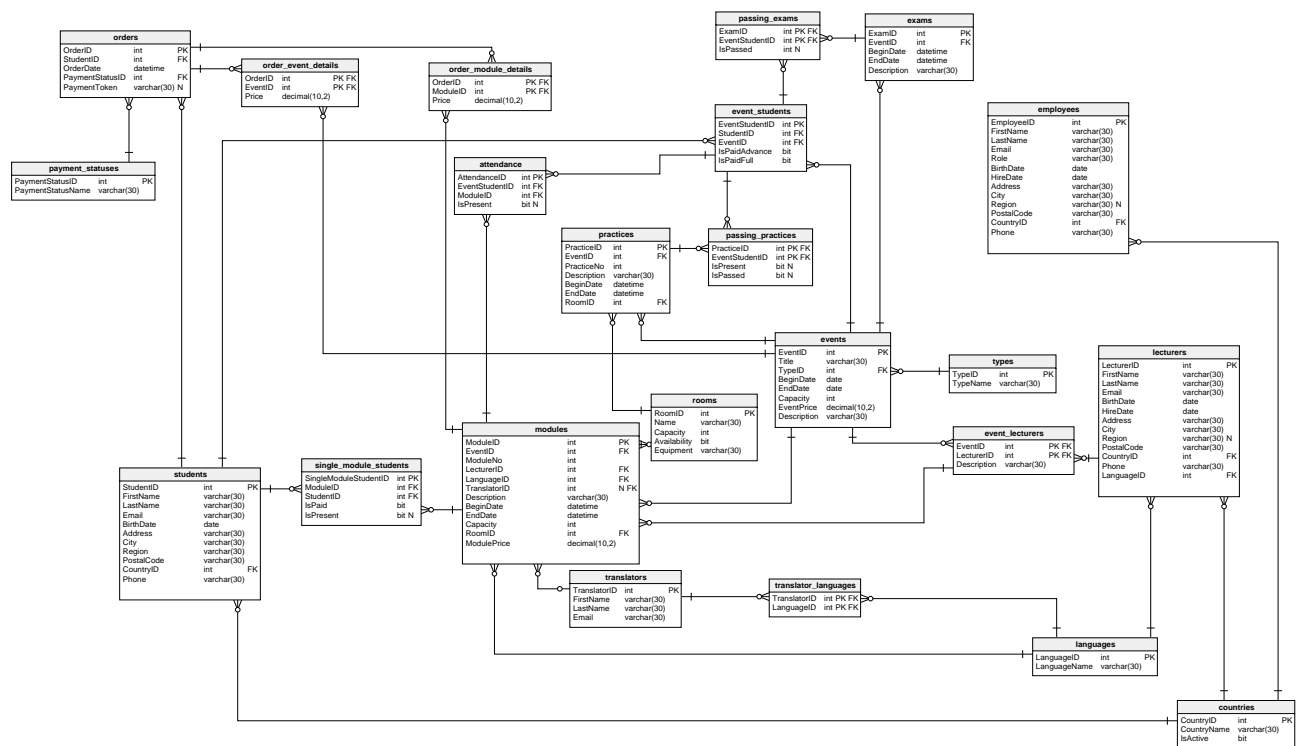
- Prowadzenie webinarów, kursów oraz studiów
- Dostęp do listy klientów i ich obecności na poszczególnych modułach zajęć

#### 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)

- 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

```
01 | CREATE TABLE attendance (  
02 |     AttendanceID int NOT NULL IDENTITY,  
03 |     EventStudentID int NOT NULL,  
04 |     ModuleID int NOT NULL,  
05 |     IsPresent bit NULL,  
06 |     CONSTRAINT attendance_pk PRIMARY KEY (AttendanceID)  
07 | );  
08 |  
09 | ALTER TABLE attendance ADD CONSTRAINT attendance_event_students  
10 |     FOREIGN KEY (EventStudentID)  
11 |     REFERENCES event_students (EventStudentID);  
12 |  
13 | ALTER TABLE attendance ADD CONSTRAINT attendance_modules  
14 |     FOREIGN KEY (ModuleID)  
15 |     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

```
01 | CREATE TABLE event_lecturers (  
02 |     EventID int NOT NULL,  
03 |     LecturerID int NOT NULL,  
04 |     Description varchar(30) NOT NULL,  
05 |     CONSTRAINT event_lecturers_pk PRIMARY KEY (EventID,LecturerID)  
06 | );  
07 |  
08 | ALTER TABLE event_lecturers ADD CONSTRAINT event_lecturers_events  
09 |     FOREIGN KEY (EventID)  
10 |     REFERENCES events (EventID);  
11 |  
12 | ALTER TABLE event_lecturers ADD CONSTRAINT event_lecturers_lecturers  
13 |     FOREIGN KEY (LecturerID)  
14 |     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

```

01 | CREATE TABLE event_students (
02 |     EventStudentID int NOT NULL IDENTITY,
03 |     StudentID int NOT NULL,
04 |     EventID int NOT NULL,
05 |     IsPaidAdvance bit NOT NULL,
06 |     IsPaidFull bit NOT NULL,
07 |     CONSTRAINT event_students_pk PRIMARY KEY (EventStudentID)
08 | );
09 |
10 | ALTER TABLE event_students ADD CONSTRAINT event_students_events
11 |     FOREIGN KEY (EventID)
12 |     REFERENCES events (EventID);
13 |
14 | ALTER TABLE event_students ADD CONSTRAINT event_students_students
15 |     FOREIGN KEY (StudentID)
16 |     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

```

01 | CREATE TABLE events (
02 |     EventID int NOT NULL IDENTITY,
03 |     Title varchar(30) NOT NULL,
04 |     TypeID int NOT NULL,
05 |     BeginDate date NOT NULL,
06 |     EndDate date NOT NULL,
07 |     Capacity int NOT NULL,
08 |     EventPrice decimal(10,2) NOT NULL,
09 |     Description varchar(30) NOT NULL,
10 |     CONSTRAINT events_pk PRIMARY KEY (EventID),
11 |     CHECK (EndDate > BeginDate),
12 |     CHECK (EventPrice >= 0)
13 | );
14 |
15 | ALTER TABLE events ADD CONSTRAINT events_types
16 |     FOREIGN KEY (TypeID)
17 |     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

```
01 | CREATE TABLE exams (  
02 |     ExamID int NOT NULL IDENTITY,  
03 |     EventID int NOT NULL,  
04 |     BeginDate datetime NOT NULL,  
05 |     EndDate datetime NOT NULL,  
06 |     Description varchar(30) NOT NULL,  
07 |     CONSTRAINT exams_pk PRIMARY KEY (ExamID),  
08 |     CHECK (EndDate > BeginDate)  
09 | );  
10 |  
11 | ALTER TABLE exams ADD CONSTRAINT exams_events  
12 |     FOREIGN KEY (EventID)  
13 |     REFERENCES events (EventID);
```

## 4.7 Tabela languages

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

**LanguageID** - Identyfikator języka

**LanguageName** - Nazwa języka

```
01 | CREATE TABLE languages (  
02 |     LanguageID int NOT NULL IDENTITY,  
03 |     LanguageName varchar(30) NOT NULL,  
04 |     CONSTRAINT languages_pk PRIMARY KEY (LanguageID)  
05 | );
```

## 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

```
01 | CREATE TABLE lecturers (  
02 |     LecturerID 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 |     HireDate date NOT NULL,  
08 |     Address varchar(30) NOT NULL,  
09 |     City varchar(30) NOT NULL,  
10 |     Region varchar(30) NULL,  
11 |     PostalCode varchar(30) NOT NULL,  
12 |     CountryID int NOT NULL,  
13 |     Phone varchar(30) NOT NULL,  
14 |     LanguageID int NOT NULL,  
15 |     CONSTRAINT lecturers_pk PRIMARY KEY (LecturerID),  
16 |     CHECK (BirthDate > '01-01-1910'),  
17 |     CHECK (HireDate < getdate())  
18 | );  
19 |  
20 | ALTER TABLE lecturers ADD CONSTRAINT lecturers_countries  
21 |     FOREIGN KEY (CountryID)  
22 |     REFERENCES countries (CountryID);  
23 |  
24 | ALTER TABLE lecturers ADD CONSTRAINT lecturers_languages  
25 |     FOREIGN KEY (LanguageID)  
26 |     REFERENCES languages (LanguageID);
```

## 4.9 Tabela modules

Zawiera informacje na temat pojedynczego 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

```
01 | CREATE TABLE modules (  
02 |     ModuleID int NOT NULL IDENTITY,  
03 |     EventID int NOT NULL,  
04 |     ModuleNo int NOT NULL,  
05 |     LecturerID int NOT NULL,  
06 |     LanguageID int NOT NULL,  
07 |     TranslatorID int NULL,
```

```

08 | Description varchar(30) NOT NULL,
09 | BeginDate datetime NOT NULL,
10 | EndDate datetime NOT NULL,
11 | Capacity int NOT NULL,
12 | RoomID int NOT NULL,
13 | ModulePrice decimal(10,2) NOT NULL,
14 | CONSTRAINT modules_pk PRIMARY KEY (ModuleID),
15 | CHECK (EndDate > BeginDate),
16 | CHECK (ModulePrice >= 0)
17 | );
18 |
19 | ALTER TABLE modules ADD CONSTRAINT modules_event_lecturers
20 | FOREIGN KEY (EventID,LecturerID)
21 | REFERENCES event_lecturers (EventID,LecturerID);
22 |
23 | ALTER TABLE modules ADD CONSTRAINT modules_events
24 | FOREIGN KEY (EventID)
25 | REFERENCES events (EventID);
26 |
27 | ALTER TABLE modules ADD CONSTRAINT modules_languages
28 | FOREIGN KEY (LanguageID)
29 | REFERENCES languages (LanguageID);
30 |
31 | ALTER TABLE modules ADD CONSTRAINT modules_rooms
32 | FOREIGN KEY (RoomID)
33 | REFERENCES rooms (RoomID);
34 |
35 | ALTER TABLE modules ADD CONSTRAINT modules_translators
36 | FOREIGN KEY (TranslatorID)
37 | 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

```

01 | CREATE TABLE order_event_details (
02 | OrderID int NOT NULL,
03 | EventID int NOT NULL,
04 | Price decimal(10,2) NOT NULL,
05 | CONSTRAINT order_event_details_pk PRIMARY KEY (OrderID,EventID),
06 | CHECK (Price >= 0)
07 | );
08 |
09 | ALTER TABLE order_event_details ADD CONSTRAINT order_event_details_orders
10 | FOREIGN KEY (OrderID)
11 | REFERENCES orders (OrderID);
12 |
13 | ALTER TABLE order_event_details ADD CONSTRAINT order_module_details_events
14 | FOREIGN KEY (EventID)
15 | 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

```
01 | CREATE TABLE order_module_details (  
02 |     OrderID int NOT NULL,  
03 |     ModuleID int NOT NULL,  
04 |     Price decimal(10,2) NOT NULL,  
05 |     CONSTRAINT order_module_details_pk PRIMARY KEY (OrderID,ModuleID),  
06 |     CHECK (Price >= 0)  
07 | );  
08 |  
09 | ALTER TABLE order_module_details ADD CONSTRAINT order_module_details_modules  
10 |     FOREIGN KEY (ModuleID)  
11 |     REFERENCES modules (ModuleID);  
12 |  
13 | ALTER TABLE order_module_details ADD CONSTRAINT order_module_details_orders  
14 |     FOREIGN KEY (OrderID)  
15 |     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 IDENTITY,  
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 |     CHECK (IsPresent IS NULL OR IsPresent = 1 OR (IsPresent = 0 AND IsPassed = 0))
08 | );
09 |
10 | ALTER TABLE passing_practices ADD CONSTRAINT passing_practices_event_students
11 |     FOREIGN KEY (EventStudentID)
12 |     REFERENCES event_students (EventStudentID);
13 |
14 | ALTER TABLE passing_practices ADD CONSTRAINT passing_practices_practices
15 |     FOREIGN KEY (PracticeID)
16 |     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 IDENTITY,
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

```
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,
```

```

08 |      City varchar(30) NOT NULL,
09 |      Region varchar(30) NOT NULL,
10 |      PostalCode varchar(30) NOT NULL,
11 |      CountryID int NOT NULL,
12 |      Phone varchar(30) NOT NULL,
13 |      CONSTRAINT students_pk PRIMARY KEY (StudentID),
14 |      CHECK (BirthDate > '01-01-1910')
15 | );
16 |
17 | ALTER TABLE students ADD CONSTRAINT students_countries
18 |     FOREIGN KEY (CountryID)
19 |     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

```

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 IDENTITY,
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 IDENTITY,  
03 |     TypeName varchar(30) NOT NULL,  
04 |     CONSTRAINT types_pk PRIMARY KEY (TypeID)  
05 | );
```

#### 4.23 Tabela countries

Tabela jest słownikiem państw.

**CountryID** - Identyfikator państwa

**CountryName** - Nazwa państwa

**isActive** - Informacja czy państwo nadal istnieje

```
01 | CREATE TABLE countries (  
02 |     CountryID int NOT NULL IDENTITY,  
03 |     CountryName varchar(30) NOT NULL,  
04 |     IsActive bit NOT NULL,  
05 |     CONSTRAINT countries_pk PRIMARY KEY (CountryID)  
06 | );
```

## 5 Widoki

### 5.1 UpcomingEvents

Wyświetla listę zaplanowanych wydarzeń.

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

### 5.3 ShowDebtors

Widok pokazuje dłużników, czyli osoby, które uczestniczyły w wydarzeniu oraz mają zaległe opłaty do zrealizowania.

```
01 | CREATE VIEW [dbo].[ShowDebtors] AS
02 | SELECT S.StudentID, S.FirstName, S.LastName, SUM(E.EventPrice) as LeftToPay FROM
03 |     event_students ES
04 | JOIN events E ON ES.EventID = E.EventID
05 | JOIN students S On S.StudentID = ES.StudentID
06 | WHERE E.EndDate < GETDATE() AND IsPaidFull <> 1
07 | GROUP BY S.StudentID, S.FirstName, S.LastName
```

## 6 Procedury

### 6.1 RegisterStudent

Rejestruje nowego studenta do bazy danych.

```
01 | CREATE PROCEDURE RegisterStudent
02 | @FirstName varchar(30),
03 | @LastName varchar(30),
04 | @Email varchar(30),
05 | @BirthDate date,
06 | @Address varchar(30),
07 | @City varchar(30),
08 | @Region varchar(30),
09 | @PostalCode varchar(30),
10 | @CountryID varchar(30),
11 | @Phone varchar(30)
12 | AS
13 | BEGIN
14 |     INSERT INTO students (FirstName, LastName, Email, BirthDate, Address, City,
15 |         Region, PostalCode, CountryID, Phone)
16 |     VALUES (@FirstName, @LastName, @Email, @BirthDate, @Address, @City, @Region,
17 |         @PostalCode, @CountryID, @Phone)
18 | END
```

### 6.2 RegisterLecturer

Rejestruje nowego wykładowcę do bazy danych

```
01 | CREATE PROCEDURE RegisterLecturer
02 | @FirstName varchar(30),
03 | @LastName varchar(30),
04 | @Email varchar(30),
05 | @BirthDate date,
06 | @HireDate date = NULL,
07 | @Address varchar(30),
08 | @City varchar(30),
09 | @Region varchar(30),
10 | @PostalCode varchar(30),
11 | @CountryID varchar(30),
12 | @Phone varchar(30),
13 | @LanguageID INT
14 | AS
15 | BEGIN
16 |     IF @HireDate IS NULL
17 |         SET @HireDate = GETDATE();
18 |
19 |     INSERT INTO lecturers (FirstName, LastName, Email, BirthDate, HireDate, Address,
20 |         City, Region, PostalCode, CountryID, Phone, LanguageID)
21 |     VALUES (@FirstName, @LastName, @Email, @BirthDate, @HireDate, @Address, @City,
22 |         @Region, @PostalCode, @CountryID, @Phone, @LanguageID)
23 | END
```

### 6.3 AddEvent

Dodaje nowe studia lub kurs.

```
01 | CREATE PROCEDURE AddEvent
02 | @Title varchar(30),
03 | @TypeID INT,
04 | @BeginDate DATE,
05 | @EndDate DATE,
06 | @Capacity INT,
07 | @EventPrice DECIMAL(10,2),
```

```

08 | @Description varchar(30)
09 |
10 | AS
11 | BEGIN
12 |     INSERT INTO events (Title, TypeID, BeginDate, EndDate, Capacity, EventPrice,
13 |         Description)
14 |     VALUES (@Title, @TypeID, @BeginDate, @EndDate, @Capacity, @EventPrice,
15 |         @Description)
16 | END

```

## 6.4 AddModule

Dodaje nowy moduł dla kursu lub dla studiów.

```

01 | CREATE PROCEDURE AddModule
02 | @EventID INT,
03 | @ModuleNo INT,
04 | @LecturerID INT,
05 | @TranslatorID INT = NULL,
06 | @Description VARCHAR(30) = '',
07 | @BeginDate DATETIME,
08 | @EndDate DATETIME,
09 | @Capacity INT = NULL,
10 | @RoomID INT,
11 | @ModulePrice INT
12 | AS
13 | BEGIN
14 |     DECLARE @LanguageID INT
15 |     IF @TranslatorID IS NULL
16 |         SELECT @LanguageID = LanguageID FROM lecturers WHERE LecturerID = @LecturerID
17 |     ELSE
18 |         SELECT @LanguageID = LanguageID FROM languages WHERE LanguageName = 'polish'
19 |
20 |     IF @Capacity IS NULL
21 |     BEGIN
22 |         DECLARE @RoomCapacity INT
23 |         SELECT @RoomCapacity = Capacity FROM rooms WHERE RoomID = @RoomID
24 |         SET @Capacity = @RoomCapacity
25 |     END
26 |
27 |     DECLARE @EventCapacity INT
28 |     SELECT @EventCapacity = Capacity FROM events WHERE EventID = @EventID
29 |     IF @Capacity < @EventCapacity
30 |         THROW 50000, 'Capacity is less than EventCapacity', 1
31 |
32 |     IF EXISTS (
33 |         SELECT 1
34 |         FROM dbo.RoomsTaken(@BeginDate, @EndDate) as sub
35 |         WHERE sub.RoomID = @RoomID
36 |     )
37 |         THROW 500001, 'Room is not available then', 1
38 |
39 |     INSERT INTO modules (EventID, ModuleNo, LecturerID, LanguageID, TranslatorID,
40 |         Description, BeginDate, EndDate, Capacity, RoomID, ModulePrice)
41 |     VALUES (@EventID, @ModuleNo, @LecturerID, @LanguageID, @TranslatorID,
42 |         @Description, @BeginDate, @EndDate, @Capacity, @RoomID, @ModulePrice)
43 | END

```

## 6.5 AddWebinar

Dodaje nowy webinar.

```

01 | CREATE PROCEDURE AddWebinar
02 | @Title varchar(30),

```

```

03 | @BeginDate DATETIME,
04 | @EndDate DATETIME,
05 | @Capacity INT = NULL,
06 | @EventPrice DECIMAL(10,2),
07 | @Description varchar(30),
08 | @LecturerID INT,
09 | @TranslatorID INT = NULL
10 | AS
11 | BEGIN
12 |     DECLARE @TypeID INT;
13 |     SELECT @TypeID =TypeID FROM types WHERE 'webinar' = TypeName
14 |
15 |     DECLARE @LanguageID INT
16 |     IF @TranslatorID IS NULL
17 |         SELECT @LanguageID = LanguageID FROM lecturers WHERE LecturerID = @LecturerID
18 |     ELSE
19 |         SELECT @LanguageID = LanguageID FROM languages WHERE LanguageName = 'polish'
20 |
21 |     IF @Capacity IS NULL
22 |     BEGIN
23 |         SET @Capacity = 1000
24 |     END
25 |
26 |     INSERT INTO events (Title,TypeID, BeginDate, EndDate, Capacity, EventPrice,
27 |     Description)
28 |     VALUES (@Title, @TypeID, CAST(@BeginDate AS DATE), CAST(@EndDate AS DATE),
29 |     @Capacity, @EventPrice, @Description)
30 |
31 |     DECLARE @LastIndex INT
32 |     SET @LastIndex = SCOPE_IDENTITY();
33 |
34 |     INSERT INTO modules(EventID, ModuleNo, LecturerID, LanguageID, TranslatorID, [
35 |     Description], BeginDate, EndDate, Capacity, RoomID, ModulePrice)
36 |     VALUES (@LastIndex, 1, @LecturerID, @LanguageID, @TranslatorID, @Description,
37 |     @BeginDate, @EndDate, @Capacity, 1, @EventPrice)
38 | END

```

## 6.6 EnrollStudent2Event

Zapisuje danego studenta do danego wydarzenia.

Wiemy, że da się tą funkcję zapisać ładniej i lepiej, ale zdecydowaliśmy się na pozostawienie takiej implementacji

```

01 | CREATE PROCEDURE EnrollStudent2Event
02 | @StudentID int,
03 | @EventID int
04 | AS
05 | BEGIN
06 |     INSERT INTO event_students (StudentID, EventID, IsPaidAdvance, IsPaidFull)
07 |     VALUES (@StudentID, @EventID, 1, 1)
08 |     DECLARE @LastIdentity INT
09 |     SET @LastIdentity = SCOPE_IDENTITY();
10 |
11 |     DECLARE @EventModules TABLE (ModuleID INT)
12 |     INSERT INTO @EventModules (ModuleID)
13 |     SELECT ModuleID FROM modules M WHERE M.EventID = @EventID
14 |
15 |     DECLARE @RowCount INT = 1
16 |     DECLARE @TotalRows INT;
17 |
18 |     SELECT @TotalRows = COUNT(*) FROM @EventModules
19 |
20 |     WHILE @RowCount <= @TotalRows
21 |     BEGIN
22 |         DECLARE @TempModuleID INT

```

```

23 |         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
13 |         @BeginDate AND @EndDate)) AND R.Name <> 'ONLINE '
14 | );
```

### 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 studentów zapisanych na dany moduł. Przyjmuje argumenty:

**@ModuleID** - Identyfikator modułu

```
01 | CREATE FUNCTION [dbo].[ShowModuleParticipants]
02 | (
03 |     @ModuleID int
04 | )
05 | RETURNS TABLE
06 | AS
07 | RETURN (
08 |     SELECT S.StudentID, S.FirstName, S.LastName FROM event_students ES
09 |     JOIN students S ON S.StudentID = ES.StudentID
10 |     JOIN modules M ON M.EventID = ES.EventID
11 |     WHERE M.ModuleID = @ModuleID
12 |     UNION ALL
13 |     SELECT SMS.StudentID, S.FirstName, S.LastName FROM single_module_students SMS
14 |     JOIN students S ON S.StudentID = SMS.StudentID
15 |     WHERE SMS.ModuleID = @ModuleID
16 | );
```

### 7.4 ShowEventParticipants

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

**@EventID** - Identyfikator wydarzenia

```
01 | CREATE FUNCTION [dbo].[ShowEventParticipants]
02 | (
03 |     @EventID int
04 | )
05 | RETURNS TABLE
06 | AS
07 | RETURN (
08 |     SELECT S.StudentID, S.FirstName, S.LastName
09 |     FROM students S
10 |     JOIN event_students ES ON ES.StudentID = S.StudentID
11 |     WHERE ES.EventID = @EventID
12 | );
```

### 7.5 ShowMostPopularEvents

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

**@BeginDate** - Data początkowa sprawdzanego zakresu

**@EndDate** - Data końcowa sprawdzanego zakresu

```
01 | CREATE FUNCTION [dbo].[ShowMostPopularEvents]
02 | (
03 |     @BeginDate DATE,
04 |     @EndDate DATE
05 | )
06 | RETURNS TABLE
07 | AS
08 | RETURN (
09 |     SELECT TOP 10 E.EventID, E.Title, COUNT(*) as Count FROM event_students ES
10 |     JOIN events E ON E.EventID = ES.EventID
```



```

11 | WHERE E.BeginDate >= @BeginDate AND E.EndDate <= @EndDate
12 | GROUP BY E.EventID, E.Title
13 | ORDER BY COUNT(*) DESC
14 | );

```

## 7.6 ShowMostPopularSingleModules

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

**@BeginDate** - Data początkowa sprawdzanego zakresu

**@EndDate** - Data końcowa sprawdzanego zakresu

```

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 | );

```

## 7.7 StudentsCountInEvent

Zwraca liczbę studentów zapisanych na dane wydarzenie.

**@EventID** - Indeks sprawdzanego wydarzenia

```

01 | CREATE FUNCTION StudentsCountInEvent(@EventID INT)
02 | RETURNS INT
03 | AS
04 | BEGIN
05 |     DECLARE @StudentsNum INT;
06 |     SELECT @StudentsNum = COUNT(*)
07 |     FROM event_students
08 |     WHERE EventID = @EventID;
09 |     RETURN @StudentsNum;
10 | END;

```

## 7.8 ShowAttendance

Zwraca listę studentów danego wydarzenia oraz pokazuje procent ich obecności na zajęciach.

**@EventID** - Indeks sprawdzanego wydarzenia

```

01 | CREATE FUNCTION ShowAttendance
02 | (
03 |     @EventID int
04 | )
05 | RETURNS TABLE
06 | AS
07 | RETURN (
08 |     SELECT FirstName, LastName, SUM(CASE WHEN IsPresent = 1 THEN 1 ELSE 0 END)
      *100/COUNT(*) as PositiveAttendancePercentage

```

```

09 | FROM attendance A
10 | JOIN event_students ES ON ES.EventStudentID = A.EventStudentID
11 | JOIN students S ON S.StudentID = ES.StudentID
12 | WHERE ES.EventID = @EventID
13 | GROUP BY S.FirstName, S.LastName, A.EventStudentID
14 | )

```

## 7.9 ShowIncomeReport

Zwraca listę wydarzeń wraz z przychodem jakie generowały w podanym zakresie czasu.

**@BeginDate** - Data początkowa sprawdzanego zakresu

**@EndDate** - Data końcowa sprawdzanego zakresu

```

01 | CREATE FUNCTION ShowIncomeReport
02 | (
03 |     @BeginDate DATE,
04 |     @EndDate DATE
05 | )
06 | RETURNS TABLE
07 | AS
08 | RETURN (
09 |     SELECT E.EventID, E.Title ,SUM(Price) as Income FROM orders O
10 |     JOIN order_event_details OED ON OED.OrderID=O.OrderID
11 |     JOIN events E ON OED.EventID = E.EventID
12 |     WHERE E.BeginDate>=@BeginDate AND E.EndDate<=@EndDate
13 |     GROUP BY E.EventID, E.Title
14 | )

```

## 7.10 ShowMostActiveStudents

Zwraca listę studentów, którzy biorą udział w jakiś wydarzeniach w zadanym przedziale czasowym oraz liczbę tych wydarzeń dla studenta.

**@BeginDate** - Data początkowa sprawdzanego zakresu

**@EndDate** - Data końcowa sprawdzanego zakresu

```

01 | CREATE FUNCTION [dbo].[ShowMostActiveStudents]
02 | (
03 |     @BeginDate DATE,
04 |     @EndDate DATE
05 | )
06 | RETURNS TABLE
07 | AS
08 | RETURN (
09 |     SELECT S.StudentID, S.FirstName, S.LastName, Count(*) as Count FROM students
10 |     S
11 |     JOIN event_students ES ON S.StudentID = ES.StudentID
12 |     JOIN events E ON E.EventID = ES.EventID
13 |     WHERE E.BeginDate>=@BeginDate AND E.EndDate<=@EndDate
14 |     GROUP BY S.StudentID, S.FirstName, S.LastName
15 | )

```

## 7.11 GetPassedExamsStudents

Zwraca indeksy studentów, którzy zaliczyli wszystkie egzaminy z danego wydarzenia.

**@EventID** - Indeks wydarzenia

```

01 | Create FUNCTION [dbo].[GetPassedExamsStudents]
02 | (
03 |     @EventID int
04 | )
05 | RETURNS TABLE
06 | AS
07 | RETURN (
08 |     SELECT ES.StudentID FROM passing_exams PE
09 |     JOIN event_students ES ON ES.EventStudentID = PE.EventStudentID
10 |     JOIN exams E ON E.ExamID = PE.ExamID
11 |     WHERE E.EventID = @EventID
12 |     GROUP BY ES.StudentID
13 |     HAVING SUM(IsPassed)/COUNT(*) = 1
14 | )

```

## 7.12 GetPassedPracticesStudents

Zwraca indeksy studentów, którzy zaliczyli wszystkie praktyki z danego wydarzenia.

**@EventID** - Indeks wydarzenia

```

01 | Create FUNCTION [dbo].[GetPassedPracticesStudents]
02 | (
03 |     @EventID int
04 | )
05 | RETURNS TABLE
06 | AS
07 | RETURN (
08 |     SELECT ES.StudentID FROM passing_practices PP
09 |     JOIN event_students ES ON PP.EventStudentID = ES.EventStudentID
10 |     WHERE ES.EventID = @EventID
11 |     GROUP BY ES.StudentID
12 |     HAVING SUM(CASE WHEN IsPassed = 1 THEN 1 ELSE 0 END) = COUNT(*)
13 | )

```

## 7.13 GetGraduatedStudentsAddresses

Zwraca adresy uczniów, którzy spełnili wymagania zaliczenia studiów.

**@EventID** - Indeks wydarzenia

```

01 | Create FUNCTION [dbo].[GetGraduatedStudentsAddresses]
02 | (
03 |     @EventID int
04 | )
05 | RETURNS TABLE
06 | AS
07 | RETURN (
08 |     SELECT S.FirstName, S.LastName, S.Address, S.City, S.Region, S.PostalCode, C
09 |     .CountryName FROM ShowAttendance(@EventID) SA
10 |     JOIN students S ON S.StudentID = SA.StudentID
11 |     JOIN countries C ON C.CountryID = S.CountryID
12 |     JOIN GetPassedExamsStudents(@EventID) GPES ON GPES.StudentID = S.StudentID
13 |     JOIN GetPassedPracticesStudents(@EventID) GPPS ON GPPS.StudentID = S.
14 |     StudentID
15 |     WHERE PositiveAttendancePercentage >=80
16 | )

```

## 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
11 |     .LecturerID
12 |     WHERE EL.EventID IS NULL AND EL.LecturerID IS NULL;
13 |
14 |     INSERT INTO modules (EventID, ModuleNo, LecturerID, LanguageID, TranslatorID,
15 |     Description, BeginDate, EndDate, Capacity, RoomID, ModulePrice)
16 |     SELECT I.EventID, I.ModuleNo, I.LecturerID, I.LanguageID, I.TranslatorID, I.
17 |     Description, I.BeginDate, I.EndDate, I.Capacity, I.RoomID, I.ModulePrice
18 |     FROM INSERTED I
19 | 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 =
18 |                 @ToEventID
19 |             END
20 |     END
21 | END;
```

## 9 Indeksy

### 9.1 Tabela employees

Po ID pracownika

```
01 | CREATE UNIQUE INDEX Employees_PKK
02 | ON employees (EmployeeID)
```

### 9.2 Tabela attendance

Po ID obecności na module

```
01 | CREATE UNIQUE INDEX Attendance_PKK
02 | ON attendance (AttendanceID)
```

### 9.3 Tabela event\_lecturers

Po ID wydarzenia

```
01 | CREATE NONCLUSTERED INDEX Event_lecturers_EventID
02 | ON event_lecturers (EventID)
```

Po ID wykładowcy

```
01 | CREATE NONCLUSTERED INDEX Event_lecturers_LecturerID
02 | ON event_lecturers (LecturerID)
```

### 9.4 Tabela event\_students

Po ID przypisania studenta do wydarzenia

```
01 | CREATE UNIQUE INDEX Event_students_PKK
02 | ON event_students (EventStudentID)
```

### 9.5 Tabela events

Po ID wydarzenia

```
01 | CREATE UNIQUE INDEX Events_PKK
02 | ON events (EventID)
```

### 9.6 Tabela exams

Po ID egzaminu

```
01 | CREATE UNIQUE INDEX Exams_PKK
02 | ON exams (ExamID)
```

### 9.7 Tabela languages

Po ID języka

```
01 | CREATE UNIQUE INDEX Languages_PKK
02 | ON languages (LanguageID)
```

## 9.8 Tabela lecturers

Po ID wykładowcy

```
01 | CREATE UNIQUE INDEX Lecturers_PKK  
02 | ON lecturers (LecturerID)
```

## 9.9 Tabela modules

Po ID modułu

```
01 | CREATE UNIQUE INDEX Modules_PKK  
02 | ON modules (ModuleID)
```

## 9.10 Tabela order\_event\_details

Po ID zamówienia

```
01 | CREATE NONCLUSTERED INDEX Order_event_details_OrderID  
02 | ON order_event_details (OrderID)
```

Po ID wydarzenia

```
01 | CREATE NONCLUSTERED INDEX Order_event_details_EventID  
02 | ON order_event_details (EventID)
```

## 9.11 Tabela order\_module\_details

Po ID zamówienia

```
01 | CREATE NONCLUSTERED INDEX Order_module_details_OrderID  
02 | ON order_module_details (OrderID)
```

Po ID modułu

```
01 | CREATE NONCLUSTERED INDEX Order_module_details_ModuleID  
02 | ON order_module_details (ModuleID)
```

## 9.12 Tabela orders

Po ID zamówienia

```
01 | CREATE UNIQUE INDEX Orders_PKK  
02 | ON orders (OrderID)
```

## 9.13 Tabela passing\_exams

Po ID egzaminu

```
01 | CREATE NONCLUSTERED INDEX Passing_exams_ExamID  
02 | ON passing_exams (ExamID)
```

Po ID przypisania studenta do danego wydarzenia

```
01 | CREATE NONCLUSTERED INDEX Passing_exams_EventStudentID  
02 | ON passing_exams (EventStudentID)
```

## 9.14 Tabela passing\_practices

Po ID praktyk

```
01 | CREATE NONCLUSTERED INDEX Passing_practices_PracticeID
02 | ON passing_practices (PracticeID)
```

Po ID przypisania studenta do danego wydarzenia

```
01 | CREATE NONCLUSTERED INDEX Passing_practices_EventStudentID
02 | ON passing_practices (EventStudentID)
```

## 9.15 Tabela payment\_statuses

Po ID statusu płatności

```
01 | CREATE UNIQUE INDEX Payment_statuses_PKK
02 | ON payment_statuses (PaymentStatusID)
```

## 9.16 Tabela practices

Po ID praktyk

```
01 | CREATE UNIQUE INDEX Practices_PKK
02 | ON practices (PracticeID)
```

## 9.17 Tabela rooms

Po ID sali

```
01 | CREATE UNIQUE INDEX Rooms_PKK
02 | ON rooms (RoomID)
```

## 9.18 Tabela single\_module\_students

Po ID przypisania studenta do pojedynczego modułu

```
01 | CREATE UNIQUE INDEX Single_module_students_PKK
02 | ON single_module_students (SingleModuleStudentID)
```

## 9.19 Tabela students

Po ID studenta

```
01 | CREATE UNIQUE INDEX Students_PKK
02 | ON students (StudentID)
```

## 9.20 Tabela translator\_languages

Po ID tłumacza

```
01 | CREATE NONCLUSTERED INDEX Translator_languages_TranslatorID
02 | ON translator_languages (TranslatorID)
```

Po ID języka

```
01 | CREATE NONCLUSTERED INDEX Translator_languages_LanguageID
02 | ON translator_languages (LanguageID)
```

## 9.21 Tabela translators

Po ID tłumacza

```
01 | CREATE UNIQUE INDEX Translators_PKK  
02 | ON translators (TranslatorID)
```

## 9.22 Tabela types

Po ID typu

```
01 | CREATE UNIQUE INDEX Types_PKK  
02 | ON types (TypeID)
```

## 9.23 Tabela countries

Po ID państwa

```
01 | CREATE UNIQUE INDEX Countries_PKK  
02 | ON countries (CountryID)
```



## 10 Uprawnienia

### 10.1 Administrator

Ma uprawnienia do wszystkiego w bazie danych

```
01 | CREATE ROLE admin
02 | GRANT ALL PRIVILEGES ON u_smyda.dbo to admin
```

### 10.2 Dyrektor

Ma uprawnienia do:

- zobaczenia nadchodzących wydarzeń
- zobaczenia studentów mających kolidujące ze sobą zajęcia
- zobaczenia dłużników (osób mających zaległe opłaty do zrealizowania)
- dodania nowych studiów lub kursu
- zobaczenia tabeli zajętych pomieszczeń w podanym przedziale czasowym
- zobaczenia tabeli z nadchodzącymi na siebie wydarzeniami dla podanego studenta
- zobaczenia tabeli z listą studentów zapisanych na dany moduł
- zobaczenia tabeli z listą studentów zapisanych na dane wydarzenie
- zobaczenia listy 10 najpopularniejszych wydarzeń w podanym przedziale czasowym
- zobaczenia listy 10 najpopularniejszych modułów w podanym przedziale czasowym
- liczby studentów zapisanych na dane wydarzenie
- zobaczenia listy studentów danego wydarzenia wraz z procentem ich obecności na zajęciach
- zobaczenia listy wydarzeń wraz z generowanymi przez nie przychodami
- zobaczenia listy studentów biorących udział w wydarzeniach w podanym przedziale czasowym

```
01 | CREATE ROLE director
02 |
03 | GRANT SELECT ON UpcomingEvents to director
04 | GRANT SELECT ON ShowBilocations to director
05 | GRANT SELECT ON ShowDebtors to director
06 | GRANT EXECUTE ON AddEvent to director
07 | GRANT SELECT ON RoomsTaken to director
08 | GRANT SELECT ON Bilocation to director
09 | GRANT SELECT ON ShowModuleParticipants to director
10 | GRANT SELECT ON ShowEventParticipants to director
11 | GRANT SELECT ON ShowMostPopularEvents to director
12 | GRANT SELECT ON ShowMostPopularSingleModules to director
13 | GRANT EXECUTE ON StudentCountInEvent to director
14 | GRANT SELECT ON ShowAttendance to director
15 | GRANT SELECT ON ShowIncomeReport to director
16 | GRANT SELECT ON ShowMostActiveStudents to director
```

### 10.3 Menadżer

Ma uprawnienia do:

- zobaczenia nadchodzących wydarzeń
- rejestrowania nowego studenta do bazy danych

- rejestrowania nowego wykładowcy do bazy danych
- dodania nowych studiów lub kursu
- dodania nowego webinaru
- zobaczenia tabeli zajętych pomieszczeń w podanym przedziale czasowym
- zobaczenia listy 10 najpopularniejszych wydarzeń w podanym przedziale czasowym
- zobaczenia listy 10 najpopularniejszych modułów w podanym przedziale czasowym
- liczby studentów zapisanych na dane wydarzenie

```

01 | CREATE ROLE manager
02 |
03 | GRANT SELECT ON UpcomingEvents to manager
04 | GRANT EXECUTE ON RegisterStudent to manager
05 | GRANT EXECUTE ON RegisterLecturer to manager
06 | GRANT EXECUTE ON AddEvent to manager
07 | GRANT EXECUTE ON AddWebinar to manager
08 | GRANT SELECT ON RoomsTaken to manager
09 | GRANT SELECT ON ShowMostPopularEvents to manager
10 | GRANT SELECT ON ShowMostPopularSingleModules to manager
11 | GRANT EXECUTE ON StudentCountInEvent to manager

```

## 10.4 Nauczyciel

Ma uprawnienia do:

- dodania nowego modułu dla kursu lub dla studiów
- zobaczenia tabeli zajętych pomieszczeń w podanym przedziale czasowym
- zobaczenia tabeli z listą studentów zapisanych na dany moduł
- zobaczenia tabeli z listą studentów zapisanych na dane wydarzenie
- zobaczenia listy 10 najpopularniejszych wydarzeń w podanym przedziale czasowym
- zobaczenia listy 10 najpopularniejszych modułów w podanym przedziale czasowym
- liczby studentów zapisanych na dane wydarzenie
- zobaczenia listy studentów danego wydarzenia wraz z procentem ich obecności na zajęciach

```

01 | CREATE ROLE teacher
02 |
03 | GRANT EXECUTE ON AddModule to teacher
04 | GRANT SELECT ON RoomsTaken to teacher
05 | GRANT SELECT ON ShowModuleParticipants to teacher
06 | GRANT SELECT ON ShowEventParticipants to teacher
07 | GRANT SELECT ON ShowMostPopularEvents to teacher
08 | GRANT SELECT ON ShowMostPopularSingleModules to teacher
09 | GRANT EXECUTE ON StudentCountInEvent to teacher
10 | GRANT SELECT ON ShowAttendance to teacher

```

## 10.5 Klient

Ma uprawnienia do:

- zapisania się na dane wydarzenie
- zobaczenia tabeli z kolidującymi ze sobą zajęciami
- zobaczenia listy 10 najpopularniejszych wydarzeń w podanym przedziale czasowym
- zobaczenia listy 10 najpopularniejszych modułów w podanym przedziale czasowym

– liczby studentów zapisanych na dane wydarzenie

```
01 | CREATE ROLE client
02 |
03 | GRANT EXECUTE ON EnrollStudent2Event to client
04 | GRANT SELECT ON Bilocation to client
05 | GRANT SELECT ON ShowMostPopularEvents to client
06 | GRANT SELECT ON ShowMostPopularSingleModules to client
07 | GRANT EXECUTE ON StudentCountInEvent to client
```

## 10.6 Niezarejestrowany użytkownik

Ma uprawnienie do:

– założenia konta

```
01 | CREATE ROLE unregistered_user
```