

# Projekt PBD

**WYKONAWCY:** Kinga Żarska Karol Markowicz Paulina Dziwak

---

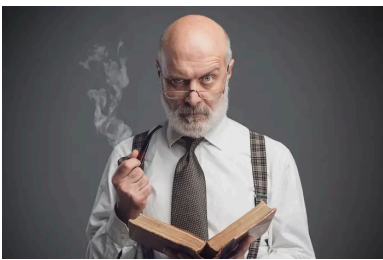
## Użytkownicy systemu i ich uprawnienia w systemie

---

- ADMINISTRATOR



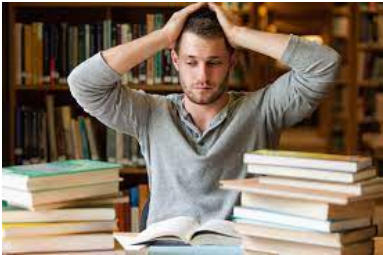
- DYREKTOR SZKOŁY



- WYKŁADOWCA



- UCZESTNIK (zalogowany/niezalogowany)



- KSIĘGOWY



- SYSTEM - Na cw pani wspominała, że wielu z nas brakuje użytkownika system

Tabela praktyk zawiera id praktykanta który odbył praktyki, informacje o nim oraz datę ukończenia praktyki.

### 1.APPRENTICESHIPS:

ApprenticeshipID: Id praktyki (INT)

StudyID: Id kierunku studiów (INT)

StudentID: Id studenta (INT)

Date: Data ukończenia praktyki (NOT NULL)

- klucz główny: ID praktyki
- klucze obce do tabel Users (UserID) i Studies (StudyID)
- domyślnie data ukończenia ustawiana na datę wprowadzania do systemu
- sprawdzamy, czy data jest późniejsza niż przyjęta data powstania systemu
- unikalna para StudentID, Date (student nie mógł zakończyć dwóch praktyk w tym samym czasie)

```
CREATE TABLE Apprenticeships (  
    ApprenticeshipID INT IDENTITY(1,1) PRIMARY KEY,  
    StudyID INT NOT NULL,  
    StudentID INT NOT NULL,  
    Date DATE DEFAULT GETDATE(),  
    FOREIGN KEY (StudyID) REFERENCES Studies (StudyID),  
    FOREIGN KEY (StudentID) REFERENCES Users (UserID),  
    CHECK (Date >= '2018-01-01' AND Date <= '9999-12-31'),  
    CONSTRAINT Unique_StudentID_Date UNIQUE (StudentID, Date)  
);
```

Tabela miała zostać wyodrębniona z tabeli users aby nie powtarzać kodu.

### 2.CITIES:

CityID: Id miasta (INT)

CityName: Nazwa (NVARCHAR(30))

CountryID: Id państwa w którym się znajduje miasto (INT)

- klucz główny: ID miasta
- klucz obcy do tabeli Countries (CountryID)
- sprawdzamy czy długość nazwy miasta wynosi maksymalnie 30 znaków
- unikalna para CityName, CountryID - nie może być dwóch tych samych miast w tym samym państwie

```
CREATE TABLE Cities (  
    CityID INT IDENTITY(1,1) PRIMARY KEY,  
    CityName NVARCHAR(30) NOT NULL,  
    CountryID INT NOT NULL,  
    FOREIGN KEY(CountryID) REFERENCES dbo.Countries(CountryID),  
    CHECK (LEN(CityName) <= 30),  
    CONSTRAINT Unique_CityName_CountryID UNIQUE (CityName, CountryID)  
);
```

Tabela konkretnych zajęć dla przedmiotów

### 3.CLASSES:

ClassID: Id zajęć (INT)

SessionID: Id zjazdu (INT)

SubjectID: Id przedmiotu (INT)

- klucz główny: ID klasy
- klucze obce do tabel Sessions (SessionID) i Subjects (SubjectID)
- unikalna para ClassID, SessionID - nie może być dwóch klas o tym samym ID w obrębie danego zjazdu

```
CREATE TABLE Classes (  
    ClassID INT IDENTITY(1,1) PRIMARY KEY,  
    SessionID INT NOT NULL,  
    SubjectID INT NOT NULL,  
    CONSTRAINT Unique_ClassID_SessionID UNIQUE (ClassID, SessionID),  
    FOREIGN KEY(SessionID) REFERENCES Sessions (SessionID),  
    FOREIGN KEY(SubjectID) REFERENCES Subjects (SubjectID)  
);
```

Tabela zajęć asynchronicznych

#### 4.CLASSESASYNC:

ClassID: Id zajęć (INT)

RecordingLink: link do nagrania (NVARCHAR(50))

- klucz główny: ID klasy
- klucz obcy do tabeli Classes (ClassID)
- długość linku wynosi maksymalnie 50 znaków
- link zaczyna się od 'http'
- RecordingLink jest unikalny, nie może być dwóch klas asynchronicznych o tym samym linku

```
CREATE TABLE ClassesAsync (  
    ClassID INT PRIMARY KEY,  
    RecordingLink NVARCHAR(50) NULL,  
    FOREIGN KEY(ClassID) REFERENCES Classes (ClassID),  
    CHECK (LEN(RecordingLink) <= 50),  
    CHECK (RecordingLink LIKE 'http%'),  
    CONSTRAINT Unique_RecordingLink UNIQUE (RecordingLink)  
);
```

Tabela obecności na zajęciach

#### 5.CLASSESATTENDANCES:

ClassID: Id zajęć (INT)

StudentID: Id studenta (INT)

Present: Czy obecny (tak/nie) (BIT)

- klucz główny złożony: ID klasy i ID studenta
- klucze obce do tabeli Classes (ClassID) i Users (UserID)

```
CREATE TABLE ClassesAttendances (  
    ClassID INT NOT NULL,  
    StudentID INT NOT NULL,  
    Present BIT NOT NULL,  
    CONSTRAINT PK_Attendances PRIMARY KEY (ClassID, StudentID),  
    FOREIGN KEY(ClassID) REFERENCES Classes (ClassID),  
    FOREIGN KEY(StudentID) REFERENCES Users (UserID));
```

Tabela zajęć stacjonarnych

## 6.CLASSESINPERSON:

ClassID: Id zajęć (INT)

LocationID: Id lokalizacji w której odbywają się zajęcia (INT)

StartTime: dzień i godzina rozpoczęcia zajęć (DATETIME)

EndTime: dzień i godzina zakończenia (może służyć np. do wyliczenia czasu trwania) (DATETIME)

- klucz główny: ID klasy
- klucze obce do tabel Classes (ClassID) i Locations (LocationID)
- sprawdzamy, czy data rozpoczęcia jest późniejsza niż przyjęta data powstania systemu
- sprawdzamy, czy data rozpoczęcia jest wcześniejsza niż data zakończenia
- sprawdzamy, czy nie ma zachodzenia na siebie zajęć w tej samej lokalizacji

```
CREATE TABLE ClassesInPerson (  
    ClassID INT NOT NULL PRIMARY KEY,  
    LocationID INT NOT NULL,  
    StartTime DATETIME NOT NULL,  
    EndTime DATETIME NOT NULL,  
    FOREIGN KEY (LocationID) REFERENCES Locations (LocationID),  
    FOREIGN KEY (ClassID) REFERENCES Classes (ClassID),  
    CHECK (StartTime < EndTime),  
    CHECK (StartTime >= '2018-01-01' AND StartTime <= '9999-12-31'),  
    CONSTRAINT Unique_Location_StartTime UNIQUE (LocationID, StartTime),  
    CONSTRAINT Check_NoTimeOverlap CHECK  
(dbo.CheckTimeOverlapClasses(LocationID, StartTime, EndTime, ClassID) = 0)  
);
```

Tabela zajęć online (synchronicznych - na żywo)

## 7.CLASSESSYNC:

ClassID: Id zajęć (INT)

StartTime: dzień i godzina rozpoczęcia (DATETIME)

EndTime: dzień i godzina zakończenia (DATETIME)

RecordingLink: link do nagrania (NVARCHAR(50))

MeetingLink: link do zewnętrznej platformy na której odbędzie się spotkanie (NVARCHAR(50))

- klucz główny: ID klasy
- klucze obce do tabel Classes (ClassID)

- sprawdzamy, czy data rozpoczęcia jest późniejsza niż przyjęta data powstania systemu
- sprawdzamy, czy data rozpoczęcia jest wcześniejsza niż data zakończenia
- długość obu linków wynosi maksymalnie 50 znaków
- linki zaczynają się od 'http'
- zarówno RecordingLink jak i MeetingLink jest unikalny, nie może być dwóch klas asynchronicznych o tym samym linku

```
CREATE TABLE ClassesSync (
    ClassID INT NOT NULL PRIMARY KEY,
    StartTime DATETIME NOT NULL,
    EndTime DATETIME NOT NULL,
    MeetingLink NVARCHAR(50) NOT NULL,
    RecordingLink NVARCHAR(50) NULL DEFAULT 'Unknown',
    FOREIGN KEY (ClassID) REFERENCES Classes (ClassID),
    CHECK (StartTime < EndTime),
    CHECK (LEN(MeetingLink) <= 50),
    CHECK (MeetingLink LIKE 'http%'),
    CHECK (LEN(RecordingLink) <= 50),
    CHECK (RecordingLink LIKE 'http%'),
    CHECK (StartTime < EndTime),
    CHECK (StartTime >= '2018-01-01' AND StartTime <= '9999-12-31'),
    CONSTRAINT Unique_MeetingLink UNIQUE (MeetingLink),
    CONSTRAINT Unique_RecordingLink UNIQUE (RecordingLink)
);
```

Tabela krajów pochodzenia użytkowników analogicznie odnosi się do tabeli cities.

### 8.COUNTRIES:

CountryID: Id państwa (INT)

CountryName: Nazwa (NVARCHAR(20))

- klucz główny: ID państwa
- długość nazwy państwa wynosi maksymalnie 50 znaków
- nazwa państwa jest unikalna

```
CREATE TABLE Countries (
    CountryID INT IDENTITY(1,1) PRIMARY KEY,
    CountryName NVARCHAR(20) NOT NULL,
    CONSTRAINT Unique_CountryName UNIQUE (CountryName),
    CHECK (LEN(CountryName) <= 20)
);
```

Tabela uczestników kursu - przechowujemy informacje o tym jaki uczestnik na jakie kursy się zapisał oraz czy opłacił zaliczkę.

#### 9.COURSEPARTICIPANTS

EnrollID: Id zapisu na kurs (INT)

ParticipantID: Id uczestnika kursu (INT)

CourseID: Id kursu (INT)

Date: Data zapisu na kurs (DATE)

- klucz główny: ID zapisu na kurs
- klucze obce do tabel Courses (CourseID) i Users (UserID)
- sprawdzamy, czy data zapisu na kurs jest późniejsza niż przyjęta data powstania systemu
- para ParticipantID, CourseID jest unikalna - nie może być dwóch uczestników o tym samym ID na jednym kursie

```
CREATE TABLE CourseParticipants (  
    EnrollID INT IDENTITY(1,1) PRIMARY KEY,  
    ParticipantID INT NOT NULL,  
    CourseID INT NOT NULL,  
    [Date] DATE NOT NULL,  
    CONSTRAINT Unique_ParticipantCourse UNIQUE (ParticipantID, CourseID),  
    CHECK (Date >= '2018-01-01' AND Date <= '9999-12-31'),  
    FOREIGN KEY (CourseID) REFERENCES Courses (CourseID),  
    FOREIGN KEY (ParticipantID) REFERENCES Users (UserID)  
);
```

Tabela przechowująca dane o płatnościach za kursy (tym razem już pełna kwota, a nie zaliczka).

#### 10.COURSEPAYMENTS:

PaymentID: Id płatności (INT)

CourseID: Id kursu (INT)

StudentID: Id studenta (INT)

Date: Data płatności (DATE)

Paid: Czy zapłacono (Tak/Nie) (BIT)

- klucz główny: ID płatności
- klucze obce do tabel Courses (CourseID) i Users (UserID)
- sprawdzamy, czy data płatności na kurs jest późniejsza niż przyjęta data powstania systemu

- para StudentID, CourseID jest unikalna - aby student o danym ID nie zapłacił dwa razy za ten sam kurs
- domyślnie przyjmujemy, że kurs nie został opłacony

```
CREATE TABLE CoursePayments (
    PaymentID INT IDENTITY(1,1) PRIMARY KEY,
    CourseID INT NOT NULL,
    StudentID INT NOT NULL,
    DueDate DATE NULL,
    Paid BIT NULL DEFAULT 0,
    CONSTRAINT Unique_CourseStudent UNIQUE (CourseID, StudentID),
    CHECK (DueDate >= '2018-01-01' AND Date <= '9999-12-31'),
    CONSTRAINT FK_CoursePayments_Courses FOREIGN KEY(CourseID) REFERENCES
Courses(CourseID),
    CONSTRAINT FK_CoursePayments_Users FOREIGN KEY(StudentID) REFERENCES
Users(UserID)
);
```

Tabela kursów

#### 11.COURSES:

CourseID: Id kursu (INT)

CourseName: Nazwa kursu (NVARCHAR(50))

ParticipantsLimit: Limit uczestników kursu (INT)

Advance: Zaliczka (MONEY)

Price: Cena kursu (MONEY)

StartDate: Data rozpoczęcia kursu (DATE)

EndDate: Data zakończenia kursu (DATE)

- klucz główny: ID kursu
- sprawdzamy, czy data rozpoczęcia kursu jest późniejsza niż przyjęta data powstania systemu
- sprawdzamy, czy data rozpoczęcia jest wcześniejsza lub równa (kursy jednodniowe) dacie zakończenia
- sprawdzamy, czy wprowadzona kwota zaliczki oraz ceny jest większa lub równa zero
- sprawdzamy, czy limit uczestników jest większy od zera
- sprawdzamy, czy nazwa kursu jest unikalna
- sprawdzamy, czy długość nazwy kursu wynosi maksymalnie 50 znaków

```
CREATE TABLE Courses (
    CourseID INT IDENTITY(1,1) PRIMARY KEY,
```



```

    CourseName NVARCHAR(50) NOT NULL,
    ParticipantsLimit INT NOT NULL,
    Advance MONEY NOT NULL,
    Price MONEY NOT NULL,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    CONSTRAINT Unique_CourseName UNIQUE (CourseName),
    CHECK (ParticipantsLimit > 0),
    CHECK (Advance >= 0),
    CHECK (Price >= 0),
    CHECK (LEN(CourseName) <= 50),
    CHECK (StartDate >= '2018-01-01' AND Date <= '9999-12-31'),
    CONSTRAINT Check_DateRange CHECK (StartDate <= EndDate)
);

```

Tabela wysłanych dyplomów

## 12. DIPLOMAS:

DiplomID INT: Id dyplomu (INT)

StudyID INT: Id studiów (INT)

StudentID INT: Id studenta (INT)

SendingDate DATE: data, kiedy dyplom został wysłany

- klucz główny: ID dyplomu
- klucze obce do tabel ProductType(TypeID) i Users (UserID)
- sprawdzamy, czy data wysłania dyplomu jest późniejsza niż przyjęta data powstania systemu
- trójka ProductID, TypeID, StudentID jest unikalna - student może mieć tylko 1 dyplom za produkt o danym ID danego typu
- sprawdzamy, czy wprowadzone ID produktu dla danego typu jest prawidłowe

```

CREATE TABLE Diplomas (
    DiplomID INT IDENTITY(1,1) PRIMARY KEY,
    ProductID INT,
    TypeID INT,
    StudentID INT,
    SendingDate DATE,
    FOREIGN KEY(StudentID) REFERENCES Users(UserID),
    FOREIGN KEY(TypeID) REFERENCES ProductType(TypeID),
    CONSTRAINT Unique_Study_Student UNIQUE (ProductID, TypeID,
StudentID),
    CHECK (SendingDate >= '2018-01-01' AND SendingDate <= '9999-12-31'),

```

```

CONSTRAINT          CHK_ProductID_Courses_Studies
(dbo.ValidateProductIDForDiplomas(TypeID, ProductID) = 1)
);

```

Tabela dziedzin nauki w których specjalizują się tłumacze oraz prowadzone są webinary.

### 13.FIELDS OF STUDY:

FieldID: Id dziedziny nauki (INT)

FieldName: Nazwa dziedziny (NVARCHAR(30))

- klucz główny: ID dziedziny nauki
- sprawdzamy, czy nazwa dziedziny jest unikalna
- sprawdzamy, czy długość nazwy dziedziny wynosi maksymalnie 30 znaków

```

CREATE TABLE FieldsOfStudy (
    FieldID INT IDENTITY(1,1) PRIMARY KEY,
    FieldName NVARCHAR(30) NOT NULL,
    CONSTRAINT Unique_FieldName UNIQUE (FieldName),
    CHECK (LEN(FieldName) <= 30)
);

```

Tabela języków, na które tłumacz potrafi tłumaczyć dane wydarzenia na język polski.

### 14.KNOWNLANGUAGES:

TranslatorID: Id tłumacza (INT)

LanguageID: Id języka, z którego potrafi tłumaczyć (INT)

- klucz główny złożony: ID tłumacza i ID języka
- klucze obce do tabelLanguages(LanguageID) i Translators(TranslatorID)

```

CREATE TABLE KnownLanguages (
    TranslatorID INT NOT NULL,
    LanguageID INT NOT NULL,
    CONSTRAINT PK_KnownLanguages PRIMARY KEY (TranslatorID, LanguageID),
    FOREIGN KEY(LanguageID) REFERENCES Languages(LanguageID),
    FOREIGN KEY(TranslatorID) REFERENCES Translators(TranslatorID)
);

```

Wyodrębniona tabela języków umożliwia w przejrzysty sposób przypisanie języka w którym prowadzi wydarzenia wykładowca oraz który potrafi tłumacz.

### 15.LANGUAGES:

LanguageID: Id języka (INT)

LanguageName: Język (np. polski) (NVARCHAR(20))

Klucz główny: LanguageID

Warunki:

Unique: LanguageName

Check: LEN(LanguageName) <= 20

```
CREATE TABLE Languages (  
    LanguageID INT IDENTITY(1,1) PRIMARY KEY,  
    LanguageName NVARCHAR(20) NOT NULL,  
    CONSTRAINT Unique_LanguageName UNIQUE (LanguageName),  
    CHECK(LEN(LanguageName) <= 20)  
);
```

Tabela wykładowców zawiera dane specyficzne dla danego wykładowcy.

### 16.LECTURERS:

LecturerID: Id wykładowcy (INT)

TitleID: ID tytułu naukowego (INT)

HireDate: Data zatrudnienia (DATE)

LanguageID: Id języka w którym prowadzi wydarzenia (INT)

Klucz główny: LecturerID

Klucze obce: LanguagesID, TitlesID, LecturerID

Warunki:

Check: HireDate >= '2018-01-01' AND HireDate <= '9999-12-31'

```
CREATE TABLE Lecturers(  
    LecturerID INT PRIMARY KEY,  
    TitleID INT NOT NULL,  
    HireDate DATE NOT NULL,  
    LanguageID INT NOT NULL,  
    FOREIGN KEY(LanguageID) REFERENCES Languages(LanguageID),  
    FOREIGN KEY(TitleID) REFERENCES Titles(TitleID),  
    FOREIGN KEY(LecturerID) REFERENCES Users (UserID),  
    CHECK(HireDate >= '2018-01-01' AND HireDate <= '9999-12-31')  
);
```

Tabela przechowująca lokalizacje prowadzenia zajęć/modułów

### 17.LOCATIONS:

LocationID: Id miejsca (INT)

CityID: Id miasta (INT)

Street: ulica (NVARCHAR(30))

BuildingNumber: numer budynku (INT)

ParticipantsLimit: Limit miejsc w danym pokoju/klasie/auli (INT)

RoomNumber: numer pokoju/klasy/auli (INT)

Description: informacje dodatkowe np. wskazówki w jaki sposób trafić (NVARCHAR(200))

Klucz główny: LocationID

Klucz obcy: CityID

Warunki:

Unique: Street, BuildingNumber, RoomNumber

Check: Street, Description

```
CREATE TABLE Locations(  
    LocationID INT IDENTITY(1,1) PRIMARY KEY,  
    CityID INT NOT NULL,  
    Street NVARCHAR(30) NOT NULL,  
    BuildingNumber INT NOT NULL,  
    ParticipantsLimit INT NOT NULL,  
    RoomNumber INT NOT NULL,  
    Description NVARCHAR(200) NULL,  
    FOREIGN KEY(CityID) REFERENCES Cities(CityID),  
    CONSTRAINT Unique_StreetBuildingRoom UNIQUE (Street, BuildingNumber,  
RoomNumber),  
    CHECK(LEN(Street)<=30),  
    CHECK(LEN(Description)<=200)  
);
```

Dane logowania użytkowników.

### 18. LOGINS:

UserID: Id użytkownika (INT)

UserName: login (NVARCHAR(20))

Password: hasło ustalone przez użytkownika (NVARCHAR(20))

Klucz główny: UserID

Klucz obcy: UserID

Warunki:

Check: LEN(UserName) <= 20, LEN>Password) <= 20

Unique: UserName

```
CREATE TABLE Logins(  
    UserID INT NOT NULL PRIMARY KEY,  
    UserName NVARCHAR(20) NOT NULL,  
    Password NVARCHAR(20) NOT NULL,  
    CHECK (LEN(UserName) <= 20),  
    CHECK (LEN>Password) <= 20),
```

```
CONSTRAINT Unique_UserName UNIQUE (UserName),  
FOREIGN KEY(UserID) REFERENCES Users (UserID)  
);
```

#### Tabela obecności poszczególnych kursantów na modułach

##### 19.MODULEATTENDANCES

ModuleID: Id modułu (INT)

StudentID: Id studenta (INT)

Present: Czy obecny (tak/nie) (BIT)

Klucz główny: ModuleID, StudentID

Klucze obce: ModuleID, StudentID

```
CREATE TABLE ModuleAttendances(  
    ModuleID INT NOT NULL,  
    StudentID INT NOT NULL,  
    Present BIT NOT NULL,  
    CONSTRAINT PK_ModuleAttendances PRIMARY KEY (ModuleID, StudentID),  
    FOREIGN KEY(ModuleID) REFERENCES Modules (ModuleID),  
    FOREIGN KEY(StudentID) REFERENCES Users (UserID)  
);
```

#### Tabela modułów prowadzonych w ramach kursów.

##### 20.MODULES:

ModuleID: Id modułu (INT)

CourseID: Id kursu (INT)

LecturerID: Id wykładowcy (INT)

Klucz główny: ModuleID

Klucze obce: CourseID, LecturerID

```
CREATE TABLE Modules(  
    ModuleID INT IDENTITY(1,1) PRIMARY KEY,  
    CourseID INT NOT NULL,  
    LecturerID INT NOT NULL,  
    FOREIGN KEY(CourseID) REFERENCES Courses (CourseID),  
    FOREIGN KEY(LecturerID) REFERENCES Lecturers (LecturerID)  
);
```

#### Tabela modułów asynchronicznych

##### 21.MODULESASYNC:

ModuleID: Id modułu (INT)

RecordingLink: Link do nagrania (NVARCHAR(50))

Klucz główny: ModuleID

Klucz obcy: Modules

Warunki:

Unique:RecordingLink

Check:LEN(RecordingLink) <= 50, RecordingLink LIKE 'http%'

```
CREATE TABLE ModulesAsync(  
    ModuleID INT NOT NULL PRIMARY KEY,  
    RecordingLink NVARCHAR(50) NULL DEFAULT 'Unknown',  
    FOREIGN KEY(ModuleID) REFERENCES Modules (ModuleID),  
    CONSTRAINT Unique_RecordingLink UNIQUE (RecordingLink),  
    CHECK (LEN(RecordingLink) <= 50),  
    CHECK (RecordingLink LIKE 'http%')  
);
```

Tabela modułów stacjonarnych

#### 22.MODULESINPERSON:

ModuleID: Id modułu (INT)

LocationID: Id lokalizacji w której odbywa się moduł (INT)

StartTime: Dzień i godzina rozpoczęcia modułu (DATETIME)

EndTime: Dzień i godzina zakończenia modułu (DATETIME)

Klucz główny: ModuleID

Klucze obce: LocationID, ModuleID

Warunki:

Unique:LocationID, StartTime

Check:StartTime <= EndTime, CheckTimeOverlapModules

```
CREATE TABLE ModulesInPerson(  
    ModuleID INT NOT NULL PRIMARY KEY,  
    LocationID INT NOT NULL,  
    StartTime DATETIME NOT NULL,  
    EndTime DATETIME NOT NULL,  
    FOREIGN KEY(LocationID) REFERENCES Locations (LocationID),  
    FOREIGN KEY(ModuleID) REFERENCES Modules (ModuleID),  
    CONSTRAINT Unique_Location_StartTime UNIQUE (LocationID, StartTime),  
    CONSTRAINT Check_NoTimeOverlapModules CHECK  
(dbo.CheckTimeOverlapModules(LocationID, StartTime, EndTime, ModuleID) =  
0),  
    CHECK (StartTime <= EndTime)  
);
```

Tabela modułów online (na żywo)

### 23.MODULESSYNC:

ModuleID: Id modułu (INT)

StartTime: Dzień i godzina rozpoczęcia (DATETIME)

EndTime: Dzień i godzina zakończenia (DATETIME)

RecordingLink: Link do nagrania (NVARCHAR(50))

MeetingLink: Link do zewnętrznej platformy na której odbędzie się spotkanie (NVARCHAR(50))

Klucz główny: ModuleID

Klucz obcy: ModuleID

Warunki:

Unique: MeetingLink

Check: LEN(MeetingLink) <= 50, MeetingLink LIKE 'http%', RecordingLink LIKE 'http%', StartTime <= EndTime

```
CREATE TABLE ModulesSync(  
    ModuleID INT NOT NULL PRIMARY KEY,  
    StartTime DATETIME NOT NULL,  
    EndTime DATETIME NOT NULL,  
    MeetingLink NVARCHAR(50) NOT NULL,  
    RecordingLink NVARCHAR(50) NULL,  
    CHECK (LEN(MeetingLink) <= 50),  
    CHECK (MeetingLink LIKE 'http%'),  
    CHECK (LEN(RecordingLink) <= 50),  
    CHECK (RecordingLink LIKE 'http%'),  
    CONSTRAINT Unique_MeetingLink UNIQUE (MeetingLink),  
    FOREIGN KEY(ModuleID) REFERENCES Modules (ModuleID),  
    CHECK (StartTime <= EndTime)  
);
```

Tabela zawierająca szczegółowe informacje nt. zamówień (co zostało zamówione w poszczególnych zamówieniach)

### 24.ORDERDETAILS:

OrderID: Id zamówienia (INT)

ProductID: Id produktu (INT)

ProductTypeID: Rodzaj produktu (webinar/kurs/studia/pojedyncze spotkanie w ramach studiów) (INT)

Klucz główny: OrderID, ProductID, ProductTypeID

Klucz obcy: OrderID, ProductTypeID,

Warunki:

Check: ProductTypeID IN (1, 2, 3, 4),

dbo.ValidateProductIDForOrders(ProductTypeID, ProductID) = 1

```

CREATE TABLE OrderDetails(
    OrderID INT NOT NULL,
    ProductID INT NOT NULL,
    ProductTypeID INT NOT NULL,
    CONSTRAINT PK_OrderDetails PRIMARY KEY (OrderID, ProductID,
ProductTypeID),
    FOREIGN KEY(OrderID) REFERENCES Orders (OrderID),
    FOREIGN KEY(ProductTypeID) REFERENCES ProductType (TypeID),
    CONSTRAINT FK_ProductType CHECK (ProductTypeID IN (1, 2, 3, 4)),
    CONSTRAINT CHK_ProductID_Courses_Studies_Webinars_Sessions CHECK
(dbo.ValidateProductIDForOrders(ProductTypeID, ProductID) = 1)
);

```

Tabela zamówień (z koszyka)

#### 25.ORDERS:

OrderID: Id zamówienia (INT)

UserID: Id osoby składającej zamówienie (INT)

OrderDate: Data zamówienia (DATETIME)

Klucz główny: OrderID

Klucz obcy: UserID

Warunki:

Check: OrderDate >= '2018-01-01' AND OrderDate <= '9999-12-31'

```

CREATE TABLE Orders (
    OrderID INT IDENTITY(1,1) PRIMARY KEY,
    UserID INT NOT NULL,
    OrderDate DATETIME NOT NULL,
    FOREIGN KEY(UserID) REFERENCES Logins(UserID),
    CHECK (OrderDate >= '2018-01-01' AND OrderDate <= '9999-12-31')
);

```

Tabela wyjątków o których decyduje Dyrektor Szkoły np. zgoda na płatność odroczoną dla stałych klientów.

#### 26.PAYMENTEXCEPTIONS:

ExceptionID: Id sytuacji wyjątkowej dla płatności (INT)

FormID: Id formy (studium/kurs/webinar) (INT)

PaymentID: Id płatności (INT)

DueDate: Termin zapłaty (DATE)

Klucz główny: ExceptionID

Klucz obcy: FormID

Warunki:



Check: DueDate >= '2018-01-01' AND DueDate <= '9999-12-31',  
CHK\_ProductID\_Courses\_Studies\_Webinars

```
CREATE TABLE PaymentExceptions (  
    ExceptionID INT IDENTITY(1,1) PRIMARY KEY,  
    FormID INT NOT NULL,  
    PaymentID INT NOT NULL,  
    DueDate DATE NOT NULL,  
    FOREIGN KEY(FormID) REFERENCES ProductType(TypeID),  
    CHECK (DueDate >= '2018-01-01' AND DueDate <= '9999-12-31'),  
    CONSTRAINT FK_FormType CHECK (FormID IN (1, 2, 3)),  
    CONSTRAINT CHK_ProductID_Courses_Studies_Webinars CHECK (  
        dbo.ValidateProductIDforPayment(FormID, PaymentID) = 1));  
);
```

Tabela przechowująca rodzaje możliwych do kupienia produktów

#### 27.PRODUCTTYPE:

TypeID: Id rodzaju (INT)

ProductName: Nazwa (np. webinar/kurs/studia/pojedyncze spotkanie w ramach studiów) (NVARCHAR(20))

Klucz główny: TypeID

Warunki:

Check:LEN(ProductName) <= 20

Unique: ProductName

```
CREATE TABLE ProductType (  
    TypeID INT IDENTITY(1,1) PRIMARY KEY,  
    ProductName NVARCHAR(20) NOT NULL,  
    CHECK (LEN(ProductName) <= 20),  
    CONSTRAINT Unique_ProductName UNIQUE (ProductName)  
);
```

Tabela ról jakie mogą pełnić użytkownicy

#### 28.ROLES:

RoleID: Id roli (INT)

RoleName: Nazwa roli (NVARCHAR(30))

Klucz główny: RoleID

Warunki:

Check:LEN(RoleName) <= 30

Unique: RoleName

```

CREATE TABLE Roles (
    RoleID INT IDENTITY(1,1) PRIMARY KEY,
    RoleName NVARCHAR(30) NOT NULL,
    CHECK (LEN(RoleName) <= 30),
    CONSTRAINT Unique_RoleName UNIQUE (RoleName)
);

```

Tabela zawierająca informacje na temat pełnionych ról przez użytkowników - zarówno pełnionych aktualnie jak i pełnionych w przeszłości

### 29.ROLESHISTORY:

UserID: Id użytkownika (INT)

RoleID: Id roli (INT)

StartDate: Data rozpoczęcia pełnienia roli (DATETIME)

EndDate: Data zakończenia pełnienia roli (DATETIME)

Klucz główny = (UserID, RoleID, StartDate)

Klucze obce: RoleID = Roles(RoleID), UserID = Users(UserID)

Warunki:

Sprawdzanie, czy data rozpoczęcia jest wcześniejsza od daty zakończenia a także sprawdzanie, czy daty są nie wcześniejsze niż rok 2018)

```

CREATE TABLE RolesHistory (
    UserID INT NOT NULL,
    RoleID INT NOT NULL,
    StartDate DATE NOT NULL,
    EndDate Date NULL,
    COINSTRANIT PK_RolesHistory PRIMARY KEY (UserID, RoleID, StartDate),
    CONSTRAINT Check_DateRange CHECK (StartDate <= EndDate),
    FOREIGN KEY(RoleID) REFERENCES Roles(RoleID),
    FOREIGN KEY(UserID) REFERENCES Users(UserID),
    CHECK (StartDate >= '2018-01-01' AND Date <= '9999-12-31'),
    CHECK (EndDate >= '2018-01-01' AND Date <= '9999-12-31'),
);

```

Tabela przechowująca zjazdy na studia

### 30.SSESSIONS:

SessionID: ID zjazdu (INT)

StudyID: Id studiów w obrebie ktorých jest zjazd (INT)

ParticipantPrice: Cena za zjazd dla uczestnika studiów (MONEY)

GuestPrice: Cena za zjazd dla osoby spoza studiów (MONEY)

Klucz główny: SessionID

Klucz obcy: StudyID = Studies(StudyID)

Warunki:

Wszystkie ceny muszą być większe od zera, warunki dla dat jak w poprzednich tabelach.

```
CREATE TABLE Sessions (  
    SessionID INT IDENTITY(1,1) PRIMARY KEY,  
    StudyID INT NOT NULL,  
    ParticipantPrice MONEY NOT NULL,  
    GuestPrice MONEY NOT NULL,  
    StartDate DATE NOT NULL,  
    EndDate DATE NOT NULL,  
    FOREIGN KEY(StudyID) REFERENCES Studies(StudyID),  
    CHECK (ParticipantPrice > 0),  
    CHECK (GuestPrice > 0)  
    CHECK (StartDate <= EndDate),  
    CHECK (StartDate >= '2018-01-01' AND Date <= '9999-12-31'),  
    CHECK (EndDate >= '2018-01-01' AND Date <= '9999-12-31')  
);
```

Tabela przechowująca adresy studentów (w celu wysłania im dyplomów)

### 31.STUDENTSADDRESSES:

StudentID: Id studenta (INT)

CityID: Id miasta (INT)

Street: Nazwa ulicy (NVARCHAR(50))

BuildingNumber: Numer budynku (INT)

FlatNumber: Numer mieszkania (opcjonalnie) (INT)

PostalCode: Kod pocztowy (NVARCHAR(10))

Klucz główny: StudentID

Klucze obce: CityID = Cities(CityID), StudentID = Users(UserID)

Warunki:

Kombinacja wartości (Street, BuildingNumber, FlatNumber) musi być unikatowa, sprawdzanie czy dane typu NVARCHAR mają odpowiednią długość.

```
CREATE TABLE StudentsAddresses (  
    StudentID INT NOT NULL PRIMARY KEY,  
    CityID INT NOT NULL,  
    Street NVARCHAR(50) NOT NULL,
```

```

        BuildingNumber INT NOT NULL,
        FlatNumber INT NULL,
        PostalCode NVARCHAR(10) NOT NULL,
        FOREIGN KEY(CityID) REFERENCES Cities (CityID),
        FOREIGN KEY(StudentID) REFERENCES Users(UserID),
        CONSTRAINT Unique_StreetBuildingFlat UNIQUE (Street, BuildingNumber,
FlatNumber),
        CHECK (LEN(Street) <= 50),
        CHECK (LEN(PostalCode) <= 10)
);

```

Tabela kierunków studiów wraz z szczegółowymi informacjami o danym kierunku.

### 32.STUDIES:

StudyID: Id kierunku studiów (INT)  
 StudyName: Nazwa kierunku (NVARCHAR(30))  
 CoordinatorID: Koordynator studiów (INT)  
 StudentsLimit: Limit studentów (INT)  
 EntryFee: Wartość wpisowego (MONEY)  
 StartDate: Data rozpoczęcia studiów (DATE)  
 EndDate: Data zakończenia studiów (DATE)

Klucz główny: StudyID

Klucz obcy: CoordinatorID = Lecturers(LecturerID)

Warunki:

Limit studentów oraz wpisowe muszą być dodatnie. Warunki dla dat jak w poprzednich tabelach, sprawdzanie długości nazwy studiów oraz kombinacja (StudyName,StartDate) musi być unikatowa - zakładamy, że nie możemy mieć dwóch studiów z tymi samymi nazwami zaczynającymi się równocześnie

```

CREATE TABLE Studies (
    StudyID INT IDENTITY(1,1) PRIMARY KEY,
    StudyName NVARCHAR(30) NOT NULL,
    CoordinatorID INT NOT NULL,
    StudentsLimit INT NOT NULL,
    EntryFee MONEY NOT NULL,
    StartDate DATE NOT NULL,
    EndDate DATE NOT NULL,
    CONSTRAINT Check_DateRange CHECK (StartDate <= EndDate),
    FOREIGN KEY(CoordinatorID) REFERENCES Lecturers(LecturerID),
    CHECK (StudentsLimit > 0),
    CHECK (EntryFee > 0),

```

```

CHECK (LEN(StudyName) <= 30),
CHECK (StartDate >= '2018-01-01' AND Date <= '9999-12-31'),
CHECK (EndDate >= '2018-01-01' AND Date <= '9999-12-31'),
CONSTRAINT Unique_StudyName_StartDate UNIQUE (StudyName, StartDate)
);

```

Tabela płatności za studia.

### 33.SESSIONPAYMENTS:

PaymentID: Id płatności (INT)  
 SessionID: Id zajęć (INT)  
 DueDate: Do kiedy płatność (DATE)  
 StudentID: Id studenta (INT)  
 Paid: czy zapłacono (0/1) (BIT)

Klucz główny: PaymentID

Klucze obce: SessionID = Sessions(SessionID), StudentID = Users(UserID)

Warunki:

Sprawdzanie poprawności dat oraz zapewnienie, aby dany student był zapisany na dany zjazd co najwyżej raz.

```

CREATE TABLE SessionPayments (
    PaymentID INT IDENTITY(1,1) PRIMARY KEY,
    SessionID INT NOT NULL,
    StudentID INT NOT NULL,
    DueDate DATE NOT NULL,
    Paid BIT NULL,
    CHECK (DueDate >= '2018-01-01' AND DueDate <= '9999-12-31'),
    FOREIGN KEY(SessionID) REFERENCES Sessions(SessionID),
    FOREIGN KEY(StudentID) REFERENCES Users(UserID),
    CONSTRAINT Unique_Session_Student UNIQUE (SessionID, StudentID)
);

```

Tabela przedmiotów, które są wykładane na danych zjazdach oraz tworzą plany zajęć studentów.

### 34.SUBJECTS:

SubjectID: Id przedmiotu (INT)  
 SubjectName: Pełna nazwa przedmiotu (NVARCHAR(50))  
 CoordinatorID: Koordynator przedmiotu (INT)  
 FieldOfStudyID: Id dziedziny nauki (INT)  
 Description: Opis przedmiotu (NVARCHAR(300))

Klucz główny: SubjectID

Klucze obce: FieldsOfStudyID = FieldsOfStudyID(FieldID), CoordinatorID = Lecturers(LecturerID)

Warunki:

Sprawdzaine długości danych typów NVARCHAR oraz zapewnienie, aby kombinacja (SubjectID, CoordinatorID, FieldOfStudy) była unikatowa.

```
CREATE TABLE Subjects (  
    SubjectID INT IDENTITY(1,1) PRIMARY KEY,  
    SubjectName NVARCHAR(50) NOT NULL,  
    CoordinatorID INT NOT NULL,  
    FieldOfStudyID INT NOT NULL,  
    Description NVARCHAR(300) NOT NULL,  
    CHECK (LEN(SubjectName) <= 50),  
    CHECK (LEN(Description) <= 300),  
    FOREIGN KEY(FieldOfStudyID) REFERENCES FieldsOfStudy(FieldID),  
    FOREIGN KEY(CoordinatorID) REFERENCES Lecturers(LecturerID),  
    CONSTRAINT Unique_SubjectName_CoordinatorID_FieldOfStudyID UNIQUE  
    (SubjectID, CoordinatorID, FieldOfStudyID)  
);
```

Tabela przechowująca plany zajęć przypisane do danego kierunku.

### 35.SYLLABUSES:

SyllabusID: Id sylabusa (INT)

StudyID: Id kierunku (INT)

SubjectID: Id przedmiotu (INT)

LecturerID: Id wykładowcy (INT)

Klucz główny: SyllabusID

Klucze obce: StudyID = Studies(StudyID), LecturerID = Lecturers(LecturerID),  
SubjectID = Subjects(SubjectID)

```
CREATE TABLE Syllabuses (  
    SyllabusID INT IDENTITY(1,1) PRIMARY KEY,  
    StudyID INT NOT NULL,  
    SubjectID INT NOT NULL,  
    LecturerID INT NOT NULL,  
    FOREIGN KEY(StudyID) REFERENCES Studies(StudyID),  
    FOREIGN KEY(LecturerID) REFERENCES Lecturers(LecturerID),  
    FOREIGN KEY(SubjectID) REFERENCES Subjects(SubjectID)  
);
```

Tabela przechowująca tytuły naukowe, aby nie powtarzać kodu w tabeli lecturers.

### 36.TITLES:

TitleID: Id tytułu naukowego (INT)

TitleName: pełny tytuł (NVARCHAR(30))

Klucz główny: TitleID

Warunki: Sprawdzanie długości TitleName a także zapewnienie, aby podany tytuł o konkretnej nazwie występował co najwyżej raz.

```
CREATE TABLE Titles (  
    TitleID INT IDENTITY(1,1) PRIMARY KEY,  
    TitleName NVARCHAR(20) NOT NULL,  
    CHECK (LEN(TitleName) <= 30),  
    CONSTRAINT Unique_TitleName UNIQUE (TitleName)  
);
```

Tabela modułów, które prowadzone są w języku obcym oraz są tłumaczone

### 37.TRANSLATEDMODULES:

ModuleID: Id modułu (INT)

TranslatorID: Id tłumacza (INT)

```
CREATE TABLE TranslatedModules (  
    ModuleID INT NOT NULL PRIMARY KEY,  
    TranslatorID INT NOT NULL,  
    FOREIGN KEY(ModuleID) REFERENCES Modules(ModuleID),  
    FOREIGN KEY(TranslatorID) REFERENCES Translators(TranslatorID)  
);
```

Tabela webinarów, które prowadzone są w języku obcym oraz są tłumaczone

### 38.TRANSLATEDWEBINARS:

WebinarID: Id webinaru (INT)

TranslatorID: Id tłumacza (INT)

Klucz główny: WebinarID

Klucze obce: WebinarID = Webinars(WebinarID), TranslatorID = Translators(TranslatorID)

```
CREATE TABLE TranslatedWebinars (  
    WebinarID INT NOT NULL PRIMARY KEY,  
    TranslatorID INT NOT NULL,  
    FOREIGN KEY(WebinarID) REFERENCES Webinars(WebinarID),  
    FOREIGN KEY(TranslatorID) REFERENCES Translators(TranslatorID)
```

);

Tabela id tłumaczy wraz z informacją z jakiej dziedziny nauki dany tłumacz posiada specjalistyczne słownictwo umożliwiające poprawne tłumaczenie.

#### 39.TRANSLATORS:

TranslatorID: Id tłumacza (INT)

FieldOfStudyID: Id dziedziny nauki (INT)

Klucz główny: TranslatorID

Klucze obce: TranslatorID = Users(UserID), FieldOfStudyID = FieldsOfStudy(FieldID)

```
CREATE TABLE Translators (  
    TranslatorID INT IDENTITY(1,1) PRIMARY KEY,  
    FieldOfStudyID INT NOT NULL,  
    FOREIGN KEY(TranslatorID) REFERENCES Users(UserID),  
    FOREIGN KEY(FieldOfStudyID) REFERENCES FieldsOfStudy(FieldID)  
);
```

Tabela wszystkich użytkowników, zawiera wspólne dane w celu uniknięcia redundancji kodu.

#### 40.USERS:

UserID: Id użytkownika (INT)

FirstName: Imię (NVARCHAR(20))

LastName: Nazwisko (NVARCHAR(20))

Nationality: Narodowość (NVARCHAR(10))

Phone: Nr telefonu (NVARCHAR(12))

Email: Adres e-mail (NVARCHAR(30))

Klucz główny: UserID

Klucze obce: –

Warunki:

Sprawdzanie długości danych typu NVARCHAR, zapewnienie aby nr telefonu oraz użytkownika był unikatowy.

```
CREATE TABLE Users (  
    UserID INT IDENTITY(1,1) PRIMARY KEY,  
    FirstName NVARCHAR(20) NOT NULL,  
    LastName NVARCHAR(20) NOT NULL,  
    Nationality NVARCHAR(10) NOT NULL,
```



```

    Phone NVARCHAR(12) NOT NULL,
    Email NVARCHAR(30) NOT NULL,
    CHECK (LEN(FirstName) <= 20),
    CHECK (LEN(LastName) <= 20),
    CHECK (LEN(Nationality) <= 10),
    CHECK (LEN(Phone) <= 12),
    CHECK (LEN(Email) <= 30),
    CONSTRAINT Unique_Email UNIQUE (Email),
    CONSTRAINT Unique_Phone UNIQUE (Phone),
    CONSTRAINT Unique_FirstName_LastName_Phone UNIQUE (FirstName,
    LastName, Phone)
);

```

Tabela przechowująca informacje o płatnościach za płatne webinary.

#### 41.WEBINARPAYMENTS:

PaymentID: Id płatności (INT)  
 WebinarID: Id webinaru (INT)  
 ParticipantID: Id uczestnika (INT)  
 DueDate: Do kiedy płatność (DATE)  
 Paid: czy zapłacono (1/0) (BIT)

Klucz główny: PaymentID

Klucze obce: WebinarID = Webinars(WebinarID), ParticipantID = Users(UserID)

Warunki:

Zapewnienie unikalności krotki (WebinarID, ParticipantID) aby uczestnik nie mógł zapłacić za dany webinar więcej niż raz.

```

CREATE TABLE WebinarPayments (
    PaymentID INT IDENTITY(1,1) PRIMARY KEY,
    WebinarID INT NOT NULL,
    ParticipantID INT NOT NULL,
    DueDate DATE NOT NULL,
    Paid BIT NULL,
    CHECK (DueDate >= '2018-01-01' AND DueDate <= '9999-12-31'),
    FOREIGN KEY(WebinarID) REFERENCES Webinars(WebinarID),
    FOREIGN KEY(ParticipantID) REFERENCES Users(UserID),
    CONSTRAINT Unique_Webinar_Participant UNIQUE (WebinarID,
    ParticipantID)
);

```

Tabela webinarów.

#### 42.WEBINARS:

WebinarId: Id webinaru (INT)  
WebinarName: Pełna nazwa webinaru (NVARCHAR(60))  
RecordingLink: Link nagrania (NVARCHAR(50))  
Date: Data przeprowadzenia webinaru (DATE)  
TranslatorID: Id tłumacza (NULL, w przypadku braku tłumaczenia) (INT)  
Price: Całkowita cena (MONEY)  
FieldOfStudyID - id dziedziny nauki (potrzebne w celu dobrania tłumacza) (ID)

Klucz główny: WebinarID

Klucz obcy: –

Warunki:

Sprawdzanie długości wprowadzanych danych typu NVARCHAR, zapewnienie aby długość webinarów była dodatnia, sprawdzanie czy link ma poprawną formę (zaczyna się od "http") oraz standardowe kontrolowanie wstawianych dat.

```
CREATE TABLE Webinars (  
    WebinarID INT IDENTITY(1,1) PRIMARY KEY,  
    WebinarName NVARCHAR(30) NOT NULL,  
    RecordingLink NVARCHAR(50) NOT NULL,  
    [Date] DATE NOT NULL,  
    Price MONEY NULL DEFAULT 0,  
    DurationInMinutes INT NOT NULL,  
    CHECK (LEN(WebinarName) <= 30),  
    CHECK (LEN(RecordingLink) <= 50),  
    CHECK (RecordingLink LIKE 'http%'),  
    CHECK (DurationInMinutes > 0),  
    CHECK (Date >= '2018-01-01' AND Date <= '9999-12-31'),  
    CONSTRAINT Unique_RecordingLink UNIQUE (RecordingLink)  
);
```

Tabela przechowująca informacje dotyczące produktów znajdujących się w koszykach użytkowników.

#### 43.CART:

UserID - Id użytkownika, w którego koszyku znajduje się dany produkt (INT)  
ProductID - ID produktu (INT)  
ProductTypeID - ID typu produktu (Webinar/Kurs/Studia/Zjazd) (INT)

Klucz główny: UserID, ProductID, ProductTypeID

Klucz obcy: ProductTypeID = ProductType(TypeID)

Warunki:

Sprawdzanie, czy ProductTypeID mieści się w zakresie (1-4) oraz czy ProductID o podanym ProductTypeID istnieje w bazie.

```

CREATE TABLE Cart (
    UserID INT NOT NULL,
    ProductID INT NOT NULL,
    ProductTypeID INT NOT NULL,
    CONSTRAINT PK_UserID_ProductID_ProductTypeID PRIMARY KEY (UserID,
ProductID, ProductTypeID),
    FOREIGN KEY(ProductTypeID) REFERENCES ProductType(TypeID),
    CONSTRAINT FK_ProductType2 CHECK (ProductTypeID IN (1,2,3,4)),
    CONSTRAINT CHK_ProductID_Couress_Studies_Webinars_Sessions2 CHECK
(dbo.ValidateProductIDForOrders(ProductTypeID, ProductID) = 1));

```

## WIDOKI

### 1. Raport finansowy - kto zapłacił całość, kto część, a kto jeszcze nic (Kinga)

```

CREATE VIEW FinancialReport AS
SELECT
    U.FirstName + ' ' + U.LastName AS StudentName,
    CP.StudentID,
    CP.Paid,
    C.CourseName AS PaymentFor,
    'Course' AS ProductType,
    CP.CourseID AS ProductID,
    CASE
        WHEN CP.Paid = 1 THEN C.Price
        ELSE C.Advance
    END AS TotalAmountPaid
FROM CoursePayments CP
JOIN Courses C ON CP.CourseID = C.CourseID
JOIN Users U ON CP.StudentID = U.UserID

UNION

SELECT
    U.FirstName + ' ' + U.LastName AS StudentName,
    WP.ParticipantID AS StudentID,
    WP.Paid,
    W.WebinarName AS PaymentFor,
    'Webinar' AS ProductType,
    WP.WebinarID AS ProductID,
    CASE
        WHEN WP.Paid = 1 THEN W.Price
        ELSE 0
    END AS TotalAmountPaid

```

```

FROM WebinarPayments WP
JOIN Webinars W ON WP.WebinarID = W.WebinarID
JOIN Users U ON WP.ParticipantID = U.UserID

UNION

SELECT
    U.FirstName + ' ' + U.LastName AS StudentName,
    SPayment.StudentID,
    SPayment.Paid,
    CONCAT('Session ', SPayment.SessionID, ' from Study ', Ses.StudyID) AS
PaymentFor,
    'Session' AS ProductType,
    SPayment.SessionID AS ProductID,
    CASE
        WHEN SPayment.Paid = 1 THEN
            CASE
                WHEN L.UserID IS NOT NULL THEN Ses.ParticipantPrice
                ELSE Ses.GuestPrice
            END
        ELSE 0
    END AS TotalAmountPaid
FROM SessionPayments SPayment
JOIN Users U ON SPayment.StudentID = U.UserID
JOIN Sessions Ses ON SPayment.SessionID = Ses.SessionID
LEFT JOIN Logins L ON SPayment.StudentID = L.UserID

UNION

SELECT
    U.FirstName + ' ' + U.LastName AS StudentName,
    SC.StudentID,
    CASE WHEN SC.PaidSessions = StudySessions.TotalSessions AND
StudySessions.TotalSessions > 0 THEN 1 ELSE 0 END AS Paid,
    Stud.StudyName AS PaymentFor,
    'Studies' AS ProductType,
    Stud.StudyID AS ProductID,
    SC.TotalAmountPaid
FROM Studies Stud
JOIN (
    SELECT
        Ses.StudyID,
        SP.StudentID,
        SUM(CAST(SP.Paid AS INT)) AS PaidSessions,
        SUM(CASE WHEN L.UserID IS NOT NULL THEN Ses.ParticipantPrice *
CAST(SP.Paid AS INT) ELSE Ses.GuestPrice * CAST(SP.Paid AS INT) END) AS
TotalAmountPaid
    FROM
        Sessions Ses
    LEFT JOIN
        SessionPayments SP ON SP.SessionID = Ses.SessionID

```

```

LEFT JOIN
    Logins L ON SP.StudentID = L.UserID
GROUP BY
    Ses.StudyID, SP.StudentID
) AS SC ON Stud.StudyID = SC.StudyID
JOIN Users U ON SC.StudentID = U.UserID
JOIN (
    SELECT StudyID, COUNT(SessionID) AS TotalSessions
    FROM Sessions
    GROUP BY StudyID
) AS StudySessions ON Stud.StudyID = StudySessions.StudyID;

GO

```

## 2. RaportFinansowyPerProdukt (Kinga)

```

SELECT
    C.CourseID AS ProductID,
    C.CourseName AS ProductName,
    'Course' AS ProductType,
    SUM(CASE WHEN CP.Paid = 1 THEN C.Price ELSE C.Advance END) AS TotalIncome
FROM CoursePayments CP
JOIN Courses C ON CP.CourseID = C.CourseID
GROUP BY C.CourseID, C.CourseName

UNION

SELECT
    WP.WebinarID AS ProductID,
    W.WebinarName AS ProductName,
    'Webinar' AS ProductType,
    SUM(CASE WHEN WP.Paid = 1 THEN W.Price ELSE 0 END) AS TotalAmountPaid
FROM WebinarPayments WP
JOIN Webinars W ON WP.WebinarID = W.WebinarID
GROUP BY WP.WebinarID, W.WebinarName

UNION

SELECT
    Ses.SessionID AS ProductID,
    CONCAT('Session ', Ses.SessionID, ' from Study ', Ses.StudyID) AS ProductName,
    'Session' AS ProductType,
    SUM(CASE WHEN SP.Paid = 1 THEN
        CASE WHEN L.UserID IS NOT NULL THEN Ses.ParticipantPrice ELSE
Ses.GuestPrice END
        ELSE 0 END) AS TotalAmountPaid
FROM SessionPayments SP
JOIN Sessions Ses ON SP.SessionID = Ses.SessionID
LEFT JOIN Logins L ON SP.StudentID = L.UserID
GROUP BY Ses.SessionID, Ses.StudyID

GO

```

### 3. Osoby z zaległymi płatnościami (Kinga)

```
CREATE VIEW OverduePaymentsView AS
SELECT U.FirstName + ' ' + U.LastName AS StudentName,
       CP.StudentID,
       CP.Paid,
       C.CourseName AS PaymentFor,
       'Course' AS ProductType,
       CP.CourseID AS ProductID
FROM CoursePayments CP
JOIN Courses C ON CP.CourseID = C.CourseID
JOIN Users U ON CP.StudentID = U.UserID
WHERE CP.Paid = 0 AND CP.DueDate < GETDATE()

UNION

SELECT U.FirstName + ' ' + U.LastName AS StudentName,
       WP.ParticipantID AS StudentID,
       WP.Paid,
       W.WebinarName AS PaymentFor,
       'Webinar' AS ProductType,
       WP.WebinarID AS ProductID
FROM WebinarPayments WP
JOIN Webinars W ON WP.WebinarID = W.WebinarID
JOIN Users U ON WP.ParticipantID = U.UserID
WHERE WP.Paid = 0 AND WP.DueDate < GETDATE()

UNION

SELECT U.FirstName + ' ' + U.LastName AS StudentName,
       SPayment.StudentID,
       SPayment.Paid,
       CONCAT('Session ', SPayment.SessionID, ' from Study ', Ses.StudyID) AS
PaymentFor,
       'Session' AS ProductType,
       SPayment.SessionID AS ProductID
FROM SessionPayments SPayment
JOIN Users U ON SPayment.StudentID = U.UserID
JOIN Sessions Ses ON SPayment.SessionID = Ses.SessionID
WHERE SPayment.Paid = 0 AND SPayment.DueDate < GETDATE()

GO
```

### 4. Pozyskiwanie kursów hybrydowych wraz z % spotkań stacjonarnych oraz informacją, ile modułów obejmuje kurs hybrydowy (Kinga)

```
CREATE VIEW HibridCourses AS
SELECT DISTINCT -- tylko hybrydowe
```

```

        C.CourseID,
        C.CourseName,
        'Hybrydowe' AS FormType,
        COUNT(DISTINCT M.ModuleID) AS NumberOfModules,
        (COUNT(DISTINCT IP.ModuleID) * 100.0) / COUNT(DISTINCT
M.ModuleID) AS PercentInPersonModules
    FROM Courses C
JOIN Modules M ON M.CourseID = C.CourseID
LEFT JOIN ModulesInPerson AS IP ON M.ModuleID = IP.ModuleID
WHERE C.CourseID IN
    (SELECT CourseID FROM Modules M1
    JOIN ModulesInPerson AS MIP
    ON M1.ModuleID = MIP.ModuleID)
    AND (C.CourseID IN (SELECT CourseID FROM Modules M2
    JOIN ModulesAsync AS MA
    ON M2.ModuleID = MA.ModuleID)
    OR C.CourseID IN (SELECT CourseID FROM Modules M3
    JOIN ModulesSync AS MS
    ON M3.ModuleID = MS.ModuleID))
GROUP BY C.CourseID, C.CourseName;

```

5. Pozyskiwanie informacji czy kurs jest stacjonarny / online asynchroniczny / online synchroniczny / hybrydowy korzystając z widoku z pkt. 4 (Kinga)

```

CREATE VIEW CourseTypes AS
SELECT DISTINCT -- tylko hybrydowe
    CourseID,
    CourseName,
    FormType,
    NumberOfModules
FROM HibridCourses

UNION

SELECT DISTINCT -- tylko stacjonarne
    C.CourseID,
    C.CourseName,
    'Stacjonarne' AS FormType,
    COUNT(DISTINCT M.ModuleID) AS NumberOfModules
FROM Courses C
JOIN Modules M ON M.CourseID = C.CourseID
WHERE C.CourseID IN
    (SELECT CourseID FROM Modules M1
    JOIN ModulesInPerson AS MIP
    ON M1.ModuleID = MIP.ModuleID)

```





```

AND C.CourseID NOT IN (SELECT CourseID FROM
Modules M3
JOIN ModulesSync AS MS
ON M3.ModuleID = MS.ModuleID)
GROUP BY C.CourseID, C.CourseName;

```

6. Pozyskiwanie zjazdów hybrydowych wraz z % spotkań (klas) stacjonarnych oraz informacją, ile klas obejmuje zjazd hybrydowy (Kinga)

```

CREATE VIEW HibridSessions AS
SELECT DISTINCT -- tylko hybrydowe
    S.SessionID,
    'Hybrydowe' AS FormType,
    COUNT(DISTINCT C.ClassID) AS NumberOfClasses,
    (COUNT(DISTINCT IP.ClassID) * 100.0) / COUNT(DISTINCT
C.ClassID) AS PercentInPersonClasses
FROM Sessions S
JOIN Classes C ON S.SessionID = C.SessionID
LEFT JOIN ClassesInPerson AS IP ON C.ClassID = IP.ClassID
WHERE S.SessionID IN
    (SELECT SessionID FROM Classes C1
    JOIN ClassesInPerson AS CIP
    ON C1.ClassID = CIP.ClassID)
    AND (S.SessionID IN (SELECT SessionID FROM Classes C2
    JOIN ClassesAsync AS CA
    ON C2.ClassID = CA.ClassID)
    OR S.SessionID IN (SELECT SessionID FROM Classes
C3
JOIN ClassesSync AS CS
ON C3.ClassID = CS.ClassID))

GROUP BY S.SessionID;

```

7. Pozyskiwanie informacji czy zjazd jest stacjonarny / online asynchroniczny / online synchroniczny / hybrydowy korzystając z widoku z pkt. 6 (Kinga)

```

CREATE VIEW SessionTypes AS
SELECT DISTINCT -- tylko hybrydowe
    SessionID,
    FormType,
    NumberOfClasses
FROM HibridSessions

UNION

SELECT DISTINCT -- tylko stacjonarne
    S.SessionID,
    'Stacjonarne' AS FormType,

```

```

        COUNT(DISTINCT C.ClassID) AS NumberOfClasses
    FROM Sessions S
JOIN Classes C ON C.SessionID = S.SessionID
WHERE S.SessionID IN
    (SELECT SessionID FROM Classes C1
     JOIN ClassesInPerson AS CIP
     ON C1.ClassID = CIP.ClassID)
    AND S.SessionID NOT IN (SELECT SessionID FROM Classes C2
                            JOIN ClassesAsync AS CA
                            ON C2.ClassID = CA.ClassID)
    AND S.SessionID NOT IN (SELECT SessionID FROM Classes C3
                            JOIN ClassesSync AS CS
                            ON C3.ClassID = CS.ClassID)

GROUP BY S.SessionID

```

UNION

```

SELECT DISTINCT -- tylko online synchroniczne
    S.SessionID,
    'Online Synchroniczne' AS FormType,
    COUNT(DISTINCT C.ClassID) AS NumberOfClasses
FROM Sessions S
JOIN Classes C ON C.SessionID = S.SessionID
WHERE S.SessionID NOT IN
    (SELECT SessionID FROM Classes C1
     JOIN ClassesInPerson AS CIP
     ON C1.ClassID = CIP.ClassID)
    AND S.SessionID NOT IN (SELECT SessionID FROM Classes C2
                            JOIN ClassesAsync AS CA
                            ON C2.ClassID = CA.ClassID)
    AND S.SessionID IN (SELECT SessionID FROM Classes C3
                       JOIN ClassesSync AS CS
                       ON C3.ClassID = CS.ClassID)

GROUP BY S.SessionID

```

UNION

```

SELECT DISTINCT -- tylko online asynchroniczne
    S.SessionID,
    'Online Asynchroniczne' AS FormType,
    COUNT(DISTINCT C.ClassID) AS NumberOfClasses
FROM Sessions S
JOIN Classes C ON C.SessionID = S.SessionID
WHERE S.SessionID NOT IN
    (SELECT SessionID FROM Classes C1
     JOIN ClassesInPerson AS CIP
     ON C1.ClassID = CIP.ClassID)

```

```

        AND S.SessionID IN (SELECT SessionID FROM Classes C2
                           JOIN ClassesAsync AS CA
                           ON C2.ClassID = CA.ClassID)
        AND S.SessionID NOT IN (SELECT SessionID FROM Classes C3
                               JOIN ClassesSync AS CS
                               ON C3.ClassID = CS.ClassID)
GROUP BY S.SessionID;

```

8. Pozyskiwanie informacji czy studia są stacjonarny / online asynchroniczny / online synchroniczny / hybrydowy korzystając z widoku z pkt. 7 (Kinga)

```

SELECT
    S.StudyID,
    S.StudyName,
    CASE
        WHEN COUNT(DISTINCT ST.FormType) = 1 AND MAX(ST.FormType) =
        'Stacjonarne' THEN 'Stacjonarne'
        WHEN COUNT(DISTINCT ST.FormType) = 1 AND MAX(ST.FormType) =
        'Online Asynchroniczne' THEN 'Online Asynchroniczne'
        WHEN COUNT(DISTINCT ST.FormType) = 1 AND MAX(ST.FormType) =
        'Online Synchroniczne' THEN 'Online Synchroniczne'
        WHEN COUNT(DISTINCT ST.FormType) = 1 AND MAX(ST.FormType) =
        'Hybrydowe' THEN 'Hybrydowe'
        WHEN COUNT(DISTINCT ST.FormType) > 1 THEN 'Hybrydowe'
    END AS StudyType
FROM
    Studies S
JOIN
    SessionTypes ST ON S.StudyID = ST.StudyID
GROUP BY
    S.StudyID, S.StudyName;

```

9. Frekwencja na kursach (Kinga)

```

CREATE VIEW CourseFrequency AS
SELECT A.CourseID, B.StudentID, A.ModuleCount AS TotalModules,
       B.AttendanceCount AS AttendanceModules,
       (B.AttendanceCount * 1.0 / A.ModuleCount) * 100.0 AS
AttendancePercentage
FROM (SELECT C.CourseID,
            COUNT(M.ModuleID) AS ModuleCount
      FROM Courses AS C
      JOIN Modules AS M ON C.CourseID = M.CourseID
      GROUP BY C.CourseID) AS A
JOIN (SELECT M.CourseID, MA.StudentID,

```

```

COUNT(MA.Present) AS AttendanceCount
FROM Modules AS M
JOIN ModulesAttendances AS MA ON M.ModuleID = MA.ModuleID
GROUP BY M.CourseID, MA.StudentID) AS B ON A.CourseID =
B.CourseID;

```

10. Pozyskiwanie informacji, komu należy wysłać dyplomy za kursy pod warunkiem, że frekwencja uczestnika wynosiła min. 80% oraz zapłacił on za kurs, wykorzystuje widok z punktu 9. (Kinga)

```

CREATE VIEW CoursesDiplomasToSend AS
SELECT CF.StudentID, CONCAT(CityName, ' ', ul. ' ', Street, ' ',
BuildingNumber,
CASE WHEN FlatNumber IS NOT NULL THEN CONCAT('/', FlatNumber)
ELSE ''
END) AS FullAddress,
CF.CourseID, CourseName, AttendancePercentage, StartDate, EndDate
FROM CourseFrequency AS CF
JOIN Courses AS C ON CF.CourseID = C.CourseID
JOIN CoursePayments AS P ON CF.CourseID = P.CourseID
LEFT JOIN StudentsAddresses AS SA ON CF.StudentID = SA.StudentID
JOIN Cities AS CIT ON SA.CityID = CIT.CityID
WHERE AttendancePercentage >= 80.0 AND EndDate < GETDATE() AND Paid = 1
AND NOT EXISTS (SELECT 1 FROM Diplomas AS D
WHERE D.ProductID = CF.CourseID AND D.StudentID = CF.StudentID AND
TypeID = 1);

```

11. Frekwencja na zjazdach (Kinga)

```

CREATE VIEW SessionsFrequency AS
SELECT A.SessionID, B.StudentID, A.ClassCount AS TotalModules,
B.AttendanceCount AS AttendanceClasses,
(B.AttendanceCount * 1.0 / A.ClassCount) * 100.0 AS
AttendancePercentage
FROM (SELECT S.SessionID,
COUNT(ClassID) AS ClassCount
FROM Sessions AS S
JOIN Classes AS C ON S.SessionID = C.SessionID
GROUP BY S.SessionID) AS A
JOIN (SELECT C.SessionID, CA.StudentID,
COUNT(CA.Present) AS AttendanceCount
FROM Classes AS C
JOIN ClassesAttendances AS CA ON C.ClassID = CA.ClassID
GROUP BY C.SessionID, CA.StudentID) AS B ON A.SessionID =
B.SessionID;

```

12. Frekwencja na studiach, liczymy frekwencje na zajeciach, aby byla wieksza lub rowna 80%, tzn. na jednym zjezdzie mozna miec 70% a na drugim 90%, srednia 80%, wiec jest okej, korzysta z widoku 11. (Kinga)

```
CREATE VIEW StudiesFrequency AS
SELECT A.StudyID, B.StudentID, A.SessionCount AS TotalSessions,
       B.AttendanceCount AS AttendanceSessions,
       (B.AttendanceCount * 1.0 / A.SessionCount) * 100.0 AS
AttendancePercentage
FROM (SELECT ST.StudyID, -- ilosc wszystkich zjazdow
      COUNT(SessionID) AS SessionCount
      FROM Studies AS ST
      JOIN Sessions AS SE ON ST.StudyID = SE.StudyID
      GROUP BY ST.StudyID) AS A
JOIN (SELECT SE.StudyID, SF.StudentID, -- obecność na zjazdach
      COUNT(AttendancePercentage) AS AttendanceCount
      FROM Sessions AS SE
      JOIN SessionsFrequency AS SF ON SE.SessionID = SF.SessionID
      GROUP BY SE.StudyID, SF.StudentID
      HAVING SUM(AttendancePercentage)/3 >= 80) AS B ON A.StudyID =
B.StudentID;
```

13. Pozyskiwanie informacji, komu należy wysłać dyplomy za studia pod warunkiem, że frekwencja studenta wynosiła min. 80% oraz zapłacił on za studia, wykorzystuje widok z punktu 12. (Kinga)

```
CREATE VIEW StudiesDiplomasToSend AS
SELECT SF.StudentID, CONCAT(CityName, ' ', ul. ' ', Street, ' ',
BuildingNumber,
      CASE WHEN FlatNumber IS NOT NULL THEN CONCAT('/', FlatNumber)
      ELSE ''
      END) AS FullAddress,
      SF.StudyID, StudyName, AttendancePercentage, StartDate, EndDate
FROM StudiesFrequency AS SF
JOIN Studies AS S ON SF.StudyID = S.StudyID
LEFT JOIN StudentsAddresses AS SA ON SF.StudentID = SA.StudentID
JOIN Cities AS CIT ON SA.CityID = CIT.CityID
JOIN (SELECT StudyID, SUM(ParticipantPrice) AS StudyPrice
      FROM Sessions
      GROUP BY StudyID) AS A ON SF.StudyID = A.StudyID
JOIN (SELECT StudyID, StudentID, SUM(ParticipantPrice) AS PaidAmount
      FROM Sessions AS SE
      JOIN SessionPayments AS SP ON SE.SessionID = SP.SessionID
      WHERE Paid = 1
```

```

        GROUP BY StudyID, StudentID) AS B ON SF.StudyID = B.StudyID
AND SF.StudentID = B.StudentID
WHERE AttendancePercentage >= 80.0 AND EndDate < GETDATE()
        AND NOT EXISTS (SELECT 1
                                FROM Diplomas AS D
                                WHERE D.ProductID = SF.StudyID
AND D.StudentID = SF.StudentID AND TypeID = 2)
        AND A.StudyPrice = B.PaidAmount;

```

14. Widok zawierający szczegółowy raport dotyczący liczby zapisanych osób na przyszłe webinary. (Paulina)

```

CREATE VIEW WebinarsInFuture AS
SELECT w.WebinarID AS EventID,
COUNT(DISTINCT wp.ParticipantID) AS TotalParticipants, 'Online-Sync' AS
EventType
FROM webinars w
JOIN WebinarPayments wp ON w.WebinarID = wp.WebinarID
WHERE w.Date > GETDATE()
GROUP BY w.WebinarID

```

15. Widok zawierający szczegółowy raport dotyczący liczby zapisanych osób na przyszłe kursy z informacją, czy kursy są stacjonarnie, czy zdalnie czy w trybie hybrydowym. -Widok pomocniczy określający typ każdego kursu. (Paulina)

```

CREATE VIEW CoursesEventTypes AS
SELECT DISTINCT
    C.CourseID,
    CASE
        WHEN (
            EXISTS (
                SELECT 1
                FROM ModulesInPerson MIP
                JOIN Modules MO ON MIP.ModuleID = MO.ModuleID
                JOIN Courses CO ON MO.CourseID = CO.CourseID
                WHERE CO.CourseID = C.CourseID
            )
        AND (
            EXISTS (
                SELECT 1
                FROM ModulesAsync MA
                JOIN Modules MO ON MA.ModuleID = MO.ModuleID
                JOIN Courses CO ON MO.CourseID = CO.CourseID
                WHERE CO.CourseID = C.CourseID
            )
        )
    END AS EventType

```

```

) OR
EXISTS (
    SELECT 1
    FROM ModulesSync MS
    JOIN Modules MO ON MS.ModuleID = MO.ModuleID
    JOIN Courses CO ON MO.CourseID = CO.CourseID
    WHERE CO.CourseID = C.CourseID
)
) ) THEN 'Hybrid'
ELSE CASE
    WHEN (EXISTS (
        SELECT 1
        FROM ModulesInPerson MIP
        JOIN Modules MO ON MIP.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    ) AND NOT EXISTS (
        SELECT 1
        FROM ModulesAsync MA
        JOIN Modules MO ON MA.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    ) AND NOT EXISTS (
        SELECT 1
        FROM ModulesSync MS
        JOIN Modules MO ON MS.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    )) THEN 'InPerson'
    WHEN (NOT EXISTS (
        SELECT 1
        FROM ModulesInPerson MIP
        JOIN Modules MO ON MIP.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    ) AND EXISTS (
        SELECT 1
        FROM ModulesAsync MA
        JOIN Modules MO ON MA.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    ) AND NOT EXISTS (
        SELECT 1
        FROM ModulesSync MS
        JOIN Modules MO ON MS.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    )) THEN 'Hybrid'
    ELSE 'Online'
END CASE

```

```

    )) THEN 'Online-Async'
    WHEN (NOT EXISTS (
        SELECT 1
        FROM ModulesInPerson MIP
        JOIN Modules MO ON MIP.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    ) AND NOT EXISTS (
        SELECT 1
        FROM ModulesAsync MA
        JOIN Modules MO ON MA.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    ) AND EXISTS (
        SELECT 1
        FROM ModulesSync MS
        JOIN Modules MO ON MS.ModuleID = MO.ModuleID
        JOIN Courses CO ON MO.CourseID = CO.CourseID
        WHERE CO.CourseID = C.CourseID
    )) THEN 'Online-Sync'
        WHEN (
            EXISTS (
                SELECT 1
                FROM ModulesAsync MA
                JOIN Modules MO ON MA.ModuleID = MO.ModuleID
                JOIN Courses CO ON MO.CourseID = CO.CourseID
                WHERE CO.CourseID = C.CourseID
            ) AND
            EXISTS (
                SELECT 1
                FROM ModulesSync MS
                JOIN Modules MO ON MS.ModuleID = MO.ModuleID
                JOIN Courses CO ON MO.CourseID = CO.CourseID
                WHERE CO.CourseID = C.CourseID
            )
        ) THEN 'Online-Async&Sync'
    ELSE 'NoModules'
END
END AS 'EventType'
FROM Courses C
JOIN Modules MO ON C.CourseID = MO.CourseID
JOIN Lecturers L ON MO.LecturerID = L.LecturerID;

CREATE VIEW CoursesInFuture AS
SELECT

```



```

        C.CourseID AS EventID,
        COUNT(DISTINCT CP.ParticipantID) AS TotalParticipants,
        CET.EventType
FROM
    CourseParticipants CP
JOIN
    Courses C ON CP.CourseID = C.CourseID
JOIN
    CoursesEventTypes CET ON C.CourseID = CET.CourseID
WHERE
    C.StartDate > GETDATE()
GROUP BY
    C.CourseID, CET.EventType;

```

16. Widok zawierający szczegółowy raport dotyczący liczby zapisanych osób na przyszłe zjazdy z informacją, czy zjazdy są stacjonarnie, czy zdalnie czy w trybie hybrydowym. (Paulina)

```

CREATE VIEW SessionsEventTypes AS
SELECT DISTINCT
    S.SessionID,
    CASE
        WHEN (
            EXISTS (
                SELECT 1
                FROM ClassesInPerson CIP
                JOIN Classes C ON CIP.ClassID = C.ClassID
                WHERE C.SessionID = S.SessionID
            )
        AND (
            EXISTS (
                SELECT 1
                FROM ClassesAsync CA
                JOIN Classes C ON CA.ClassID = C.ClassID
                WHERE C.SessionID = S.SessionID
            ) OR
            EXISTS (
                SELECT 1
                FROM ClassesSync CS
                JOIN Classes C ON CS.ClassID = C.ClassID
                WHERE C.SessionID = S.SessionID
            )
        ) THEN 'Hybrid'
    ELSE CASE
        WHEN (EXISTS (

```

```

SELECT 1
FROM ClassesInPerson CIP
JOIN Classes C ON CIP.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
) AND NOT EXISTS (
SELECT 1
FROM ClassesAsync CA
JOIN Classes C ON CA.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
) AND NOT EXISTS (
SELECT 1
FROM ClassesSync CS
JOIN Classes C ON CS.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
)) THEN 'InPerson'
WHEN (NOT EXISTS (
SELECT 1
FROM ClassesInPerson CIP
JOIN Classes C ON CIP.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
) AND EXISTS (
SELECT 1
FROM ClassesAsync CA
JOIN Classes C ON CA.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
) AND NOT EXISTS (
SELECT 1
FROM ClassesSync CS
JOIN Classes C ON CS.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
)) THEN 'Online-Async'
WHEN (NOT EXISTS (
SELECT 1
FROM ClassesInPerson CIP
JOIN Classes C ON CIP.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
) AND NOT EXISTS (
SELECT 1
FROM ClassesAsync CA
JOIN Classes C ON CA.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
) AND EXISTS (
SELECT 1
FROM ClassesSync CS
JOIN Classes C ON CS.ClassID = C.ClassID
WHERE C.SessionID = S.SessionID
)) THEN 'Online-Sync'

```

```

        WHEN (
            EXISTS (
                SELECT 1
                FROM ClassesAsync CA
                JOIN Classes C ON CA.ClassID = C.ClassID
                WHERE C.SessionID = S.SessionID
            ) AND
            EXISTS (
                SELECT 1
                FROM ClassesSync CS
                JOIN Classes C ON CS.ClassID = C.ClassID
                WHERE C.SessionID = S.SessionID
            )
        ) THEN 'Online-Async&Sync'
    ELSE 'NoClasses'
END
END AS 'EventType'
FROM Sessions S
JOIN Classes C ON C.SessionID = S.SessionID;

```

```

CREATE VIEW SessionsInFuture AS
SELECT
    S.SessionID AS EventID,
    COUNT(DISTINCT SP.StudentID) AS TotalParticipants,
    SE.EventType
FROM
    SessionPayments SP
JOIN
    Sessions S ON SP.SessionID = S.SessionID
JOIN
    SessionsEventTypes SE ON S.SessionID = SE.SessionID
WHERE
    S.StartDate > GETDATE() AND SP.StudentID IN (SELECT DISTINCT(O.UserID)
FROM Orders O
JOIN OrderDetails OD ON O.OrderID = OD.OrderID
WHERE ProductTypeID = 4)
GROUP BY
    S.SessionID, SE.EventType;

```

17. Widok zawierający szczegółowy raport dotyczący liczby zapisanych osób na przyszłe studia z informacją, czy są stacjonarnie, czy zdalnie czy w trybie hybrydowym. (Paulina)

```

CREATE VIEW StudiesInFuture AS
SELECT

```

```

S.StudyID AS EventID,
COUNT(DISTINCT SP.StudentID) AS 'TotalParticipants',
CASE
    WHEN COUNT(DISTINCT ST.EventType) = 1 THEN
        MAX(CASE
            WHEN ST.EventType = 'Hybrid' THEN 'Hybrid'
            WHEN ST.EventType = 'InPerson' THEN 'InPerson'
            WHEN ST.EventType = 'Async' THEN 'Async'
            WHEN ST.EventType = 'Sync' THEN 'Sync'
            WHEN ST.EventType = 'Async&Sync' THEN 'Async&Sync'
            ELSE 'NoClasses'
        )
    ELSE 'HybridStudy' -- lub dowolny inny domyślny typ w przypadku
mieszanych sesji
END AS 'EventType'
FROM Studies S
JOIN Sessions SE ON SE.StudyID = S.StudyID
JOIN SessionsEventTypes ST ON ST.SessionID = SE.SessionID
JOIN SessionPayments SP ON SP.SessionID = SE.SessionID
WHERE SP.StudentID IN (
    SELECT DISTINCT O.UserID
    FROM Orders O
    JOIN OrderDetails OD ON O.OrderID = OD.OrderID
    WHERE OD.ProductTypeID = 2
)
AND S.StartDate > GETDATE()
GROUP BY S.StudyID;

```

18.Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia (z informacją, czy wydarzenie jest stacjonarnie, czy zdalnie). (Paulina)

```

CREATE VIEW EventsInFuture AS
SELECT 'session' AS Type, EventID, TotalParticipants, EventType
FROM SessionsInFuture
UNION ALL
SELECT 'course', EventID, TotalParticipants, EventType
FROM CoursesInFuture
UNION ALL
SELECT 'webinar', EventID, TotalParticipants, EventType
FROM WebinarsInFuture
UNION ALL
SELECT 'study', EventID, TotalParticipants, EventType
FROM StudiesInFuture;

```

19. Widok dla frekwencji na zajęciach zakończonych (Inperson oraz Sync)  
(Paulina)

```
CREATE VIEW AttendanceListClassesInPast AS
SELECT
    u.UserID AS UserId,
    u.FirstName,
    u.LastName,
    c.ClassID AS EventId,
    CASE
        WHEN ip.StartTime IS NOT NULL THEN ip.StartTime
        WHEN cs.StartTime IS NOT NULL THEN cs.StartTime
    END AS EventStart,
    CASE
        WHEN ip.EndTime IS NOT NULL THEN ip.EndTime
        WHEN cs.EndTime IS NOT NULL THEN cs.EndTime
    END AS EventEnd,
    ca.Present AS Attendance
FROM
    Users u
JOIN
    ClassesAttendances ca ON u.UserID = ca.StudentID
JOIN
    Classes c ON ca.ClassID = c.ClassID
LEFT JOIN
    ClassesInPerson ip ON c.ClassID = ip.ClassID
LEFT JOIN
    ClassesSync cs ON c.ClassID = cs.ClassID
WHERE
    (
        (ip.StartTime IS NOT NULL AND ip.EndTime IS NOT NULL)
        OR (cs.StartTime IS NOT NULL AND cs.EndTime IS NOT NULL)
    )
    AND (COALESCE(ip.EndTime, cs.EndTime) < GETDATE());
```

20. Raport frekwencji na zajęciach, które się już zakończyły (InPerson, Sync).  
(Paulina)

```
CREATE VIEW AttendanceReportClassesInPast AS
SELECT
    EventId,
    EventStart,
    EventEnd,
    COUNT(UserId) AS TotalParticipants,
    SUM(CASE WHEN Attendance = 1 THEN 1 ELSE 0 END) AS TotalAttendance,
```

```

        CAST((SUM(CASE WHEN Attendance = 1 THEN 1 ELSE 0 END) * 100.0 /
NULLIF(COUNT(UserId), 0)) AS DECIMAL(5,2)) AS AttendancePercentage
FROM
    AttendanceListClassesInPast
GROUP BY
    EventId,
    EventStart,
    EventEnd;

```

## 21. Widok dla frekwencji na modułach zakończonych (Inperson oraz Sync) (Paulina)

```

CREATE VIEW AttendanceListModulesInPast AS
SELECT
    u.UserID AS UserId,
    u.FirstName,
    u.LastName,
    m.ModuleID AS EventId,
    CASE
        WHEN ip.StartTime IS NOT NULL THEN ip.StartTime
        WHEN ms.StartTime IS NOT NULL THEN ms.StartTime
    END AS EventStart,
    CASE
        WHEN ip.EndTime IS NOT NULL THEN ip.EndTime
        WHEN ms.EndTime IS NOT NULL THEN ms.EndTime
    END AS EventEnd,
    ma.Present AS Attendance
FROM
    Users u
JOIN
    ModulesAttendances ma ON u.UserID = ma.StudentID
JOIN
    Modules m ON ma.ModuleID = m.ModuleID
LEFT JOIN
    ModulesInPerson ip ON m.ModuleID = ip.ModuleID
LEFT JOIN
    ModulesSync ms ON m.ModuleID = ms.ModuleID
WHERE
    (
        (ip.StartTime IS NOT NULL AND ip.EndTime IS NOT NULL)
        OR (ms.StartTime IS NOT NULL AND ms.EndTime IS NOT NULL)
    )
    AND (COALESCE(ip.EndTime, ms.EndTime) < GETDATE());

```

## 22. Raport frekwencji na modułach, które się już zakończyły (InPerson,Sync). (Paulina)

```

CREATE VIEW AttendanceReportModulesInPast AS
SELECT
    EventId,
    EventStart,
    EventEnd,
    COUNT(UserId) AS TotalParticipants,
    SUM(CASE WHEN Attendance = 1 THEN 1 ELSE 0 END) AS TotalAttendance,
    CAST((SUM(CASE WHEN Attendance = 1 THEN 1 ELSE 0 END) * 100.0 /
    NULLIF(COUNT(UserId), 0)) AS DECIMAL(5,2)) AS AttendancePercentage

FROM
    AttendanceListModulesInPast
GROUP BY
    EventId,
    EventStart,
    EventEnd;

```

23. Ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach.  
(Paulina)

```

CREATE VIEW AttendanceReportInPast AS
SELECT 'module' AS EventType, EventId
    ,EventStart
    ,EventEnd
    ,TotalParticipants
    ,TotalAttendance
    ,AttendancePercentage
FROM AttendanceReportModulesInPast
UNION ALL
SELECT 'class', EventId
    ,EventStart
    ,EventEnd
    ,TotalParticipants
    ,TotalAttendance
    ,AttendancePercentage
FROM AttendanceReportClassesInPast

```

24. Obecność na poszczególnych zajęciach (Karol) - wraz z datami rozpoczęcia oraz zakończenia zajęć (NULL w przypadku zajęć asynchronicznych)

```

CREATE VIEW AttendancesAtClasses AS
SELECT CA.StudentID, U.FirstName, U.LastName, CP.ClassID, CP.StartTime,
CP.EndTime, CA.Present

```

```

FROM ClassesAttendances CA
JOIN Users U ON CA.StudentID = U.UserID
JOIN ClassesInPerson CP ON CP.ClassID = CA.ClassID
WHERE EndTime < GETDATE()
UNION
SELECT CA.StudentID, U.FirstName, U.LastName, CAS.ClassID, NULL AS
StartTime, NULL AS EndTime, CA.Present
FROM ClassesAttendances CA
LEFT JOIN ClassesInPerson CP ON CP.ClassID = CA.ClassID
LEFT JOIN Users U ON CA.StudentID = U.UserID
LEFT JOIN ClassesAsync CAS ON CAS.ClassID = CA.ClassID
WHERE CAS.ClassID NOT IN (SELECT ModuleID FROM ModulesInPerson)
UNION
SELECT CA.StudentID, U.FirstName, U.LastName, CA.ClassID, CS.StartTime,
CS.EndTime, CA.Present
FROM ClassesAttendances CA
JOIN Users U ON CA.StudentID = U.UserID
JOIN ClassesSync CS ON CS.ClassID = CA.ClassID
WHERE EndTime < GETDATE()

```

## 25. Obecność na poszczególnych zjazdach (Karol)

```

CREATE VIEW AttendancesAtSessions AS
SELECT S.StudyID, S.SessionID, AC.ClassID, AC.StudentID, S.StartDate,
S.EndDate, AC.Present
From Sessions S JOIN Classes C
ON S.SessionID = C.SessionID
JOIN AttendancesAtClasses AC
ON AC.ClassID = C.ClassID
WHERE StartDate < GETDATE()

```

## 26. Obecność na poszczególnych modułach wraz z datą (Karol)

```

CREATE VIEW AttendancesAtModules AS
SELECT MA.StudentID, U.FirstName, U.LastName, Ma.ModuleID, MS.StartTime,
MS.EndTime, MA.Present
FROM ModulesAttendances MA
JOIN Users U ON MA.StudentID = U.UserID
JOIN ModulesInPerson MS ON MS.ModuleID = MA.ModuleID
WHERE EndTime < GETDATE()
UNION
SELECT MA.StudentID, U.FirstName, U.LastName, MAS.ModuleID, NULL AS
StartTime, NULL AS EndTime, MA.Present
FROM ModulesAttendances MA

```



```

LEFT JOIN ModulesInPerson MI ON MI.ModuleID = MA.ModuleID
LEFT JOIN Users U ON MA.StudentID = U.UserID
LEFT JOIN ModulesAsync MAS ON MAS.ModuleID = MA.ModuleID
WHERE MAS.ModuleID NOT IN (SELECT ModuleID FROM ModulesInPerson)
UNION
SELECT MA.StudentID, U.FirstName, U.LastName, Ma.ModuleID, MS.StartTime,
MS.EndTime, MA.Present
FROM ModulesAttendances MA
JOIN Users U ON MA.StudentID = U.UserID
JOIN ModulesSync MS ON MS.ModuleID = MA.ModuleID
WHERE EndTime < GETDATE()

```

## 27. Obecność na kursach (Karol)

```

CREATE VIEW AttendancesAtCourses AS
SELECT M.CourseID, M.ModuleID, AM.StudentID, AM.StartTime, AM.EndTime,
AM.Present
FROM Modules M JOIN AttendancesAtModules AM
ON M.ModuleID = AM.ModuleID
WHERE StartTime < GETDATE() OR StartTime IS NULL

```

## 28. Lista przyszłych kursów wraz z dokładnymi datami rozpoczęcia podlegających im modułów (Karol)

```

CREATE VIEW FutureCourses AS
SELECT M.CourseID, MIP.ModuleID, MIP.StartTime, MIP.EndTime
FROM ModulesInPerson MIP JOIN Modules M
ON MIP.ModuleID = M.ModuleID
WHERE StartTime > GETDATE()
UNION
SELECT M.CourseID, MS.ModuleID, MS.StartTime, MS.EndTime
FROM ModulesSync MS JOIN Modules M
ON MS.ModuleID = M.ModuleID
WHERE StartTime > GETDATE()
UNION
SELECT M.CourseID, MAS.ModuleID, C.StartDate, C.EndDate
FROM ModulesAsync MAS JOIN Modules M
ON MAS.ModuleID = M.ModuleID
JOIN Courses C ON C.CourseID = M.CourseID
WHERE StartDate > GETDATE()

```

## 29. Osoby zapisane na przyszłe kursy (Karol)

```

CREATE VIEW FutureCoursesParticipants AS

```

```

SELECT    CP.ParticipantID,    CP.CourseID,    FC.ModuleID,    FC.StartTime,
FC.EndTime
FROM CourseParticipants CP
JOIN FutureCourses FC
ON CP.CourseID = FC.CourseID

```

### 30. Raport bilokacji dla kursów (Karol)

```

CREATE VIEW CoursesBilocationReport AS
SELECT  FCP1.ParticipantID,  U.FirstName,  U.LastName,  FCP1.ModuleID  AS
ModuleID1, --ewenrualnie courseID
FCP1.CourseID AS CourseID1, FCP1.StartTime AS StartTime1, FCP1.EndTime AS
EndTime1,
FCP2.ModuleID AS ModuleID2, FCP2.CourseID AS CourseID2, FCP2.StartTime AS
StartTime2, FCP2.EndTime AS EndTime2
FROM Users U
JOIN FutureCoursesParticipants FCP1
ON U.UserID = FCP1.ParticipantID
JOIN FutureCoursesParticipants FCP2
ON U.UserID = FCP2.ParticipantID
WHERE FCP1.ModuleID <> FCP2.ModuleID AND
NOT(FCP1.EndTime <= FCP2.StartTime OR FCP2.EndTime <= FCP1.StartTime)
AND FCP1.ModuleID < FCP2.ModuleID
AND FCP2.ModuleID NOT IN (SELECT ModuleID FROM ModulesAsync)
AND FCP1.ModuleID NOT IN (SELECT ModuleID FROM ModulesAsync)

```

### 31. Lista przyszłych studiów (Karol)

```

CREATE VIEW FutureStudies AS
SELECT S.StudyID, C.SessionID, CP.ClassID, CP.StartTime, CP.EndTime
FROM ClassesInPerson CP JOIN Classes C
ON CP.ClassID = C.ClassID
JOIN Sessions S ON S.SessionID = C.SessionID
WHERE StartTime > GETDATE()
UNION
SELECT S.StudyID, C.SessionID, CS.ClassID, CS.StartTime, CS.EndTime
FROM ClassesSync CS JOIN Classes C
ON CS.ClassID = C.ClassID
JOIN Sessions S ON S.SessionID = C.SessionID
WHERE StartTime > GETDATE()

```

### 32. Studenci zapisani na przyszłe zajęcia (Karol)

```

CREATE VIEW FutureClassesParticipants AS
SELECT  SP.StudentID,  SP.SessionID,  C.ClassID,  CP.StartTime,  CP.EndTime
FROM SessionPayments SP

```

```

JOIN Sessions S ON S.SessionID = SP.SessionID
JOIN Classes C ON S.SessionID = C.SessionID
JOIN ClassesInPerson CP ON CP.ClassID = C.ClassID
WHERE S.StartDate > GETDATE()
UNION
SELECT SP.StudentID, SP.SessionID, C.ClassID, CS.StartTime, CS.EndTime
FROM SessionPayments SP
JOIN Sessions S ON S.SessionID = SP.SessionID
JOIN Classes C ON S.SessionID = C.SessionID
JOIN ClassesSync CS ON CS.ClassID = C.ClassID
WHERE S.StartDate > GETDATE()
UNION
SELECT SP.StudentID, SP.SessionID, C.ClassID, NULL AS StartTime, NULL AS
EndTime FROM SessionPayments SP
JOIN Sessions S ON S.SessionID = SP.SessionID
JOIN Classes C ON S.SessionID = C.SessionID
JOIN ClassesAsync CA ON CA.ClassID = C.ClassID
WHERE S.StartDate > GETDATE()

```

### 33. Studenci zapisani na przyszłe studia (Karol)

```

CREATE VIEW FutureStudiesParticipants AS
SELECT FCP.StudentID, ST.StudyID, ST.StartDate, ST.EndDate FROM
FutureClassesParticipants FCP JOIN Sessions S
ON FCP.SessionID = S.SessionID
JOIN Studies ST ON ST.StudyID = S.StudyID

```

### 34. Raport bilokacji dla zajęć (Karol)

```

CREATE VIEW ClassesBilocationReport AS
SELECT FCP1.StudentID, U.FirstName, U.LastName, FCP1.ClassID AS ClassID1,
FCP1.StartTime AS StartDate1, FCP1.EndTime AS EndTime1,
FCP2.ClassID AS ClassID2, FCP2.StartTime AS StartTime2, FCP2.EndTime AS
EndTime2
FROM Users U
JOIN FutureClassesParticipants FCP1
ON U.UserID = FCP1.StudentID
JOIN FutureClassesParticipants FCP2
ON U.UserID = FCP2.StudentID
WHERE FCP1.ClassID <> FCP2.ClassID AND
NOT(FCP1.EndTime <= FCP2.StartTime OR FCP2.EndTime <= FCP1.StartTime)
AND FCP1.ClassID < FCP2.ClassID
AND FCP2.ClassID NOT IN (SELECT ClassID FROM ClassesAsync)
AND FCP1.ClassID NOT IN (SELECT ClassID FROM ClassesAsync)

```

### 35. Wyświetlanie zawartości koszyków w bardziej przystępny sposób (Karol)

```
CREATE VIEW DisplayCart AS
SELECT C.UserID, U.FirstName, U.LastName, C.ProductID, P.ProductName,
W.WebinarName AS [Name]
FROM Cart C
JOIN Users U ON C.UserID = U.UserID
JOIN ProductType P ON C.ProductTypeID = P.TypeID
JOIN Webinars W ON W.WebinarID = C.ProductID
WHERE ProductTypeID = 3
UNION
SELECT C.UserID, U.FirstName, U.LastName, C.ProductID, P.ProductName,
S.StudyName AS [Name]
FROM Cart C
JOIN Users U ON C.UserID = U.UserID
JOIN ProductType P ON C.ProductTypeID = P.TypeID
JOIN Studies S ON S.StudyID = C.ProductID
WHERE ProductTypeID = 2
UNION
SELECT C.UserID, U.FirstName, U.LastName, C.ProductID, P.ProductName,
Co.CourseName AS [Name]
FROM Cart C
JOIN Users U ON C.UserID = U.UserID
JOIN ProductType P ON C.ProductTypeID = P.TypeID
JOIN Courses Co ON Co.CourseID = C.ProductID
WHERE ProductTypeID = 1
UNION
SELECT C.UserID, U.FirstName, U.LastName, C.ProductID, P.ProductName, NULL
AS [Name]
FROM Cart C
JOIN Users U ON C.UserID = U.UserID
JOIN ProductType P ON C.ProductTypeID = P.TypeID
JOIN Sessions S ON S.SessionID = C.ProductID
WHERE ProductTypeID = 4
```

## FUNKCJE

1. Funkcja sprawdzająca czy nie ma sytuacji, że dwa zajęcia rozpoczną się w tym samym czasie - bo w sali nie może być dwóch zajęć na raz (Kinga)

```

CREATE FUNCTION CheckTimeOverlapClasses(@LocationID INT, @StartTime
DATETIME, @EndTime DATETIME, @ClassID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @Overlap BIT = 0;

    IF EXISTS (
        SELECT 1
        FROM ClassesInPerson c1
        WHERE c1.LocationID = @LocationID
            AND c1.ClassID <> @ClassID
            AND ((@StartTime BETWEEN c1.StartTime AND c1.EndTime) OR
(@EndTime BETWEEN c1.StartTime AND c1.EndTime))
        )
        SET @Overlap = 1;

    RETURN @Overlap;
END;

```

2. Funkcja do tabeli Diplomas - sprawdza czy dany identyfikator produktu jest prawidłowy dla określonego typu produktu

```

CREATE FUNCTION dbo.ValidateProductIDForDiplomas(
    @ProductTypeID INT,
    @ProductID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsValid BIT = 0;

    IF (@ProductTypeID = 1 AND EXISTS (SELECT 1 FROM Courses WHERE
CourseID = @ProductID))
        SET @IsValid = 1;
    ELSE IF (@ProductTypeID = 2 AND EXISTS (SELECT 1 FROM Studies WHERE
StudyID = @ProductID))
        SET @IsValid = 1;
    RETURN @IsValid;
END;
GO

```

3. Funkcja sprawdzająca czy nie ma sytuacji, że dwa moduły rozpoczną się w tym samym czasie - bo w sali nie może być dwóch modułów na raz (Kinga)

```

CREATE FUNCTION CheckTimeOverlapModules(@LocationID INT, @StartTime
DATETIME, @EndTime DATETIME, @ModuleID INT)

```

```

RETURNS BIT
AS
BEGIN
    DECLARE @Overlap BIT = 0;

    IF EXISTS (
        SELECT 1
        FROM ModulesInPerson m1
        WHERE m1.LocationID = @LocationID
            AND m1.ModuleID <> @ModuleID
            AND ((@StartTime BETWEEN m1.StartTime AND m1.EndTime) OR
                (@EndTime BETWEEN m1.StartTime AND m1.EndTime))
    )
        SET @Overlap = 1;

    RETURN @Overlap;
END;

```

4. Funkcja dla zamówienia do sprawdzania, jaki typ produktu został kupiony - bo dopuszczamy sytuację, że jakiś Webinar ma ID 1 i jakiś Kurs też ma ID 1, więc aby je rozróżnić potrzebujemy TypeID (Kinga)

```

CREATE FUNCTION dbo.ValidateProductIDForOrders(
    @ProductTypeID INT,
    @ProductID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsValid BIT = 0;

    IF (@ProductTypeID = 1 AND EXISTS (SELECT 1 FROM Courses WHERE
        CourseID = @ProductID))
        SET @IsValid = 1;
    ELSE IF (@ProductTypeID = 2 AND EXISTS (SELECT 1 FROM Studies WHERE
        StudyID = @ProductID))
        SET @IsValid = 1;
    ELSE IF (@ProductTypeID = 3 AND EXISTS (SELECT 1 FROM Webinar WHERE
        WebinarID = @ProductID))
        SET @IsValid = 1;
    ELSE IF (@ProductTypeID = 4 AND EXISTS (SELECT 1 FROM Sessions WHERE
        SessionID = @ProductID))
        SET @IsValid = 1;

    RETURN @IsValid;
END;

```

GO

5. Analogicznie do 4, funkcja dla płatności do sprawdzania, jaki typ produktu został opłacony - bo dopuszczamy sytuację, że jakiś Webinar ma ID 1 i jakiś Kurs też ma ID 1, więc aby je rozróżnić potrzebujemy TypeID (Kinga)

```
CREATE FUNCTION dbo.ValidateProductIDforPayment(
    @ProductTypeID INT,
    @ProductID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsValid BIT = 0;

    IF (@ProductTypeID = 1 AND EXISTS (SELECT 1 FROM Courses WHERE
    CourseID = @ProductID))
        SET @IsValid = 1;
    ELSE IF (@ProductTypeID = 2 AND EXISTS (SELECT 1 FROM Studies WHERE
    StudyID = @ProductID))
        SET @IsValid = 1;
    ELSE IF (@ProductTypeID = 3 AND EXISTS (SELECT 1 FROM Webinar WHERE
    WebinarID = @ProductID))
        SET @IsValid = 1;
    ELSE IF (@ProductTypeID = 4 AND EXISTS (SELECT 1 FROM Sessions WHERE
    SessionID = @ProductID))
        SET @IsValid = 1;
    RETURN @IsValid;
END;
GO
```

6. Sprawdzanie, jaki jest postęp ukończenia kursu przez danego studenta (Kinga)

```
CREATE FUNCTION CalculateStudentCourseProgressAndReturnMessage(
    @StudentID INT,
    @CourseID INT
)
RETURNS NVARCHAR(200)
AS
BEGIN
    DECLARE @TotalModules INT;
    DECLARE @CompletedModules INT;
```

```

        SELECT @TotalModules = COUNT(*) FROM Modules WHERE CourseID =
@CourseID;

```

```

        SELECT @CompletedModules = COUNT(*) FROM ModulesAttendances A
        INNER JOIN Modules M ON A.ModuleID = M.ModuleID
        WHERE M.CourseID = @CourseID AND A.StudentID = @StudentID AND
A.Present = 1;

```

```

        DECLARE @Progress DECIMAL(5, 2);
        SET @Progress = CASE WHEN @TotalModules > 0 THEN (@CompletedModules *
100.0 / @TotalModules) ELSE 0 END;

```

```

        RETURN 'Postęp zaliczania modułów dla studenta o ID ' + CAST(@StudentID
AS NVARCHAR(10)) + ' w kursie o ID ' + CAST(@CourseID AS NVARCHAR(10)) + '
wynosi ' + CONVERT(NVARCHAR(10), @Progress) + '%.';
END;

```

## 7. Sprawdzanie postępu kursu w oparciu o przeprowadzone do tej pory moduły (Kinga)

```

CREATE FUNCTION CalculateCourseProgress(
    @CourseID INT
)
RETURNS NVARCHAR(100)
AS
BEGIN

```

```

    DECLARE @TotalModules INT;
    DECLARE @CompletedModules INT;

```

```

        SELECT @TotalModules = COUNT(*) FROM Modules WHERE CourseID =
@CourseID;

```

```

        SELECT @CompletedModules = COUNT(*) FROM Modules AS M
        WHERE M.CourseID = @CourseID
        AND (EXISTS ( -- moduły zakończone w ModulesInPerson
            SELECT 1 FROM ModulesInPerson AS MP
            WHERE MP.ModuleID = M.ModuleID AND MP.EndTime <= GETDATE()
        )
        OR EXISTS ( -- moduły zakończone w ModulesSync
            SELECT 1 FROM ModulesSync AS MS
            WHERE MS.ModuleID = M.ModuleID AND MS.EndTime <= GETDATE()
        )
        OR EXISTS ( -- moduły zakończone w ModulesAsync
            SELECT 1 FROM ModulesAsync AS MA

```



```

        WHERE MA.ModuleID = M.ModuleID AND (SELECT EndDate FROM
Courses WHERE CourseID = @CourseID) <= GETDATE()
    )
);

DECLARE @Progress DECIMAL(5, 2);
SET @Progress = CASE WHEN @TotalModules > 0 THEN (@CompletedModules *
100.0 / @TotalModules) ELSE 0 END;

DECLARE @Message NVARCHAR(100);
SET @Message = 'Postęp kursu ' + CAST(@CourseID AS NVARCHAR(10)) + ' w
oparciu o przeprowadzone moduły wynosi ' + CAST(@Progress AS NVARCHAR(10))
+ '%.';
RETURN @Message;
END;

DECLARE @CourseID INT = 1;

DECLARE @ProgressMessage NVARCHAR(100);
SET @ProgressMessage = dbo.CalculateCourseProgress(@CourseID);

PRINT @ProgressMessage;

```

## 8. Sprawdzanie statusu płatności i odpowiedź zwrotna do użytkownika (Paulina)

```

CREATE FUNCTION CheckPaymentStatus (
    @UserID INT,
    @ProductID INT,
    @ProductType INT
)
RETURNS NVARCHAR(50)
AS
BEGIN
    DECLARE @Message NVARCHAR(50);

    IF EXISTS (SELECT 1 FROM Logins WHERE UserID = @UserID)
    BEGIN
        IF @ProductType = 1
        BEGIN
            SELECT @Message = 'Status płatności: ' +
CASE WHEN Paid = 1 THEN 'Opłacony' ELSE
'Nieopłacony' END
FROM CoursePayments
WHERE CourseID = @ProductID AND StudentID = @UserID;

```

```

END
ELSE IF @ProductType = 3
BEGIN
    SELECT @Message = 'Status płatności: ' +
        CASE WHEN Paid = 1 THEN 'Opłacony' ELSE
'Nieopłacony' END
    FROM WebinarPayments
    WHERE WebinarID = @ProductID AND ParticipantID =
@UserID;
END
ELSE IF @ProductType = 4
BEGIN
    SELECT @Message = 'Status płatności: ' +
        CASE WHEN Paid = 1 THEN 'Opłacony' ELSE
'Nieopłacony' END
    FROM SessionPayments
    WHERE SessionID = @ProductID AND StudentID = @UserID;
END
ELSE
BEGIN
    SET @Message = 'Nieprawidłowy typ produktu.';
END
END
ELSE
BEGIN
    SET @Message = 'Użytkownik o podanym ID nie posiada konta.';
END
END

RETURN @Message;
END;
-Test

```

```

SELECT dbo.CheckPaymentStatus(11, 8, 4) AS 'PaymentStatus';
SELECT dbo.CheckPaymentStatus(11, 24, 4) AS 'PaymentStatus';
SELECT dbo.CheckPaymentStatus(14, 2, 3) AS 'PaymentStatus';
SELECT dbo.CheckPaymentStatus(15, 2, 3) AS 'PaymentStatus';
SELECT dbo.CheckPaymentStatus(3, 3, 1) AS 'PaymentStatus';
SELECT dbo.CheckPaymentStatus(4, 3, 1) AS 'PaymentStatus';

```

## 9. Podliczenie wartości koszyka dla podanego użytkownika (Karol)

```

CREATE FUNCTION GetCartValue (
    @UserID INT
)
RETURNS MONEY
AS

```

```

BEGIN
    DECLARE @SummedPrice MONEY;
    IF EXISTS (SELECT 1 FROM Cart WHERE UserID = @UserID)
    BEGIN
        SELECT @SummedPrice = SUM(ObligatoryPay) FROM
DisplayCart WHERE UserID = @UserID
    END
    ELSE
    BEGIN
        SET @SummedPrice = 0
    END
    RETURN @SummedPrice
END

-Test
DECLARE @UserIDTest INT = 29;
SELECT dbo.GetCartValue(@UserIDTest) AS CartValue;

```

## PROCEDUREY

### 1. Zakładanie konta (Paulina)

```

CREATE PROCEDURE CreateAccount (
    @UserID INT,
    @UserName NVARCHAR(20),
    @Password NVARCHAR(20)
)
AS
BEGIN
    -- Sprawdź, czy użytkownik istnieje w tabeli Users
    IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
    BEGIN
        -- Sprawdź, czy użytkownik już ma konto w tabeli Logins
        IF NOT EXISTS (SELECT 1 FROM Logins WHERE UserID = @UserID)
        BEGIN
            -- Dodaj nowe konto do tabeli Logins
            INSERT INTO Logins (UserID, UserName, Password)
            VALUES (@UserID, @UserName, @Password)

            PRINT 'Konto zostało pomyślnie utworzone.'
        END
        ELSE
        BEGIN
            -- Jeżeli konto już istnieje
            PRINT 'Użytkownik ma już konto.'
        END
    END

```

```

        END
    ELSE
    BEGIN
        -- Jeżeli użytkownik nie istnieje w tabeli Users
        PRINT 'Użytkownik nie istnieje.'
    END
END;

--Test
SELECT * FROM Logins

EXEC CreateAccount @UserID = 29, @UserName = 'NowyUzytkownik',
@Password = 'Haslo123';
SELECT * FROM Logins

```

## 2. Usuwanie konta (Paulina)

```

CREATE PROCEDURE DeleteAccount (
    @UserID INT
)
AS
BEGIN
    -- Sprawdź, czy użytkownik ma konto w tabeli Logins
    IF EXISTS (SELECT 1 FROM Logins WHERE UserID = @UserID)
    BEGIN
        -- Usuń konto z tabeli Logins
        DELETE FROM Logins WHERE UserID = @UserID

        PRINT 'Konto zostało pomyślnie usunięte.'
    END
    ELSE
    BEGIN
        -- Jeżeli użytkownik nie ma konta, zwróć odpowiedni
        komunikat
        PRINT 'Użytkownik nie ma przypisanego konta.'
    END
END;

--Test
SELECT * FROM Logins
EXEC DeleteAccount @UserID = 29;
SELECT * FROM Logins

```

## 3. Deaktywacja linków do webinarów po upływie 30 dni (Paulina)

```

CREATE PROCEDURE DeactivateWebinarLinks
AS
BEGIN
    UPDATE WebinarPayments
    SET IsLinkActive = 0
    FROM WebinarPayments wp
    INNER JOIN Webinars w ON wp.WebinarID = w.WebinarID
    WHERE wp.IsLinkActive = 1
        AND w.[Date] <= DATEADD(DAY, -30, GETDATE()); -- Dezaktywacja po
        upływie 30 dni

    IF @@ROWCOUNT > 0
    BEGIN
        PRINT 'Linki zostały dezaktywowane.';
    END
    ELSE
    BEGIN
        PRINT 'Brak linków do dezaktywacji.';
    END
END;

--Test
SELECT * FROM Webinars
UPDATE WebinarPayments
SET IsLinkActive = 1
WHERE WebinarID IN (13)
SELECT * FROM Webinars

EXEC DeactivateWebinarLinks;

```

4. Aktywacja linku do przyszłego webinaru jeśli ktoś ma konto oraz go opłacił.(Paulina)

```

CREATE PROCEDURE ActivateWebinarLink (
    @WebinarID INT
)
AS
BEGIN
    -- Aktualizuj IsLinkActive, jeśli spełnione są warunki
    UPDATE WebinarPayments
    SET IsLinkActive = 1
    FROM WebinarPayments wp
    INNER JOIN Webinars w ON wp.WebinarID = w.WebinarID
    WHERE wp.WebinarID = @WebinarID
        AND wp.ParticipantID IN (SELECT UserID FROM Logins)

```

```

        AND wp.Paid = 1
        AND wp.IsLinkActive = 0
        AND w.[Date] > GETDATE());

-- Sprawdź, czy wykonano aktualizację
IF @@ROWCOUNT > 0
BEGIN
    PRINT 'Linki zostały aktywowane.';
END
END;

```

```

--Test
SELECT * FROM WebinarPayments

```

```

UPDATE WebinarPayments
SET IsLinkActive = 0
WHERE WebinarID IN (2)

```

```

SELECT * FROM WebinarPayments
EXEC ActivateWebinarLink @WebinarID = 2;
SELECT * FROM WebinarPayments

```

## 5. Skasowanie webinaru. (Paulina)

```

CREATE PROCEDURE CancelWebinar (
    @WebinarID INT
)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM TranslatedWebinars WHERE WebinarID =
@WebinarID)
    BEGIN
        DELETE FROM TranslatedWebinars WHERE WebinarID = @WebinarID;
    END
    -- Sprawdź, czy istnieją powiązane wpisy w tabeli OrderDetails
    IF EXISTS (SELECT 1 FROM OrderDetails WHERE ProductTypeID = 3 AND
ProductID = @WebinarID)
    BEGIN
        -- Usuń powiązane wpisy z koszyka
        DELETE FROM OrderDetails WHERE ProductTypeID = 3 AND ProductID =
@WebinarID;

        -- Usuń związane płatności z tabeli WebinarPayments
        DELETE FROM WebinarPayments WHERE WebinarID = @WebinarID;

        -- Wyślij powiadomienie do użytkowników
    
```

```

        PRINT 'Anulowano webinar o ID ' + CAST(@WebinarID AS NVARCHAR(10))
+ ' z powodu decyzji administratora.';
    END

    -- Usuń webinar z tabeli Webinars
    DELETE FROM Webinars WHERE WebinarID = @WebinarID;

    -- Wyślij dodatkowe powiadomienie, jeśli to konieczne
    PRINT 'Webinar o ID ' + CAST(@WebinarID AS NVARCHAR(10)) + ' został
anulowany.';
END;

--Test
--Webinar bez powiazan
SELECT * FROM Webinars
EXEC CancelWebinar 20;
SELECT * FROM Webinars

--Webinar z powiazaniami
SELECT * FROM Webinars
EXEC CancelWebinar 17;
SELECT * FROM Webinars

--najlepiej przywrócić dane przez usunięcie bazy i postawienie na nowo.

```

## 5v2. Dodawanie webinaru z odpowiednim tłumaczem.(Paulina)

```

CREATE PROCEDURE AddWebinar
    @WebinarName NVARCHAR(60),
    @TranslatorID INT,
    @RecordingLink NVARCHAR(50),
    @Date DATE,
    @Price MONEY,
    @DurationInMinutes INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @IsValidTranslator BIT;

    -- Sprawdź czy podany tłumacz istnieje
    SELECT @IsValidTranslator = CASE WHEN EXISTS (SELECT 1 FROM
Translators WHERE TranslatorID = @TranslatorID) THEN 1 ELSE 0 END;

    IF @IsValidTranslator = 1

```

```

BEGIN
    -- Dodaj webinar
    INSERT INTO Webinars (WebinarName, RecordingLink, [Date], Price,
DurationInMinutes)
        VALUES (@WebinarName, @RecordingLink, @Date, @Price,
@DurationInMinutes);

    -- Pobierz ID dodanego webinaru
    DECLARE @WebinarID INT;
    SET @WebinarID = SCOPE_IDENTITY();

    -- Dodaj informację o tłumaczu dla tego webinaru
    INSERT INTO TranslatedWebinars (WebinarID, TranslatorID)
    VALUES (@WebinarID, @TranslatorID);
END
ELSE
BEGIN
    PRINT 'Błąd: Nieprawidłowy tłumacz.';
END;
END;

```

-Test

```

SELECT * FROM Webinars
SELECT * FROM Translators

```

EXEC AddWebinar

```

@WebinarName = 'TestowyWebinar',
@TranslatorID = 5,
@RecordingLink = 'http://example.com/webinar',
@Date = '2024-01-18',
@Price = 19.99,
@DurationInMinutes = 90;

```

```

SELECT * FROM TranslatedWebinars
SELECT * FROM Webinars

```

6. Czy produkt został w pełni opłacony webinar/zjazd/kurs?  
(Paulina)

```

CREATE PROCEDURE PayForProduct (
    @UserID INT,
    @ProductID INT,
    @ProductType INT
)

```



```

AS
BEGIN
    -- Sprawdzenie czy użytkownik istnieje w tabeli Logins
    IF EXISTS (SELECT 1 FROM Logins WHERE UserID = @UserID)
    BEGIN
        IF @ProductType = 3 -- Webinar
        BEGIN
            UPDATE WebinarPayments
            SET Paid = 1
            WHERE WebinarID = @ProductID
            AND ParticipantID = @UserID;
        END
        ELSE IF @ProductType = 4 -- Zjazd
        BEGIN
            UPDATE SessionPayments
            SET Paid = 1
            WHERE SessionID = @ProductID
            AND StudentID = @UserID;
        END
        ELSE IF @ProductType = 1 -- Kurs
        BEGIN
            UPDATE CoursePayments
            SET Paid = 1
            WHERE CourseID = @ProductID
            AND StudentID = @UserID;
        END

        PRINT 'Produkt został w pełni opłacony.';
    END
    ELSE
    BEGIN
        PRINT 'Użytkownik o podanym ID nie posiada konta.';
    END
END;

```

```

--Test
UPDATE CoursePayments
SET Paid = 0
WHERE CourseID IN (13) AND StudentID = 10

```

```

EXEC PayForProduct 10, 13, 1

```

```

UPDATE WebinarPayments
SET Paid = 0
WHERE WebinarID IN (13) AND ParticipantID = 18

```

```
EXEC PayForProduct 18, 13, 3
```

```
UPDATE SessionPayments  
SET Paid = 0  
WHERE SessionID IN (39) AND StudentID = 13
```

```
EXEC PayForProduct 13, 39, 4
```

## 7. Zmiana danych użytkowników w tabeli Users (Kinga)

```
CREATE PROCEDURE UpdateUserData
```

```
    @UserID INT,  
    @NewFirstName NVARCHAR(20) = NULL,  
    @NewLastName NVARCHAR(20) = NULL,  
    @NewNationality NVARCHAR(10) = NULL,  
    @NewPhone NVARCHAR(12) = NULL,  
    @NewEmail NVARCHAR(30) = NULL
```

```
AS
```

```
BEGIN
```

```
    -- sprawdzenie czy istnieje użytkownik o podanym UserID
```

```
    IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
```

```
    BEGIN
```

```
        UPDATE Users
```

```
        SET
```

```
            FirstName = ISNULL(@NewFirstName, FirstName),
```

```
            LastName = ISNULL(@NewLastName, LastName),
```

```
            Nationality = ISNULL(@NewNationality, Nationality),
```

```
            Phone = ISNULL(@NewPhone, Phone),
```

```
            Email = ISNULL(@NewEmail, Email)
```

```
        WHERE UserID = @UserID;
```

```
    END
```

```
    ELSE
```

```
    BEGIN
```

```
        PRINT 'Użytkownik o podanym UserID nie istnieje.';
```

```
    END
```

```
END;
```

```
EXEC UpdateUserData @UserID = 1, @NewFirstName = 'Michał', @NewLastName =  
'Ziomek';
```

```
-- aktualizacja tylko numeru telefonu użytkownika o ID 2
```

```
EXEC UpdateUserData @UserID = 2, @NewPhone = '48521134453';
```

## 8. Zmiana hasła dla użytkownika o danym UserName w tabeli Logins (Kinga)

```
CREATE PROCEDURE ChangeUserPassword
```

```

        @UserName NVARCHAR(20),
        @NewPassword NVARCHAR(20)
AS
BEGIN
    -- sprawdzenie czy istnieje użytkownik o podanym UserName
    IF EXISTS (SELECT 1 FROM Logins WHERE UserName = @UserName)
    BEGIN
        UPDATE Logins
        SET Password = @NewPassword
        WHERE UserName = @UserName;

        PRINT 'Hasło zostało pomyślnie zmienione.';
    END
    ELSE
    BEGIN
        PRINT 'Użytkownik o podanym UserName nie istnieje.';
    END
END;

EXEC dbo.ChangeUserPassword @UserName = 'JanKowalski', @NewPassword =
'hejJan255'

```

## 9. Sprawdzanie, jakie lokalizacje są wolne w danym czasie (Kinga)

```

CREATE PROCEDURE CheckLocationAvailability
    @StartTime DATETIME,
    @EndTime DATETIME
AS
BEGIN
    SELECT LocationID
    FROM Locations
    WHERE LocationID NOT IN (
        -- Lokalizacje zajęte przez ClassInPerson w danym czasie
        SELECT DISTINCT CIP.LocationID
        FROM ClassesInPerson CIP
        WHERE @StartTime < CIP.EndTime AND @EndTime > CIP.StartTime
        UNION
        -- Lokalizacje zajęte przez ModuleInPerson w danym czasie
        SELECT DISTINCT MIP.LocationID
        FROM ModulesInPerson MIP
        INNER JOIN Modules M ON MIP.ModuleID = M.ModuleID
        WHERE @StartTime < MIP.EndTime AND @EndTime > MIP.StartTime
    );
END;

```

```
DECLARE @StartTime DATETIME = '2023-03-01 9:30:00';
```

```
DECLARE @EndTime DATETIME = '2023-03-01 12:30:00';
```

```
EXEC CheckLocationAvailability @StartTime, @EndTime;
```

10. Sprawdzanie, czy jakakolwiek lokalizacja jest dostępna w danym czasie, jeśli tak zwraca jedną z nich (Kinga)

```
CREATE PROCEDURE CheckIfAnyLocationIsAvailable
```

```
    @StartTime DATETIME,
```

```
    @EndTime DATETIME,
```

```
    @LocationID INT OUTPUT,
```

```
    @LocationAvailable BIT OUTPUT
```

```
AS
```

```
BEGIN
```

```
    SELECT TOP 1 @LocationID = LocationID
```

```
    FROM Locations
```

```
    WHERE LocationID NOT IN (
```

```
        SELECT DISTINCT CIP.LocationID
```

```
        FROM ClassesInPerson CIP
```

```
        WHERE @StartTime < CIP.EndTime AND @EndTime > CIP.StartTime
```

```
        UNION
```

```
        SELECT DISTINCT MIP.LocationID
```

```
        FROM ModulesInPerson MIP
```

```
        INNER JOIN Modules M ON MIP.ModuleID = M.ModuleID
```

```
        WHERE @StartTime < MIP.EndTime AND @EndTime > MIP.StartTime
```

```
    );
```

```
    SET @LocationAvailable = CASE WHEN @LocationID IS NOT NULL THEN 1 ELSE  
0 END;
```

```
END;
```

11. Sprawdzanie obecności uczestników na zajęciach - Classes (Kinga)

```
CREATE PROCEDURE CheckAttendance(
```

```
    @ClassID INT,
```

```
    @StudentID INT,
```

```
    @IsPresent BIT
```

```
)
```

```
AS
```

```
BEGIN
```

```
    -- Sprawdź, czy klasa istnieje
```

```
    IF NOT EXISTS (SELECT 1 FROM Classes WHERE ClassID = @ClassID)
```

```

BEGIN
    -- Obsługa nieistniejącej klasy
    PRINT 'Nieprawidłowa klasa.';
    RETURN;
END;

-- Sprawdź, czy student istnieje
IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @StudentID)
BEGIN
    -- Obsługa nieistniejącego studenta
    PRINT 'Nieprawidłowy student.';
    RETURN;
END;

-- Sprawdź, czy uczestnik już jest zarejestrowany w ClassAttendances
IF NOT EXISTS (SELECT 1 FROM ClassesAttendances WHERE ClassID =
@ClassID AND StudentID = @StudentID)
BEGIN
    -- Dodaj nowy wpis do ClassesAttendances, jeśli jeszcze nie
istnieje
    INSERT INTO ClassesAttendances (ClassID, StudentID, Present)
    VALUES (@ClassID, @StudentID, @IsPresent);

    PRINT 'Uczestnik zarejestrowany na lekcji.';
END
ELSE
BEGIN
    -- Zaktualizuj obecność uczestnika
    UPDATE ClassesAttendances
    SET Present = @IsPresent
    WHERE ClassID = @ClassID AND StudentID = @StudentID;

    PRINT 'Obecność zaktualizowana.';
END;
END;

EXEC CheckAttendance @ClassID = 11, @StudentID = 18, @IsPresent = 0;
EXEC CheckAttendance @ClassID = 11, @StudentID = 18, @IsPresent = 1;
EXEC CheckAttendance @ClassID = 1111, @StudentID = 18, @IsPresent = 1; --
nieistniejąca klasa
EXEC CheckAttendance @ClassID = 11, @StudentID = 1811, @IsPresent = 1; --
nieistniejący student
EXEC CheckAttendance @ClassID = 55, @StudentID = 16, @IsPresent = 1; --
wpis jeszcze nie istnieje

```

## 12. Analogicznie do procedury 7, sprawdzanie obecności uczestników na modułach - Modules (Kinga)

```
CREATE PROCEDURE CheckModuleAttendance(
    @ModuleID INT,
    @StudentID INT,
    @IsPresent BIT
)
AS
BEGIN
    -- Sprawdź, czy moduł istnieje
    IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
    BEGIN
        -- Obsługa nieistniejącego modułu
        PRINT 'Nieprawidłowy moduł.';
        RETURN;
    END;

    -- Sprawdź, czy student istnieje
    IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @StudentID)
    BEGIN
        -- Obsługa nieistniejącego studenta
        PRINT 'Nieprawidłowy student.';
        RETURN;
    END;

    -- Sprawdź, czy uczestnik już jest zarejestrowany w ModulesAttendances
    IF NOT EXISTS (SELECT 1 FROM ModulesAttendances WHERE ModuleID =
@ModuleID AND StudentID = @StudentID)
    BEGIN
        -- Dodaj nowy wpis do ModulesAttendances, jeśli jeszcze nie
istnieje
        INSERT INTO ModulesAttendances (ModuleID, StudentID, Present)
        VALUES (@ModuleID, @StudentID, @IsPresent);

        PRINT 'Uczestnik zarejestrowany na module.';
    END
    ELSE
    BEGIN
        -- Zaktualizuj obecność uczestnika
        UPDATE ModulesAttendances
        SET Present = @IsPresent
        WHERE ModuleID = @ModuleID AND StudentID = @StudentID;

        PRINT 'Obecność zaktualizowana.';
    END;
END;
```

END;

### 13. Sprawdzanie, czy sala jest dostępna w danym przedziale czasowym (Kinga)

```
CREATE PROCEDURE CheckRoomAvailability (  
    @LocationID INT,  
    @StartTime DATETIME,  
    @EndTime DATETIME  
)  
AS  
BEGIN  
    -- Sprawdź, czy sala istnieje  
    IF NOT EXISTS (SELECT 1 FROM Locations WHERE LocationID = @LocationID)  
    BEGIN  
        -- Obsługa nieistniejącej sali  
        PRINT 'Nieprawidłowa sala.';  
        RETURN;  
    END;  
  
    -- Sprawdź dostępność sali w podanym przedziale czasowym dla zajęć  
    stacjonarnych  
    IF EXISTS (  
        SELECT 1  
        FROM ClassesInPerson  
        WHERE LocationID = @LocationID  
        AND @StartTime < EndTime  
        AND @EndTime > StartTime  
    )  
    BEGIN  
        -- Sala zajęta w tym czasie przez zajęcia stacjonarne  
        PRINT 'Sala jest zajęta w tym przedziale czasowym na zajęcia  
    stacjonarne.';  
        RETURN;  
    END;  
  
    -- Sprawdź dostępność sali w podanym przedziale czasowym dla modułów  
    stacjonarnych  
    IF EXISTS (  
        SELECT 1  
        FROM ModulesInPerson  
        WHERE LocationID = @LocationID  
        AND @StartTime < EndTime  
        AND @EndTime > StartTime  
    )  
    BEGIN
```

```

        -- Sala zajęta w tym czasie przez moduły stacjonarne
        PRINT 'Sala jest zajęta w tym przedziale czasowym przez moduły
stacjonarne.';
    END
    ELSE
    BEGIN
        -- Sala dostępna
        PRINT 'Sala jest dostępna w tym przedziale czasowym.';
    END;
END;

```

#### 14. Pobieranie adresu korespondencji dla każdego absolwenta studium/kursu (Kinga)

```

CREATE PROCEDURE GetAlumniAddresses
AS
BEGIN
    -- Deklaracja zmiennych
    DECLARE @StudentID INT;
    DECLARE @CityName NVARCHAR(30);
    DECLARE @CountryName NVARCHAR(20);
    DECLARE @Street NVARCHAR(50);
    DECLARE @BuildingNumber INT;
    DECLARE @FlatNumber INT;
    DECLARE @PostalCode NVARCHAR(10);

    -- Kursor do pobrania danych
    DECLARE AlumniCursor CURSOR FOR
    SELECT
        SA.StudentID,
        CIT.CityName,
        COU.CountryName,
        SA.Street,
        SA.BuildingNumber,
        SA.FlatNumber,
        SA.PostalCode
    FROM
        StudentsAddresses SA
        INNER JOIN Cities CIT ON SA.CityID = CIT.CityID
        INNER JOIN Countries COU ON CIT.CountryID = COU.CountryID;

    -- Otwarcie kursora
    OPEN AlumniCursor;

    -- Pętla przetwarzająca każdego absolwenta
    FETCH NEXT FROM AlumniCursor INTO

```



```

        @StudentID, @CityName, @CountryName, @Street, @BuildingNumber,
        @FlatNumber, @PostalCode;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Wyświetlenie adresu korespondencyjnego (możesz dostosować to do
        -- swoich potrzeb)
        PRINT 'Absolwent o ID ' + CAST(@StudentID AS NVARCHAR(10)) + ' ma
        adres korespondencyjny w mieście ' + @CityName + ', ' + @CountryName + ':
        '
            + @Street + ' ' + CAST(@BuildingNumber AS NVARCHAR(10))
            + CASE WHEN @FlatNumber IS NOT NULL THEN '/' +
            CAST(@FlatNumber AS NVARCHAR(10)) ELSE '' END
            + ', ' + @PostalCode;

        -- Pobranie kolejnego rekordu
        FETCH NEXT FROM AlumniCursor INTO
            @StudentID, @CityName, @CountryName, @Street, @BuildingNumber,
            @FlatNumber, @PostalCode;
        END;

        -- Zamknięcie kursora
        CLOSE AlumniCursor;
        DEALLOCATE AlumniCursor;
    END;

EXEC GetAlumniAddresses;

```

## 15. Ustawianie limitu miejsc dla kursu / studiów (Kinga)

```

CREATE PROCEDURE SetCapacityLimit(
    @ProductID INT,
    @TypeID INT,
    @CapacityLimit INT
)
AS
BEGIN
    DECLARE @IsCourse INT;
    DECLARE @IsStudy INT;

    -- Sprawdź w ProductType, czy typ produktu to kurs
    SELECT @IsCourse = COUNT(*) FROM ProductType WHERE TypeID = @TypeID
    AND ProductName = 'Course';

    -- Sprawdź w ProductType, czy typ produktu to studia

```

```
SELECT @IsStudy = COUNT(*) FROM ProductType WHERE TypeID = @TypeID AND
ProductName = 'Study';
```

```
IF @IsCourse > 0
BEGIN
    -- Aktualizuj limit miejsc dla kursu
    UPDATE Courses
    SET ParticipantsLimit = @CapacityLimit
    WHERE CourseID = @ProductID;

    PRINT 'Limit miejsc dla kursu został zaktualizowany.';
END
ELSE IF @IsStudy > 0
BEGIN
    -- Aktualizuj limit miejsc dla studiów
    UPDATE Studies
    SET StudentsLimit = @CapacityLimit
    WHERE StudyID = @ProductID;

    PRINT 'Limit miejsc dla studiów został zaktualizowany.';
END
ELSE
BEGIN
    PRINT 'Nieprawidłowy typ produktu.';
    RETURN;
END
END;
GO

DECLARE @ProductTypeStudy INT = 2;
DECLARE @StudyID INT = 4;
DECLARE @CapacityLimit INT = 30;

EXEC SetCapacityLimit @StudyID, @ProductTypeStudy, @CapacityLimit;

DECLARE @ProductTypeCourse INT = 1; -- Przykładowy identyfikator typu
produktu dla modułu
DECLARE @CourseID INT = 1; -- Przykładowy identyfikator modułu

EXEC SetCapacityLimit @CourseID, @ProductTypeCourse, @CapacityLimit;
```

## 16. Wyświetlanie programu (sylabusu) wybranych studiów (Kinga)

```
CREATE PROCEDURE DisplayStudyProgram(
    @StudyID INT
```

```

)
AS
BEGIN
    -- Sprawdź, czy istnieją studia o podanym ID
    IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudyID = @StudyID)
    BEGIN
        PRINT 'Studia o podanym ID nie istnieją.';
        RETURN;
    END;

    DECLARE @StudyName NVARCHAR(255);
    SELECT @StudyName = StudyName FROM Studies WHERE StudyID = @StudyID;

    PRINT 'Program studiów dla ' + @StudyName + ':';

    DECLARE @SyllabusID INT;

    -- Kursor do przechodzenia przez dostępne SyllabusID
    DECLARE SyllabusCursor CURSOR FOR
    SELECT SyllabusID FROM Syllabuses WHERE StudyID = @StudyID;

    OPEN SyllabusCursor;

    -- Przechodź przez dostępne SyllabusID
    FETCH NEXT FROM SyllabusCursor INTO @SyllabusID;

    -- Tabela tymczasowa do przechowywania wyników
    CREATE TABLE #StudyProgram (
        SubjectID INT,
        SubjectName NVARCHAR(255),
        Description NVARCHAR(MAX)
    );

    WHILE @@FETCH_STATUS = 0
    BEGIN
        INSERT INTO #StudyProgram
        SELECT
            S.SubjectID,
            S.SubjectName,
            S.Description
        FROM Subjects S
        INNER JOIN Syllabuses SS ON S.SubjectID = SS.SubjectID
        WHERE SS.SyllabusID = @SyllabusID;

        FETCH NEXT FROM SyllabusCursor INTO @SyllabusID;
    END;

```

```

-- Zamknij i usuń kursor
CLOSE SyllabusCursor;
DEALLOCATE SyllabusCursor;

SELECT * FROM #StudyProgram;

-- Usuń tabelę tymczasową
DROP TABLE #StudyProgram;

END;

DECLARE @StudyID INT = 3;

EXEC DisplayStudyProgram @StudyID;

SELECT * FROM Syllabuses

```

## 17. Wyświetlanie harmonogramu wybranego kursu (Kinga)

```

CREATE PROCEDURE GenerateCourseSchedule(
    @CourseID INT
)
AS
BEGIN
    -- Sprawdź, czy istnieje kurs o podanym ID
    IF NOT EXISTS (SELECT 1 FROM Courses WHERE CourseID = @CourseID)
    BEGIN
        PRINT 'Kurs o podanym ID nie istnieje.';
        RETURN;
    END;

    -- Deklaruj kursor do przechodzenia przez dostępne moduły
    DECLARE ModuleCursor CURSOR FOR
    SELECT  MO.ModuleID, A.ModuleType, A.LocationID,
            A.StartTime AS ModuleStartDate,
            A.EndTime AS ModuleEndDate
    FROM Modules MO
    JOIN (SELECT M.ModuleID, 'Stacjonarny' AS ModuleType,
                LocationID, StartTime, EndTime
         FROM Modules AS M
         JOIN ModulesInPerson AS MIP ON MIP.ModuleID = M.ModuleID

        UNION

        SELECT M.ModuleID, 'Online Synchroniczny' AS ModuleType,
        NULL AS LocationID, StartTime, EndTime

```

```

        FROM Modules AS M
        JOIN ModulesSync AS MS ON MS.ModuleID = M.ModuleID

        UNION

        SELECT      M.ModuleID,      'Online      Asynchroniczny'      AS
ModuleType,
        NULL AS LocationID, StartDate, EndDate
        FROM Modules AS M
        JOIN ModulesAsync AS MA ON MA.ModuleID = M.ModuleID
        JOIN Courses AS C ON M.CourseID = C.CourseID)
        AS A ON A.ModuleID = MO.ModuleID
WHERE MO.CourseID = @CourseID;

OPEN ModuleCursor;

-- Wyświetl nagłówek harmonogramu
PRINT 'Harmonogram kursu o ID ' + CAST(@CourseID AS NVARCHAR(10)) +
':';
PRINT '-----';
PRINT 'ModuleID | ModuleType | ModuleLocation | ModuleStartDate |
ModuleEndDate';

        DECLARE @ModuleID INT, @ModuleType NVARCHAR(50), @ModuleLocation
NVARCHAR(255), @ModuleStartDate DATETIME, @ModuleEndDate DATETIME;

        FETCH NEXT FROM ModuleCursor INTO @ModuleID, @ModuleType,
@ModuleLocation, @ModuleStartDate, @ModuleEndDate;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Wyświetl informacje o module
            PRINT CAST(@ModuleID AS NVARCHAR(10)) + ' | ' + @ModuleType + ' |
' + COALESCE(@ModuleLocation, 'Brak') + ' | ' + CAST(@ModuleStartDate AS
NVARCHAR(20)) + ' | ' + CAST(@ModuleEndDate AS NVARCHAR(20));

            FETCH NEXT FROM ModuleCursor INTO @ModuleID, @ModuleType,
@ModuleLocation, @ModuleStartDate, @ModuleEndDate;
            END;

            -- Zamknij i usuń kursor
            CLOSE ModuleCursor;
            DEALLOCATE ModuleCursor;

            PRINT '-----';
END;

```

```
DECLARE @CourseID INT = 3;
```

```
EXEC dbo.GenerateCourseSchedule @CourseID;
```

## 18. Wyświetlanie harmonogramu wybranych studiów (Kinga)

```
CREATE PROCEDURE GenerateStudySchedule(
    @StudyID INT
)
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Studies WHERE StudyID = @StudyID)
    BEGIN
        PRINT 'Studia o podanym ID nie istnieją.';
        RETURN;
    END;

    -- kursor do przechodzenia przez dostępne moduły
    DECLARE SessionCursor CURSOR FOR

    SELECT ClassID, ClassType, A.SessionID,
           LocationID, StartTime, EndTime
    FROM Studies AS ST
    JOIN Sessions AS SE ON ST.StudyID = SE.StudyID
    JOIN (SELECT C.ClassID, 'Stacjonarne' AS ClassType, SessionID,
              LocationID, StartTime, EndTime
         FROM Classes AS C
         JOIN ClassesInPerson AS CIP ON CIP.ClassID = C.ClassID

        UNION

        SELECT C.ClassID, 'Online Synchroniczne' AS ClassType,
        SessionID,
        NULL AS LocationID, StartTime, EndTime
        FROM Classes AS C
        JOIN ClassesSync AS CS ON CS.ClassID = C.ClassID

        UNION

        SELECT C.ClassID, 'Online Asynchroniczne' AS ClassType,
        C.SessionID,
        NULL AS LocationID, StartDate, EndDate
        FROM Classes AS C
        JOIN ClassesAsync AS CA ON CA.ClassID = C.ClassID
        JOIN Sessions AS S ON C.SessionID = S.SessionID)
    AS A ON SE.SessionID = A.SessionID
```

```

WHERE ST.StudyID = @StudyID;

OPEN SessionCursor;

-- Wyświetl nagłówek harmonogramu
PRINT 'Harmonogram studiów o ID ' + CAST(@StudyID AS NVARCHAR(10)) +
':';
PRINT '-----';
PRINT 'ClassID | ClassType | SessionID | ClassLocation |
ClassStartDate | ClassEndDate';

DECLARE @ClassID INT, @ClassType NVARCHAR(50), @SessionID INT,
@ClassLocation NVARCHAR(255), @ClassStartDate DATETIME, @ClassEndDate
DATETIME;

FETCH NEXT FROM SessionCursor INTO @ClassID, @ClassType, @SessionID,
@ClassLocation, @ClassStartDate, @ClassEndDate;

WHILE @@FETCH_STATUS = 0
BEGIN
    -- Wyświetl informacje o klasie
    PRINT CAST(@ClassID AS NVARCHAR(10)) + ' | ' + @ClassType + ' | '
+ CAST(@SessionID AS NVARCHAR(10)) + ' | ' + COALESCE(@ClassLocation,
'Brak') + ' | ' + CAST(@ClassStartDate AS NVARCHAR(20)) + ' | ' +
CAST(@ClassEndDate AS NVARCHAR(20));

    FETCH NEXT FROM SessionCursor INTO @ClassID, @ClassType,
@SessionID, @ClassLocation, @ClassStartDate, @ClassEndDate;
    END;

    -- Zamknij i usuń kursor
    CLOSE SessionCursor;
    DEALLOCATE SessionCursor;

    PRINT '-----';
END;
GO

DECLARE @StudyID1 INT = 3;
DECLARE @StudyID2 INT = 21;

EXEC dbo.GenerateStudySchedule @StudyID1;
EXEC dbo.GenerateStudySchedule @StudyID2;

```

19. Sprawdzanie, czy lektor o podanym ID jest dostępny w podanym przedziale czasowym (Kinga)

```

CREATE PROCEDURE CheckLecturerAvailability
    @LecturerID INT,
    @StartTime DATETIME,
    @EndTime DATETIME,
    @LecturerAvailable BIT = NULL OUTPUT
AS
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM Modules M
        JOIN ModulesInPerson MIP ON M.ModuleID = MIP.ModuleID
        WHERE M.LecturerID = @LecturerID
            AND @StartTime < MIP.EndTime
            AND @EndTime > MIP.StartTime
    )
    BEGIN
        -- Lektor jest dostępny
        SET @LecturerAvailable = 1;
        PRINT 'Lektor jest dostępny w podanym przedziale czasowym.';
    END
    ELSE
    BEGIN
        -- Lektor jest zajęty
        SET @LecturerAvailable = 0;
        PRINT 'Lektor prowadzi inny moduł w podanym przedziale czasowym.';
    END
END;

EXEC CheckLecturerAvailability @LecturerID = 26, @StartTime = '2023-03-01
9:00', @EndTime = '2023-03-01 11:30';

EXEC CheckLecturerAvailability @LecturerID = 27, @StartTime = '2023-03-01
9:00', @EndTime = '2023-03-01 11:30';

```

20. Dodanie kursu o dwóch modułach - stacjonarnym i online synchronicznym (Kinga)

```

CREATE PROCEDURE AddCourseWithModules
    @CourseName NVARCHAR(255),
    @ParticipantsLimit INT,
    @Advance MONEY,
    @Price MONEY,
    @StartDate DATETIME,
    @EndDate DATETIME,

```



```

    @LecturerID1 INT,
    @StartTime1 DATETIME,
    @EndTime1 DATETIME,
    @LecturerID2 INT,
    @StartTime2 DATETIME,
    @EndTime2 DATETIME,
    @MeetingLink NVARCHAR(50)
AS
BEGIN
    DECLARE @LocationID INT;
    DECLARE @LocationAvailable BIT;
    EXEC CheckIfAnyLocationIsAvailable @StartTime1, @EndTime1, @LocationID
    OUTPUT, @LocationAvailable OUTPUT;

    IF @LocationAvailable = 0
    BEGIN
        PRINT 'Żadna lokalizacja dla modułu stacjonarnego nie jest dostępna
w podanym przedziale czasowym.';
        PRINT 'Sprawdź podane dane i spróbuj ponownie!';
        RETURN;
    END

    -- sprawdzamy czy lektor 1 jest dostępny
    DECLARE @Lecturer1Available BIT;
    EXEC CheckLecturerAvailability @LecturerID1, @StartTime1, @EndTime1,
    @Lecturer1Available OUTPUT;

    IF @Lecturer1Available = 0
    BEGIN
        PRINT 'Lektor o ID ' + CAST(@LecturerID1 AS NVARCHAR(10)) + ' nie
jest dostępny w podanym przedziale czasowym.';
        RETURN;
    END

    -- analogicznie czy lektor 2 jest dostępny
    DECLARE @Lecturer2Available BIT;
    EXEC CheckLecturerAvailability @LecturerID2, @StartTime2, @EndTime2,
    @Lecturer2Available OUTPUT;

    IF @Lecturer2Available = 0
    BEGIN
        PRINT 'Lektor o ID ' + CAST(@LecturerID2 AS NVARCHAR(10)) + ' nie
jest dostępny w podanym przedziale czasowym.';
        RETURN;
    END

    -- Dodaj kurs do tabeli Courses

```

```

DECLARE @CourseID INT;

INSERT INTO Courses (CourseName, ParticipantsLimit, Advance, Price,
StartDate, EndDate)
VALUES (@CourseName, @ParticipantsLimit, @Advance, @Price, @StartDate,
@EndDate);

SET @CourseID = SCOPE_IDENTITY();

-- Dodaj pierwszy moduł do tabeli Modules
INSERT INTO Modules (CourseID, LecturerID)
VALUES (@CourseID, @LecturerID1)

DECLARE @ModuleID1 INT;
SET @ModuleID1 = SCOPE_IDENTITY();

-- Dodaj pierwszy moduł do tabeli ModulesInPerson
INSERT INTO ModulesInPerson (ModuleID, LocationID, StartTime, EndTime)
VALUES (@ModuleID1, @LocationID, @StartTime1, @EndTime1);

-- Dodaj drugi moduł do tabeli Modules
INSERT INTO Modules (CourseID, LecturerID)
VALUES (@CourseID, @LecturerID2)

DECLARE @ModuleID2 INT;
SET @ModuleID2 = SCOPE_IDENTITY();

-- Dodaj drugi moduł do tabeli ModulesSync
INSERT INTO ModulesSync (ModuleID, StartTime, EndTime, MeetingLink)
VALUES (@ModuleID2, @StartTime2, @EndTime2, @MeetingLink);

PRINT 'Dodano pomyślnie kurs o ID' + @CourseID + ' i jego moduły'
END;

EXEC AddCourseWithModules @CourseName = 'Computational Complexity Theory',
@ParticipantsLimit = 20,
@Advance = 100, @Price = 450, @StartDate = '2024-03-24', @EndDate =
'2024-03-26',
@LecturerID1 = 26, @StartTime1 = '2024-03-24 17:00', @EndTime1 =
'2024-03-24 18:30',
@LecturerID2 = 26, @StartTime2 = '2024-03-26 17:00', @EndTime2 =
'2024-03-26 18:30',
@MeetingLink = 'http://cc_theory.com'

```

21. Uczestnictwo w bezpłatnym webinarze (Karol) - (Czyli de facto zwracanie linku do webinaru o podanym ID wtedy, gdy jest bezpłatny)

```
CREATE FUNCTION GetLinkToFreeWebinar (  
    @WebinarID INT  
)  
RETURNS NVARCHAR(50)  
AS  
BEGIN  
    --sprawdzamy, czy podany webinar istnieje  
    DECLARE @WebinarLink NVARCHAR(50);  
    IF EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID)  
    BEGIN  
        IF (SELECT Price FROM Webinars WHERE WebinarID = @WebinarID) =  
0  
        BEGIN  
            SELECT @WebinarLink = RecordingLink FROM Webinars WHERE  
WebinarID = @WebinarID  
        END  
        ELSE  
        BEGIN  
            SET @WebinarLink = 'Webinar o podanym ID jest płatny.'  
        END  
    END  
    ELSE  
    BEGIN  
        SET @WebinarLink = 'Webinar o podanym ID nie istnieje';  
    END  
    RETURN @WebinarLink  
END
```

```
SELECT TOP 2 * FROM Webinars
```

```
DECLARE @WebinarLink1 NVARCHAR(50);  
DECLARE @WebinarLink2 NVARCHAR(50);
```

```
SET @WebinarLink1 = dbo.GetLinkToFreeWebinar(1);  
SET @WebinarLink2 = dbo.GetLinkToFreeWebinar(2);
```

```
PRINT @WebinarLink1;  
PRINT @WebinarLink2;
```

22. Przypisanie roli danemu użytkownikowi - (Paulina)

```

CREATE PROCEDURE AssignRoleToUser
    @UserID INT,
    @RoleName NVARCHAR(30),
    @StartDate DATE,
    @EndDate DATE = NULL
AS
BEGIN
    DECLARE @RoleID INT;

    -- Sprawdź, czy rola istnieje
    SELECT @RoleID = RoleID FROM Roles WHERE RoleName = @RoleName;

    IF @RoleID IS NOT NULL
    BEGIN
        -- Sprawdź, czy użytkownik istnieje
        IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
        BEGIN
            -- Sprawdź, czy użytkownik już ma przypisaną daną rolę w danym
            -- okresie
            IF NOT EXISTS (
                SELECT 1
                FROM RolesHistory
                WHERE UserID = @UserID
                AND RoleID = @RoleID
                AND (
                    (@StartDate >= StartDate AND @StartDate <=
ISNULL(EndDate, '9999-12-31'))
                    OR (@EndDate >= StartDate AND @EndDate <=
ISNULL(EndDate, '9999-12-31'))
                )
            )
            BEGIN
                INSERT INTO RolesHistory (UserID, RoleID, StartDate,
EndDate)
                VALUES (@UserID, @RoleID, @StartDate, @EndDate);

                PRINT 'Rola została przypisana użytkownikowi.';
            END
        ELSE
        BEGIN
            PRINT 'Użytkownik już ma przypisaną daną rolę w danym
            -- okresie.';
        END
    END
    ELSE
    BEGIN
        PRINT 'Użytkownik o podanym ID nie istnieje.';
    END
END

```

```
        END
    END
    ELSE
    BEGIN
        PRINT 'Rola o podanej nazwie nie istnieje.';
    END
END;
```

--Test

```
SELECT * FROM Users
SELECT * FROM RolesHistory
SELECT * FROM Roles
```

```
EXEC AssignRoleToUser @UserID = 30, @RoleName = 'Participant', @StartDate
= '2024-01-22';
```

```
EXEC AssignRoleToUser @UserID = 10, @RoleName = 'Participant', @StartDate
= '2019-01-01', @EndDate = '2019-10-01';
```

### 23. Dodawanie do koszyka -(Paulina)

```
CREATE PROCEDURE AddToCart
    @UserID INT,
    @ProductID INT,
    @ProductTypeID INT
AS
BEGIN
    -- Sprawdź, czy użytkownik istnieje
    IF NOT EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID)
    BEGIN
        PRINT 'Użytkownik o podanym ID nie istnieje.';
        RETURN;
    END

    -- Sprawdź, czy produkt istnieje
    IF NOT EXISTS (SELECT 1 FROM ProductType WHERE TypeID =
@ProductTypeID)
    BEGIN
        PRINT 'Typ produktu o podanym ID nie istnieje.';
        RETURN;
    END

    -- Sprawdź, czy produkt pasuje do zamówień
    IF dbo.ValidateProductIDForOrders(@ProductTypeID, @ProductID) <>
1
    BEGIN
        PRINT 'Podany produkt nie istnieje';
        RETURN;
    END

    -- Dodaj rekord do koszyka
    INSERT INTO Cart (UserID, ProductID, ProductTypeID)
    VALUES (@UserID, @ProductID, @ProductTypeID);

    PRINT 'Produkt został dodany do koszyka.';
END;

--Test
---courses
EXEC AddToCart 30, 30, 1
EXEC AddToCart 30, 29, 1

---study
EXEC AddToCart 19, 31, 2
EXEC AddToCart 19, 21, 2

---Webinars
```

```
EXEC AddToCart 31, 40, 3
EXEC AddToCart 31, 10, 3
```

```
---sessions
EXEC AddToCart 31, 40, 4
EXEC AddToCart 31, 39, 4
```

```
SELECT * FROM Users
SELECT * FROM Courses
SELECT * FROM Webinars
SELECT * FROM Studies
SELECT * FROM Sessions
```

24. Funkcja sprawdzająca dostępność tłumacza (Czy w podanym czasie nie prowadzi jakiegoś modułu synchronicznego lub stacjonarnego) - (Paulina) -używana w funkcji poniżej.

```
CREATE FUNCTION dbo.IsTranslatorAvailable
(
    @TranslatorID INT,
    @StartTime DATETIME,
    @EndTime DATETIME
)
RETURNS BIT
AS
BEGIN
    DECLARE @IsAvailable BIT;

    -- Sprawdź, czy tłumacz jest dostępny w danym czasie dla modułów
    synchronicznych
    SELECT @IsAvailable = CASE
        WHEN NOT EXISTS (
            SELECT 1
            FROM TranslatedModules tm
            INNER JOIN ModulesSync ms ON
                tm.ModuleID = ms.ModuleID
            WHERE tm.TranslatorID =
                @TranslatorID
            AND (@StartTime BETWEEN
                ms.StartTime AND ms.EndTime
                OR @EndTime BETWEEN
                ms.StartTime AND ms.EndTime)
        )
        THEN 1
        ELSE 0
    
```

```

END;

-- Jeśli tłumacz jest dostępny dla modułów synchronicznych,
sprawdź dla modułów InPerson
IF @IsAvailable = 1
BEGIN
    SELECT @IsAvailable = CASE
        WHEN NOT EXISTS (
            SELECT 1
            FROM TranslatedModules tm
            INNER JOIN ModulesInPerson mip
            ON tm.ModuleID = mip.ModuleID
            WHERE tm.TranslatorID =
@TranslatorID
            AND (@StartTime BETWEEN
mip.StartTime AND mip.EndTime
            OR @EndTime BETWEEN
mip.StartTime AND mip.EndTime)
        )
        THEN 1
        ELSE 0
    END;

END;

RETURN @IsAvailable;
END;

--Test

SELECT * FROM ModulesSync
SELECT * FROM ModulesInPerson
SELECT * FROM Translators
SELECT * FROM TranslatedModules

-- Zajęty bo w tym czasie tłumaczy moduleInPerson nr.1
SELECT dbo.IsTranslatorAvailable(5, '2023-03-01 09:00:00.000',
'2023-03-01 12:00:00.000') AS IsAvailable;

-- Zajęty bo w tym czasie tłumaczy moduleSync nr.20
SELECT dbo.IsTranslatorAvailable(5, '2024-06-07 13:00:00.000',
'2024-06-07 16:00:00.000') AS IsAvailable;

-- Wolny
SELECT dbo.IsTranslatorAvailable(7, '2024-01-01 14:00:00',
'2024-01-01 16:00:00') AS IsAvailable;

```



## 25. Przypisanie tłumacza do wybranego modułu - (Paulina)

```
CREATE PROCEDURE AddTranslatorToModule
(
    @ModuleID INT,
    @TranslatorID INT
)
AS
BEGIN
    DECLARE @IsSyncModule BIT;
        -- Sprawdź, czy moduł istnieje w tabeli Modules
    IF NOT EXISTS (SELECT 1 FROM Modules WHERE ModuleID = @ModuleID)
    BEGIN
        PRINT 'Podany moduł nie istnieje.';
        RETURN;
    END

    -- Sprawdź, czy tłumacz istnieje w tabeli Translators
    IF NOT EXISTS (SELECT 1 FROM Translators WHERE TranslatorID =
@TranslatorID)
    BEGIN
        PRINT 'Podany tłumacz nie istnieje.';
        RETURN;
    END

    -- Sprawdź, czy moduł należy do tabeli ModulesSync
    SELECT @IsSyncModule = CASE WHEN EXISTS (SELECT 1 FROM
ModulesSync WHERE ModuleID = @ModuleID) THEN 1 ELSE 0 END;

    IF @IsSyncModule = 1
    BEGIN
        DECLARE @StartTime DATETIME, @EndTime DATETIME;

        -- Pobierz czasy rozpoczęcia i zakończenia dla modułu z
tabeli ModulesSync
        SELECT @StartTime = StartTime, @EndTime = EndTime
        FROM ModulesSync
        WHERE ModuleID = @ModuleID;

        -- Sprawdź, czy tłumacz jest dostępny w danym czasie
        IF dbo.IsTranslatorAvailable(@TranslatorID, @StartTime,
@EndTime) = 1
        BEGIN
            -- Sprawdź, czy tłumacz nie jest już przypisany do tego
modułu
```

```

        IF NOT EXISTS (SELECT 1 FROM TranslatedModules WHERE
ModuleID = @ModuleID AND TranslatorID = @TranslatorID)
        BEGIN
            -- Dodaj tłumacza do tabeli TranslatedModules
            INSERT INTO TranslatedModules (ModuleID,
TranslatorID)
            VALUES (@ModuleID, @TranslatorID);

            PRINT 'Tłumacz został dodany do modułu.';
        END
        ELSE
        BEGIN
            PRINT 'Tłumacz jest już przypisany do tego modułu.';
        END
    END
    ELSE
    BEGIN
        PRINT 'Tłumacz jest już przypisany do innego modułu w tym
czasie.';
    END
END
ELSE
BEGIN
    -- Jeśli moduł nie należy do tabeli ModulesSync, dodaj
tłumacza bez sprawdzania dostępności czasowej
    -- oraz sprawdź, czy tłumacz nie jest już przypisany do tego
modułu

    IF NOT EXISTS (SELECT 1 FROM TranslatedModules WHERE
ModuleID = @ModuleID AND TranslatorID = @TranslatorID)
    BEGIN
        INSERT INTO TranslatedModules (ModuleID, TranslatorID)
        VALUES (@ModuleID, @TranslatorID);

        PRINT 'Tłumacz został dodany do modułu.';
    END
    ELSE
    BEGIN
        PRINT 'Tłumacz jest już przypisany do tego modułu.';
    END
END
END;

--Test
EXEC AddTranslatorToModule 22, 24

SELECT * FROM ModulesSync

```

```

SELECT * FROM ModulesInPerson
SELECT * FROM Translators
SELECT * FROM TranslatedModules

```

26. Usunięcie zawartości koszyka i przeniesienie informacji o zakupionych produktach do tabeli Orders oraz OrderDetails (wykonywana po pomyślnym statusie płatności) (Karol)

```

CREATE PROCEDURE ProcessOrderForUser (
    @UserID INT
)
AS
BEGIN
    --SET NOCOUNT ON

    IF EXISTS (SELECT 1 FROM Cart WHERE UserID = @UserID)
    BEGIN
        DECLARE @OrderID INT;

        INSERT INTO Orders (UserID, OrderDate)
        VALUES (@UserID, GETDATE());
        SET @OrderID = SCOPE_IDENTITY(); --zwraca nam id
ostatnio wstawionego wiersza
        INSERT INTO OrderDetails (OrderID, ProductID,
ProductTypeID)
        SELECT
            @OrderID,
            ProductID,
            ProductTypeID
        FROM Cart
        WHERE UserID = @UserID;
        DELETE FROM Cart WHERE UserID = @UserID;
    END
    ELSE
    BEGIN
        PRINT 'Koszyk dla podanego użytkownika jest pusty.'
    END
END;

```

## Triggery

- logika dla niektórych triggerów została zawarta w wybranych procedurach

## 1. Do sprawdzania poprawności ProductID iTypeID w tabeli Diplomas (Kinga)

```
CREATE TRIGGER CheckDiplomaProductType
ON Diplomas
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE NOT EXISTS (
            SELECT 1
            FROM ProductType p
            WHERE p.TypeID = i.TypeID AND p.ProductID = i.ProductID
        )
    )
    BEGIN
        RAISEERROR('Niepoprawny ProductID lub TypeID w tabeli Diplomas.',
16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

## 2. Do sprawdzania poprawność długości pól w tabeli Users (Kinga)

```
CREATE TRIGGER CheckUserFieldsLength
ON Users
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE LEN(i.FirstName) > 20 OR LEN(i.LastName) > 20 OR
LEN(i.Nationality) > 10 OR LEN(i.Phone) > 12 OR LEN(i.Email) > 30
    )
    BEGIN
        RAISERROR('Niepoprawna długość pól w tabeli Users.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

### 3. Do sprawdzania daty zamówienia w tabeli Orders (Kinga)

```
CREATE TRIGGER CheckOrderDate
ON Orders
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE i.OrderDate < '2018-01-01' OR i.OrderDate > '9999-12-31'
    )
    BEGIN
        RAISERROR('Niepoprawna data zamówienia w tabeli Orders.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

### 4. Do sprawdzania poprawności zaliczki i ceny w tabeli Courses (Kinga)

```
CREATE TRIGGER CheckCoursePrices
ON Courses
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE i.Advance < 0 OR i.Price < 0
    )
    BEGIN
        RAISERROR('Niepoprawna cena kursu w tabeli Courses.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

### 5. Do sprawdzania poprawności wprowadzanego do tabeli lektora (Kinga)

```
CREATE TRIGGER CheckModuleLecturers
ON Modules
AFTER INSERT, UPDATE
AS
```

```

BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE NOT EXISTS (
            SELECT 1
            FROM Lecturers l
            WHERE l.LecturerID = i.LecturerID
        )
    )
    BEGIN
        RAISERROR('Niepoprawne LektorID w tabeli Modules.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;

```

6. Do sprawdzania unikalności kombinacji imię, nazwisko, telefon w tabeli Users (Kinga)

```

CREATE TRIGGER CheckUniqueUser
ON Users
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE NOT EXISTS (
            SELECT 1
            FROM Users u
            WHERE u.FirstName = i.FirstName AND u.LastName = i.LastName
            AND u.Phone = i.Phone
        )
    )
    BEGIN
        RAISERROR('Kombinacja FirstName, LastName, Phone musi być unikalna
w tabeli Users.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        INSERT INTO Users (FirstName, LastName, Nationality, Phone, Email)
        SELECT FirstName, LastName, Nationality, Phone, Email
        FROM inserted;
    END
END;

```

## 7. Do sprawdzania poprawności liczby uczestników dla kursu (Kinga)

```
CREATE TRIGGER CheckParticipantsLimit
ON Courses
INSTEAD OF INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE i.ParticipantsLimit <= 0
    )
    BEGIN
        RAISERROR('Limit uczestników musi być większy niż 0.', 16, 1);
        ROLLBACK TRANSACTION;
    END
    ELSE
    BEGIN
        INSERT INTO Courses (CourseName, ParticipantsLimit, Advance,
Price, StartDate, EndDate)
        SELECT CourseName, ParticipantsLimit, Advance, Price, StartDate,
EndDate
        FROM inserted;
    END
END;
```

## 8. Do sprawdzania unikalności nazwy kursu (Kinga)

```
CREATE TRIGGER CheckUniqueCourseName
ON Courses
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE EXISTS (
            SELECT 1
            FROM Courses
            WHERE CourseID <> i.CourseID AND CourseName = i.CourseName
        )
    )
    BEGIN
```

```

        RAISERROR('Nazwa kursu musi być unikalna.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;

```

## 9. Do sprawdzania, czy link do nagrania jest poprawny (Kinga)

```

CREATE TRIGGER CheckRecordingLink
ON ClassesAsync
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE LEN(i.RecordingLink) > 50 OR (i.RecordingLink IS NOT NULL
AND i.RecordingLink NOT LIKE 'http%')
    )
    BEGIN
        RAISERROR('Link nagrania musi mieć maksymalnie 50 znaków i
zaczynać się od "http".', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;

```

## 10. Jak wyżej, ale dla tabeli kursów synchronicznych (Kinga)

```

CREATE TRIGGER CheckRecordingLinkSync
ON ClassesSync
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE LEN(i.RecordingLink) > 50 OR (i.RecordingLink IS NOT NULL
AND i.RecordingLink NOT LIKE 'http%')
    )
    BEGIN
        RAISERROR('Link nagrania musi mieć maksymalnie 50 znaków i
zaczynać się od "http".', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;

```



11. Sprawdzanie, czy data zakończenia nie jest wcześniejsza niż data rozpoczęcia (Kinga)

```
CREATE TRIGGER CheckStudyDates
ON Studies
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE i.StartDate > i.EndDate
    )
    BEGIN
        RAISERROR('Data rozpoczęcia studiów nie może być późniejsza niż data zakończenia.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

12. Sprawdzanie, czy istnieje użytkownik o danym UserID przy dodawaniu do tabeli Lecturers (Kinga)

```
CREATE TRIGGER CheckLecturerExistence
ON Lecturers
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE NOT EXISTS (
            SELECT 1
            FROM Users
            WHERE UserID = i.LecturerID
        )
    )
    BEGIN
        RAISERROR('Wykładowca musi być zarejestrowany jako użytkownik.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

### 13. Sprawdzanie, czy LocationID jest unikalne w tabeli Locations (Kinga)

```
CREATE TRIGGER CheckUniqueLocationID
ON Locations
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT LocationID, COUNT(*)
        FROM inserted
        GROUP BY LocationID
        HAVING COUNT(*) > 1
    )
    BEGIN
        RAISERROR('LocationID musi być unikalne.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

### 14. Sprawdzanie, czy email użytkownika jest unikalny (Kinga)

```
CREATE TRIGGER CheckUniqueUserEmail
ON Users
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT Email, COUNT(*)
        FROM inserted
        GROUP BY Email
        HAVING COUNT(*) > 1
    )
    BEGIN
        RAISERROR('Adres email musi być unikalny.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

### 15. Sprawdzanie, czy telefon użytkownika jest unikalny (Kinga)

```
CREATE TRIGGER CheckUniqueUserPhone
```

```

ON Users
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT Phone, COUNT(*)
        FROM inserted
        GROUP BY Phone
        HAVING COUNT(*) > 1
    )
    BEGIN
        RAISERROR('Numer telefonu musi być unikalny.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;

```

## INDEKSY

1. Indeks równocześnie dla ProductID,TypeID w tabeli Diplomas (Kinga)

```

CREATE INDEX IX_ProductID_TypeID_Diplomas
ON Diplomas (ProductID,TypeID);

```

2. Indeks równocześnie dla ProductID, ProductTypeID w tabeli OrderDetails (Kinga)

```

CREATE INDEX IX_ProductID_ProductTypeID_OrderDetails
ON OrderDetails (ProductID, ProductTypeID);

```

```

SET SHOWPLAN_TEXT ON;
GO
SELECT * FROM Diplomas WHERE ProductID = 123 ANDTypeID = 456;
GO
SET SHOWPLAN_TEXT OFF;

```

3. Indeks równocześnie dla ProductID, ProductTypeID w tabeli Cart (Kinga)

```

CREATE INDEX IX_ProductID_ProductTypeID_Cart
ON Cart (ProductID, ProductTypeID);

```

4. Indeks dla LecturerID w tabeli Modules (Kinga)

```
CREATE INDEX IX_LecturerID_Modules  
ON Modules (LecturerID);
```

## UPRAWNIENIA

### 1. Administrator (Kinga)

```
CREATE ROLE Admin;  
GRANT UPDATE (ProductID,TypeID,StudentID, SendingDate) ON Diplomas TO  
Admin;  
GRANT DELETE ON Diplomas TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON Users TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON Courses TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON Logins TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON Modules TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON Syllabuses TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON Subjects TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON Locations TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON ModulesInPerson TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON ModulesSync TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON ModulesAsync TO Admin;  
GRANT SELECT, INSERT, UPDATE, DELETE ON PaymentExceptions TO Admin;  
GRANT EXECUTE ON CancelWebinar TO Admin;  
GRANT EXECUTE ON DeleteAccount TO Admin;  
GRANT EXECUTE ON SetCapacityLimit TO Admin;
```

### 2. Księgowa (Kinga)

```
CREATE ROLE Ksiegowa;  
GRANT SELECT ON WebinarPayments TO Ksiegowa;  
GRANT SELECT ON SessionPayments TO Ksiegowa;  
GRANT SELECT ON CoursePayments TO Ksiegowa;  
GRANT SELECT ON FinancialReport TO Ksiegowa;  
GRANT SELECT ON FinancialReportsPerProduct TO Ksiegowa;  
GRANT SELECT ON OverduePaymentsView TO Ksiegowa;  
GRANT EXECUTE ON CheckPaymentStatus TO Ksiegowa;
```

## CASE STUDY

--przypisz rolę danemu użytkownikowi

-Test

```
SELECT * FROM Users
```

```
SELECT * FROM RolesHistory
```

```
SELECT * FROM Roles
```

-Przypisanie nowej roli, która jeszcze się nie zakończyła

```
EXEC AssignRoleToUser @UserID = 31, @RoleName = 'Participant', @StartDate  
= '2024-01-25';
```

- możemy przypisać z zakończeniem pełnienia danej roli

```
EXEC AssignRoleToUser @UserID = 10, @RoleName = 'Participant', @StartDate  
= '2018-01-01', @EndDate = '2018-10-01';
```

-- dodaj moduł poprawnie

```
EXEC AddCourseWithModules @CourseName = 'Computational Complexity Theory',  
@ParticipantsLimit = 20,  
    @Advance = 100, @Price = 450, @StartDate = '2024-03-24', @EndDate =  
'2024-03-26',  
    @LecturerID1 = 26, @StartTime1 = '2024-03-24 17:00', @EndTime1 =  
'2024-03-24 18:30',  
    @LecturerID2 = 26, @StartTime2 = '2024-03-26 17:00', @EndTime2 =  
'2024-03-26 18:30',  
    @MeetingLink = 'http://cc\_theory.com'
```

-- nie da się dodać modułu w tym samym czasie

```
EXEC AddCourseWithModules @CourseName = 'Programming For Beginners 1',  
@ParticipantsLimit = 20,  
    @Advance = 100, @Price = 450, @StartDate = '2024-03-24', @EndDate =  
'2024-03-26',  
    @LecturerID1 = 12, @StartTime1 = '2024-03-24 17:00', @EndTime1 =  
'2024-03-24 18:30',  
    @LecturerID2 = 12, @StartTime2 = '2024-03-26 17:00', @EndTime2 =  
'2024-03-26 18:30',  
    @MeetingLink = 'http://program\_begin1.com'
```

-- nie da się dodać modułu, jeśli lektor ma w tym czasie inne zajęcia

```
EXEC AddCourseWithModules @CourseName = 'Einstein Theory',  
@ParticipantsLimit = 20,  
    @Advance = 100, @Price = 450, @StartDate = '2024-03-24', @EndDate =  
'2024-03-26',  
    @LecturerID1 = 26, @StartTime1 = '2024-03-24 17:00', @EndTime1 =  
'2024-03-24 18:30',
```

```
@LecturerID2 = 26, @StartTime2 = '2024-03-26 17:00', @EndTime2 =  
'2024-03-26 18:30',  
@MeetingLink = 'http://einstein\_theory.com'
```

```
-- próba dodania zerowego Limitu uczestników zakończy się niepowodzeniem  
EXEC AddCourseWithModules @CourseName = 'Programming in PHP',  
@ParticipantsLimit = 0,  
@Advance = 100, @Price = 450, @StartDate = '2024-03-24', @EndDate =  
'2024-03-26',  
@LecturerID1 = 12, @StartTime1 = '2024-03-24 17:00', @EndTime1 =  
'2024-03-24 18:30',  
@LecturerID2 = 12, @StartTime2 = '2024-03-26 17:00', @EndTime2 =  
'2024-03-26 18:30',  
@MeetingLink = 'http://program\_begin1.com'
```

```
--przypisanie tłumacza do modułu  
--Test
```

```
SELECT * FROM ModulesSync  
SELECT * FROM ModulesInPerson  
SELECT * FROM Translators  
SELECT * FROM TranslatedModules
```

```
EXEC AddTranslatorToModule @ModuleID = 12, @TranslatorID = 7  
-Już jest przypisany  
EXEC AddTranslatorToModule @ModuleID = 22, @TranslatorID = 24
```

```
-dodaj do koszyka wybrany kurs
```

```
--Test  
---courses  
SELECT * FROM Courses  
SELECT * FROM Cart  
EXEC AddToCart @UserID = 31, @ProductID = 30, @ProductTypeID = 1  
EXEC AddToCart @UserID = 31, @ProductID = 29, @ProductTypeID = 1
```

```
-zakup dla podanego użytkownika produktów z jego koszyka
```

```
EXEC ProcessOrderForUser @UserID = 31
```

## Diagram bazy danych

