Systemy Baz Danych 2023/2024 – projekt

Autorzy:

Bartosz Ludwin, Bartłomiej Kaczyński, Kacper Wąchała

Funkcje systemu i uprawnienia użytkowników

Konta i uprawnienia

- **Zakładanie konta uczestnika**: Gość
- **Usuwanie swojego konta, ustawianie adresu korespondencyjnego**: Uczestnik
- **Zakładanie i dezaktywowanie kont tłumaczy, prowadzących, koordynatorów**: Administrator uprawnień
- **Dodawanie i usuwanie pracowników sekretariatu oraz administratorów**: Dyrektor

---

Webinary

- **Przeglądanie listy webinarów**: Gość
- **Uzyskiwanie dostępu do szczegółowych danych webinaru po zapisaniu**: Uczestnik
- **Modyfikowanie szczegółów, przypisywanie tłumacza**: Prowadzący
- **Dodawanie webinarów**: Administrator planista
- **Usuwanie webinarów**: Administrator główny
- **Umożliwianie darmowego dostępu do płatnych webinarów**: Dyrektor
- **Zapisywanie uczestnika po płatności, generowanie linków do płatności i rejestrowanie transakcji**: System

---

Kursy

- **Przeglądanie listy kursów i modułów**: Gość
- **Uzyskiwanie dostępu do szczegółowych danych kursu po zapisaniu**: Uczestnik
- **Modyfikowanie kursów i modułów, wystawianie zaliczeń**: Koordynator
- **Dodawanie kursów i tworzenie harmonogramów**: Administrator planista
- **Usuwanie kursów**: Administrator główny
- **Przypisywanie tłumaczy do modułów**: Prowadzący
- **Umożliwianie darmowego dostępu do płatnych kursów**: Dyrektor
- **Automatyczne zapisywanie uczestnika, generowanie dyplomów, wyliczanie dostępności miejsc**: System

---

Studia

- **Przeglądanie listy studiów i modułów**: Gość
- **Uzyskiwanie dostępu do szczegółowych danych studiów po zapisaniu**: Uczestnik

- **Dodawanie studiów, przedmiotów, praktyk, tworzenie harmonogramów**: Administrator planista
- **Wystawianie zaliczeń**: Koordynator
- **Sprawdzanie obecności, wskazywanie możliwości odrabiania, przypisywanie tłumaczy**: Prowadzący
- **Umożliwianie darmowego dostępu do płatnych studiów**: Dyrektor
- **Automatyczne zapisywanie uczestnika, zarządzanie miejscami, generowanie dyplomów**: System

---

Raporty

- **Generowanie raportów finansowych, frekwencji, zapisów i kolizji terminów**: Pracownik sekretariatu
- **Przeglądanie raportów frekwencji i kolizji**: Uczestnik
- **Generowanie list obecności, raportów wydarzeń**: Prowadzący
- **Przeglądanie list tłumaczonych wydarzeń**: Tłumacz

Koszyk

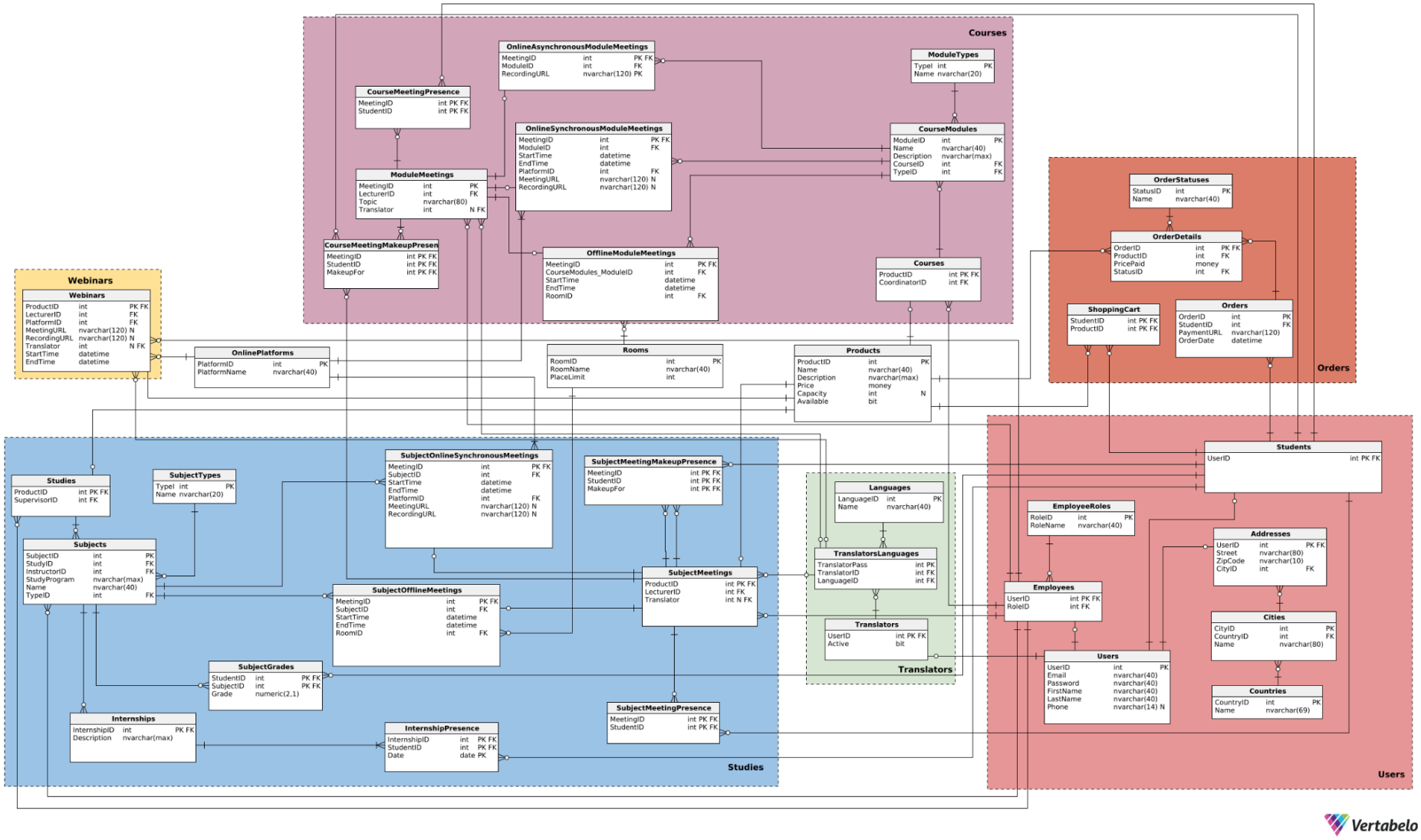
- **Dodawanie produktów, tworzenie zamówień, dokonywanie płatności**: Uczestnik
- **Generowanie linków do płatności, rejestrowanie transakcji**: System

---

Zarządzanie systemem

- **Backupy ręczne i odtwarzanie danych**: Administrator backupów
- **Zarządzanie dostępem użytkowników i rolami**: Administrator uprawnień
- **Automatyczne backupy, wysyłanie przypomnień o płatnościach**: System

Schemat bazy danych



Vertabelo

Users

Users

Opis: Tabela `Users` przechowuje dane dotyczące każdego użytkownika zapisanego w bazie, w tym ID, adres e-mail, hasło (przechowywane jako wynik funkcji skrótu SHA256), imię, nazwisko oraz opcjonalny numer telefonu.

Kolumny:

- `UserID`: Klucz główny, identyfikator użytkownika (int, NOT NULL).
- `Email`: Adres e-mail użytkownika (nvarchar(40), NOT NULL).
- `Password`: Hasło użytkownika (nvarchar(40), NOT NULL).
- `FirstName`: Imię użytkownika (nvarchar(40), NOT NULL).
- `LastName`: Nazwisko użytkownika (nvarchar(40), NOT NULL).
- `Phone`: Opcjonalny numer telefonu użytkownika (nvarchar(15), DEFAULT NULL).

Warunki integralności:

- `Email` musi być unikalny i w formacie `'%_@%_.%_'`.

```
CHECK (Email LIKE '%_@%_.%_')
```

- `Phone` może być `NULL` lub w formacie `'+## ### ### ###'` gdzie `'#'` to liczba z zakresu od 0 do 9.

```
CHECK (Phone IS NULL OR Phone LIKE '+[0-9][0-9]
[0-9][0-9][0-9] [0-9][0-9][0-9] [0-9][0-9][0-9]')
```

```
CREATE TABLE Users (
   UserID int  NOT NULL,
   Email nvarchar(40)  NOT NULL CHECK (Email LIKE '%_@%_.%_'),
   Password nvarchar(40)  NOT NULL CHECK (LEN(Password) >= 5),
   FirstName nvarchar(40)  NOT NULL,
   LastName nvarchar(40)  NOT NULL,
   Phone nvarchar(14)  NULL DEFAULT NULL CHECK (Phone IS NULL OR
Phone LIKE '[0-9][0-9]+[0-9][0-9][0-9] [0-9][0-9][0-9]
[0-9][0-9][0-9]'),
   CONSTRAINT Users_unique_email UNIQUE (Email),
   CONSTRAINT Users_pk PRIMARY KEY  (UserID)
);
```

Students

Opis: Tabela `Students` przechowuje identyfikatory użytkowników będących studentami, uczestnikami kursów lub webinarów.

Kolumny:

- `UserID`: Klucz główny, identyfikator użytkownika (int, NOT NULL).

Relacje:

- `UserID` jest kluczem obcym odwołującym się do `Users(UserID)`.

```
CREATE TABLE Students (
   UserID int  NOT NULL,
   CONSTRAINT Students_pk PRIMARY KEY  (UserID)
);
```

Employees

Opis: Tabela `Employees` przechowuje identyfikatory pracowników wraz z ich rolą (np. admin).

Kolumny:

- `UserID`: Klucz główny, identyfikator użytkownika (int, NOT NULL).
- `Role`: Rola użytkownika specjalnego (bit, DEFAULT 1).

Relacje:

- `UserID` jest kluczem obcym odwołującym się do `Users(UserID)`.
- `RoleID` jest kluczem obcym odwołującym się do `EmployeeRoles(RoleID)`.

```
CREATE TABLE Employees (
   UserID int  NOT NULL,
   RoleID int  NOT NULL,
   CONSTRAINT Employees_pk PRIMARY KEY  (UserID)
);
```

Employee Roles

Opis: Tabela `EmployeeRoles` jest tabelą słownikową zawierającą nazwy ról.

Kolumny:

- `RoleID`: Klucz główny, identyfikator roli (int, NOT NULL).
- `RoleName`: Nazwa roli (nvarchar(40), NOT NULL).

```
CREATE TABLE EmployeeRoles (
   RoleID int  NOT NULL,
   RoleName nvarchar(40)  NOT NULL,
   CONSTRAINT EmployeeRoles_pk PRIMARY KEY  (RoleID)
);
```

Countries

Opis: Tabela `Countries` jest tabelą słownikową, przechowującą identyfikatory oraz nazwy krajów.

Kolumny:

- `CountryID`: Klucz główny, identyfikator kraju (int, NOT NULL).
- `Name`: Nazwa kraju (nvarchar(80), NOT NULL).

Warunki integralności:

- `Name` musi być unikalny.

```
CREATE TABLE Countries (
   CountryID int  NOT NULL,
   Name nvarchar(69)  NOT NULL,
   CONSTRAINT Countries_pk PRIMARY KEY  (CountryID)
   CONSTRAINT UQ_Countries_Name UNIQUE (Name)
);
```

Cities

Opis: Tabela `Cities` przechowuje informacje o miastach, w tym ich identyfikatory, nazwy oraz przypisanie do krajów.

Kolumny:

- `CityID`: Klucz główny, identyfikator miasta (int, NOT NULL).
- `CountryID`: Identyfikator kraju (int, NOT NULL).
- `Name`: Nazwa miasta (nvarchar(80), NOT NULL).

Warunki integralności:

- Kombinacja `Name` i `CountryID` musi być unikalna.

  ```
  UNIQUE (CountryID, Name)
  ```

Relacje:

- `CountryID` jest kluczem obcym odwołującym się do `Countries(CountryID)`.

```
CREATE TABLE Cities (
   CityID int  NOT NULL,
   CountryID int  NOT NULL,
   Name nvarchar(80)  NOT NULL,
   CONSTRAINT Cities_unique_name UNIQUE (CountryID, Name),
   CONSTRAINT Cities_pk PRIMARY KEY  (CityID)
);
```

Addresses

Opis: Tabela `Addresses` przechowuje dane adresowe użytkowników, w tym ulicę, kod pocztowy oraz przypisanie do miasta.

Kolumny:

- `UserID`: Klucz główny, identyfikator studenta (int, NOT NULL).
- `Street`: Ulica (nvarchar(80), NOT NULL).
- `ZipCode`: Kod pocztowy (nvarchar(10), NOT NULL).
- `CityID`: Identyfikator miasta (int, NOT NULL).

Warunki integralności:

- `ZipCode` musi składać się wyłącznie z cyfr.

```
CHECK (ISNUMERIC(ZipCode) = 1)
```

Relacje:

- `CityID` jest kluczem obcym odwołującym się do `Cities(CityID)`.
- `UserID` jest kluczem obcym odwołującym się do `Users(UserID)`.

```
CREATE TABLE Addresses (
   UserID int  NOT NULL,
   Street nvarchar(80)  NOT NULL,
   ZipCode nvarchar(10)  NOT NULL CHECK (ISNUMERIC(ZipCode) = 1),
   CityID int  NOT NULL,
   CONSTRAINT Addresses_pk PRIMARY KEY  (UserID)
);
```

Shared

Products

Opis: Tabela `Products` przechowuje dane o produktach dostępnych do zakupu przez studentów. Produktami są: webinary, kursy i studia, a także pojedyncze spotkania studyjne. Ilość miejsc na danym produkcie może być ograniczona i jest reprezentowana przez kolumnę `Capacity` (wartość NULL oznacza nieograniczoną ilość miejsc).

Kolumny:

- `ProductID`: Klucz główny, identyfikator aktywności (int, NOT NULL).
- `Name`: Nazwa produktu (nvarchar(40), NOT NULL).
- `Description`: Opis produktu (nvarchar(max), NOT NULL).
- `Price`: Cena produktu (money, DEFAULT 0).
- `Capacity`: Ilość miejsc na produkcie(int, DEFAULT NULL).
- `Avaible`: Czy produkt można jeszcze kupić (bit, DEFAULT 1).

Warunki integralności:

- `Price` nie może być ujemna.

```
CHECK (Price >= 0)
```

- `Capacity` wartość może być dodatnia lub nullem.

```
CHECK (Capacity IS NULL OR Capacity > 0)
```

```
CREATE TABLE Products (
   ProductID int  NOT NULL,
   Name nvarchar(40)  NOT NULL,
   Description nvarchar(max)  NOT NULL,
   Price money  NOT NULL DEFAULT 0 CHECK (Price >= 0),
   Capacity int  NULL DEFAULT NULL CHECK (Capacity IS NULL OR
Capacity > 0),
   Available bit  NOT NULL DEFAULT 1,
   CONSTRAINT Products_pk PRIMARY KEY  (ProductID)
);
```

Online Platforms

Opis: Tabela `OnlinePlatforms` jest tabelą słownikową, która przechowuje dane o platformach na których mogą odbywać się spotkania online.

Kolumny:

- `PlatformID`: Identyfikator platformy (int, NOT NULL).
- `PlatformName`: Nazwa platformy (nvarchar(40), NOT NULL).

```
CREATE TABLE OnlinePlatforms (
   PlatformID int  NOT NULL,
   PlatformName nvarchar(40)  NOT NULL,
   CONSTRAINT OnlinePlatforms_unique UNIQUE (PlatformName),
   CONSTRAINT OnlinePlatforms_pk PRIMARY KEY  (PlatformID)
);
```

Rooms

Opis: Tabela `Rooms` jest tabelą słownikową, która przechowuje dane o pokojach, w których mogą się odbywać spotkania offline.

Kolumny:

- `RoomID`: Identyfikator pokoju (int, NOT NULL).
- `RoomName`: Nazwa pokoju (nvarchar(40), NOT NULL).
- `PlaceLimit`: Ilość miejsc w pokoju (int, NOT NULL).

Warunki integralności:

- `PlaceLimit` musi być większe od 0.

```
CHECK (PlaceLimit > 0)
```

- `RoomName` nie może się powtarzać.

```
UNIQUE (RoomName)
```

```
CREATE TABLE Rooms (
   RoomID int  NOT NULL,
   RoomName nvarchar(40)  NOT NULL,
   PlaceLimit int  NOT NULL CHECK (PlaceLimit > 0),
   CONSTRAINT Rooms_name UNIQUE (RoomName),
   CONSTRAINT Rooms_pk PRIMARY KEY  (RoomID)
);
```

Webinars

Webinars

Opis: Tabela `Webinars` przechowuje dane o webinarach, które są jednym z produktów oferowanych przez firmę.

Kolumny:

- `ProductID`: Klucz główny, identyfikator produktu (int, NOT NULL).
- `LecturerID`: Identyfikator prowadzącego spotkanie (int, NOT NULL).
- `PlatformID`: Identyfikator platformy (int, NOT NULL).
- `MeetingURL`: Link do spotkania (nvarchar(120), DEFAULT NULL).
- `RecordingURL`: Link do nagrania (nvarchar(120), DEFAULT NULL).
- `Translator`: Opcjonalny translator spotkania (int, DEFAULT NULL).

Warunki integralności:

- `MeetingURL` musi być nullem albo poprawny.

```
CHECK (MeetingURL IS NULL OR MeetingURL LIKE 'https://%_.%_')
```

- `RecordingURL` musi być nullem albo poprawny.

```
CHECK (RecordingURL IS NULL OR RecordingURL LIKE
'https://%_.%_')
```

Relacje:

- `ProductID` jest kluczem obcym odwołującym się do `Products(ProductID)`.
- `LecturerID` jest kluczem obcym odwołującym się do `Employees(UserID)`.

- `PlatformID` jest kluczem obcym odwołującym się do `OnlinePlatforms(PlatformID)`.

```
CREATE TABLE Webinars (
   ProductID int  NOT NULL,
   LecturerID int  NOT NULL,
   PlatformID int  NOT NULL,
   MeetingURL nvarchar(120)  NULL DEFAULT NULL CHECK (MeetingURL
IS NULL OR MeetingURL LIKE 'https://%_.%_'),
   RecordingURL nvarchar(120)  NULL DEFAULT NULL CHECK
(RecordingURL IS NULL OR RecordingURL LIKE 'https://%_.%_'),
   Translator int  NULL DEFAULT NULL,
   CONSTRAINT Webinars_pk PRIMARY KEY  (ProductID)
);
```

Courses

Courses

Opis: Tabela `Courses` przechowuje ID produktu związanego z kursami oraz przypisanego koordynatora.

Kolumny:

- `ProductID`: Klucz główny, identyfikator kursu (int, NOT NULL).
- `CoordinatorID`: Identyfikator koordynatora kursu (int, NOT NULL).

Relacje:

- `ProductID` jest kluczem obcym odwołującym się do `Products(ProductID)`.
- `CoordinatorID` jest kluczem obcym odwołującym się do `Employees(UserID)`.

```
CREATE TABLE Courses (
   ProductID int  NOT NULL,
   CoordinatorID int  NOT NULL,
   CONSTRAINT Courses_pk PRIMARY KEY  (ProductID)
);
```

Course Modules

Opis: Tabela `CourseModules` przechowuje dane o modułach kursów i ich powiązaniach z kursami.

Kolumny:

- `ModuleID`: Klucz główny, identyfikator modułu kursu (int, NOT NULL).
- `Name`: Identyfikator kursu, do którego moduł należy (nvarchar(40), NOT NULL).
- `Description`: Opis modułu (nvarchar(max), NOT NULL).

- `CourseID`: Identyfikator kursu, do którego moduł należy (int, NOT NULL).
- `TypeID`: ID typu modułu (int, NOT NULL)

Relacje:

- `ModuleID` jest kluczem obcym odwołującym się do `ModuleMeetings(ModuleID)`.
- `CouseID` jest kluczem obcym odwołującym się do `Courses(ProductID)`.
- `TypeID` jest kluczem obcym odwołującym się do `ModuleTypes(TypeID)`.

```
CREATE TABLE CourseModules (
   ModuleID int  NOT NULL,
   Name nvarchar(40)  NOT NULL,
   Description nvarchar(max)  NOT NULL,
   CourseID int  NOT NULL,
   TypeID int  NOT NULL,
   CONSTRAINT CourseModules_pk PRIMARY KEY  (ModuleID)
);
```

Module Types

Opis: Tabela słownikowa `ModuleTypes` przechowuje możliwe typy modułów.

Kolumny:

- `TypeID`: Klucz główny, identyfikator typu kursu (int, NOT NULL).
- `Name`: Nazwa typu kursu (nvarchar(20), NOT NULL).

```
CREATE TABLE ModuleTypes (
   TypeID int  NOT NULL IDENTITY,
   Name nvarchar(20)  NOT NULL,
   CONSTRAINT ModuleTypes_pk PRIMARY KEY  (TypeID)
);
```

Module Meetings

Opis: Tabela `ModuleMeetings` przechowuje ogólne dane o spotkaniach powiązanych z modułami kursów. Bardziej szczegółowe dane są w tabelach dotyczących odpowiednich rodzajów spotkań.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `LecturerID`: Identyfikator prowadzącego spotkanie (int, NOT NULL).
- `Topic`: Temat spotkania (nvarchar(80), NOT NULL).
- `Translator`: Opcjonalny tłumacz spotkania (int, DEFAULT NULL).

Relacje:

- `LecturerID`: jest kluczem obcym odwołującym się do `Employees(UserID)`.

- `Translator`: jest kluczem obcym odwołującym się do `TranslatorLanguages(TranslatorPass)`.

```
CREATE TABLE ModuleMeetings (
   MeetingID int  NOT NULL,
   LecturerID int  NOT NULL,
   Topic nvarchar(80)  NOT NULL,
   Translator int  NULL,
   CONSTRAINT ModuleMeetings_pk PRIMARY KEY  (MeetingID)
);
```

Online Asynchronous Module Meetings

Opis: Tabela `OnlineAsynchronousModuleMeetings` przechowuje informacje o nagraniach udostępnianych w ramach kursów.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `SubjectID`: Identyfikator przedmiotu (int, NOT NULL).
- `RecordingURL`: Link do nagrania (nvarchar(120), DEFAULT NULL).

Warunki integralności:

- `RecordingURL` musi być NULL lub poprawnym URL

  ```
  CHECK (RecordingURL IS NULL OR RecordingURL LIKE
  'https://%_.%_')
  ```

Relacje:

- `MeetingID` jest kluczem obcym odwołującym się do `ModuleMeetings(MeetingID)`.
- `SubjectID` jest kluczem obcym odwołującym się do `OnlineAsynchronousModule(ModuleID)` lub `HybridModules(ModuleID)`.
- `PlatformID` jest kluczem obcym odwołującym się do `Platforms(PlatformID)`.

```
CREATE TABLE OnlineAsynchronousModuleMeetings (
   MeetingID int  NOT NULL,
   ModuleID int  NOT NULL,
   RecordingURL nvarchar(120)  NOT NULL,
   CONSTRAINT OnlineAsynchronousModuleMeetings_unique UNIQUE
(RecordingURL),
   CONSTRAINT OnlineAsynchronousModuleMeetings_pk PRIMARY KEY
(RecordingURL,MeetingID)
);
```

Online Synchronous Module Meetings

Opis: Tabela `OnlineSynchronousModuleMeetings` przechowuje informacje o spotkaniach online na żywo, prowadzonych w ramach kursów. Każdy rekord reprezentuje jedno spotkanie, jego czas trwania, platformę używaną do spotkania, oraz odnośniki do spotkania i nagrania.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `SubjectID`: Identyfikator przedmiotu (int, NOT NULL).
- `StartTime`: Identyfikator języka (date, NOT NULL).
- `EndTime`: Identyfikator tłumacza (date, NOT NULL).
- `PlatformID`: Identyfikator platformy na której odbywa się spotkanie (int).
- `MeetingURL`: : Link do spotkania (nvarchar(120), DEFAULT NULL).
- `RecordingURL`: Link do nagrania spotkania (nvarchar(120), DEFAULT NULL).

Warunki integralności:

- `StartTime` musi być wcześniej niż `EndTime`

```
CHECK (StartTime < EndTime)
```

- `MeetingURL` musi być NULL lub poprawnym URL

```
CHECK (MeetingURL IS NULL OR MeetingURL LIKE 'https://%_.%_')
```

- `RecordingURL` musi być NULL lub poprawnym URL

```
CHECK (RecordingURL IS NULL OR RecordingURL LIKE
'https://%_.%_')
```

Relacje:

- `MeetingID` jest kluczem obcym odwołującym się do `ModuleMeetings(MeetingID)`.
- `SubjectID` jest kluczem obcym odwołującym się do `OnlineSynchronousModule(ModuleID)` lub `HybridModules(ModuleID)`.
- `PlatformID` jest kluczem obcym odwołującym się do `Platforms(PlatformID)`.

```
CREATE TABLE OnlineSynchronousModuleMeetings (
   MeetingID int  NOT NULL,
   ModuleID int  NOT NULL,
   StartTime datetime  NOT NULL,
   EndTime datetime  NOT NULL,
   PlatformID int  NOT NULL,
   MeetingURL nvarchar(120)  NULL DEFAULT NULL CHECK (MeetingURL
IS NULL OR MeetingURL LIKE 'https://%_.%_'),
```

```
   RecordingURL nvarchar(120)  NULL DEFAULT NULL CHECK
(RecordingURL IS NULL OR RecordingURL LIKE 'https://%_.%_'),
   CONSTRAINT OnlineSynchronousModuleMeetings_uniqueURL UNIQUE
(MeetingURL),
   CONSTRAINT OnlineSynchronousModuleMeetings_uniqueURL2 UNIQUE
(RecordingURL),
   CONSTRAINT date CHECK (StartTime < EndTime ),
   CONSTRAINT OnlineSynchronousModuleMeetings_pk PRIMARY KEY
(MeetingID)
);
```

Offline Module Meetings

Opis: Tabela `OfflineModuleMeetings` przechowuje informacje o spotkaniach stacjonarnych, prowadzonych w ramach kursów. Każdy rekord reprezentuje jedno spotkanie, jego czas trwania, oraz pokój, w którym się odbywa .

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `SubjectID`: Identyfikator przedmiotu (int, NOT NULL).
- `StartTime`: Identyfikator języka (date, NOT NULL).
- `EndTime`: Identyfikator tłumacza (date, NOT NULL).
- `RoomID`: Identyfikator sali w której odbywa się spotkanie (int, NOT NULL).

Warunki integralności:

- `StartTime` musi być wcześniej niż `EndTime`.

  ```
  CHECK (StartTime < EndTime)
  ```

Relacje:

- `MeetingID` jest kluczem obcym odwołującym się do `ModuleMeetings(MeetingID)`.
- `ModuleID` jest kluczem obcym odwołującym się do `OfflineModules(ModuleID)` lub `HybridModules(ModuleID)`.
- `RoomID` jest kluczem obcym odwołującym się do `Rooms(RoomID)`.

Course Meeting Presence

Opis: Tabela `CourseMeetingPresence` przechowuje informacje o obecności na spotkaniach dotyczących danego kursu. Każdy rekord reprezentuje jedno spotkanie, zawierając szczegóły dotyczące prowadzącego i identyfikatora studenta.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).

- `StudentID`: Identyfikator studenta (int, NOT NULL).

Relacje:

- `MeetingID` jest kluczem obcym odwołującym się do `ModuleMeetings(MeetingID)`.

```
CREATE TABLE CourseMeetingPresence (
   MeetingID int  NOT NULL,
   StudentID int  NOT NULL,
   CONSTRAINT CourseMeetingPresence_pk PRIMARY KEY
(MeetingID,StudentID)
);
```

Course Meeting Makeup Presence

Opis: Tabela `CourseMeetingMakeupPresence` przechowuje informacje o obecności na spotkaniach służących odrabianiu obecności. Każdy rekord reprezentuje jedno spotkanie, zawierając szczegóły dotyczące prowadzącego i identyfikatora studenta oraz uzupełnianych spotkań.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `StudentID`: Identyfikator studenta (int, NOT NULL).
- `MakeupFor`: Identyfikator spotkania które się odrabia (int, NOT NULL).

Relacje:

- `MeetingID` jest kluczem obcym odwołującym się do `ModuleMeetings(MeetingID)`.
- `StudentID` jest kluczem obcym odwołującym się do `Students(UserID)`.
- `MeetingID` jest kluczem obcym odwołującym się do `SubjectMeetings(ProductID)`.

```
CREATE TABLE CourseMeetingMakeupPresence (
   MeetingID int  NOT NULL,
   StudentID int  NOT NULL,
   MakeupFor int  NOT NULL,
   CONSTRAINT CourseMeetingMakeupPresence_pk PRIMARY KEY
(MeetingID,StudentID,MakeupFor)
);
```

Studies

Studies

Opis: Tabela `Studies` przechowuje dane o studiach.

Kolumny:

- `ProductID`: Klucz główny, identyfikator kierunku (int, NOT NULL).
- `SupervisorID`: Opiekun kierunku studiów (int, NOT NULL).

Relacje:

- `ProductID` jest kluczem obcym odwołującym się do `Products(ProductID)`.
- `SupervisorID` jest kluczem obcym odwołującym się do `Employes(UserID)`.

```
CREATE TABLE Studies (
    ProductID int  NOT NULL,
    SupervisorID int  NOT NULL,
    CONSTRAINT Studies_pk PRIMARY KEY  (ProductID)
);
```

Subjects

Opis: Tabela `Subjects` przechowuje dane o kierunkach studiów.

Kolumny:

- `SubjectID`: Klucz główny, identyfikator produktu (int, NOT NULL).
- `StudyID`: Identyfikator kierunku studiów (int, NOT NULL).
- `InstructorID`: Identyfikator prowadzącego przedmiot (int, NOT NULL).
- `StudyProgram`: Sylabus kierunku (nvarchar(max), NOT NULL).
- `Name`: Nazwa kierunku studiów (nvarchar(40), NOT NULL).
- `TypeID`: Typ kierunku studiów (nvarchar(40), NOT NULL).

Relacje:

- `StudyID` jest kluczem obcym odwołującym się do `Studies(ProductID)`.
- `SubjectID` jest kluczem obcym odwołującym się do `Internships(InternshipID)`.
- `TypeID` jest kluczem obcym odwołującym się do `SubjectTypes(TypeID)`.

```
CREATE TABLE Subjects (
    SubjectID int  NOT NULL,
    StudyID int  NOT NULL,
    InstructorID int  NOT NULL,
    StudyProgram nvarchar(max)  NOT NULL,
    Name nvarchar(40)  NOT NULL,
    TypeID int  NOT NULL,
    CONSTRAINT Subjects_pk PRIMARY KEY  (SubjectID)
);
```

Subject Grades

Opis: Tabela `SubjectGrades` przechowuje dane o ocenach uzyskanych przez studentów.

Kolumny:

- `SubjectID`: Klucz główny, identyfikator przedmiotu (int, NOT NULL).
- `StudentID`: Klucz główny, identyfikator studenta (int, NOT NULL).
- `Grade`: Uzyskana ocena (int, NOT NULL).

Warunki integralności:

- `Grade` musi być z przedziału [2, 5].

```
CHECK (Grade BETWEEN 2 AND 5)
```

Relacje:

- `SubjectID` jest kluczem obcym odwołującym się do `Subject(SubjectID)`.
- `StudentID` jest kluczem obcym odwołującym się do `Students(UserID)`.

```
CREATE TABLE SubjectGrades (
   StudentID int  NOT NULL,
   SubjectID int  NOT NULL,
   Grade int  NOT NULL CHECK (Grade BETWEEN 2 AND 5),
   CONSTRAINT SubjectGrades_pk PRIMARY KEY  (StudentID,SubjectID)
);
```

Subject Types

Opis: Tabela słownikowa `SubjectTypes` zawiera możliwe typy przedmiotów na studiach

Kolumny:

- `TypeID`: Klucz główny, identyfikator typu (int, NOT NULL).
- `Name`: Nazwa typu (int, NOT NULL).

```
CREATE TABLE SubjectTypes (
   TypeID int  NOT NULL,
   Name nvarchar(20)  NOT NULL,
   CONSTRAINT SubjectTypes_pk PRIMARY KEY  (TypeID)
);
```

Subjects Online Synchronous Meetings

Opis: Tabela `SubjectOnlineSynchronousMeetings` przechowuje informacje o spotkaniach online na żywo. Każdy rekord reprezentuje jedno spotkanie, jego czas trwania, platformę używaną do spotkania, oraz odnośniki do spotkania i nagrania.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `SubjectID`: Identyfikator przedmiotu (int, NOT NULL).

- `StartTime`: Identyfikator języka (date, NOT NULL).
- `EndTime`: Identyfikator tłumacza (date, NOT NULL).
- `PlatformID`: Identyfikator platformy na której odbywa się spotkanie (int).
- `MeetingURL`: : Link do spotkania (nvarchar(120), DEFAULT NULL).
- `RecordingURL`: Link do nagrania spotkania (nvarchar(120), DEFAULT NULL).

Warunki integralności:

- `StartTime` musi być wcześniej niż `EndTime`

```
CHECK (StartTime < EndTime)
```

- `MeetingURL` musi być NULL lub poprawnym URL

```
CHECK (MeetingURL IS NULL OR MeetingURL LIKE 'https://%_.%_')
```

- `RecordingURL` musi być NULL lub poprawnym URL

```
CHECK (RecordingURL IS NULL OR RecordingURL LIKE
'https://%_.%_')
```

Relacje:

- `MeetingID` jest kluczem obcym odwołującym się do `SubjectMeetings(ProductID)`.
- `SubjectID` jest kluczem obcym odwołującym się do `SubjectOnlineSynchronous(SubjectID)` lub `SubjectHybrid(SubjectID)`.
- `PlatformID` jest kluczem obcym odwołującym się do `Platforms(PlatformID)`.

```
CREATE TABLE SubjectOnlineSynchronousMeetings (
   MeetingID int  NOT NULL,
   SubjectID int  NOT NULL,
   StartTime datetime  NOT NULL,
   EndTime datetime  NOT NULL,
   PlatformID int  NOT NULL,
   MeetingURL nvarchar(120)  NULL DEFAULT NULL CHECK (MeetingURL
IS NULL OR MeetingURL LIKE 'https://%_.%_'),
   RecordingURL nvarchar(120)  NULL DEFAULT NULL CHECK
(RecordingURL IS NULL OR RecordingURL LIKE 'https://%_.%_'),
   CONSTRAINT OnlineSynchronousMeetings_unique1 UNIQUE
(MeetingURL),
   CONSTRAINT OnlineSynchronousMeetings_unique2 UNIQUE
(RecordingURL),
   CONSTRAINT date CHECK (StartTime < EndTime),
   CONSTRAINT SubjectOnlineSynchronousMeetings_pk PRIMARY KEY
(MeetingID)
);
```

Subjects Offline Meetings

Opis: Tabela `SubjectOfflineMeetings` przechowuje informacje o spotkaniach stacjonarnych. Każdy rekord reprezentuje jedno spotkanie, jego czas trwania, oraz pokój w którym się ono odbywa.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `SubjectID`: Identyfikator przedmiotu (int, NOT NULL).
- `StartTime`: Identyfikator języka (date, NOT NULL).
- `EndTime`: Identyfikator tłumacza (date, NOT NULL).
- `RoomID`: Identyfikator sali w której odbywa się spotkanie (int, NOT NULL).

Warunki integralności:

- `StartTime` musi być wcześniej niż `EndTime`.

```
CHECK (StartTime < EndTime)
```

Relacje:

- `MeetingID` jest kluczem obcym odwołującym się do `SubjectMeetings(ProductID)`.
- `SubjectID` jest kluczem obcym odwołującym się do `SubjectOffline(SubjectID)` lub `SubjectHybrid(SubjectID)`.
- `RoomID` jest kluczem obcym odwołującym się do `Rooms(RoomID)`.

```
CREATE TABLE SubjectOfflineMeetings (
   MeetingID int  NOT NULL,
   SubjectID int  NOT NULL,
   StartTime datetime  NOT NULL,
   EndTime datetime  NOT NULL,
   RoomID int  NOT NULL,
   CONSTRAINT date CHECK (StartTime < EndTime),
   CONSTRAINT SubjectOfflineMeetings_pk PRIMARY KEY  (MeetingID)
);
```

Subject Meeting Makeup Presence

Opis: Tabela `SubjectMeetingMakeupPresence` przechowuje informacje o obecności na spotkaniach służących odrabianiu obecności. Każdy rekord reprezentuje jedno spotkanie, zawierając szczegóły dotyczące prowadzącego i identyfikatora studenta oraz uzupełnianych spotkań.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).

- `StudentID`: Identyfikator studenta (int, NOT NULL).
- `MakeupFor`: Identyfikator spotkania które się odrabia (int, NOT NULL).

```
CREATE TABLE SubjectMeetingMakeupPresence (
   MeetingID int  NOT NULL,
   StudentID int  NOT NULL,
   MakeupFor int  NOT NULL,
   CONSTRAINT SubjectMeetingMakeupPresence_pk PRIMARY KEY
(MeetingID,StudentID,MakeupFor)
);
```

Subject Meeting Presence

Opis: Tabela `SubjectMeetingPresence` przechowuje informacje o obecności na spotkaniach dotyczących danego przedmiotu. Każdy rekord reprezentuje jedno spotkanie, zawierając szczegóły dotyczące prowadzącego i identyfikatora studenta.

Kolumny:

- `MeetingID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `StudentID`: Identyfikator studenta (int, NOT NULL).

```
CREATE TABLE SubjectMeetingPresence (
   MeetingID int  NOT NULL,
   StudentID int  NOT NULL,
   CONSTRAINT SubjectMeetingPresence_pk PRIMARY KEY
(MeetingID,StudentID)
);
```

Subject Meetings

Opis: Tabela `SubjectMeetings` przechowuje ID produktu związanego z przedmiotami oraz przypisanego prowadzącego i tłumacza.

Kolumny:

- `ProductID`: Klucz główny, identyfikator spotkania (int, NOT NULL).
- `LecturerID`: Identyfikator prowadzącego (int, NOT NULL).
- `Translator`: Identyfikator tłumacza (int, NOT NULL).

Relacje:

- `ProductID` jest kluczem obcym odwołującym się do `Products(ProductID)`.
- `LecturerID` jest kluczem obcym odwołującym się do `Employees(UserID)`.
- `Translator` jest kluczem obcym odwołującym się do `TranslatorsLanguages(TranslatorPass)`.

```
CREATE TABLE SubjectMeetings (
   ProductID int  NOT NULL,
```

```
    LecturerID int  NOT NULL,
    Translator int  NULL,
    CONSTRAINT SubjectMeetings_pk PRIMARY KEY  (ProductID)
);
```

Internships

Opis: Tabela `Internships` przechowuje informacje o przedmiotach będących praktykami. Każdy rekord reprezentuje jeden przedmiot wraz z jego opisem.

Kolumny:

- `InternshipID`: Klucz główny, identyfikator praktyk (int, NOT NULL).
- `Description`: Opis praktyk (nvarchar(max), NOT NULL).

Relacje:

- `InternshipID` jest kluczem obcym odwołującym się do `Subjects(SubjectID)`.

```
CREATE TABLE Internships (
    InternshipID int  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    CONSTRAINT Internships_pk PRIMARY KEY  (InternshipID)
);
```

Internship Presence

Opis: Tabela `InternshipPresence` przechowuje informacje o obecności na praktykach. Każdy rekord reprezentuje jeden przedmiot, identyfikator studenta odbywającego praktyki oraz datę praktyk.

Kolumny:

- `InternshipID`: Klucz główny, identyfikator praktyk (int, NOT NULL).
- `StudentID`: Identyfikator studenta (int, NOT NULL).
- `Date`: Data odbycia praktyk (Date, NOT NULL).

Relacje:

- `InternshipID` jest kluczem obcym odwołującym się do `Internships(InternshipID)`.
- `StudentID` jest kluczem obcym odwołującym się do `Students(UserID)`.

```
CREATE TABLE InternshipPresence (
    InternshipID int  NOT NULL,
    StudentID int  NOT NULL,
    Date date  NOT NULL,
    CONSTRAINT InternshipPresence_pk PRIMARY KEY
(InternshipID,StudentID)
);
```

Translators

Translators

Opis: Tabela `Translators` przechowuje identyfikatory użytkowników będących tłumaczami, wraz ze statusem ich aktywności.

Kolumny:

- `UserID`: Klucz główny, identyfikator użytkownika (int, NOT NULL).
- `Active`: Status aktywności konta (bit, DEFAULT 1).

```
CREATE TABLE Translators (
   UserID int  NOT NULL,
   Active bit  NOT NULL,
   CONSTRAINT Translators_pk PRIMARY KEY  (UserID)
);
```

Languages

Opis: Tabela `Languages` przechowuje dane o językach.

Kolumny:

- `LanguageID`: Klucz główny, identyfikator języka (int, NOT NULL).
- `Name`: Nazwa języka (nvarchar(40), NOT NULL).

Warunki integralności:

- `Name` musi być unikalny.

  ```
  Unique(Name)
  ```

```
CREATE TABLE Languages (
   LanguageID int  NOT NULL,
   Name nvarchar(40)  NOT NULL,
   CONSTRAINT Languages_unique UNIQUE (Name),
   CONSTRAINT NameUnique CHECK (Unique(Name)),
   CONSTRAINT Languages_pk PRIMARY KEY  (LanguageID)
);
```

Translators Languages

Opis: Tabela `TranslatorsLanguages` przechowuje dane o językach obsługiwanych przez tłumaczy.

Kolumny:

- `TranslatorPass`: Połączony identyfikator tłumacza i języka (int, NOT NULL).

- `TranslatorID`: Identyfikator tłumacza (int, NOT NULL).
- `LanguageID`: Identyfikator języka (int, NOT NULL).

Relacje:

- `TranslatorID` jest kluczem obcym odwołującym się do `Translators(UserID)`.
- `LanguageID` jest kluczem obcym odwołującym się do `Language(LanguageID)`.

```
CREATE TABLE TranslatorsLanguages (
   TranslatorPass int  NOT NULL,
   TranslatorID int  NOT NULL,
   LanguageID int  NOT NULL,
   CONSTRAINT TranslatorsLanguages_unique_pair UNIQUE
(TranslatorID, LanguageID),
   CONSTRAINT TranslatorsLanguages_pk PRIMARY KEY
(TranslatorPass)
);
```

Orders

Orders

Opis: Tabela `Orders` przechowuje dane o zamówieniach składanych przez studentów.

Kolumny:

- `OrderID`: Klucz główny, identyfikator zamówienia (int, NOT NULL).
- `StudentID`: Identyfikator studenta (int, NOT NULL).
- `PaymentURL`: Link do płatności (nvarchar(120), NULL).
- `OrderDate`: Data złożenia zamówienia (date, DEFAULT GETDATE()).

Warunki integralności:

- `PaymentURL` musi być unikalny i jest poprawny.

```
CHECK (PaymentURL LIKE 'https://%_.%_')

UNIQUE (PaymentURL)
```

Relacje:

- `StudentID` jest kluczem obcym odwołującym się do `Students(UserID)`.
- `OrderID` jest kluczem obcym odwołującym się do `OrderDetails(OrderID)`.

```
CREATE TABLE Orders (
   OrderID int  NOT NULL,
   StudentID int  NOT NULL,
   PaymentURL nvarchar(120)  NOT NULL CHECK (PaymentURL LIKE
'https://%_.%_'),
```

```
   OrderDate datetime  NOT NULL DEFAULT GETDATE(),
   CONSTRAINT Orders_payment_unique UNIQUE (PaymentURL),
   CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);
```

Shopping Cart

Opis: Tabela `ShoppingCart` przechowuje dane o produktach dodanych do koszyków studentów.

Kolumny:

- `StudentID`: Identyfikator studenta (int, NOT NULL).
- `ProductID`: Identyfikator produktu (int, NOT NULL).

Relacje:

- `ProductID` jest kluczem obcym odwołującym się do `Products(ProductID)`.
- `StudentID` jest kluczem obcym odwołującym się do `Students(UserID)`.

```
CREATE TABLE ShoppingCart (
   StudentID int  NOT NULL,
   ProductID int  NOT NULL,
   CONSTRAINT ShoppingCart_pk PRIMARY KEY  (StudentID,ProductID)
);
```

Order Statuses

Opis: Tabela `OrderStatuses` przechowuje dane o statusach zamówień.

Kolumny:

- `StatusID`: Klucz główny, identyfikator statusu (int, NOT NULL).
- `Name`: Nazwa statusu (nvarchar(40), NOT NULL).

```
CREATE TABLE OrderStatuses (
   StatusID int  NOT NULL,
   Name nvarchar(40)  NOT NULL,
   CONSTRAINT OrderStatuses_pk PRIMARY KEY  (StatusID)
);
```

Order Details

Opis: Tabela `OrderDetails` przechowuje szczegóły zamówień, w tym produkty, ich ceny i statusy.

Kolumny:

- `OrderID`: Identyfikator zamówienia (int, NOT NULL).

- `ProductID`: Identyfikator produktu (int, NOT NULL).
- `PricePaid`: Cena produktu (money, NOT NULL).
- `StatusID`: Identyfikator statusu płatności (int, NOT NULL).

Warunki integralności:

- `PricePaid` nie może być ujemna.

```
CHECK (PricePaid >= 0)
```

Relacje:

- `OrderID` jest kluczem obcym odwołującym się do `Orders(OrderID)`.
- `ProductID` jest kluczem obcym odwołującym się do `Products(ProductID)`.
- `StatusID` jest kluczem obcym odwołującym się do `OrderStatuses(StatusID)`.

```
CREATE TABLE OrderDetails (
   OrderID int  NOT NULL,
   ProductID int  NOT NULL,
   PricePaid money  NOT NULL CHECK (PricePaid >= 0),
   StatusID int  NOT NULL,
   CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID)
);
```

Widoki

CanGetStipend

Zwraca listę studentów (`StudentID`) z wyliczoną średnią ocen (`AvarageGrade`) we wszystkich przedmiotach, gdzie średnia wynosi co najmniej 4.0. Autor: Bartosz Ludwin.

```
CREATE VIEW CanGetStipend
AS
SELECT
   sg.StudentID,
   AVG(sg.Grade) AS AvarageGrade
FROM SubjectGrades AS sg
GROUP BY
   sg.StudentID
HAVING
   AVG(sg.Grade) >= 4.0;
```

StudyAttendanceList

Zwraca listę obecności studentów (`StudentID`) na spotkaniach (`MeetingID`) oraz do jakiego kierunku one należą (`StudyID`). Autor: Bartosz Ludwin.

```
select
ESM.MeetingID as MeetingID,
ESM.StudyID as StudyID,
SMP.StudentID as StudentID
from EveryStudyMeeting ESM
join SubjectMeetingPresence SMP on SMP.MeetingID = ESM.MeetingID
union
select
ESM.MeetingID as MeetingID,
ESM.StudyID as StudyID,
SMMP.StudentID as StudentID
from EveryStudyMeeting ESM
join SubjectMeetingMakeupPresence SMMP on SMMP.MeetingID = ESM.MeetingID
```

CourseAttendanceList

Zwraca listę obecności studentów (`StudentID`) na spotkaniach (`MeetingID`) oraz do jakiego kursu one należą (`CourseID`). Autor: Bartosz Ludwin.

```
select
ECM.MeetingID as MeetingID,
ECM.CourseID as CourseID,
SMP.StudentID as StudentID
from EveryCourseMeeting ECM
join SubjectMeetingPresence SMP on SMP.MeetingID = ECM.MeetingID

union

select
ECM.MeetingID as MeetingID,
ECM.CourseID as StudyID,
CMMP.StudentID as StudentID
from EveryCourseMeeting ECM
join CourseMeetingMakeupPresence CMMP on CMMP.MeetingID = ECM.MeetingID
```

StudentCoursePresencePercentage

Zwraca informacje o procentowym udziale obecności każdego studenta (`StudentID`) w poszczególnych kursach (`CourseID`), obliczonym jako liczba obecności na spotkaniach podzielona przez liczbę wszystkich spotkań w danym kursie. Autor: Bartosz Ludwin.

```
select
CAL.StudentID,
CAL.CourseID,
100 * (select COUNT(*) from CourseAttendanceList CAL2
  where CAL2.CourseID = CAL.CourseID)
      /
      NULLIF((select COUNT(*) from EveryCourseMeeting ECM2
              where ECM2.CourseID = CAL.CourseID
      ), 0) as PresencePercentage
from CourseAttendanceList CAL;
```

StudentStudyPresencePercentage

Zwraca informacje o procentowym udziale obecności każdego studenta (`StudentID`) na danych studiach (`StudyID`), obliczonym jako liczba obecności na spotkaniach podzielona przez liczbę wszystkich spotkań przypisanych do danego kierunku studiów. Autor: Bartosz Ludwin

```
select
SAL.StudentID,
SAL.StudyID,
100 * (select COUNT(*) from StudyAttendanceList SAL2
  where SAL2.StudyID = SAL.StudyID)
      /
      (select COUNT(*) from EveryStudyMeeting ESM2
              where ESM2.StudyID = SAL.StudyID
      ) as PresencePercentage
from StudyAttendanceList SAL
```

CourseProduct

Zwraca listę wszystkich kursów wraz z ich wszystkimi informacjami z tabel `Courses` i `Products`. -Bartłomiej Kaczyński

```
CREATE VIEW CourseProduct
```

```
AS
SELECT
p.ProductID,
c.CoordinatorID,
p.Name,
p.Description,
p.Price,
p.Capacity,
p.Available
FROM Products AS p
JOIN Courses AS c
ON p.ProductID = c.ProductID;
```

EveryModuleMeeting

Zwraca listę wszystkich spotkań kursowych wraz z ich typem (`ModuleID`, `MeetingID`, Typ spotkania, `LecturerID`, Temat spotkania, Tłumacz spotkania (jeśli jest tłumaczone),`StartTime`, `EndTime`). -Bartłomiej Kaczyński

```
CREATE VIEW EveryModuleMeeting
AS
SELECT
   oamm.ModuleID,
   mm.MeetingID,
   'OnlineAsynchronous' as [Type],
   mm.LecturerID,
   mm.Topic,
   mm.Translator,
   NULL as [StartTime],
   NULL as [EndTime]
FROM ModuleMeetings AS mm
JOIN OnlineAsynchronousModuleMeetings AS oamm
ON mm.MeetingID = oamm.MeetingID
UNION
SELECT
   osmm.ModuleID,
   mm.MeetingID,
   'OnlineSynchronous' as [Type],
   mm.LecturerID,
   mm.Topic,
   mm.Translator,
   osmm.StartTime,
   osmm.EndTime
FROM ModuleMeetings AS mm
JOIN OnlineSynchronousModuleMeetings AS osmm
ON mm.MeetingID = osmm.MeetingID
UNION
SELECT
   omm.ModuleID,
   mm.MeetingID,
   'Offline' as [Type],
   mm.LecturerID,
```

```
    mm.Topic,
    mm.Translator,
    omm.StartTime,
    omm.EndTime
FROM ModuleMeetings AS mm
JOIN OfflineModuleMeetings AS omm
ON mm.MeetingID = omm.MeetingID;
```

EveryCourseModule

Zwraca listę wszystkich modułów wraz z ich typem (`CourseID`, `ModuleID`, Typ modułu, nazwa modułu, opis modułu) . -Bartłomiej Kaczyński

```
CREATE VIEW EveryCourseModule
AS
SELECT
cm.CourseID,
cm.ModuleID,
ct.Name,
cm.Name,
cm.Description
FROM CourseModules AS cm
JOIN ModuleTypes AS mt
ON cm.TypeID = mt.TypeID
```

EveryCourseMeeting

Zwraca listę wszystkich spotkań na wszystkich kursach wraz z typem modułu i typem spotkania (`CourseID`, `ModuleID`, Typ modułu, `MeetingID`, Typ spotkania, `StartTime`, `EndTime`) . -Bartłomiej Kaczyński

```
CREATE VIEW EveryCourseMeeting
AS
SELECT
ecm.CourseID,
ecm.ModuleID,
ecm.ModuleType,
emm.MeetingID,
emm.Type as MeetingType,
emm.StartTime,
emm.EndTime
FROM EveryCourseModule AS ecm
JOIN EveryModuleMeeting AS emm
ON ecm.ModuleID = emm.ModuleID;
```

StudyProduct

Zwraca listę wszystkich studiów wraz z ich wszystkimi informacjami z tabel `Studies` i `Products.` -Bartłomiej Kaczyński

```sql
CREATE VIEW StudyProduct
AS
SELECT
p.ProductID as StudyID,
p.Name,
p.Description,
p.Price,
p.Capacity,
p.Available,
s.SupervisorID
FROM Products p
JOIN Studies s ON s.ProductID = p.ProductID;
```

EveryStudyMeeting

Zwraca listę wszystkich spotkań (`MeetingID`, `LecturerID`, `Translator`, `Topic`) dla każdego kierunku studiów (`StudyID`). Autor: Bartosz Ludwin.

```sql
SELECT
    s.ProductID AS StudyID,
    SUB.SubjectID AS SubjectID,
    SOSM.MeetingID AS MeetingID,
    SOSM.StartTime AS StartTime,
    SOSM. EndTime AS EndTime,
    'Online' AS MeetingType
FROM Studies AS S
JOIN Subjects sub on sub.StudyID = s.ProductID
JOIN SubjectonlineSynchronousMeetings SOSM on SOSM.SubjectID = sub.SubjectID
union
SELECT
    S.ProductID AS StudyID,
    SUB.SubjectID AS SubjectID,
    SOM.MeetingID AS MeetingID,
    SOM.StartTime AS StartTime,
    SOM.EndTime AS EndTime,
    'Offline' AS MeetingType
FROM Studies AS S
JOIN Subjects sub on sub.StudyID = s.ProductID
JOIN SubjectofflineMeetings SOM on SOM.SubjectID = sub.SubjectID
```

RemainingPlacesOnCourse

Zwraca dla każdego produktu (`ProductID`) informacje o nazwie, pojemności (`Capacity`) oraz wyliczoną liczbę wolnych miejsc (`RemainingSlots`), a także informację, czy produkt jest dostępny (`Available`). Autor: Bartosz Ludwin.

```
SELECT
    p.ProductID,
    p.Name,
    p.Capacity,
    (p.Capacity
      - ISNULL(
          (
            SELECT COUNT(*)
            FROM OrderDetails AS od
            WHERE od.ProductID = p.ProductID
          ),
          0
        )
    ) AS RemainingSlots,
    p.Available
FROM Products AS p;
```

InternshipParticipants

Zwraca listę praktyk (`InternshipID`) wraz z ich nazwami (`Name`) i datami ich odbycia (`Date`), a także danymi studentów odbywających te praktyki (`StudentID`, `FirstName`, `LastName`). -Kacper Wąchała

```
CREATE VIEW InternshipParticipants
AS
SELECT ip.InternshipID,
s.Name,
ip.Date,
ip.StudentID,
u.FirstName,
u.LastName
FROM InternshipPresence AS ip
JOIN Internships as i
ON ip.InternshipID = i.InternshipID
JOIN Subjects as s
ON i.InternshipID = s.SubjectID
JOIN Students as st
ON ip.StudentID = st.UserID
JOIN Users as u
ON st.UserID = u.UserID
```

CourseMeetingsAttendanceList

Zwraca listę wszystkich spotkań kursowych wraz z ich typem (`MeetingID`, `Topic`, `Type`), a także danymi studentów (`StudentID`, `FirstName`, `LastName`) będących na tych spotkaniach -Kacper Wąchała

```
CREATE VIEW CourseMeetingsAttendanceList
AS
```

```
SELECT emm.MeetingID,
emm.Topic,
emm.Type,
cmp.StudentID,
u.FirstName,
u.LastName
FROM EveryModuleMeeting AS emm
JOIN CourseMeetingPresence AS cmp
ON emm.MeetingID = cmp.MeetingID
JOIN Students AS s
ON cmp.StudentID = s.UserID
JOIN Users as u
ON s.UserID = u.UserID
```

SubjectMeetingsAttendanceList

Zwraca listę wszystkich spotkań przedmiotowych (`MeetingID`, `Topic`), a także dane studentów (`StudentID`, `FirstName`, `LastName`) będących na tych spotkaniach -Kacper Wąchała

```
CREATE VIEW SubjectMeetingsAttendanceList
AS
SELECT esm.MeetingID,
s.Name,
esm.MeetingType,
smp.StudentID,
u.FirstName,
u.LastName
FROM EveryStudyMeeting as esm
JOIN Subjects AS s
ON (esm.SubjectID = s.SubjectID)
JOIN SubjectMeetingPresence AS smp
ON esm.MeetingID = smp.MeetingID
JOIN Students as st
ON smp.StudentID = st.UserID
JOIN Users as u
ON st.UserID = u.UserID;
```

StudentStudyMeetings

Zwraca listę wszystkich spotkań przedmiotowych (`MeetingID`), na które zapisani są dani studenci (`StudentID`), czyli te spotkania w których mogą uczestniczyć. Bierze pod uwagę zasadę 3 dni. -Bartłomiej Kaczyński

```
CREATE VIEW StudentStudyMeetings
AS
SELECT o.StudentID, esm.MeetingID
FROM OrderDetails od
JOIN EveryStudyMeeting esm ON od.ProductID = esm.MeetingID
JOIN Orders o ON od.OrderID = o.OrderID
AND DATEDIFF(day,o.OrderDate,esm.StartTime) >= 3
JOIN OrderStatuses os ON os.StatusID = od.StatusID
```

```
AND os.Name IN ('Done','Given')
```

StudentModuleMeetings

Zwraca listę wszystkich spotkań kursowych (`MeetingID`), na które zapisani są dani studenci (`StudentID`), czyli te spotkania w których mogą uczestniczyć. Bierze pod uwagę zasadę 3 dni. -Bartłomiej Kaczyński

```
CREATE VIEW StudentModuleMeetings
AS
SELECT o.StudentID, ecm.MeetingID
FROM OrderDetails od
JOIN EveryCourseMeeting ecm ON od.ProductID = ecm.CourseID
JOIN Orders o ON od.OrderID = o.OrderID
AND ecm.StartTime IS NULL OR
(ecm.StartTime IS NOT NULL AND
DATEDIFF(day,o.OrderDate,ecm.StartTime) >= 3)
JOIN OrderStatuses os ON os.StatusID = od.StatusID
AND os.Name IN ('Done','Given')
```

EmployeeInformation

Zwraca listę wszystkich pracowników wraz z pełnioną przez nich rolą, a także ich danymi (`UserID`, `Role`, `FirstName`, `LastName`, `Email`, `Phone`, `Street`, `ZipCode`, `City`, `Country`) -Kacper Wąchała

```
CREATE VIEW EmployeeInformation
AS
SELECT e.UserID,
er.RoleName AS Role,
u.FirstName,
u.LastName,
u.Email,
u.Phone,
a.Street,
a.ZipCode,
cit.Name as City,
cnt.Name as Country
FROM Employees AS e
JOIN Users AS u
ON e.UserID = u.UserID
JOIN EmployeeRoles AS er
ON e.RoleID = er.RoleID
LEFT JOIN Addresses AS a
ON u.UserID = a.UserID
LEFT JOIN Cities AS cit
ON a.CityID = cit.CityID
LEFT JOIN Countries AS cnt
ON cit.CountryID = cnt.CountryID
```

StudentInformation

Zwraca listę wszystkich studentów wraz z ich danymi (`UserID`, `FirstName`, `LastName`, `Email`, `Phone`, `Street`, `ZipCode`, `City`, `Country`) -Kacper Wąchała

```
CREATE VIEW StudentInformation
AS
SELECT st.UserID,
u.FirstName,
u.LastName,
u.Email,
u.Phone,
a.Street,
a.ZipCode,
cit.Name as City,
cnt.Name as Country
FROM Students AS st
JOIN Users AS u
ON st.UserID = u.UserID
LEFT JOIN Addresses AS a
ON u.UserID = a.UserID
LEFT JOIN Cities AS cit
ON a.CityID = cit.CityID
LEFT JOIN Countries AS cnt
ON cit.CountryID = cnt.CountryID
```

TranslatorInformation

Zwraca listę wszystkich translatorów wraz z ich danymi oraz językami które umieją (`UserID`, `Email` , `FirstName`, `LastName`,`Phone`, `Active`, `TranslatorPass`, `Language`) -Bartłomiej Kaczyński

```
CREATE VIEW TranslatorInformation AS
SELECT u.UserID, u.Email, u.FirstName,
u.LastName, u.Phone, t.active, tl.TranslatorPass,
l.Name as Language
FROM Users u
JOIN Translators t on u.UserID = t.UserID
LEFT JOIN TranslatorsLanguages tl ON t.UserID = tl.TranslatorID
LEFT JOIN Languages l ON tl.LanguageID = l.LanguageID
```

Procedury

Create Student

Procedura `CreateStudent` tworzy nowego użytkownika typu student, zapisując jego dane w tabeli `users` oraz `students`. Procedura zwraca ID nowo utworzonego użytkownika poprzez argument `@userID`. Autor: Bartosz Ludwin.

Argumenty procedury:

- `@email` – Adres e-mail studenta.
- `@password` – Hasło studenta.
- `@firstName` – Imię studenta.
- `@lastName` – Nazwisko studenta.
- `@phone` – Numer telefonu studenta.
- `@userID` – Zwracana wartość ID nowo utworzonego studenta.

```
CREATE PROCEDURE CreateStudent
@email NVARCHAR(40),
@password NVARCHAR(40),
@firstName NVARCHAR(40),
@lastName NVARCHAR(40),
@phone NVARCHAR(14),
@userID INT OUTPUT
AS BEGIN
DECLARE @inserted_user TABLE (id INT);
INSERT INTO users (email, password, firstName, lastName, phone)
OUTPUT INSERTED.UserID INTO @inserted_user
VALUES (@email, @password, @firstName, @lastName, @phone);
SELECT @userID = id FROM @inserted_user;
INSERT INTO students (UserID)
VALUES (@userID);
END
```

Delete Student

Procedura `DeleteStudent` usuwa studenta z bazy danych wraz ze wszystkimi powiązanymi danymi, takimi jak produkty w koszyku oraz adresy. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza istnienie student przed usunięciem, jeśli student nie istnieje, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@student_id` – ID studenta, którego dane mają zostać usunięte.

```
CREATE PROCEDURE DeleteStudent
@student_id INT
AS
BEGIN
IF @student_id NOT IN (SELECT UserID FROM students)
THROW 50000, 'Student not found', 11;
DELETE ShoppingCart WHERE StudentID = @student_id;
DELETE Addresses WHERE UserID = @student_id;
DELETE Students WHERE UserID = @student_id;
DELETE Users WHERE UserID = @student_id;
END
```

Assign Student Address

Procedura `AssignStudentAddress` przypisuje adres korespondencyjny studentowi, zapisując jego dane w tabeli `Addresses`, a także dodając miasto do `Cities`, czy kraj do `Countries`, jeśli nie istnieją. Autor: Kacper Wąchała

Argumenty procedury:

- `@studentID` – ID studenta.
- `@street` – Ulica, numer budynku i numer mieszkania.
- `@zipCode` – Kod pocztowy.
- `@city` – Nazwa miasta.
- `@country` – Nazwa kraju..

```
CREATE PROCEDURE AssignStudentAddress
   @studentID INT,
   @street NVARCHAR(80),
   @zipCode NVARCHAR(10),
   @city NVARCHAR(80),
   @country NVARCHAR(69)
AS
BEGIN
   DECLARE @countryID INT;
   DECLARE @cityID INT;
   IF @studentID NOT IN (SELECT UserID FROM Students)
      THROW 50000, 'Student not found', 11;
   IF @country NOT IN (SELECT Name FROM Countries)
   BEGIN
      DECLARE @insertedCountry TABLE (CountryID INT);
      INSERT INTO Countries (Name)
      OUTPUT INSERTED.CountryID INTO @insertedCountry
      VALUES (@country);
```

```
      SELECT @countryID = CountryID FROM @insertedCountry;
   END
   ELSE
   BEGIN
      SELECT @countryID = CountryID FROM Countries WHERE Name =
@country;
   END;
   IF @city NOT IN (SELECT Name FROM Cities)
   BEGIN
      DECLARE @insertedCity TABLE (CityID INT);
      INSERT INTO Cities (Name, CountryID)
      OUTPUT INSERTED.CityID INTO @insertedCity
      VALUES (@city, @countryID);
      SELECT @cityID = CityID FROM @insertedCity;
   END
   ELSE
   BEGIN
      SELECT @cityID = CityID FROM Cities WHERE Name = @city;
   END;
   IF @studentID IN (SELECT UserID FROM Addresses)
   BEGIN
      UPDATE Addresses
      SET Street = @street, ZipCode = @zipCode, CityID = @cityID
      WHERE UserID = @studentID;
   END
   ELSE
   BEGIN
      SET IDENTITY_INSERT Addresses ON
      INSERT INTO Addresses (UserID, Street, ZipCode, CityID)
      VALUES (@studentID, @street, @zipCode, @cityID);
      SET IDENTITY_INSERT Addresses OFF
   END;
END;
```

Create Coordinator

Procedura `CreateCoordinator` tworzy nowego użytkownika typu koordynator, zapisując jego dane w tabeli `users` oraz `employees`. Procedura przypisuje użytkownikowi rolę koordynatora o ID `10` (ID przypisane dla koordynatora) i zwraca jego ID poprzez argument `@userID`. Autor: Bartosz Ludwin.

Argumenty procedury:

- `@email` – Adres e-mail koordynatora.
- `@password` – Hasło koordynatora.
- `@firstName` – Imię koordynatora.
- `@lastName` – Nazwisko koordynatora.
- `@phone` – Numer telefonu koordynatora.
- `@userID` – Zwracana wartość ID nowo utworzonego koordynatora.

```
CREATE PROCEDURE CreateCoordinator
@email NVARCHAR(40),
@password NVARCHAR(40),
@firstName NVARCHAR(40),
@lastName NVARCHAR(40),
@phone NVARCHAR(14),
@userID INT OUTPUT
AS BEGIN
DECLARE @inserted_user TABLE (id INT);
INSERT INTO users (email, password, firstName, lastName, phone)
OUTPUT INSERTED.UserID INTO @inserted_user
VALUES (@email, @password, @firstName, @lastName, @phone);
SELECT @userID = id FROM @inserted_user;
INSERT INTO Employees(UserID, RoleID)
VALUES (@userID, 10); -- let coordinator role id be 10
END
```

Delete Coordinator

Procedura `DeleteCoordinator` usuwa koordynatora z bazy danych na podstawie podanego ID. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy użytkownik ma przypisaną rolę koordynatora, jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@coordinator_id` – ID koordynatora, który ma zostać usunięty.

```
CREATE PROCEDURE DeleteCoordinator
@coordinator_id INT
AS BEGIN
IF @coordinator_id NOT IN (SELECT UserID FROM Employees E
join EmployeeRoles ER on E.RoleID = ER.RoleID
WHERE ER.RoleID = 'Coordinator')
THROW 50000, 'Coordinator not found', 11;
DELETE Users WHERE UserID = @coordinator_id;
DELETE Employees WHERE UserID = @coordinator_id;
END
```

Create Principal

Procedura `CreatePrincipal` tworzy nowego użytkownika typu dyrektor, zapisując jego dane w tabeli `users` oraz `employees`. Procedura przypisuje użytkownikowi rolę dyrektora o

ID 1 (ID przypisywane dyrektorowi) i zwraca jego ID poprzez argument `@userID`. Autor: Bartosz Ludwin.

Argumenty procedury:

- `@email` – Adres e-mail dyrektora.
- `@password` – Hasło dyrektora.
- `@firstName` – Imię dyrektora.
- `@lastName` – Nazwisko dyrektora.
- `@phone` – Numer telefonu dyrektora.
- `@userID` – Zwracana wartość ID nowo utworzonego dyrektora.

```
CREATE PROCEDURE CreatePrincipal
@email NVARCHAR(40),
@password NVARCHAR(40),
@firstName NVARCHAR(40),
@lastName NVARCHAR(40),
@phone NVARCHAR(14),
@userID INT OUTPUT
AS BEGIN
DECLARE @inserted_user TABLE (id INT);
INSERT INTO users (email, password, firstName, lastName, phone)
OUTPUT INSERTED.UserID INTO @inserted_user
VALUES (@email, @password, @firstName, @lastName, @phone);
SELECT @userID = id FROM @inserted_user;
INSERT INTO Employees(UserID, RoleID)
VALUES (@userID, 1); -- let principal role id be 1
END
```

Delete Principal

Procedura `DeletePrincipal` usuwa dyrektora z bazy danych na podstawie podanego ID. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy użytkownik ma przypisaną rolę dyrektora, jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@principal_id` – ID dyrektora, który ma zostać usunięty.

```sql
CREATE PROCEDURE DeletePrincipal
@principal_id INT
AS BEGIN
IF @principal_id NOT IN (SELECT UserID FROM Employees E
join EmployeeRoles ER on E.RoleID = ER.RoleID
WHERE ER.RoleID = 'Principal')
THROW 50000, 'Principal not found', 11;
DELETE Users WHERE UserID = @principal_id;
DELETE Employees WHERE UserID = @principal_id;
END
```

Create Translator

Procedura `CreateTranslator` tworzy nowego użytkownika typu tłumacz, zapisując jego dane w tabeli `users`, a także przypisuje odpowiedni język z tabeli `Languages`. Procedura zwraca ID nowo utworzonego tłumacza poprzez argument `@userID`. Autor: Bartosz Ludwin.

Argumenty procedury:

- `@email` – Adres e-mail tłumacza.
- `@password` – Hasło tłumacza.
- `@firstName` – Imię tłumacza.
- `@lastName` – Nazwisko tłumacza.
- `@phone` – Numer telefonu tłumacza.
- `@language` – Język, którym posługuje się tłumacz.
- `@userID` – Zwracana wartość ID nowo utworzonego tłumacza.

```
CREATE PROCEDURE CreateTranslator
@email NVARCHAR(40),
@password NVARCHAR(40),
@firstName NVARCHAR(40),
@lastName NVARCHAR(40),
@phone NVARCHAR(14),
@language NVARCHAR(40),
@userID INT OUTPUT
AS BEGIN
DECLARE @inserted_user TABLE (id INT);
DECLARE @LanguageID INT;
SELECT @LanguageID = LanguageID from Languages where Languages.Name = @language
INSERT INTO users (email, password, firstName, lastName, phone)
OUTPUT INSERTED.UserID INTO @inserted_user
VALUES (@email, @password, @firstName, @lastName, @phone);
SELECT @userID = id FROM @inserted_user
INSERT INTO Translators(UserID, Active)
VALUES (@userID, 1);
INSERT INTO TranslatorsLanguages(TranslatorID, LanguageID)
VALUES (@userID, @LanguageID)
END
```

Delete Translator

Procedura `DeleteTranslator` usuwa tłumacza z bazy danych na podstawie podanego ID. Autor: Bartosz Ludwin

Warunki integralnościowe:

- Procedura sprawdza, czy użytkownik ma przypisaną rolę tłumacza, jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@translator_id` – ID tłumacza, który ma zostać usunięty.

```
CREATE PROCEDURE DeleteTranslator
@translator_id INT
AS BEGIN
IF @translator_id NOT IN (SELECT UserID FROM Employees E
join EmployeeRoles ER on E.RoleID = ER.RoleID
WHERE ER.RoleID = 'Translator')
THROW 50000, 'Translator not found', 11;
DELETE Users WHERE UserID = @translator_id;
DELETE Translators WHERE UserID = @translator_id;
DELETE TranslatorsLanguages WHERE TranslatorID = @translator_id;
END
```

Add Translator Language

Procedura `AddTranslatorLanguage` dodaje dodatkowy język do tłumacza a podanym ID. Autor: Bartłomiej Kaczyński

Warunki integralnościowe:

- Procedura sprawdza, czy podany język istnieje w bazie (co wykrywa np. literówki), jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy użytkownik ma przypisaną rolę tłumacza, jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy użytkownik już ma przypisany ten język, jeśli tak, rzuca błąd o kodzie `50002`.

Argumenty procedury:

- `@TranslatorID` – ID tłumacza, do którego ma zostać przypisany język.
- `@LanguageID` – Język który ma zostać przypisany do tłumacza.
- `@TranslatorPass` – Zwracana wartość ID przepustki na tłumaczenie danego języka.

```
CREATE PROCEDURE AddTranslatorLanguage
@TranslatorID INT,
@Language NVARCHAR(40),
@TranslatorPass INT OUTPUT
AS BEGIN
   IF dbo.GetLanguageID(@Language) IS NULL
       THROW 50000, 'Language not in the database', 11;
   IF @TranslatorID NOT IN (SELECT UserID FROM Translators)
       THROW 50001, 'Specified translator does not exist', 11;
   IF dbo.GetLanguageID(@Language) IN
      (SELECT LanguageID FROM TranslatorsLanguages WHERE
TranslatorID = @TranslatorID)
      THROW 50002, 'This language is already assigned to this
translator', 16

   INSERT INTO TranslatorsLanguages(TranslatorID, LanguageID)
VALUES
   (@TranslatorID,dbo.GetLanguageID(@Language))
END
```

Remove Translator Language

Procedura `RemoveTranslatorLanguage` usuwa język z tłumacza a podanym ID. Autor: Bartłomiej Kaczyński

Warunki integralnościowe:

- Procedura sprawdza, czy podany język istnieje w bazie (co wykrywa np. literówki), jeśli nie, rzuca błąd o kodzie `50000`.

- Procedura sprawdza, czy użytkownik ma przypisaną rolę tłumacza, jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy użytkownik ma przypisany ten język, jeśli nie, rzuca błąd o kodzie `50002`.

Argumenty procedury:

- `@TranslatorID` – ID tłumacza, do którego ma zostać przypisany język.
- `@LanguageID` – Język który ma zostać usunięty z tłumacza.
- `@TranslatorPass` – Zwracana wartość ID przepustki na tłumaczenie danego języka.

```
CREATE PROCEDURE RemoveTranslatorLanguage
@TranslatorID INT,
@Language NVARCHAR(40)
AS BEGIN
   IF dbo.GetLanguageID(@Language) IS NULL
       THROW 50000, 'Language not in the database', 11;
   IF @TranslatorID NOT IN (SELECT UserID FROM Translators)
       THROW 50001, 'Specified translator does not exist', 11;
   IF dbo.GetLanguageID(@Language) NOT IN
      (SELECT LanguageID FROM TranslatorsLanguages WHERE
TranslatorID = @TranslatorID)
      THROW 50002, 'This translator is already not assigned to
this language', 16

   DELETE TranslatorsLanguages WHERE LanguageID =
dbo.GetLanguageID(@Language) AND TranslatorID = @TranslatorID
END
```

Create Lecturer

Procedura `CreateLecturer` tworzy nowego użytkownika typu wykładowca, zapisując jego dane w tabeli `users` oraz `employees`. Procedura przypisuje użytkownikowi rolę wykładowcy o ID `1000` (ID przypisane dla wykładowcy) i zwraca jego ID poprzez argument `@userID`. Autor: Bartosz Ludwin.

Argumenty procedury:

- `@email` – Adres e-mail wykładowcy.
- `@password` – Hasło wykładowcy.
- `@firstName` – Imię wykładowcy.
- `@lastName` – Nazwisko wykładowcy.
- `@phone` – Numer telefonu wykładowcy.
- `@userID` – Zwracana wartość ID nowo utworzonego wykładowcy.

```sql
CREATE PROCEDURE CreateLecturer
@email NVARCHAR(40),
@password NVARCHAR(40),
@firstName NVARCHAR(40),
@lastName NVARCHAR(40),
@phone NVARCHAR(14),
@userID INT OUTPUT
AS BEGIN
DECLARE @inserted_user TABLE (id INT);
INSERT INTO users (email, password, firstName, lastName, phone)
OUTPUT INSERTED.UserID INTO @inserted_user
VALUES (@email, @password, @firstName, @lastName, @phone);
SELECT @userID = id FROM @inserted_user;
INSERT INTO Employees(UserID, RoleID)
VALUES (@userID, 1000); -- let Lecturer role id be 1000
END
```

Delete Lecturer

Procedura `DeleteLecturer` usuwa wykładowcę z bazy danych na podstawie podanego ID. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy użytkownik ma przypisaną rolę wykładowcy, jeśli nie, rzuca błąd o kodzie `50000`.
- Usunięcie użytkownika odbywa się z tabel `users` i `employees`.

Argumenty procedury:

- `@lecturer_id` – ID wykładowcy, który ma zostać usunięty.

```sql
CREATE PROCEDURE DeleteLecturer
@lecturer_id INT
AS BEGIN
IF @lecturer_id NOT IN (SELECT UserID FROM Employees E
join EmployeeRoles ER on E.RoleID = ER.RoleID
WHERE ER.RoleID = 'Lecturer')
THROW 50000, 'Lecturer not found', 11;
DELETE Users WHERE UserID = @lecturer_id;
DELETE Employees WHERE UserID = @lecturer_id;
END
```

Create Instructor

Procedura `CreateInstructor` tworzy nowego użytkownika typu instruktor, zapisując jego dane w tabeli `users` oraz `employees`. Procedura przypisuje użytkownikowi rolę instruktora o ID `10000` (ID przypisane dla instruktora) i zwraca jego ID poprzez argument `@userID`. Autor: Bartosz Ludwin.

Argumenty procedury:

- `@email` – Adres e-mail instruktora.
- `@password` – Hasło instruktora.
- `@firstName` – Imię instruktora.
- `@lastName` – Nazwisko instruktora.
- `@phone` – Numer telefonu instruktora.
- `@userID` – Zwracana wartość ID nowo utworzonego instruktora.

```sql
CREATE PROCEDURE CreateInstuctor
@email NVARCHAR(40),
@password NVARCHAR(40),
@firstName NVARCHAR(40),
@lastName NVARCHAR(40),
@phone NVARCHAR(14),
@userID INT OUTPUT
AS BEGIN
DECLARE @inserted_user TABLE (id INT);
INSERT INTO users (email, password, firstName, lastName, phone)
OUTPUT INSERTED.UserID INTO @inserted_user
VALUES (@email, @password, @firstName, @lastName, @phone);
SELECT @userID = id FROM @inserted_user;
INSERT INTO Employees(UserID, RoleID)
VALUES (@userID, 10000); -- let Instuctor role id be 10000
END
```

Delete Instructor

Procedura `DeleteInstructor` usuwa instruktora z bazy danych na podstawie podanego ID. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy użytkownik ma przypisaną rolę instruktora, jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@instructor_id` – ID instruktora, który ma zostać usunięty.

```sql
CREATE PROCEDURE DeleteInstuctor
@instructor_id INT
AS BEGIN
IF @instructor_id NOT IN (SELECT UserID FROM Employees E
join EmployeeRoles ER on E.RoleID = ER.RoleID
WHERE ER.RoleID = 'Instuctor')
THROW 50000, 'Instuctor not found', 11;
DELETE Users WHERE UserID = @instructor_id;
DELETE Employees WHERE UserID = @instructor_id;
END
```

Create Secretary

Procedura `CreateSecretary` tworzy nowego użytkownika typu sekretarka, zapisując jej dane w tabeli `users` oraz `employees`. Procedura przypisuje użytkownikowi rolę instruktora o ID `100000` (ID przypisane dla sekretarki) i zwraca jego ID poprzez argument `@userID`. Autor: Bartosz Ludwin.

Argumenty procedury:

- `@email` – Adres e-mail sekretarki.
- `@password` – Hasło sekretarki.
- `@firstName` – Imię sekretarki.
- `@lastName` – Nazwisko sekretarki.
- `@phone` – Numer telefonu sekretarki.
- `@userID` – Zwracana wartość ID nowo utworzonego instruktora.

```
CREATE PROCEDURE CreateSecretary
@email NVARCHAR(40),
@password NVARCHAR(40),
@firstName NVARCHAR(40),
@lastName NVARCHAR(40),
@phone NVARCHAR(40),
@userID INT OUTPUT
AS BEGIN
DECLARE @inserted_user TABLE (id INT);
INSERT INTO users (email, password, firstName, lastName, phone)
OUTPUT INSERTED. UserID INTO @inserted_user
VALUES (@email, @password, @firstName, @lastName, @phone);
SELECT @userID = id FROM @inserted_user;
INSERT INTO Employees (UserID, RoleID)
VALUES (@userID, 100000); -- let Secretary role id be 100000
END
```

Delete Instructor

Procedura `DeleteSecretary` usuwa sekretarkę z bazy danych na podstawie podanego ID. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy użytkownik ma przypisaną rolę instruktora, jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@secretary_id` – ID instruktora, który ma zostać usunięty.

```
CREATE PROCEDURE DeleteSecretary
@secretary_id INT
AS BEGIN
IF @secretary_id NOT IN (SELECT UserID FROM Employees E
join EmployeeRoles ER on E.RoleID = ER.RoleID WHERE ER. RoleID = 'Secretary')
THROW 50000, 'Secretary not found', 11;
DELETE Users WHERE UserID = @secretary_id;
DELETE Employees WHERE UserID = @secretary_id;
END
```

Create Webinar

Procedura `CreateWebinar` tworzy nowy webinar, zapisując dane w tabelach `Products` oraz `Webinars`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego webinaru poprzez argument `@webinarID`. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy `@lecturerID` istnieje jako użytkownik z rolą wykładowcy. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@platformID` istnieje w tabeli `OnlinePlatforms`. Jeśli nie, rzuca błąd o kodzie `50001`.
- `@startTime` musi być wcześniejsze niż `@endTime`. W przeciwnym razie procedura rzuca błąd o kodzie `50002`.
- `@price` musi być większe lub równe 0. Jeśli nie, procedura rzuca błąd o kodzie `50003`.

Argumenty procedury:

- `@name` – Nazwa webinaru.
- `@description` – Opis webinaru.
- `@lecturerID` – ID wykładowcy prowadzącego webinar.
- `@platformID` – ID platformy, na której odbędzie się webinar.
- `@startTime` – Czas rozpoczęcia webinaru.
- `@endTime` – Czas zakończenia webinaru.
- `@translatorID` – ID tłumacza (opcjonalnie).
- `@meetingURL` – Link do spotkania online.
- `@recordingURL` – Link do nagrania (opcjonalnie).
- `@price` – Cena webinaru.
- `@capacity` – Maksymalna liczba uczestników.
- `@available` – Flaga wskazująca, czy webinar jest dostępny.
- `@webinarID` – Zwracana wartość ID nowo utworzonego webinaru.

```
CREATE PROCEDURE CreateWebinar
@name NVARCHAR(40),
@description NVARCHAR(MAX),
@lecturerID INT,
@platformID INT,
@startTime DATETIME,
@endTime DATETIME,
@translatorID INT,
@meetingURL NVARCHAR(120),
@recordingURL NVARCHAR(120),
@price MONEY,
@capacity INT,
@available BIT,
@webinarID INT OUTPUT
AS BEGIN
    IF @lecturerID NOT IN (SELECT userID FROM Employees E
                           JOIN EmployeeRoles ER ON ER.RoleID =
E.RoleID WHERE ER.RoleName = 'Lecturer')
        THROW 50000, 'Lecturer not found', 11;
    ELSE IF @platformID NOT IN (SELECT PlatformID FROM
OnlinePlatforms)
        THROW 50001, 'Online platform not found', 11;
    ELSE IF @startTime > @endTime
        THROW 50002, 'Start time must be before end time', 16;
    ELSE IF @price < 0
        THROW 50003, 'Price cannot be negative', 16;

    DECLARE @insertedProduct TABLE (ProductID INT);
    INSERT INTO Products (Name, Description, Price, Capacity,
Available)
    OUTPUT INSERTED.ProductID INTO @insertedProduct
    VALUES (@name, @description, @price, @capacity, @available);

    SELECT @webinarID = ProductID FROM @insertedProduct;
    INSERT INTO Webinars (ProductID, LecturerID, PlatformID,
MeetingURL, RecordingURL, Translator, StartTime, EndTime)
    VALUES (@webinarID, @lecturerID, @platformID, @meetingURL,
@recordingURL, @translatorID, @startTime, @endTime);
END
```

Modify Webinar

Procedura `ModifyWebinar` modyfikuje dane webinaru w tabeli `Webinars`. Procedura przeprowadza walidację danych wejściowych. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@webinarID` istnieje w tabeli `Webinars`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@lecturerID` istnieje jako użytkownik z rolą wykładowcy. Jeśli nie, rzuca błąd o kodzie `50001`.

- Procedura sprawdza, czy `@platformID` istnieje w tabeli `OnlinePlatforms`. Jeśli nie, rzuca błąd o kodzie `50002`.
- Procedura sprawdza, czy `@translatorID` istnieje jako użytkownik z rolą tłumacza. Jeśli nie, rzuca błąd o kodzie `50003`.
- `@startTime` musi być wcześniejsze niż `@endTime`. W przeciwnym razie procedura rzuca błąd o kodzie `50004`.
- `@meetingURL` musi być poprawny lub być nullem. Jeśli nie, procedura rzuca błąd o kodzie `50005`.
- `@recordingURL` musi być poprawny lub być nullem. Jeśli nie, procedura rzuca błąd o kodzie `50006`.

Argumenty procedury:

- `@webinarID` – ID webinaru.
- `@name` – Nazwa webinaru.
- `@description` – Opis webinaru.
- `@lecturerID` – ID wykładowcy prowadzącego webinar.
- `@platformID` – ID platformy, na której odbędzie się webinar.
- `@startTime` – Czas rozpoczęcia webinaru.
- `@endTime` – Czas zakończenia webinaru.
- `@translatorID` – ID tłumacza (opcjonalnie).
- `@meetingURL` – Link do spotkania online.
- `@recordingURL` – Link do nagrania (opcjonalnie).
- `@price` – Cena webinaru.
- `@capacity` – Maksymalna liczba uczestników.

```
CREATE PROCEDURE ModifyWebinar
   @webinarID INT,
   @name NVARCHAR(40),
   @description NVARCHAR(MAX),
   @lecturerID INT,
   @platformID INT,
   @startTime DATETIME,
   @endTime DATETIME,
   @translatorID INT,
   @meetingURL NVARCHAR(120),
   @recordingURL NVARCHAR(120),
   @price MONEY,
   @capacity INT
AS
BEGIN
   IF @webinarID NOT IN (SELECT ProductID FROM Webinars)
      THROW 50000, 'Webinar not found', 11;
   IF @lecturerID NOT IN (SELECT UserID FROM EmployeeInformation
WHERE Role = 'Lecturer')
      THROW 50001, 'Lecturer not found', 11;
   IF @platformID NOT IN (SELECT PlatformID FROM OnlinePlatforms)
```

```
      THROW 50002, 'PlatformID is not correct', 16;
   IF NOT (@startTime < @endTime)
      THROW 50003, 'The specified webinar time is wrong', 16;
   IF (@translatorID IS NOT NULL AND @translatorID NOT IN (SELECT
UserID from Translators))
      THROW 50004, 'Translator not found', 11;
   IF (@meetingURL IS NOT NULL AND @meetingURL NOT LIKE
'https://%_.%_')
      THROW 50005, 'MeetingURL is not correct', 16;
   IF (@recordingURL IS NOT NULL AND @recordingURL NOT LIKE
'https://%_.%_')
      THROW 50006, 'RecordingURL is not correct', 16;
   SET LecturerID = @lecturerID, PlatformID = @platformID,
MeetingURL = @meetingURL, RecordingURL = @recordingURL, Translator
= @translatorID, StartTime = @startTime, EndTime = @endTime
   WHERE ProductID = @webinarID;
END;
```

Delete Webinar

Procedura `DeleteWebinar` oznacza webinar jako niedostępny poprzez ustawienie wartości `available` na `0` w tabeli `Products`. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy webinar istnieje w tabeli `Webinars`. Jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@webinarID` – ID webinaru, który ma zostać oznaczony jako niedostępny.

```
CREATE PROCEDURE DeleteWebinar
@webinarID INT
AS BEGIN
IF @webinarID NOT IN (SELECT ProductID FROM Webinars)
THROW 50000, 'Webinar not found', 11;
UPDATE products
SET available = 0 WHERE ProductID = @webinarID;
END
```

Create Course

Procedura `CreateCourse` tworzy nowy kurs, zapisując dane w tabelach `Products` oraz `Courses`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego kursu poprzez argument `@courseID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@coordinatorID` istnieje jako użytkownik z rolą koordynatora. Jeśli nie, rzuca błąd o kodzie `50000`.
- `@price` musi być większe lub równe 0. Jeśli nie, procedura rzuca błąd o kodzie `50001`.

Argumenty procedury:

- `@name` – Nazwa kursu.
- `@description` – Opis kursu.
- `@coordinatorID` – ID koordynatora kursu.
- `@price` – Cena kursu.
- `@capacity` – Maksymalna liczba uczestników.
- `@available` – Flaga wskazująca, czy kurs jest dostępny.
- `@courseID` – Zwracana wartość ID nowo utworzonego kursu.

```
CREATE PROCEDURE CreateCourse
@name NVARCHAR(40),
@description NVARCHAR(MAX),
@coordinatorID INT,
@price MONEY,
@capacity INT,
@available BIT,
@courseID INT OUTPUT
AS BEGIN
IF @coordinatorID NOT IN (SELECT e.UserID FROM Employees AS e
JOIN EmployeeRoles AS er ON e.RoleID = er.RoleID
WHERE er.RoleName = 'Coordinator')
THROW 50000, 'Coordinator not found', 11;
ELSE IF @price < 0
THROW 50001, 'Price cannot be negative', 16;
DECLARE @insertedProduct TABLE (ProductID INT);
INSERT INTO Products (Name, Description, Price, Capacity,
Available)
OUTPUT INSERTED.ProductID INTO @insertedProduct
VALUES (@name, @description, @price, @capacity, @available);
SELECT @courseID = ProductID FROM @insertedProduct;
INSERT INTO Courses (ProductID, CoordinatorID)
VALUES (@courseID, @coordinatorID);
END
```

Create Offline Course Module

Procedura `CreateOfflineCourseModule` tworzy nowy moduł stacjonarny kursu, zapisując dane w tabelach `CourseModules`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego modułu poprzez argument `@moduleID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@courseID` istnieje w tabeli `Courses`. Jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@courseID` – ID kursu, do którego dodawany jest moduł.
- `@name` – Nazwa modułu.
- `@description` – Opis modułu.
- `@moduleID` – Zwracana wartość ID nowo utworzonego modułu.

```
CREATE PROCEDURE CreateOfflineCourseModule
@courseID INT,
@name NVARCHAR(40),
@description NVARCHAR(MAX),
@moduleID INT OUTPUT
AS BEGIN
IF @courseID NOT IN (SELECT ProductID FROM Courses)
THROW 50000, 'Course not found', 11;
DECLARE @insertedModule TABLE (ModuleID INT);
DECLARE @typeID INT;
SET @typeID = dbo.GetModuleTypeID('Offline');
INSERT INTO CourseModules (Name, Description, CourseID, TypeID)
OUTPUT INSERTED.ModuleID INTO @insertedModule
VALUES (@name, @description, @courseID, @typeID);
SELECT @moduleID = ModuleID FROM @insertedModule;
END
```

Create Online Synchronous Course Module

Procedura `CreateOnlineSynchronousCourseModule` tworzy nowy moduł synchroniczny online kursu, zapisując dane w tabelach `CourseModules`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego modułu poprzez argument `@moduleID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@courseID` istnieje w tabeli `Courses`. Jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@courseID` – ID kursu, do którego dodawany jest moduł.
- `@name` – Nazwa modułu.
- `@description` – Opis modułu.
- `@moduleID` – Zwracana wartość ID nowo utworzonego modułu.

```
CREATE PROCEDURE CreateOnlineSynchronousCourseModule
```

```
@courseID INT,
@name NVARCHAR(40),
@description NVARCHAR(MAX),
@moduleID INT OUTPUT
AS BEGIN
IF @courseID NOT IN (SELECT ProductID FROM Courses)
THROW 50000, 'Course not found', 11;
DECLARE @insertedModule TABLE (ModuleID INT);
DECLARE @typeID INT;
SET @typeID = dbo.GetModuleTypeID('Online Synchronous');
INSERT INTO CourseModules (Name, Description, CourseID, TypeID)
OUTPUT INSERTED.ModuleID INTO @insertedModule
VALUES (@name, @description, @courseID, @typeID);
SELECT @moduleID = ModuleID FROM @insertedModule;
END
```

Create Online Asynchronous Course Module

Procedura `CreateOnlineAsynchronousCourseModule` tworzy nowy moduł asynchroniczny online kursu, zapisując dane w tabelach `CourseModules`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego modułu poprzez argument `@moduleID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@courseID` istnieje w tabeli `Courses`. Jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@courseID` – ID kursu, do którego dodawany jest moduł.
- `@name` – Nazwa modułu.
- `@description` – Opis modułu.
- `@moduleID` – Zwracana wartość ID nowo utworzonego modułu.

```
CREATE PROCEDURE CreateOnlineAsynchronousCourseModule
@courseID INT,
@name NVARCHAR(40),
@description NVARCHAR(MAX),
@moduleID INT OUTPUT
AS BEGIN
IF @courseID NOT IN (SELECT ProductID FROM Courses)
THROW 50000, 'Course not found', 11;
DECLARE @insertedModule TABLE (ModuleID INT);
DECLARE @typeID INT;
SET @typeID = dbo.GetModuleTypeID('Online Asynchronous');
INSERT INTO CourseModules (Name, Description, CourseID, TypeID)
OUTPUT INSERTED.ModuleID INTO @insertedModule
VALUES (@name, @description, @courseID, @typeID);
SELECT @moduleID = ModuleID FROM @insertedModule;
```

```
END
```

Create Hybrid Course Module

Procedura `CreateHybridCourseModule` tworzy nowy moduł hybrydowy kursu, zapisując dane w tabelach `CourseModules`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego modułu poprzez argument `@moduleID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@courseID` istnieje w tabeli `Courses`. Jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@courseID` – ID kursu, do którego dodawany jest moduł.
- `@name` – Nazwa modułu.
- `@description` – Opis modułu.
- `@moduleID` – Zwracana wartość ID nowo utworzonego modułu.

```
CREATE PROCEDURE CreateHybridCourseModule
@courseID INT,
@name NVARCHAR(40),
@description NVARCHAR(MAX),
@moduleID INT OUTPUT
AS BEGIN
IF @courseID NOT IN (SELECT ProductID FROM Courses)
THROW 50000, 'Course not found', 11;
DECLARE @insertedModule TABLE (ModuleID INT);
DECLARE @typeID INT;
SET @typeID = dbo.GetModuleTypeID('Hybrid');
INSERT INTO CourseModules (Name, Description, CourseID, TypeID)
OUTPUT INSERTED.ModuleID INTO @insertedModule
VALUES (@name, @description, @courseID, @typeID);
SELECT @moduleID = ModuleID FROM @insertedModule;
END
```

Add Online Asynchronous Module Meeting

Procedura `AddOnlineAsynchronousModuleMeeting` tworzy nowe spotkanie online asynchroniczne zapisując wyniki w tabelach `ModuleMeetings` oraz `OnlineAsynchronousModuleMeetings`. Autor: Bartłomiej Kaczyński.

Warunki integralnościowe:

- Procedura sprawdza, czy `@LecturerID` istnieje w tabeli `Employees`. Jeśli nie, rzuca błąd o kodzie `50000`.

- Procedura sprawdza, czy `@ModuleID` należy do kompatybilnego typu spotkania. Jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy `@RecordingURL` jest poprawny. Jak nie to rzuca błąd o kodzie `50002`.
- Procedura sprawdza, czy `@RecordingURL` jest unikalny. Jak nie to rzuca błąd o kodzie `50003`.

Argumenty procedury:

- `@LecturerID` – ID prowadzącego spotkanie
- `@Topic` – Temat spotkania.
- `@Translator` – Opcjonalny translator spotkania
- `@ModuleID` – ID modułu do którego zostanie przypisane to spotkanie
- `@RecordingURL` – link do nagrania tego spotkania.
- `@MeetingID` – zwracane ID tego spotkania.

```
CREATE PROCEDURE AddOnlineAsynchronousModuleMeeting
   @LecturerID INT,
   @Topic NVARCHAR(80),
   @Translator INT,
   @ModuleID INT,
   @RecordingURL NVARCHAR(120),
   @MeetingID INT OUTPUT
AS BEGIN
   IF @LecturerID NOT IN (SELECT UserID FROM Employees where
 RoleID = 1000)
       THROW 50000, 'This lecturer does not exist', 11;
   ELSE IF @ModuleID NOT IN (SELECT ModuleID
                          FROM CourseModules WHERE TypeID IN
(dbo.GetModuleTypeID('Online Asynchronous'),
dbo.GetModuleTypeID('Hybrid')))
       THROW 50001, 'The specified module is not compatible
       with this type of meeting or it does not exist', 11;
   ELSE IF @RecordingURL NOT LIKE 'https://%_.%_'
       THROW 50002, 'RecordingURL is not correct', 16;
   ELSE IF @RecordingURL IN (SELECT RecordingURL FROM
 OnlineAsynchronousModuleMeetings)
       THROW 50003, 'RecordingURL is not unique', 16;

   DECLARE @Meeting Table (ID INT);

   INSERT INTO ModuleMeetings (LecturerID,Topic,Translator)
   OUTPUT INSERTED.MeetingID INTO @Meeting
   VALUES (@LecturerID,@Topic,@Translator)

   SELECT @MeetingID = ID FROM @Meeting

   INSERT INTO OnlineAsynchronousModuleMeetings
(MeetingID,ModuleID,RecordingURL)
   VALUES (@MeetingID,@ModuleID,@RecordingURL)
```

```
END
```

Add Offline Module Meeting

Procedura `AddOfflineModuleMeeting` tworzy nowe spotkanie offline zapisując wyniki w tabelach `ModuleMeetings` oraz `OfflineModuleMeetings`. Autor: Bartłomiej Kaczyński.

Warunki integralnościowe:

- Procedura sprawdza, czy `@LecturerID` istnieje w tabeli `Employees`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@ModuleID` należy do kompatybilnego typu spotkania. Jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy `@StartTime` jest mniejsze od `@EndTime`. Jeśli nie, rzuca błąd o kodzie `50002`.
- Procedura sprawdza, czy `@RoomID` jest poprawnym id pokoju. Jak nie to rzuca błąd o kodzie `50003`.

Argumenty procedury:

- `@LecturerID` – ID prowadzącego spotkanie
- `@Topic` – Temat spotkania.
- `@Translator` – Opcjonalny translator spotkania
- `@ModuleID` – ID modułu do którego zostanie przypisane to spotkanie
- `@StartTime` – Początek spotkania
- `@EndTime` – Koniec spotkania
- `@RoomID` – id pokoju w którym odbyć ma się to spotkanie.
- `@MeetingID` – zwracane ID tego spotkania.

```
CREATE PROCEDURE AddOfflineModuleMeeting
   @LecturerID INT,
   @Topic NVARCHAR(80),
   @Translator INT,
   @ModuleID INT,
   @StartTime DATETIME,
   @EndTime DATETIME,
   @RoomID INT,
   @MeetingID INT OUTPUT
AS BEGIN
   IF @LecturerID NOT IN (SELECT UserID FROM Employees where
 RoleID = 1000)
       THROW 50000, 'This lecturer does not exist', 11;
   ELSE IF @ModuleID NOT IN (SELECT ModuleID FROM CourseModules
 WHERE TypeID IN (dbo.GetModuleTypeID('Offline'),
dbo.GetModuleTypeID('Hybrid')))
       THROW 50001, 'The specified module is not compatible
            with this type of meeting or it does not exist', 11;
   ELSE IF (@StartTime > @EndTime)
```

```
        THROW 50002, 'The specified meeting time is wrong', 16;
    ELSE IF @RoomID NOT IN (SELECT RoomID FROM Rooms)
        THROW 50003, 'PlatformID is not correct',11;
    ELSE IF dbo.GetModuleCapacity(@ModuleID) >
          (SELECT PlaceLimit FROM Rooms WHERE RoomID = @RoomID)
        THROW 50004, 'The specified room is too small for this
meeting', 16;


    DECLARE @Meeting Table (ID INT);

    INSERT INTO ModuleMeetings (LecturerID,Topic,Translator)
    OUTPUT INSERTED.MeetingID INTO @Meeting
    VALUES (@LecturerID,@Topic,@Translator)

    SELECT @MeetingID = ID FROM @Meeting

    INSERT INTO OfflineModuleMeetings
(MeetingID,ModuleID,StartTime,EndTime,RoomID)
    VALUES (@MeetingID,@ModuleID,@StartTime,@EndTime,@RoomID)
END
```

Add Online Synchronous Module Meeting

Procedura `AddOnlineSynchronousModuleMeeting` tworzy nowe spotkanie online synchroniczne zapisując wyniki w tabelach `ModuleMeetings` oraz `OnlineSynchronousModuleMeetings`. Autor: Bartłomiej Kaczyński.

Warunki integralnościowe:

- Procedura sprawdza, czy `@LecturerID` istnieje w tabeli `Employees`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@ModuleID` należy do kompatybilnego typu spotkania. Jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy `@StartTime` jest mniejsze od `@EndTime`. Jeśli nie, rzuca błąd o kodzie `50002`.
- Procedura sprawdza, czy `@RecordingURL` jest poprawny lub nullem. Jak nie to rzuca błąd o kodzie `50003`.
- Procedura sprawdza, czy `@RecordingURL` jest unikalny lub nullem. Jak nie to rzuca błąd o kodzie `50004`.
- Procedura sprawdza, czy `@MeetingURL` jest poprawny lub nullem. Jak nie to rzuca błąd o kodzie `50005`.
- Procedura sprawdza, czy `@MeetingURL` jest unikalny lub nullem. Jak nie to rzuca błąd o kodzie `50006`.
- Procedura sprawdza, czy `@PlatformID` jest prawdziwym id jakiejś platformy. Jak nie to rzuca błąd o kodzie `50007`.

Argumenty procedury:

- `@LecturerID` – ID prowadzącego spotkanie
- `@Topic` – Temat spotkania.
- `@Translator` – Opcjonalny translator spotkania
- `@ModuleID` – ID modułu do którego zostanie przypisane to spotkanie
- `@StartTime` – Początek spotkania
- `@EndTime` – Koniec spotkania
- `@PlatformID` – id platformy na której ma się odbyć to spotkanie
- `@MeetingURL` – link do spotkania.
- `@RecordingURL` – link do nagrania tego spotkania.
- `@MeetingID` – zwracane ID tego spotkania.

```
CREATE PROCEDURE AddOnlineSynchronousModuleMeeting
   @LecturerID INT,
   @Topic NVARCHAR(80),
   @Translator INT,
   @ModuleID INT,
   @StartTime DATETIME,
   @EndTime DATETIME,
   @PlatformID INT,
   @MeetingURL NVARCHAR(120),
   @RecordingURL NVARCHAR(120),
   @MeetingID INT OUTPUT
AS BEGIN
   IF @LecturerID NOT IN (SELECT UserID FROM Employees where
 RoleID = 1000)
       THROW 50000, 'This lecturer does not exist', 11;
   ELSE IF @ModuleID NOT IN (SELECT ModuleID FROM CourseModules
 WHERE TypeID IN
   (dbo.GetModuleTypeID('Online
Synchronous'),dbo.GetModuleTypeID('Hybrid')))
       THROW 50001, 'The specified module is not compatible
                     with this type of meeting or it does not
exist', 11;
   ELSE IF (@StartTime > @EndTime)
       THROW 50002, 'The specified meeting time is wrong', 16;
   ELSE IF (@RecordingURL NOT LIKE 'https://%_.%_' AND
@RecordingURL IS NOT NULL)
       THROW 50003, 'RecordingURL is not correct', 16;
   ELSE IF @RecordingURL IS NOT NULL AND @RecordingURL IN (SELECT
RecordingURL FROM OnlineSynchronousModuleMeetings)
       THROW 50004, 'RecordingURL is not unique', 16;
   ELSE IF (@MeetingURL NOT LIKE 'https://%_.%_' AND @MeetingURL
 IS NOT NULL)
       THROW 50005, 'MeetingURL is not correct', 16;
   ELSE IF @MeetingURL IS NOT NULL AND @MeetingURL IN (SELECT
MeetingURL FROM OnlineSynchronousModuleMeetings)
       THROW 50006, 'MeetingURL is not unique', 16;
   ELSE IF @PlatformID NOT IN (SELECT PlatformID FROM
```

```
OnlinePlatforms)
       THROW 50007, 'PlatformID is not correct',11;

   DECLARE @Meeting Table (ID INT);

   INSERT INTO ModuleMeetings (LecturerID,Topic,Translator)
   OUTPUT INSERTED.MeetingID INTO @Meeting
   VALUES (@LecturerID,@Topic,@Translator)

   SELECT @MeetingID = ID FROM @Meeting

   INSERT INTO OnlineSynchronousModuleMeetings
 (MeetingID,ModuleID,StartTime,EndTime,PlatformID,MeetingURL,
RecordingURL)
   VALUES (@MeetingID,@ModuleID,@StartTime,@EndTime,@PlatformID,
@MeetingURL,@RecordingURL)
END
```

Create Study

Procedura `CreateStudy` tworzy nowy kierunek studiów, zapisując dane w tabelach `Products` oraz `Studies`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego kierunku poprzez argument `@studyID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@supervisorID` istnieje jako użytkownik z rolą koordynatora. Jeśli nie, rzuca błąd o kodzie `50000`.
- `@price` musi być większe lub równe 0. Jeśli nie, procedura rzuca błąd o kodzie `50001`.

Argumenty procedury:

- `@name` – Nazwa kierunku.
- `@description` – Opis kierunku.
- `@supervisorID` – ID opiekuna kierunku.
- `@price` – Opłata za studia.
- `@capacity` – Maksymalna liczba studentów.
- `@available` – Flaga wskazująca, czy kierunek jest dostępny.
- `@studyID` – Zwracana wartość ID nowo utworzonego kierunku.

```
CREATE PROCEDURE CreateStudy
@name NVARCHAR(40),
@description NVARCHAR(MAX),
@supervisorID INT,
@price MONEY,
@capacity INT,
@available BIT,
```

```
@studyID INT OUTPUT
AS BEGIN
IF @supervisorID NOT IN (SELECT e.UserID FROM Employees AS e
JOIN EmployeeRoles AS er ON e.RoleID = er.RoleID
WHERE er.RoleName = 'Coordinator')
THROW 50000, 'Coordinator not found', 11;
ELSE IF @price < 0
THROW 50001, 'Price cannot be negative', 16;
DECLARE @insertedProduct TABLE (ProductID INT);
INSERT INTO Products (Name, Description, Price, Capacity,
Available)
OUTPUT INSERTED.ProductID INTO @insertedProduct
VALUES (@name, @description, @price, @capacity, @available);
SELECT @studyID = ProductID FROM @insertedProduct;
INSERT INTO Studies (ProductID, SupervisorID)
VALUES (@studyID, @supervisorID);
END
```

Create Online Synchronous Subject

Procedura `CreateOnlineSynchronousSubject` tworzy nowy przedmiot online, zapisując dane w tabeli `Subjects`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego przedmiotu poprzez argument `@subjectID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@studyID` istnieje w tabeli `Studies`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@instructorID` istnieje jako użytkownik z rolą wykładowcy. Jeśli nie, rzuca błąd o kodzie `50001`.

Argumenty procedury:

- `@studyID` – ID kierunku studiów, do którego dodawany jest przedmiot.
- `@name` – Nazwa przedmiotu.
- `@studyProgram` – Sylabus przedmiotu.
- `@instructorID` – ID prowadzącego dany przedmiot.
- `@subjectID` – Zwracana wartość ID nowo utworzonego modułu.

```
CREATE PROCEDURE CreateOnlineSynchronousSubject
@studyID INT,
@name NVARCHAR(40),
@studyProgram NVARCHAR(MAX),
@instructorID INT,
@subjectID INT OUTPUT
AS BEGIN
IF @studyID NOT IN (SELECT ProductID FROM Studies)
THROW 50000, 'Study not found', 11;
```

```
IF @instructorID NOT IN (SELECT e.UserID FROM Employees AS e
JOIN EmployeeRoles AS er ON e.RoleID = er.RoleID
WHERE er.RoleName = 'Lecturer')
THROW 50001, 'Lecturer not found', 11;
DECLARE @insertedSubject TABLE (SubjectID INT);
DECLARE @typeID INT;
SET @typeID = dbo.GetSubjectTypeID('Online Synchronous');
INSERT INTO Subjects (Name, StudyProgram, StudyID, InstructorID,
TypeID)
OUTPUT INSERTED.SubjectID INTO @insertedSubject
VALUES (@name, @studyProgram, @studyID, @instructorID, @typeID);
SELECT @subjectID = SubjectID FROM @insertedSubject;
END
```

Create Offline Subject

Procedura `CreateOfflineSubject` tworzy nowy przedmiot stacjonarny, zapisując dane w tabeli `Subjects`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego przedmiotu poprzez argument `@subjectID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@studyID` istnieje w tabeli `Studies`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@instructorID` istnieje jako użytkownik z rolą wykładowcy. Jeśli nie, rzuca błąd o kodzie `50001`.

Argumenty procedury:

- `@studyID` – ID kierunku studiów, do którego dodawany jest przedmiot.
- `@name` – Nazwa przedmiotu.
- `@studyProgram` – Sylabus przedmiotu.
- `@instructorID` – ID prowadzącego dany przedmiot.
- `@subjectID` – Zwracana wartość ID nowo utworzonego modułu.

```
CREATE PROCEDURE CreateOfflineSubject
@studyID INT,
@name NVARCHAR(40),
@studyProgram NVARCHAR(MAX),
@instructorID INT,
@subjectID INT OUTPUT
AS BEGIN
IF @studyID NOT IN (SELECT ProductID FROM Studies)
THROW 50000, 'Study not found', 11;
IF @instructorID NOT IN (SELECT e.UserID FROM Employees AS e
JOIN EmployeeRoles AS er ON e.RoleID = er.RoleID
WHERE er.RoleName = 'Lecturer')
THROW 50001, 'Lecturer not found', 11;
```

```
DECLARE @insertedSubject TABLE (SubjectID INT);
DECLARE @typeID INT;
SET @typeID = dbo.GetSubjectTypeID('Offline');
INSERT INTO Subjects (Name, StudyProgram, StudyID, InstructorID,
TypeID)
OUTPUT INSERTED.SubjectID INTO @insertedSubject
VALUES (@name, @studyProgram, @studyID, @instructorID, @typeID);
SELECT @subjectID = SubjectID FROM @insertedSubject;
END
```

Create Hybrid Subject

Procedura `CreateHybridSubject` tworzy nowy przedmiot prowadzony w formie hybrydowej, zapisując dane w tabeli `Subjects`. Procedura przeprowadza walidację danych wejściowych, a następnie zwraca ID nowo utworzonego przedmiotu poprzez argument `@subjectID`. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@studyID` istnieje w tabeli `Studies`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@instructorID` istnieje jako użytkownik z rolą wykładowcy. Jeśli nie, rzuca błąd o kodzie `50001`.

Argumenty procedury:

- `@studyID` – ID kierunku studiów, do którego dodawany jest przedmiot.
- `@name` – Nazwa przedmiotu.
- `@studyProgram` – Sylabus przedmiotu.
- `@instructorID` – ID prowadzącego dany przedmiot.
- `@subjectID` – Zwracana wartość ID nowo utworzonego modułu.

```
CREATE PROCEDURE CreateHybridSubject
@studyID INT,
@name NVARCHAR(40),
@studyProgram NVARCHAR(MAX),
@instructorID INT,
@subjectID INT OUTPUT
AS BEGIN
IF @studyID NOT IN (SELECT ProductID FROM Studies)
THROW 50000, 'Study not found', 11;
IF @instructorID NOT IN (SELECT e.UserID FROM Employees AS e
JOIN EmployeeRoles AS er ON e.RoleID = er.RoleID
WHERE er.RoleName = 'Lecturer')
THROW 50001, 'Lecturer not found', 11;
DECLARE @insertedSubject TABLE (SubjectID INT);
DECLARE @typeID INT;
SET @typeID = dbo.GetSubjectTypeID('Hybrid');
INSERT INTO Subjects (Name, StudyProgram, StudyID, InstructorID,
```

```
TypeID)
OUTPUT INSERTED.SubjectID INTO @insertedSubject
VALUES (@name, @studyProgram, @studyID, @instructorID, @typeID);
SELECT @subjectID = SubjectID FROM @insertedSubject;
END
```

Add Offline Subject Meeting

Procedura `AddOfflineSubjectMeeting` tworzy nowe spotkanie studyjne offline zapisując wyniki w tabelach `Products`, `SubjectMeetings` oraz `SubjectOfflineMeetings` Autor: Bartłomiej Kaczyński.

Warunki integralnościowe:

- Procedura sprawdza, czy `@LecturerID` istnieje w tabeli `Employees`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@SubjectID` należy do kompatybilnego typu przedmiotu. Jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy `@StartTime` jest mniejsze od `@EndTime`. Jeśli nie, rzuca błąd o kodzie `50002`.
- Procedura sprawdza, czy `@RoomID` jest poprawnym id pokoju. Jak nie to rzuca błąd o kodzie `50003`.
- Procedura sprawdza, czy `Capacity` jest większe od pojemności spotkania oraz mniejsze lub równe pojemności pokoju. Jak nie to rzuca odpowiednio błędy o kodach `50004` i `50005`
- Procedura sprawdza, czy `@Price` jest dodatnia. Jak nie to rzuca błąd o kodzie `50006`.

Argumenty procedury:

- `@Name` – Nazwa spotkania.
- `@Description` – Opis spotkania.
- `@Price` – Cena spotkania dla osób spoza studium.
- `@Capacity` – Ilość miejsc na spotkaniu.
- `@LecturerID` – ID prowadzącego spotkanie
- `@Translator` – Opcjonalny translator spotkania
- `@SubjectID` – ID przedmiotu do którego zostanie przypisane to spotkanie
- `@StartTime` – Początek spotkania
- `@EndTime` – Koniec spotkania
- `@RoomID` – id pokoju w którym odbyć ma się to spotkanie.
- `@MeetingID` – zwracane ID tego spotkania.

```
CREATE PROCEDURE AddOfflineSubjectMeeting
   @Name NVARCHAR(40),
   @Description NVARCHAR(MAX),
   @Price MONEY,
```

```
   @Capacity INT,
   @Available BIT,
   @LecturerID INT,
   @Translator INT,
   @SubjectID INT,
   @StartTime DATETIME,
   @EndTime DATETIME,
   @RoomID INT,
   @MeetingID INT OUTPUT
AS BEGIN
   IF @LecturerID NOT IN (SELECT UserID FROM Employees where
 RoleID = 1000)
       THROW 50000, 'This lecturer does not exist', 11;
   ELSE IF @SubjectID NOT IN (SELECT SubjectID FROM Subjects
 WHERE TypeID IN

(dbo.GetSubjectTypeID('Offline'),dbo.GetSubjectTypeID('Hybrid')))
       THROW 50001, 'The specified subject is not compatible
            with this type of meeting or it does not exist', 11;
   ELSE IF (@StartTime > @EndTime)
       THROW 50002, 'The specified meeting time is wrong', 16;
   ELSE IF @RoomID NOT IN (SELECT RoomID FROM Rooms)
       THROW 50003, 'PlatformID is not correct',11;
   ELSE IF @Capacity < dbo.GetSubjectCapacity(@SubjectID)
 THROW 50004, 'Meetings capacity is too small for this study',16;
   ELSE IF @Capacity > (SELECT PlaceLimit FROM Rooms
WHERE RoomID = @RoomID)
  THROW 50005, 'Meetings capacity is too large for this room',16;
   ELSE IF @Price < 0
       THROW 50006, 'Price cant be negative', 16;

   DECLARE @Meeting Table (ID INT);

   INSERT INTO Products (Name, Description,Price, Capacity,
 Available)
   OUTPUT INSERTED.ProductID INTO @Meeting
   VALUES (@Name, @Description, @Price, @Capacity, @Available)

   SELECT @MeetingID = ID FROM @Meeting

   INSERT INTO SubjectMeetings (ProductID, LecturerID,
 Translator)
   VALUES (@MeetingID,@LecturerID,@Translator)

   INSERT INTO SubjectOfflineMeetings (MeetingID, SubjectID,
 StartTime, EndTime,RoomID)
   VALUES (@MeetingID,@SubjectID,@StartTime,@EndTime,@RoomID)
END
```

Add Online Synchronous Subject Meeting

Procedura `AddOfflineSubjectMeeting` tworzy nowe spotkanie studyjne online synchroniczne zapisując wyniki w tabelach `Products`, `SubjectMeetings` oraz `SubjectOfflineMeetings` Autor: Bartłomiej Kaczyński.

Warunki integralnościowe:

- Procedura sprawdza, czy `@LecturerID` istnieje w tabeli `Employees`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@SubjectID` należy do kompatybilnego typu przedmiotu. Jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy `@StartTime` jest mniejsze od `@EndTime`. Jeśli nie, rzuca błąd o kodzie `50002`.
- Procedura sprawdza, czy `@PlatformID` jest prawdziwym id jakiejś platformy. Jak nie to rzuca błąd o kodzie `50003`.
- Procedura sprawdza, czy `Capacity` jest większe od pojemności studium. Jak nie to rzuca błąd o kodzie `50004.`
- Procedura sprawdza, czy `@Price` jest dodatnia. Jak nie to rzuca błąd o kodzie `50005`.
- Procedura sprawdza, czy `@RecordingURL` jest poprawny lub nullem. Jak nie to rzuca błąd o kodzie `50006`.
- Procedura sprawdza, czy `@RecordingURL` jest unikalny lub nullem. Jak nie to rzuca błąd o kodzie `50007`.
- Procedura sprawdza, czy `@MeetingURL` jest poprawny lub nullem. Jak nie to rzuca błąd o kodzie `50008`.
- Procedura sprawdza, czy `@MeetingURL` jest unikalny lub nullem. Jak nie to rzuca błąd o kodzie `50009`.

Argumenty procedury:

- `@Name` – Nazwa spotkania.
- `@Description` – Opis spotkania.
- `@Price` – Cena spotkania dla osób spoza studium.
- `@Capacity` – Ilość miejsc na spotkaniu.
- `@LecturerID` – ID prowadzącego spotkanie
- `@Translator` – Opcjonalny translator spotkania
- `@SubjectID` – ID przedmiotu do którego zostanie przypisane to spotkanie
- `@StartTime` – Początek spotkania
- `@EndTime` – Koniec spotkania
- `@PlatformID` – id platformy na której odbyć ma się to spotkanie.
- `@MeetingURL` – link do spotkania.
- `@RecordingURL` – link do nagrania tego spotkania.
- `@MeetingID` – zwracane ID tego spotkania.

```
CREATE PROCEDURE AddOnlineSynchronousSubjectMeeting
   @Name NVARCHAR(40),
   @Description NVARCHAR(MAX),
   @Price MONEY,
   @Capacity INT,
   @Available BIT,
   @LecturerID INT,
   @Translator INT,
   @SubjectID INT,
   @StartTime DATETIME,
   @EndTime DATETIME,
   @PlatformID INT,
   @MeetingURL NVARCHAR(120),
   @RecordingURL NVARCHAR(120),
   @MeetingID INT OUTPUT
AS BEGIN
   IF @LecturerID NOT IN (SELECT UserID FROM Employees where
 RoleID = 1000)
       THROW 50000, 'This lecturer does not exist', 11;
   ELSE IF @SubjectID NOT IN (SELECT SubjectID FROM Subjects
 WHERE TypeID IN

(dbo.GetSubjectTypeID('Offline'),dbo.GetSubjectTypeID('Hybrid')))
       THROW 50001, 'The specified subject is not compatible
            with this type of meeting or it does not exist', 11;
   ELSE IF (@StartTime > @EndTime)
       THROW 50002, 'The specified meeting time is wrong', 16;
   ELSE IF @PlatformID NOT IN (SELECT PlatformID FROM
 OnlinePlatforms)
       THROW 50003, 'PlatformID is not correct',11;
   ELSE IF @Capacity < dbo.GetSubjectCapacity(@SubjectID)
       THROW 50004, 'Meetings capacity is too small for this
 study',16;
   ELSE IF @Price < 0
       THROW 50005, 'Price cant be negative', 16;
   ELSE IF (@RecordingURL NOT LIKE 'https://%_.%_' AND
 @RecordingURL IS NOT NULL)
     THROW 50006, 'RecordingURL is not correct', 16;
   ELSE IF @RecordingURL IS NOT NULL AND @RecordingURL IN (SELECT
 RecordingURL FROM SubjectOnlineSynchronousMeetings)
     THROW 50007, 'RecordingURL is not unique', 16;
   ELSE IF (@MeetingURL NOT LIKE 'https://%_.%_' AND @MeetingURL
 IS NOT NULL)
     THROW 50008, 'MeetingURL is not correct', 16;
   ELSE IF @MeetingURL IS NOT NULL AND @MeetingURL IN (SELECT
 MeetingURL FROM SubjectOnlineSynchronousMeetings)
     THROW 50009, 'MeetingURL is not unique', 16;

   DECLARE @Meeting Table (ID INT);

   INSERT INTO Products (Name, Description,Price, Capacity,
 Available)
   OUTPUT INSERTED.ProductID INTO @Meeting
```

```
   VALUES (@Name, @Description, @Price, @Capacity, @Available)

   SELECT @MeetingID = ID FROM @Meeting

   INSERT INTO SubjectMeetings (ProductID, LecturerID,
 Translator)
   VALUES (@MeetingID,@LecturerID,@Translator)

   INSERT INTO SubjectOnlineSynchronousMeetings (MeetingID,
SubjectID, StartTime, EndTime, PlatformID, MeetingURL,
RecordingURL)
   VALUES (@MeetingID, @SubjectID, @StartTime, @EndTime,
@PlatformID, @MeetingURL, @RecordingURL)
END
```

Add Internship

Procedura `AddInternship` tworzy praktyki z danego przedmiotu, zapisując dane w tabeli `Internships`. Procedura przeprowadza walidację danych wejściowych. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@subjectID` istnieje w tabeli `Subjects`. Jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy `@subjectID` istnieje już w tabeli `Internships`. Jeśli tak, rzuca błąd o kodzie `50001`.

Argumenty procedury:

- `@subjectID` – ID przedmiotu.
- `@description` – Opis praktyk.

```
CREATE PROCEDURE AddInternship
   @subjectID INT,
   @description NVARCHAR(MAX)
AS BEGIN
   IF @subjectID NOT IN (SELECT s.SubjectID FROM Subjects AS s)
      THROW 50000, 'Subject not found', 1;
   IF @subjectID IN (SELECT i.InternshipID FROM Internships as i)
      THROW 50001, 'Internship for this subject already exists',
1;
   INSERT INTO Internships (InternshipID, Description)
   VALUES (@subjectID, @description);
END
```

Modify Internship

Procedura `ModifyInternship` modyfikuje dane praktyk z danego przedmiotu, zapisując zmiany w tabeli `Internships`. Procedura przeprowadza walidację danych wejściowych. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@internshipID` istnieje w tabeli `Internships`. Jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@internshipID` – ID praktyk.
- `@description` – Opis praktyk.

```
CREATE PROCEDURE ModifyInternship
   @internshipID INT,
   @description NVARCHAR(MAX)
AS BEGIN
   IF @internshipID NOT IN (SELECT i.InternshipID FROM Internships
as i)
      THROW 50000, 'Internship not found', 1;
   UPDATE Internships
   SET Description = @description
   WHERE InternshipID = @internshipID
END
```

Add Internship Presence

Procedura `AddInternshipPresence` dodaje obecność studenta na praktykach do tabeli `InternshipPresence`. Procedura przeprowadza walidację danych wejściowych, aby zapewnić integralność danych. Autor**:** Bartosz Ludwin.

Warunki integralnościowe:

1. Procedura sprawdza, czy `@InternshipID` istnieje w tabeli `Internships`. Jeśli nie, rzuca błąd o kodzie `50000`.
2. Procedura sprawdza, czy `@StudentID` istnieje w tabeli `Students`. Jeśli nie, rzuca błąd o kodzie `50001`.
3. Procedura sprawdza, czy obecność dla danego studenta i praktyki na podaną datę już istnieje. Jeśli tak, rzuca błąd o kodzie `50001`.
4. Jeśli wszystkie warunki są spełnione, procedura wstawia nowy wpis do tabeli `InternshipPresence`.

Argumenty procedury:

- `@InternshipID` – ID praktyk.
- `@StudentID` – ID studenta, dla którego dodawana jest obecność.
- `@Date` – Data obecności.

```sql
CREATE PROCEDURE AddInternshipPresence
    @InternshipID INT,
    @StudentID INT,
    @Date DATE
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM Internships WHERE InternshipID = @InternshipID)
        THROW 50000, 'Praktyka o podanym ID nie istnieje.', 1;

    IF NOT EXISTS (SELECT 1 FROM Students WHERE UserID = @StudentID)
        THROW 50001, 'Student o podanym ID nie istnieje', 1;

    IF EXISTS (SELECT 1 FROM InternshipPresence
               WHERE InternshipID = @InternshipID
               AND StudentID = @StudentID
               AND Date = @Date)
        THROW 50001, 'Obecność na tę datę już istnieje', 1;

    INSERT INTO InternshipPresence (InternshipID, StudentID, Date)
    VALUES (@InternshipID, @StudentID, @Date);
END;
```

Delete Internship

Procedura `DeleteInternship` usuwa praktyki z danego przedmiotu z tabeli `Internships`, a także powiązane dane z tabeli `InternshipPresence`. Procedura przeprowadza walidację danych wejściowych. Autor: Kacper Wąchała.

Warunki integralnościowe:

- Procedura sprawdza, czy `@internshipID` istnieje w tabeli `Internships`. Jeśli nie, rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@internshipID` – ID praktyk.

```sql
CREATE PROCEDURE DeleteInternship
   @internshipID INT
AS BEGIN
   IF @internshipID NOT IN (SELECT InternshipID FROM Internships)
      THROW 50002, 'Internship not found', 1;
   DELETE FROM InternshipPresence
   WHERE InternshipID = @internshipID;
   DELETE FROM Internships
   WHERE InternshipID = @internshipID;
END
```

Add Product To Shopping Cart

Procedura `addProductToShoppingCart` dodaje produkt do koszyka studenta. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy produkt może zostać dodany do koszyka, korzystając z funkcji `CanAddProductToShoppingCart`.

Argumenty procedury:

- `@studentID` – ID studenta.
- `@productID` – ID produktu.

```
CREATE PROCEDURE addProductToShoppingCart
@studentID INT,
@productID INT
AS BEGIN
IF (dbo.CanAddProductToShoppingCart(@studentID, @productID) = 1)
    INSERT INTO ShoppingCart (StudentID, ProductID)
    VALUES (@productID, @studentID);
END
```

Create New Order

Procedura `createNewOrder` tworzy nowe zamówienie dla studenta, przetwarzając zawartość jego koszyka zakupowego. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Sprawdza, czy podany `studentID` istnieje w tabeli `students`.
- Weryfikuje, czy koszyk zakupowy (`shoppingCart`) dla danego studenta nie jest pusty.
- Zapewnia unikalność `paymentURL`, sprawdzając, czy nie jest on już używany w tabeli `orders`.
- Dla każdego produktu w koszyku sprawdza, czy można go zamówić, korzystając z funkcji `CanOrderProductInShoppingCart`.

Argumenty procedury:

- `@student_id` – ID studenta.
- `@payment_url` – URL płatności.
- `@order_id` – ID zamówienia (OUTPUT).

```
CREATE PROCEDURE [dbo].[createNewOrder]
@student_id INT,
@payment_url NVARCHAR(128),
@order_id INT OUTPUT
AS BEGIN
IF @student_id NOT IN (SELECT userID FROM students)
THROW 50000, 'Student not found', 1;
ELSE IF 0 = (SELECT COUNT(*) FROM shoppingCart WHERE studentID =
@student_id)
THROW 50001, 'Shopping cart is empty', 1;
ELSE IF @payment_url IN (SELECT PaymentURL FROM orders)
THROW 50002, 'Payment URL must be unique', 1;
DECLARE @product_id INT, @price MONEY;
DECLARE student_shopping_cart_cursor CURSOR FOR
SELECT
products.ProductID,
products.price
FROM ShoppingCart
JOIN products ON ShoppingCart.ProductID = products.ProductID
WHERE StudentID = @student_id;
OPEN student_shopping_cart_cursor;
FETCH NEXT FROM student_shopping_cart_cursor INTO @product_id,
@price;
WHILE @@FETCH_STATUS = 0
BEGIN
IF 0 = dbo.CanOrderProductInShoppingCart(@student_id, @product_id)
THROW 50003, 'Product cannot be ordered', 1;
FETCH NEXT FROM student_shopping_cart_cursor INTO @product_id,
@price;
END
CLOSE student_shopping_cart_cursor;
DEALLOCATE student_shopping_cart_cursor;
DECLARE @inserted_order TABLE (id INT);
INSERT INTO orders(StudentID, PaymentURL, OrderDate)
OUTPUT INSERTED.OrderID INTO @inserted_order
VALUES (@student_id, @payment_url, GETDATE())
SELECT @order_id = id FROM @inserted_order;
INSERT INTO OrderDetails(orderID, ProductID, pricePaid, StatusID)
SELECT
@order_id,
shoppingcart.ProductID,
dbo.GetProductPrice(@student_ID, @product_id),
IIF(
price = 0,
(SELECT statusID FROM orderStatuses WHERE name = 'Done'),
(SELECT statusID FROM orderStatuses WHERE name = 'In realization')
)
FROM ShoppingCart
JOIN products ON ShoppingCart.ProductID= products.ProductID
WHERE StudentID = @student_id;
DELETE ShoppingCart
WHERE StudentID = @student_id;
END
```

Remove Product From Shopping Cart

Procedura `RemoveProductFromShoppingCart` usuwa produkt z koszyka studenta.
Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy student istnieje, jeśli nie, rzuca błąd o kodzie `50000`.
- Procedura sprawdza, czy produkt istnieje, jeśli nie, rzuca błąd o kodzie `50001`.
- Procedura sprawdza, czy produkt znajduje się w koszyku studenta, jeśli nie, rzuca błąd o kodzie `50002`.

Argumenty procedury:

- `@student_id` – ID studenta.
- `@product_id` – ID produktu.

```
CREATE PROCEDURE RemoveProductFromShoppingCart
@student_id INT,
@product_id INT
AS BEGIN
IF @student_id NOT IN (SELECT UserID FROM students)
THROW 50000, 'Student not found', 11;
ELSE IF @product_id NOT IN (SELECT ProductID FROM products)
THROW 50001, 'Product not found', 11;
ELSE IF @product_id NOT IN (SELECT productID FROM ShoppingCart WHERE
studentID = @student_id)
THROW 50002, 'Product not in shopping cart', 11
DELETE ShoppingCart WHERE studentID = @student_id AND productID = @product_id
END
```

Change Order Status

Procedura `changeOrderStatus` zmienia status zamówienia w tabeli `OrderDetails`.
Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy `OrderID` istnieje w tabeli `OrderDetails`. Jeśli nie, rzuca błąd o kodzie `50010`.
- Procedura sprawdza, czy `StatusID` istnieje w tabeli `OrderStatuses`. Jeśli nie, rzuca błąd o kodzie `50011`.

Argumenty procedury:

- `@orderID` – ID zamówienia.
- `@statusID` – Nowy ID statusu.

```
CREATE PROCEDURE [dbo].[changeOrderStatus]
    @orderID INT,
    @statusID INT
AS
BEGIN
    -- Sprawdzenie, czy OrderID istnieje w OrderStatuses
    IF NOT EXISTS (SELECT 1 FROM OrderDetails WHERE OrderID = @orderID)
        THROW 50010, 'OrderID not found in OrderStatuses.', 1;

    IF NOT EXISTS (SELECT 1 FROM OrderStatuses WHERE StatusID = @statusID)
        THROW 50011, 'StatusID does not exist in Statuses.', 1;

    UPDATE OrderDetails
    SET statusID = @statusID
    WHERE OrderID = @orderID;
END
```

Change Translator

Procedura `changeTranslator` zmienia wartość `translatorID` dla określonego `meetingID` w tabelach `SubjectMeetings`, `ModuleMeetings` lub `Webinars`. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Procedura sprawdza, czy podane `meetingID` istnieje w jednej z tabel: `SubjectMeetings`, `ModuleMeetings` lub `Webinars`.
- Jeśli `productID` nie istnieje w żadnej z tych tabel, procedura rzuca błąd o kodzie `50000`.

Argumenty procedury:

- `@productID` – ID webinaru lub spotkania.
- `@translatorID` - ID Tłumacza (`TranslatorPass`)

```
CREATE PROCEDURE [dbo].[ChangeTranslator]
    @meetingID INT,
    @translatorID INT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM SubjectMeetings WHERE ProductID =
@meetingID)
    BEGIN
        UPDATE SubjectMeetings
        SET Translator = @translatorID
        WHERE ProductID = @meetingID;
```

```
        PRINT 'TranslatorID updated in SubjectMeetings.';
    END
    ELSE IF EXISTS (SELECT 1 FROM ModuleMeetings WHERE MeetingID =
@meetingID)
    BEGIN
        UPDATE ModuleMeetings
        SET Translator = @translatorID
        WHERE MeetingID = @meetingID;
        PRINT 'TranslatorID updated in ModuleMeetings.';
    END
    ELSE IF EXISTS (SELECT 1 FROM Webinars WHERE ProductID =
@meetingID)
    BEGIN
        UPDATE Webinars
        SET Translator = @translatorID
        WHERE ProductID = @meetingID;
        PRINT 'TranslatorID updated in Webinars.';
    END
    ELSE
    THROW 50000, 'ProductID not found in SubjectMeetings,
ModuleMeetings, or Webinars.', 1;
END
```

Add Subject Meeting Presence

Procedura `AddSubjectMeetingPresence` zaznacza obecność danego studenta na danym spotkaniu przedmiotowym. Autor: Bartłomiej Kaczyński

Warunki integralnościowe:

- Procedura sprawdza, czy podane `StudentID` istnieje, jeśli nie, to rzuca błąd o kodzie `50003`.
- Procedura sprawdza, czy podane `MeetingID` istnieje, jeśli nie, to rzuca błąd o kodzie `50004`.
- Procedura sprawdza, czy podany student nie ma już obecności na podanym spotkaniu, jak tak to rzuca błąd o kodzie `50005`.
- Reszta warunków jest już w triggerze 🙂

Argumenty procedury:

- `@StudentID` – ID produktu.
- `@MeetingID` - ID spotkania przedmiotowego

```
CREATE PROCEDURE AddSubjectMeetingPresence
     @StudentID INT,
     @MeetingID INT
AS BEGIN
     IF @StudentID NOT IN (SELECT UserID FROM Students)
          THROW 50003, 'Student by that ID does not exist',11;
     IF @MeetingID NOT IN (SELECT ProductID FROM SubjectMeetings)
```

```
          THROW 50004, 'Meeting by that ID does not exist',11;
     IF @MeetingID IN (SELECT MeetingID FROM
SubjectMeetingPresence
               WHERE StudentID = @StudentID)
          THROW 50005, 'Student already has presence on this
meeting',16;

     INSERT INTO SubjectMeetingPresence (MeetingID,StudentID)
     VALUES (@MeetingID,@StudentID)
END
```

Add Course Meeting Presence

Procedura `AddCourseMeetingPresence` zaznacza obecność danego studenta na danym spotkaniu kursowym. Autor: Bartłomiej Kaczyński

Warunki integralnościowe:

- Procedura sprawdza, czy podane `StudentID` istnieje, jeśli nie, to rzuca błąd o kodzie `50003`.
- Procedura sprawdza, czy podane `MeetingID` istnieje, jeśli nie, to rzuca błąd o kodzie `50004`.
- Procedura sprawdza, czy podany student nie ma już obecności na podanym spotkaniu, jak tak to rzuca błąd o kodzie `50005`.
- Reszta warunków jest już w triggerze 🙂

Argumenty procedury:

- `@StudentID` – ID produktu.
- `@MeetingID` - ID spotkania kursowego

```
CREATE PROCEDURE AddCourseMeetingPresence
     @StudentID INT,
     @MeetingID INT
AS BEGIN
     IF @StudentID NOT IN (SELECT UserID FROM Students)
          THROW 50003, 'Student by that ID does not exist',11;
     IF @MeetingID NOT IN (SELECT MeetingID FROM ModuleMeetings)
          THROW 50004, 'Meeting by that ID does not exist',11;
     IF @MeetingID IN (SELECT MeetingID FROM
CourseMeetingPresence
               WHERE StudentID = @StudentID)
          THROW 50005, 'Student already has presence on this
meeting',16;

     INSERT INTO CourseMeetingPresence (MeetingID,StudentID)
     VALUES (@MeetingID,@StudentID)
END
```

Add Course Meeting Makeup Presence

Procedura `AddCourseMeetingMakeupPresence` zaznacza odrobienie obecności danego studenta na danym spotkaniu przedmiotowym danym spotkaniem kursowym. Autor: Bartłomiej Kaczyński

Warunki integralnościowe:

- Procedura sprawdza, czy podane `StudentID` istnieje, jeśli nie, to rzuca błąd o kodzie `50004`.
- Procedura sprawdza, czy podane `MeetingID` istnieje, jeśli nie, to rzuca błąd o kodzie `50005`.
- Procedura sprawdza, czy podane `MakeupFor` istnieje, jeśli nie, to rzuca błąd o kodzie `50006`.
- Procedura sprawdza, czy podany student nie ma już odrobionej obecności na danym spotkaniu, jak tak to rzuca błąd o kodzie `50007`.
- Reszta warunków jest już w triggerze 🙂

```
CREATE PROCEDURE AddCourseMeetingMakeupPresence
     @StudentID INT,
     @MeetingID INT,
     @MakeupFor INT
AS BEGIN
     IF @StudentID NOT IN (SELECT UserID FROM Students)
          THROW 50003, 'Student by that ID does not exist',11;
     IF @MeetingID NOT IN (SELECT MeetingID FROM ModuleMeetings)
          THROW 50004, 'Meeting by that ID does not exist',11;
     IF @MakeupFor NOT IN (SELECT ProductID FROM SubjectMeetings)
          THROW 50005, 'Meeting for makeup by that ID does not
exist',11;
     IF @MakeupFor IN (SELECT MakeupFor FROM
CourseMeetingMakeupPresence
                WHERE StudentID = @StudentID)
     OR @MakeupFor IN (SELECT MakeupFor FROM
SubjectMeetingMakeupPresence
                WHERE StudentID = @StudentID)
          THROW 50006, 'Student already has makeup presence on
that meeting',16;
     INSERT INTO CourseMeetingMakeupPresence
(MeetingID,StudentID,MakeupFor)
     VALUES (@MeetingID,@StudentID,@MakeupFor)
END
```

Add Subject Meeting Makeup Presence

Procedura `AddSubjectMeetingMakeupPresence` zaznacza odrobienie obecności danego studenta na danym spotkaniu przedmiotowym innym (danym) spotkaniem przedmiotowym. Autor: Bartłomiej Kaczyński.

Warunki integralnościowe:

- Procedura sprawdza, czy podane `StudentID` istnieje, jeśli nie, to rzuca błąd o kodzie `50004`.
- Procedura sprawdza, czy podane `MeetingID` istnieje, jeśli nie, to rzuca błąd o kodzie `50005`.
- Procedura sprawdza, czy podane `MakeupFor` istnieje, jeśli nie, to rzuca błąd o kodzie `50006`.
- Procedura sprawdza, czy podany student nie ma już odrobionej obecności na danym spotkaniu, jak tak to rzuca błąd o kodzie `50007`.
- Reszta warunków jest już w triggerze 🙂

```
CREATE PROCEDURE AddSubjectMeetingMakeupPresence
     @StudentID INT,
     @MeetingID INT,
     @MakeupFor INT
AS BEGIN
     IF @StudentID NOT IN (SELECT UserID FROM Students)
          THROW 50004, 'Student by that ID does not exist',11;
     IF @MeetingID NOT IN (SELECT ProductID FROM SubjectMeetings)
          THROW 50005, 'Meeting by that ID does not exist',11;
     IF @MakeupFor NOT IN (SELECT ProductID FROM SubjectMeetings)
          THROW 50006, 'Meeting for makeup by that ID does not
exist',11;
     IF @MakeupFor IN (SELECT MakeupFor FROM
CourseMeetingMakeupPresence
                WHERE StudentID = @StudentID)
                OR @MakeupFor IN (SELECT MakeupFor FROM
SubjectMeetingMakeupPresence
                WHERE StudentID = @StudentID)
          THROW 50007, 'Student already has makeup presence on
that meeting',16;
     INSERT INTO SubjectMeetingMakeupPresence
(MeetingID,StudentID,MakeupFor)
     VALUES (@MeetingID,@StudentID,@MakeupFor)
END
```

Funkcje

Can Add Product To Shopping Cart

Funkcja `CanAddProductToShoppingCart` sprawdza, czy dany produkt może zostać dodany do koszyka studenta. Funkcja zwraca wartość typu `BIT` (`1`, jeśli można dodać produkt; `0`, jeśli nie można). Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Produkt może zostać dodany, jeśli istnieje w tabeli `Products`, student istnieje w tabeli `Students`, a produkt nie został wcześniej zamówiony przez tego studenta.

Argumenty funkcji:

- `@userID` – ID studenta.
- `@productID` – ID produktu.

```
CREATE FUNCTION [dbo].[CanAddProductToShoppingCart](@userID INT, @productID INT)
RETURNS BIT
AS
BEGIN
DECLARE @result BIT = 0;

IF EXISTS (SELECT 1 FROM Students WHERE UserID = @userID)
    AND EXISTS (SELECT 1 FROM Products WHERE ProductID = @productID)
    AND NOT EXISTS (
        SELECT 1
        FROM Orders O
        JOIN OrderDetails OD ON O.OrderID = OD.OrderID
        WHERE O.StudentID = @userID
            AND OD.ProductID = @productID
    )
BEGIN
    -- Jeśli wszystkie warunki są spełnione, produkt może być dodany
    SET @result = 1;
END

RETURN @result;
END
```

Get Product Price

Funkcja `GetProductPrice` oblicza cenę produktu dla studenta, uwzględniając ewentualne zniżki jeśli student wcześniej zakupił produkt powiązany z danym przedmiotem. Cena zostaje wtedy obniżona o 50%. Funkcja zwraca wartość typu `MONEY`, reprezentującą końcową cenę produktu.

Warunki integralnościowe:
Cena produktu jest obliczana na podstawie danych z tabel `Products`, `Orders`, `OrderDetails`, `Students` oraz `SubjectMeetings`. Jeśli student jest już zapisany na studia związane z danym przedmiotem (`StudyID`), funkcja stosuje rabat w wysokości 50%.

Argumenty funkcji:

- `@StudentID` – ID studenta, dla którego obliczana jest cena.
- `@ProductID` – ID produktu, którego cena ma zostać zwrócona.

```sql
CREATE FUNCTION dbo.GetProductPrice
(
    @StudentID INT,
    @ProductID INT
)
RETURNS MONEY
AS
BEGIN
    DECLARE @Price MONEY;
    DECLARE @OriginalPrice MONEY;
    DECLARE @BuyedProductsByStudent TABLE (ProductID INT);
    DECLARE @AssignedStudy INT

    SELECT @Price = price from Products
    WHERE ProductID = @ProductID

    INSERT INTO @BuyedProductsByStudent(ProductID) SELECT od.ProductID FROM OrderDetails od
    join Orders o on od.OrderID = o.OrderID
    join Students s on o.StudentID = s.UserID
    join SubjectMeetings sm on sm.ProductID = od.ProductID
    where s.UserID = @StudentID

    select @AssignedStudy = StudyID from EveryStudyMeeting
    where MeetingID = @ProductID

    IF EXISTS (
    SELECT 1
        FROM @BuyedProductsByStudent
        WHERE ProductID = @AssignedStudy
    )
    BEGIN
        SET @Price = @Price * 0.5; -- Zastosowanie rabatu 50%
    END

    RETURN @Price;
END
GO
```

Can Order Product In Shopping Cart

Funkcja `CanOrderProductInShoppingCart` sprawdza, czy dany produkt znajdujący się w koszyku studenta może zostać zamówiony. Zwraca wartość typu `BIT` – `1`, jeśli można złożyć zamówienie, oraz `0`, jeśli nie można. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Funkcja korzysta z `CanAddProductToShoppingCart`, aby sprawdzić, czy produkt może być dodany do koszyka.
- Sprawdzana jest dostępność miejsc dla danego produktu (`RemainingSlots` w widoku `RemainingPlacesOnProduct`). Produkt może zostać zamówiony, jeśli liczba pozostałych miejsc jest większa niż 0.

Argumenty funkcji:

- `@studentID` – ID studenta.

- `@productID` – ID produktu.

Zwracana wartość:

- `1`, jeśli produkt może zostać zamówiony.
- `0`, jeśli produkt nie może zostać zamówiony.

```
CREATE FUNCTION CanOrderProductInShoppingCart(@studentID INT, @productID INT)
RETURNS BIT
BEGIN
DECLARE @result BIT = dbo.canAddProductToShoppingCart(@studentID,@productID);
IF 0 >= (SELECT RPOP.RemainingSlots FROM RemainingPlacesOnProduct RPOP WHERE
RPOP.productID = @productID)
SET @result = 0;
RETURN @result;
END
```

Get Shopping Cart Value

Funkcja `GetShoppingCartValue` oblicza całkowitą wartość produktów znajdujących się w koszyku danego studenta. Zwraca wartość typu `MONEY`. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Jeśli koszyk jest pusty, funkcja zwraca wartość `0` (dzięki zastosowaniu `ISNULL`).

Argumenty funkcji:

- `@studentID` – ID studenta.

Zwracana wartość:

- Całkowita wartość produktów w koszyku danego studenta jako typ `MONEY`.

```
CREATE FUNCTION GetShoppingCartValue(@studentID INT)
RETURNS MONEY
BEGIN
DECLARE @result MONEY;
SELECT @result = ISNULL(SUM(products.price), 0)
FROM ShoppingCart
JOIN products ON ShoppingCart.productID = products.ProductID
WHERE ShoppingCart.studentID = @studentID
RETURN @result;
END
```

Get Module Type ID

Funkcja `GetModuleTypeID` zwraca id modułu po nazwie. Autor: Bartłomiej Kaczyński.

Argumenty funkcji:

- `@Name` – Nazwa typu kursu.

Zwracana wartość:

- ID tego typu kursu lub null jeśli nie istnieje.

```
CREATE FUNCTION GetModuleTypeID(@Name NVARCHAR(20))
RETURNS INT
BEGIN
   RETURN (
       SELECT TypeID FROM ModuleTypes WHERE Name = @Name
   )
END
```

Get Subject Type ID

Funkcja `GetSubjectTypeID` zwraca ID przedmiotu po nazwie. Autor: Kacper Wąchała.

Argumenty funkcji:

- `@Name` – Nazwa typu przedmiotu.

Zwracana wartość:

- ID tego typu przedmiotu lub null jeśli nie istnieje.

```
CREATE FUNCTION GetSubjectTypeID(@Name NVARCHAR(20))
RETURNS INT
BEGIN
RETURN (SELECT TypeID FROM SubjectTypes WHERE Name = @Name)
```

```
END
```

Get Subject Capacity

Pomocnicza funkcja `GetSubjectCapacity` zwraca ilość miejsc na danym przedmiocie kierunkowym po jego ID. Autor: Bartłomiej Kaczyński

Argumenty funkcji:

- `@SubjectID` – ID przedmiotu.

Zwracana wartość:

- Ilość miejsc tego typu przedmiotu lub null jeśli podany przedmiot nie istnieje

```
CREATE FUNCTION GetSubjectCapacity(@SubjectID int)
RETURNS INT
BEGIN
RETURN (
SELECT p.Capacity
FROM Products p
JOIN Studies s ON p.ProductID = s.ProductID
JOIN Subjects s2 ON s2.StudyID = s.ProductID
WHERE s2.SubjectID = @SubjectID
)
END
```

Get Module Capacity

Pomocnicza funkcja `GetModuleCapacity` zwraca ilość miejsc na danym module po jego ID. Autor: Bartłomiej Kaczyński

Argumenty funkcji:

- `@ModuleID` – ID modułu.

Zwracana wartość:

- Ilość miejsc tego modułu lub null jeśli podany moduł nie istnieje

```
CREATE FUNCTION GetModuleCapacity(@ModuleID INT)
RETURNS INT
BEGIN
RETURN (SELECT p.Capacity
FROM Products p
JOIN Courses c ON p.ProductID = c.ProductID
JOIN CourseModules cm ON cm.CourseID = c.ProductID
WHERE cm.ModuleID = @ModuleID)
END
```

Get Language ID

Funkcja `GetLanguageID` zwraca id języka po nazwie. Autor: Bartłomiej Kaczyński.

Argumenty funkcji:

- `@Language` – Nazwa języka.

Zwracana wartość:

- ID tego języka lub null jeśli nie istnieje.

```
CREATE FUNCTION GetLanguageID(@Language NVARCHAR(40))
RETURNS INT
BEGIN
RETURN (SELECT LanguageID FROM Languages WHERE Name = @Language)
END
```

Triggery

validate study presence

Trigger `validate_study_presence` pozwala przypisać obecność w tabeli `SubjectMeetingPresence` na przedmiocie tylko tym studentom którzy są do niego przypisani. Autor: Kacper Wąchała.

```
CREATE TRIGGER validate_study_presence
ON SubjectMeetingPresence
AFTER INSERT
AS
BEGIN
   DECLARE @StudentID INT;
   DECLARE @MeetingID INT;
   SELECT @StudentID = StudentID, @MeetingID = MeetingID
   FROM INSERTED

   IF @StudentID NOT IN (SELECT o.StudentID
     FROM OrderDetails od
     JOIN Orders o ON o.OrderID = od.OrderID
     WHERE od.ProductID = @MeetingID
     or od.ProductID IN (SELECT StudyID
     FROM EveryStudyMeeting
     WHERE MeetingID = @MeetingID))
       BEGIN
           ROLLBACK TRANSACTION;
           THROW 50000, 'Student not assigned to this meeting',
11;
```

```
        END
END;
```

validate course presence

Trigger `validate_course_presence` pozwala przypisać obecność w tabeli `CourseMeetingPresence` na kursie tylko osobom zapisanym na ten kurs. Autor: Bartosz Ludwin

```
CREATE TRIGGER validate__course_presence
ON CourseMeetingPresence
AFTER INSERT
AS
BEGIN
   DECLARE @StudentID INT;
   DECLARE @MeetingID INT;
   SELECT @StudentID = StudentID, @MeetingID = MeetingID
   FROM INSERTED


   IF @StudentID NOT IN (SELECT o.StudentID
     FROM OrderDetails od
     JOIN Orders o ON o.OrderID = od.OrderID
     WHERE od.ProductID IN (SELECT p.ProductID
     FROM Products p
     JOIN EveryCourseMeeting ecm ON p.ProductID = CourseID
     AND ecm.MeetingID = @MeetingID))
       BEGIN
           ROLLBACK TRANSACTION;
           THROW 50000, 'Student not assigned to this meeting', 11;
       END
END;
```

validate course makeup presence

Trigger `validate_course_makeup_presence` pozwala przypisać obecność w tabeli `CourseMeetingMakeupPresence` na kursie tylko osobom zapisanym i na obecny kurs i na spotkanie studyjne za które nadrabiają. Sprawdza też czy na pewno ten student ma obecność na tym spotkaniu i nieobecność na tamtym. Autor: Bartłomiej Kaczyński

```
CREATE TRIGGER validate_course_makeup_presence
ON CourseMeetingMakeupPresence
AFTER INSERT
AS
BEGIN
     DECLARE @StudentID INT;
     DECLARE @MeetingID INT;
```

```
      DECLARE @MakeupFor INT;
      SELECT @StudentID = StudentID, @MeetingID = MeetingID,
      @MakeupFor = MakeupFor
      FROM INSERTED

      IF @StudentID NOT IN (SELECT StudentID FROM
            StudentStudyMeetings WHERE MeetingID = @MakeupFor)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50000, 'Student was not assigned to the
missed meeting', 11;
            END
      IF @StudentID NOT IN (SELECT StudentID FROM
            StudentModuleMeetings WHERE MeetingID = @MeetingID)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50001, 'Student not assigned to this
meeting', 11;
            END
      IF @StudentID IN (SELECT StudentID
        FROM SubjectMeetingPresence
        WHERE MeetingID = @MakeupFor)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50002, 'Student was present on that
meeting', 16;
            END
      IF @StudentID NOT IN (SELECT StudentID
        FROM CourseMeetingPresence
        WHERE MeetingID = @MeetingID)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50003, 'Student was not present on this
meeting', 11;
            END
END;
```

validate study makeup presence

Trigger `validate_study_makeup_presence` pozwala przypisać obecność w tabeli `SubjectMeetingMakeupPresence` na kursie tylko osobom zapisanym i na obecne spotkanie studyjne i na spotkanie studyjne za które nadrabiają. Sprawdza też czy na pewno ten student ma obecność na tym i nieobecność na tamtym spotkaniu. Autor: Bartłomiej Kaczyński

```
CREATE TRIGGER validate_study_makeup_presence
ON SubjectMeetingMakeupPresence
AFTER INSERT
AS
BEGIN
```

```
      DECLARE @StudentID INT;
      DECLARE @MeetingID INT;
      DECLARE @MakeupFor INT;
      SELECT @StudentID = StudentID, @MeetingID = MeetingID,
      @MakeupFor = MakeupFor
      FROM INSERTED

      IF @StudentID NOT IN (SELECT StudentID FROM
            StudentStudyMeetings WHERE MeetingID = @MakeupFor)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50000, 'Student was not assigned to the
missed meeting', 11;
            END
      IF @StudentID NOT IN (SELECT StudentID FROM
            StudentStudyMeetings WHERE MeetingID = @MeetingID)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50001, 'Student not assigned to this
meeting', 11;
            END
      IF @StudentID IN (SELECT StudentID
        FROM SubjectMeetingPresence
        WHERE MeetingID = @MakeupFor)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50002, 'Student was present on that
meeting', 16;
            END
      IF @StudentID NOT IN (SELECT StudentID
        FROM SubjectMeetingPresence
        WHERE MeetingID = @MeetingID)
            BEGIN
                  ROLLBACK TRANSACTION;
                  THROW 50003, 'Student was not present on this
meeting', 11;
            END
END;
```

Uprawnienia:

Visitor (Gość)

Rola `visitor` umożliwia tworzenie konta studenta oraz przeglądanie informacji o produktach, przedmiotach i modułach kursów. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Użytkownik może tworzyć konto studenta poprzez procedurę `createStudent`.

- Użytkownik ma dostęp do tabel `Products`, `Subjects` i `CourseModules` w trybie SELECT.

```
CREATE ROLE visitor -- (gość)
GRANT EXECUTE ON createStudent TO visitor
GRANT SELECT ON Products TO visitor
GRANT SELECT ON Subjects TO visitor
GRANT SELECT ON CourseModules to visitor
```

Student (Uczestnik)

Rola `student` umożliwia zarządzanie kontem studenta oraz korzystanie z funkcji związanych z zakupami i zamówieniami. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Użytkownik może usuwać swoje konto poprzez procedurę `DeleteStudent`.
- Użytkownik może dodawać i usuwać produkty z koszyka poprzez procedury `AddProductToShoppingCart` i `RemoveProductFromShoppingCart`.
- Użytkownik może tworzyć nowe zamówienia za pomocą procedury `CreateNewOrder`.
- Użytkownik może przypisywać adres korespondencyjny poprzez procedurę `AssignStudentAddress`.
- Użytkownik ma dostęp do tabel `Products`, `Subjects`, `CourseModules`, `Webinars`, `SubjectGrades` oraz `InternshipPresence` w trybie SELECT.

```
CREATE ROLE student -- (uczestnik)
GRANT EXECUTE ON DeleteStudent TO student
GRANT EXECUTE ON AddProductToShoppingCart  TO student
GRANT EXECUTE ON RemoveProductFromShoppingCart TO student
GRANT EXECUTE ON CreateNewOrder TO student
GRANT EXECUTE ON AssignStudentAddress TO student
GRANT SELECT ON Products TO student
GRANT SELECT ON Subjects TO student
GRANT SELECT ON CourseModules TO student
GRANT SELECT ON Webinars TO student
GRANT SELECT ON SubjectGrades TO student
GRANT SELECT ON InternshipPresence TO student
```

Permission Admin

Rola `permissionAdmin` umożliwia zarządzanie rolami pracowników oraz przeglądanie informacji o pracownikach i ich rolach. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Rola ma uprawnienia do wykonywania procedur `CreateCoordinator`, `DeleteCoordinator`, `CreateLecturer`, `DeleteLecturer`, `CreateTranslator`, `DeleteTranslator`, `CreatePrincipal`, `DeletePrincipal`, `CreateInstructor`, `DeleteInstructor`, `CreateSecretary` oraz `DeleteSecretary`.
- Rola ma dostęp do tabel `Employees` i `EmployeeRoles` w trybie SELECT.

```
CREATE ROLE permissionAdmin
GRANT EXECUTE ON CreateCoordinator TO permissionAdmin
GRANT EXECUTE ON DeleteCoordinator TO permissionAdmin
GRANT EXECUTE ON CreateLecturer TO permissionAdmin
GRANT EXECUTE ON DeleteLecturer TO permissionAdmin
GRANT EXECUTE ON CreateTranslator TO permissionAdmin
GRANT EXECUTE ON DeleteTranslator TO permissionAdmin
GRANT EXECUTE ON CreatePrincipal TO permissionAdmin
GRANT EXECUTE ON DeletePrincipal TO permissionAdmin
GRANT EXECUTE ON CreateInstructor TO permissionAdmin
GRANT EXECUTE ON DeleteInstructor TO permissionAdmin
GRANT EXECUTE ON CreateSecretary TO permissionAdmin
GRANT EXECUTE ON DeleteSecretary TO permissionAdmin
GRANT SELECT ON Employees TO permissionAdmin
GRANT SELECT ON EmployeeRoles to permissionAdmin
```

Principal

Rola `principal` umożliwia zmianę statusu zamówień poprzez procedurę `changeOrderStatus`. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Rola ma uprawnienie do wykonywania procedury `changeOrderStatus`, która zmienia status zamówienia.

```
CREATE ROLE principal
GRANT EXEC ON changeOrderStatus TO principal
```

Lecturer (Wykładowca)

Rola `lecturer` umożliwia przeglądanie list obecności oraz informacji o studentach, a także zarządzanie tłumaczami. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Rola ma uprawnienia do wykonywania procedury `changeTranslator`.
- Rola ma dostęp do tabel `StudyAttendanceList`, `CourseAttendanceList`, `StudentCoursePresencePercentage`, `StudentStudyPresencePercentage` oraz `StudentInformation` w trybie SELECT.

```
CREATE ROLE lecturer
GRANT SELECT ON StudyAttendanceList TO lecturer
GRANT SELECT ON CourseAttendanceList TO lecturer
GRANT SELECT ON StudentCoursePresencePercentage TO lecturer
GRANT SELECT ON StudentStudyPresencePercentage TO lecturer
GRANT SELECT ON StudentInformation TO lecturer
GRANT EXEC ON changeTranslator TO lecturer
```

Planner Admin

Rola `plannerAdmin` umożliwia tworzenie webinarów, kursów oraz studiów, a także przeglądanie informacji o spotkaniach. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Rola ma uprawnienia do wykonywania procedur `createWebinar`, `createCourse` oraz `createStudy`.
- Rola ma dostęp do tabel `SubjectOnlineSynchronousMeetings`, `SubjectOfflineMeetings`, `OfflineModuleMeetings`, `OnlineSynchronousModuleMeetings` oraz `OnlineAsynchronousModuleMeetings` w trybie SELECT.

```
CREATE ROLE plannerAdmin
GRANT EXECUTE ON createWebinar TO plannerAdmin
GRANT EXECUTE ON createCourse TO plannerAdmin
GRANT EXECUTE ON createStudy TO plannerAdmin
GRANT SELECT ON SubjectOnlineSynchronousMeetings TO plannerAdmin
GRANT SELECT ON SubjectOfflineMeetings TO plannerAdmin
GRANT SELECT ON OfflineModuleMeetings TO plannerAdmin
GRANT SELECT ON OnlineSynchronousModuleMeetings TO plannerAdmin
GRANT SELECT ON OnlineAsynchronousModuleMeetings TO plannerAdmin
```

Coordinator

Rola `Coordinator` umożliwia przeglądanie obecności studentów oraz ocen. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Rola ma dostęp do tabel `StudentCoursePresencePercentage`, `StudentStudyPresencePercentage` oraz `subjectGrades` w trybie SELECT.

```
CREATE ROLE Coordinator
GRANT SELECT ON StudentCoursePresencePercentage TO Coordinator
GRANT SELECT ON StudentStudyPresencePercentage TO Coordinator
GRANT SELECT ON subjectGrades TO Coordinator
```

Translator

Rola `translator` umożliwia przeglądanie informacji o spotkaniach związanych z przedmiotami i modułami. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Rola ma dostęp do tabel `SubjectMeetings`, `SubjectOnlineSynchronousMeetings`, `SubjectOfflineMeetings`, `ModuleMeetings`, `OnlineSynchronousModuleMeetings`, `OfflineModuleMeetings` oraz `OnlineAsynchronousModuleMeetings` w trybie SELECT.

```
CREATE ROLE translator
GRANT SELECT ON SubjectMeetings TO translator
GRANT SELECT ON SubjectOnlineSynchronousMeetings TO translator
GRANT SELECT ON SubjectOfflineMeetings TO translator
GRANT SELECT ON ModuleMeetings TO translator
GRANT SELECT ON OnlineSynchronousModuleMeetings TO translator
GRANT SELECT ON OfflineModuleMeetings TO translator
GRANT SELECT ON OnlineAsynchronousModuleMeetings TO translator
```

Main Admin

Rola `mainAdmin` umożliwia usuwanie webinarów, kursów oraz studiów. Autor: Kacper Wąchała

Warunki integralnościowe:

- Rola ma uprawnienia do wykonywania procedur `deleteWebinar`, `deleteCourse` oraz `deleteStudies`.

```
CREATE ROLE mainAdmin
GRANT EXECUTE ON deleteWebinar TO mainAdmin
GRANT EXECUTE ON deleteCourse TO mainAdmin
GRANT EXECUTE ON deleteStudies TO mainAdmin
GRANT EXECUTE ON deleteWebinar TO mainAdmin
```

Secretary

Rola `secretary` umożliwia przeglądanie informacji o stypendiach, obecności studentów oraz list obecności na spotkaniach kursów i przedmiotów. Autor: Bartosz Ludwin.

Warunki integralnościowe:

- Rola ma dostęp do tabel `CanGetStipend`, `StudentStudyPresencePercentage`, `CourseStudyPresencePercentage`, `CourseMeetingsAttendanceList` oraz `SubjectMeetingsAttendanceList` w trybie SELECT.
- Rola ma uprawnienia do generowania raportów związanych z kolizjami terminów oraz finansami.

```
CREATE ROLE secretary
GRANT SELECT ON CanGetStipend TO secretary
GRANT SELECT ON StudentStudyPresencePercentage TO secretary
GRANT SELECT ON CourseStudyPresencePercentage TO secretary
GRANT SELECT ON CourseMeetingsAttendanceList TO secretary
GRANT SELECT ON SubjectMeetingsAttendanceList TO secretary
```

Backup Admin

Rola `backupAdmin` umożliwia dostęp do wszystkich obiektów w trybie SELECT, co jest przydatne do tworzenia kopii zapasowych danych. Autor: Bartłomiej Kaczyński

Warunki integralnościowe:

- Rola ma uprawnienia do przeglądania wszystkich tabel, widoków i procedur w trybie SELECT.

```
CREATE ROLE backupAdmin
GRANT SELECT ON SCHEMA :: [dbo] TO backupAdmin
```

Indeksy:

Courses:

Module Meetings

Autor: Bartłomiej Kaczyński

```
CREATE INDEX lecturer_module_meetings ON ModuleMeetings
(LecturerID)
CREATE INDEX translator_module_meetings ON ModuleMeetings
(Translator)
```

Offline Module Meetings

Autor: Bartłomiej Kaczyński

```
CREATE INDEX module_offline_module_meetings ON
OfflineModuleMeetings (ModuleID)
CREATE INDEX room_offline_module_meetings ON OfflineModuleMeetings
(RoomID)
CREATE INDEX start_offline_module_meetings ON
OfflineModuleMeetings (StartTime)
CREATE INDEX end_offline_module_meetings ON OfflineModuleMeetings
(EndTime)
```

Online Synchronous Module Meetings

Autor: Bartłomiej Kaczyński

```
CREATE INDEX module_online_synchronous_module_meetings ON
OnlineSynchronousModuleMeetings (ModuleID)
CREATE INDEX platform_online_synchronous_module_meetings ON
OnlineSynchronousModuleMeetings (PlatformID)
CREATE INDEX start_online_synchronous_module_meetings ON
OnlineSynchronousModuleMeetings (StartTime)
CREATE INDEX end_online_synchronous_module_meetings ON
OnlineSynchronousModuleMeetings (EndTime)
```

Online Asynchronous Module Meetings

Autor: Bartłomiej Kaczyński

```
CREATE INDEX module_online_asynchronous_module_meetings ON
OnlineAsynchronousModuleMeetings (ModuleID)
```

Course Modules

Autor: Bartłomiej Kaczyński

```
CREATE INDEX course_course_modules ON CourseModules (CourseID)
CREATE INDEX types_course_modules ON CourseModules (TypeID)
```

Courses

Autor: Bartłomiej Kaczyński

```
CREATE INDEX coordinator_course ON Courses (CoordinatorID)
```

Studies:

Subject Meetings

Autor: Bartłomiej Kaczyński

```
Create INDEX lecturer_subject_meetings ON SubjectMeetings
(LecturerID)
CREATE INDEX translator_subject_meetings ON SubjectMeetings
(Translator)
```

Subject Online Synchronous Meetings

Autor: Bartłomiej Kaczyński

```
CREATE INDEX subject_subject_online_synchronous_meetings ON
SubjectOnlineSynchronousMeetings (SubjectID)
CREATE INDEX platform_subject_online_synchronous_meetings ON
SubjectOnlineSynchronousMeetings (PlatformID)
CREATE INDEX start_subject_online_synchronous_meetings ON
SubjectOnlineSynchronousMeetings (StartTime)
CREATE INDEX end_subject_online_synchronous_meetings ON
SubjectOnlineSynchronousMeetings (EndTime)
```

Subject Offline Meetings

Autor: Bartłomiej Kaczyński

```
CREATE INDEX subject_subject_offline_meetings ON
SubjectOfflineMeetings (SubjectID)
CREATE INDEX room_subject_offline_meetings ON
SubjectOfflineMeetings (RoomID)
CREATE INDEX start_subject_offline_meetings ON
SubjectOfflineMeetings (StartTime)
CREATE INDEX end_subject_offline_meetings ON
SubjectOfflineMeetings (EndTime)
```

Internship Presence:

Autor: Bartosz Ludwin

```
CREATE INDEX InternshipPresence_StudentID ON InternshipPresence (StudentID);
CREATE INDEX InternshipPresence_Date ON InternshipPresence (Date);
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0,003125
Estimated Operator Cost	0,0033777 (100%)
Estimated CPU Cost	0,0002527
Estimated Subtree Cost	0,0033777
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	87
Estimated Number of Rows Per Execution	87
Estimated Number of Rows to be Read	87
Estimated Row Size	11 B
Ordered	False
Node ID	0

Object
[u_bludwin].[dbo].[InternshipPresence].[InternshipPresence_pk]
Output List
[u_bludwin].[dbo].[InternshipPresence].StudentID

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated I/O Cost	0,003125
Estimated Operator Cost	0,0033777 (100%)
Estimated CPU Cost	0,0002527
Estimated Subtree Cost	0,0033777
Estimated Number of Executions	1
Estimated Number of Rows for All Executions	87
Estimated Number of Rows Per Execution	87
Estimated Number of Rows to be Read	87
Estimated Row Size	10 B
Ordered	False
Node ID	0

Object
[u_bludwin].[dbo].[InternshipPresence].[InternshipPresence_pk]
Output List
[u_bludwin].[dbo].[InternshipPresence].Date

Subjects

Autor: Bartłomiej Kaczyński

```
CREATE INDEX study_subjects ON Subjects (StudyID)
CREATE INDEX types_subjects ON Subjects (TypeID)
CREATE INDEX instructor_subjects ON Subjects (InstructorID)
```

Studies:

Autor: Bartłomiej Kaczyński

```
CREATE INDEX supervisor_studies ON Studies (SupervisorID)
```

Webinars:

Autor: Bartłomiej Kaczyński

```
CREATE INDEX lecturer_webinars ON Webinars (LecturerID)
CREATE INDEX platform_webinars ON Webinars (PlatformID)
CREATE INDEX start_webinars ON Webinars (StartTime)
CREATE INDEX end_webinars ON Webinars (EndTime)
CREATE INDEX translator_webinars ON Webinars (TranslatorID)
```

Orders:

Orders:

Autor: Bartosz Ludwin

```
CREATE INDEX Orders_StudentID ON Orders(StudentID);
CREATE INDEX Orders_OrderDate ON Orders(OrderDate);
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0032831 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0032831
Estimated CPU Cost	0,0001581
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	147 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Orders].[StudentID]=CONVERT_IMPLICIT(int,[@1],0)
Object
[u_bludwin].[dbo].[Orders].[Orders_pk]
Output List
[u_bludwin].[dbo].[Orders].OrderID; [u_bludwin].[dbo].[Orders].StudentID; [u_bludwin].[dbo].[Orders].PaymentURL; [u_bludwin].[dbo].[Orders].OrderDate

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0049736 (100%)
Estimated I/O Cost	0,0046065
Estimated Subtree Cost	0,0049736
Estimated CPU Cost	0,0003671
Estimated Number of Executions	1
Estimated Number of Rows to be Read	191
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	147 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Orders].[OrderDate]=CONVERT_IMPLICIT(datetime,[@1],0)
Object
[u_bludwin].[dbo].[Orders].[Orders_pk]
Output List
[u_bludwin].[dbo].[Orders].OrderID; [u_bludwin].[dbo].[Orders].StudentID; [u_bludwin].[dbo].[Orders].PaymentURL; [u_bludwin].[dbo].[Orders].OrderDate

Order Details:

Autor: Bartosz Ludwin

```
CREATE INDEX OrderDetails_ProductID ON OrderDetails(ProductID);
CREATE INDEX OrderDetails_StatusID ON OrderDetails(StatusID);
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0034646 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0034646
Estimated CPU Cost	0,0003396
Estimated Number of Executions	1
Estimated Number of Rows to be Read	166
Estimated Number of Rows for All Executions	6,91667
Estimated Number of Rows Per Execution	6,91667
Estimated Row Size	27 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[OrderDetails].[ProductID]=CONVERT_IMPLICIT(int,[@1],0)
Object
[u_bludwin].[dbo].[OrderDetails].[OrderDetails_pk]
Output List
[u_bludwin].[dbo].[OrderDetails].OrderID; [u_bludwin].[dbo].[OrderDetails].ProductID; [u_bludwin].[dbo].[OrderDetails].PricePaid; [u_bludwin].[dbo].[OrderDetails].StatusID

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0034646 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0034646
Estimated CPU Cost	0,0003396
Estimated Number of Executions	1
Estimated Number of Rows to be Read	166
Estimated Number of Rows for All Executions	6
Estimated Number of Rows Per Execution	6
Estimated Row Size	27 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[OrderDetails].[StatusID]=CONVERT_IMPLICIT(int,[@1],0)
Object
[u_bludwin].[dbo].[OrderDetails].[OrderDetails_pk]
Output List
[u_bludwin].[dbo].[OrderDetails].OrderID; [u_bludwin].[dbo].[OrderDetails].ProductID; [u_bludwin].[dbo].[OrderDetails].PricePaid; [u_bludwin].[dbo].[OrderDetails].StatusID

Users:

Users:

Autor: Bartosz Ludwin

```
CREATE INDEX Users_Email ON Users(Email);
CREATE INDEX Users_LastName ON Users(LastName);
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0208917 (100%)
Estimated I/O Cost	0,0194213
Estimated Subtree Cost	0,0208917
Estimated CPU Cost	0,0014704
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1194
Estimated Number of Rows for All Executions	1,5
Estimated Number of Rows Per Execution	1,5
Estimated Row Size	171 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Users].[LastName]=N'Kowalski'
Object
[u_bludwin].[dbo].[Users].[Users_pk]
Output List
[u_bludwin].[dbo].[Users].UserID; [u_bludwin].[dbo].[Users].Email; [u_bludwin].[dbo].[Users].Password; [u_bludwin].[dbo].[Users].FirstName; [u_bludwin].[dbo].[Users].LastName; [u_bludwin].[dbo].[Users].Phone

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0032831 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0032831
Estimated CPU Cost	0,0001581
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	51 B
Ordered	True
Node ID	0

Object
[u_bludwin].[dbo].[Users].[Users_unique_email]
Output List
[u_bludwin].[dbo].[Users].Email
Seek Predicates
Seek Keys[1]: Prefix: [u_bludwin].[dbo].[Users].Email = Scalar Operator (CONVERT_IMPLICIT(nvarchar(4000),[@1],0))

Addresses:

Autor: Bartosz Ludwin

```
CREATE INDEX Addresses_CityID ON Addresses(CityID);
CREATE INDEX Addresses_ZipCode ON Addresses(ZipCode);
CREATE INDEX Addresses_UserID ON Addresses(UserID);
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0135166 (100%)
Estimated I/O Cost	0,0127546
Estimated Subtree Cost	0,0135166
Estimated CPU Cost	0,000762
Estimated Number of Executions	1
Estimated Number of Rows to be Read	550
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	110 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Addresses].[ZipCode]=CONVERT_IMPLICIT(nvarchar(4000),[@1],0)
Object
[u_bludwin].[dbo].[Addresses].[Addresses_pk]
Output List
[u_bludwin].[dbo].[Addresses].UserID; [u_bludwin].[dbo].[Addresses].Street; [u_bludwin].[dbo].[Addresses].ZipCode; [u_bludwin].[dbo].[Addresses].CityID

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0135166 (100%)
Estimated I/O Cost	0,0127546
Estimated Subtree Cost	0,0135166
Estimated CPU Cost	0,000762
Estimated Number of Executions	1
Estimated Number of Rows to be Read	550
Estimated Number of Rows for All Executions	2,4932
Estimated Number of Rows Per Execution	2,4932
Estimated Row Size	111 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Addresses].[CityID]=CONVERT_IMPLICIT(int,[@1],0)
Object
[u_bludwin].[dbo].[Addresses].[Addresses_pk]
Output List
[u_bludwin].[dbo].[Addresses].UserID; [u_bludwin].[dbo].[Addresses].Street; [u_bludwin].[dbo].[Addresses].ZipCode; [u_bludwin].[dbo].[Addresses].CityID

Clustered Index Seek (Clustered)

Scanning a particular range of rows from a clustered index.

Physical Operation	Clustered Index Seek
Logical Operation	Clustered Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0032831 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0032831
Estimated CPU Cost	0,0001581
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	111 B
Ordered	True
Node ID	0

Object
[u_bludwin].[dbo].[Addresses].[Addresses_pk]
Output List
[u_bludwin].[dbo].[Addresses].UserID; [u_bludwin].[dbo].[Addresses].Street; [u_bludwin].[dbo].[Addresses].ZipCode; [u_bludwin].[dbo].[Addresses].CityID
Seek Predicates
Seek Keys[1]: Prefix: [u_bludwin].[dbo].[Addresses].UserID = Scalar Operator(CONVERT_IMPLICIT(int,[@1],0))

Cities:

Autor: Bartosz Ludwin

```
CREATE INDEX Cities_Name ON Cities(Name);
CREATE INDEX Cities_CountryID ON Cities(CountryID);
```

Index Scan (NonClustered)

Scan a nonclustered index, entirely or only a range.

Physical Operation	Index Scan
Logical Operation	Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0032886 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0032886
Estimated CPU Cost	0,0001636
Estimated Number of Executions	1
Estimated Number of Rows to be Read	6
Estimated Number of Rows for All Executions	1,2
Estimated Number of Rows Per Execution	1,2
Estimated Row Size	30 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Cities].[Name]=CONVERT_IMPLICIT(nvarchar(4000),[@1],0)
Object
[u_bludwin].[dbo].[Cities].[Cities_unique_name]
Output List
[u_bludwin].[dbo].[Cities].CityID; [u_bludwin].[dbo].[Cities].CountryID; [u_bludwin].[dbo].[Cities].Name

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0032831 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0032831
Estimated CPU Cost	0,0001581
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	99 B
Ordered	True
Node ID	0

Object
[u_bludwin].[dbo].[Cities].[Cities_unique_name]
Output List
[u_bludwin].[dbo].[Cities].CityID; [u_bludwin].[dbo].[Cities].CountryID; [u_bludwin].[dbo].[Cities].Name
Seek Predicates
Seek Keys[1]: Prefix: [u_bludwin].[dbo].[Cities].CountryID = Scalar Operator(CONVERT_IMPLICIT(int,[@1],0))

Countries:

Autor: Bartosz Ludwin

```
CREATE INDEX Countries_Name ON Countries(Name);
```

Index Seek (NonClustered)

Scan a particular range of rows from a nonclustered index.

Physical Operation	Index Seek
Logical Operation	Index Seek
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0032831 (100%)
Estimated I/O Cost	0,003125
Estimated Subtree Cost	0,0032831
Estimated CPU Cost	0,0001581
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	84 B
Ordered	True
Node ID	0

Object
[u_bludwin].[dbo].[Countries].[UQ_Countries_Name]
Output List
[u_bludwin].[dbo].[Countries].CountryID; [u_bludwin].[dbo].[Countries].Name
Seek Predicates
Seek Keys[1]: Prefix: [u_bludwin].[dbo].[Countries].Name = Scalar Operator(CONVERT_IMPLICIT(nvarchar(4000),[@1],0))

Products:

Autor: Bartosz Ludwin

```
CREATE INDEX Products_Name ON Products(Name);
CREATE INDEX Products_Available ON Products(Available);
CREATE INDEX Products_Price ON Products(Price);
```

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0303838 (100%)
Estimated I/O Cost	0,0290509
Estimated Subtree Cost	0,0303838
Estimated CPU Cost	0,0013329
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1069
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	4102 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Products].[Name]=CONVERT_IMPLICIT(nvarchar(4000),[@1],0)
Object
[u_bludwin].[dbo].[Products].[Products_pk]
Output List
[u_bludwin].[dbo].[Products].ProductID; [u_bludwin].[dbo].[Products].Name; [u_bludwin].[dbo].[Products].Description; [u_bludwin].[dbo].[Products].Price; [u_bludwin].[dbo].[Products].Capacity; [u_bludwin].[dbo].[Products].Available

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0303838 (100%)
Estimated I/O Cost	0,0290509
Estimated Subtree Cost	0,0303838
Estimated CPU Cost	0,0013329
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1069
Estimated Number of Rows for All Executions	1069
Estimated Number of Rows Per Execution	1069
Estimated Row Size	4094 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Products].[Available]=[@1]
Object
[u_bludwin].[dbo].[Products].[Products_pk]
Output List
[u_bludwin].[dbo].[Products].ProductID; [u_bludwin].[dbo].[Products].Name; [u_bludwin].[dbo].[Products].Description; [u_bludwin].[dbo].[Products].Price; [u_bludwin].[dbo].[Products].Capacity; [u_bludwin].[dbo].[Products].Available

Clustered Index Scan (Clustered)

Scanning a clustered index, entirely or only a range.

Physical Operation	Clustered Index Scan
Logical Operation	Clustered Index Scan
Estimated Execution Mode	Row
Storage	RowStore
Estimated Operator Cost	0,0303838 (100%)
Estimated I/O Cost	0,0290509
Estimated Subtree Cost	0,0303838
Estimated CPU Cost	0,0013329
Estimated Number of Executions	1
Estimated Number of Rows to be Read	1069
Estimated Number of Rows for All Executions	1
Estimated Number of Rows Per Execution	1
Estimated Row Size	4094 B
Ordered	False
Node ID	0

Predicate
[u_bludwin].[dbo].[Products].[Price]=[@1]
Object
[u_bludwin].[dbo].[Products].[Products_pk]
Output List
[u_bludwin].[dbo].[Products].ProductID; [u_bludwin].[dbo].[Products].Name; [u_bludwin].[dbo].[Products].Description; [u_bludwin].[dbo].[Products].Price; [u_bludwin].[dbo].[Products].Capacity; [u_bludwin].[dbo].[Products].Available

Opis generowania danych:

Listy nazw i opisów produktów zostały wygenerowane za pomocą ChataGPT, a konkretniej o1-mini. Użyte polecenie:
“wygeneruj listę nazw i opisów produktów w bazie danych. Mają być wstawione do bazy danych uczelni jako produkty uczelni (kurs, webinar i studia) więc zachowaj tematykę uczelnianą. Napisz to w wygodnym formacie do obróbki przez program Python)“
po czym z tych list były dalej generowane listy nazw i opisów modułów/przedmiotów na każdym kursie/studium, i jeszcze dalej z tych list nazwy i opisy spotkań danych modułów/przedmiotów. Użyte polecenie:
“wygeneruj teraz do wyżej wygenerowanych produktów listę zawierającą nazwy i opisy modułów/przedmiotów na każdym kursie lub studium (ich podkategorie). Wygeneruj kilka dla każdego produktu, zapisz w wygodnym do obróbki w Pythonie formacie”
Reszta danych była generowana pseudolosowo w pythonie na podstawie bardzo fikuśnych wytycznych (np. Przypisywanie losowego tłumacza z danego języka w spotkaniach z modułów których id są liczbami pierwszymi, w reszcie spotkań brak tłumacza). Ten sam skrypt również generował listę poleceń EXECUTE przygotowanych wcześniej procedur do tworzenia kursów/studiów/spotkań itd. po czym wykonywane były one na bazie danych. Z uwagi na losowość danych wiele z nich nie przeszło warunków integralnościowych.

Szczegóły skryptów w języku Python

Do przetwarzania plików CSV i generowania skryptów SQL wykorzystano bibliotekę pandas. Skrypty te odczytują dane i konwertują je na zapytania SQL w następujących krokach:

1. **Wczytanie pliku CSV**
    - Użyto `pandas.read_csv()` do odczytu danych.
2. **Generowanie zapytań SQL**
    - Dla każdej tabeli utworzono odpowiednie instrukcje **INSERT**.
    - Dla pól **IDENTITY** (`ProductID`, `SubjectID`) zastosowano `SCOPE_IDENTITY()` do pobrania nowo utworzonego ID.
    - `SupervisorID` oraz `InstructorID` były losowane z określonych zakresów.
3. **Zapisywanie skryptów SQL**
    - Gotowe zapytania SQL zostały zapisane do plików `.sql`, które można uruchomić w SQL Server Management Studio (SSMS).

Wartości znajdujące się w bazie danych zostały wygenerowane z pomocą strony: https://sqldatagenerator.com/generator oraz chataGPT, a następnie za pomocą skryptu utworzonego przez kod w języku Python wdrożone w bazę danych.