Projekt Bazy Danych na rok akademicki 2024/2025

Autorzy: Szymon Tyburczy oraz Marek Swakoń

Spis treści

Studies_Stationary_Meeting
Subjects
Translator
Users
Webinar_Details
Webinar_OrdersWebinars

Procedury

AddCourse
AddEmployee
AddInternship
AddLanguage
AddOrder
AddOrderDetails
AddParticipantToCourse
AddPosition
AddStudies
AddStudent
AddSubjectToStudies
AddTranslator
AddUser
AddWebinar
DeleteOrder
UpdateCourse
UpdateMeetingDuration
AddModuleToCourse
DeleteInternship
DeleteUser
AddPosition
Add_Studies_Online_Sync_Meeting
Add_Studies_Online_Async_Meeting
Delete_Studies_Online_Sync_Meeting
Delete_Studies_Online_Async_Meeting
Add_Studies_Meeting
Delete_Studies_Meeting

CalculateAttendancePercentage CalculateMeetingDurationInMinutes GetAvailableCourseSeats GetAvailableSeats

Funkcje
GetAverageCoursePrice
GetParticipantAttendance
GetParticipantCount
GetFinancialReport
GetStudentGrades
GetStudentSchedule
GetStudiesSchedule
GetTotalOrderValue
HasStudentCompletedInternships
IsStudentEnrolledInMeeting
CheckStudentInternship
GetAttendanceSummary
GetCourseSchedule

```sql
CREATE INDEX EmplotyeePoistionID_Index ON Employees(Position_ID);

CREATE INDEX AccountStatus ON Users(Account_Status);

CREATE INDEX Users_PostalCode ON Users(Postal_Code);

CREATE INDEX Students_StudiesID ON Students(Studies_ID);

CREATE INDEX Courses_CoordinatorID ON Courses(Coordinator_ID);

CREATE INDEX ParticipantCourse_CompletionPercentage
ON Participant_Course(Completion_Percentage);

CREATE INDEX ParticipantWebinar_WebinarAccess
ON Participant_Webinar(Webinar_Access);

CREATE INDEX StudentsInternship_Completion
ON Student_Internship(Completion);

CREATE INDEX StudiesMeeting_InstructorID ON Studies_Meeting(Instructor_ID);

CREATE INDEX StudiesMeeting_LanguageID ON Studies_Meeting(Language_ID);

CREATE INDEX Webinars_InstructorID ON Webinars(Instructor_ID);

CREATE INDEX Webinars_WebinarDate ON Webinars(Webinar_Date);

CREATE INDEX OrdersDetails_CurrencyID ON Orders_Details(Currency_ID);

CREATE INDEX IX_Grades_Grades ON Grades(Grades);
```

Widoki
AttendanceList
CoordinatorAssignments
EventAttendanceSummary
FinancialSummaryPerUser
InactiveUsers
InstructorScheduleConflicts
InstructorsList
InternshipEndDates
LowAttendanceUsers
RevenueSummary
StudentsInternshipStatus
StudyMeetingsOverview
UserEnrollments
UserPostalData
UserScheduleConflicts

```sql
CREATE INDEX OrderDate ON Orders(Order_Date);

CREATE INDEX WebinarDate ON Webinars(Webinar_Date);

CREATE INDEX WebinarInstructorID ON Webinars(Instructor_ID);

CREATE INDEX StudiesMeeting_InstructorID ON Studies_Meeting(Instructor_ID);

CREATE INDEX CourdeMeeting_InstructorID ON Course_Meeting(Instructor_ID);

CREATE INDEX CourseMeeting_Date ON Course_Meeting(Date);

CREATE INDEX StudiesMeeting_Date ON Studies_Meeting_Details(Date);

CREATE INDEX StudiesMeeting_Form ON Studies_Meeting_Details(Form);

CREATE INDEX StudiesMeeting_Price ON Studies_Meeting_Details(Price);

CREATE INDEX StudiesMeeting_Subject ON Studies_Meeting(Subject_ID);
```

TO DO
Harmonogram zajęć dla studenta
Harmonogram danego kursu
Sprawdzamy czy student jest zapisany do jakiegoś spotkania studyjnego z studiów
Harmonogram danego kierunku studiów
Obliczenie łącznej wartości zamówienia
Obliczenie łącznej wartości zamówienia dla kursow Obliczenie łącznej wartości zamówienia dla studiow Sprawdzenie czy student odbył praktyki raz na pół roku i czy miał na nich 100% frekwencji

OGOLNIE POPATRZ SOBIE NA UPELA BO TAM SA DOKLADNE RZECZY KTORE ONI CHCA A NIE MAMY U NAS - DODALEM NA PRZYKLAD TABELE COURSES_MODULE

Użytkownicy systemu

- Gość
- Użytkownik systemu:
    - Student/Kursant
    - Prowadzący zajęcia
    - Administrator systemu
    - Księgowy
    - Dyrektor
    - Tłumacz
    - Pracownik administracji firmy
    - Koordynator praktyk

Funkcje użytkowników

- **Gość:**
    - Tworzenie konta
    - Przegląd kursów, studiów i webinarów

- **Użytkownik systemu (wszyscy poniżej dodatkowo posiadają):**
    - Zmiana hasła (przez admina na prośbę użytkownika)
    - Zmiana danych / usunięcie konta (przez admina na prośbę użytkownika)
    - Dostęp przez link do przypisanych webinarów/kursów on-line/nagrań

- **Student/Kursant:**
    - Przegląd kursów, studiów, dostępnych pojedynczych spotkań studyjnych i webinarów
    - Zapisanie się (dodanie do koszyka) na kursy, studia, płatne webinary
    - Zapisanie się za opłatą (dodanie do koszyka) na pojedyncze spotkanie studyjne
    - Zapisanie się na bezpłatne webinary
    - Obejrzenie nagrania ogólnodostępnego webinaru
    - Wykupienie dostępu (dodanie do koszyka) do nagrania z płatnego webinaru
    - Wykupienie koszyka
    - Wysłanie potwierdzenia płatności
    - Podgląd ocen
    - Podgląd frekwencji
    - Podgląd terminarza zajęć
    - Przegląd prowadzących
    - Generowanie raportu bilokacji
    - Dostęp przez link do wykupionych nagrań

- **Prowadzący zajęcia:**
  - Wpisywanie ocen
  - Zaznaczanie frekwencji na zajęciach
  - Podgląd terminarza zajęć
  - Dodawanie wydarzeń na studiach/kursach
  - Usprawiedliwianie nieobecności studentów
  - Przegląd tłumaczy oraz przypisanie tłumacza do swoich zajęć
  - Generowanie raportu listy obecności

- **Administrator systemu:**
  - Dostęp do wszystkich tabel w bazie danych i możliwość ręcznego modyfikowania, usuwania bądź dodawania dowolnego wpisu

- **Księgowy:**
  - Generowanie raportu zestawienia przychodów dla każdego webinaru/kursu/studium
  - Generowanie listy “dłużników”
  - Zatwierdzanie wpłat

- **Dyrektor:**
  - Wyrażenie zgody na odroczenie płatności w wyjątkowych przypadkach
  - Przegląd terminarzy studentów oraz prowadzących
  - Przegląd ocen oraz frekwencji studentów
  - Wykreślenie lub dopisanie studenta do kursu
  - Generowanie wszystkich dostępnych raportów

- **Tłumacz:**
  - Podgląd terminarza zajęć

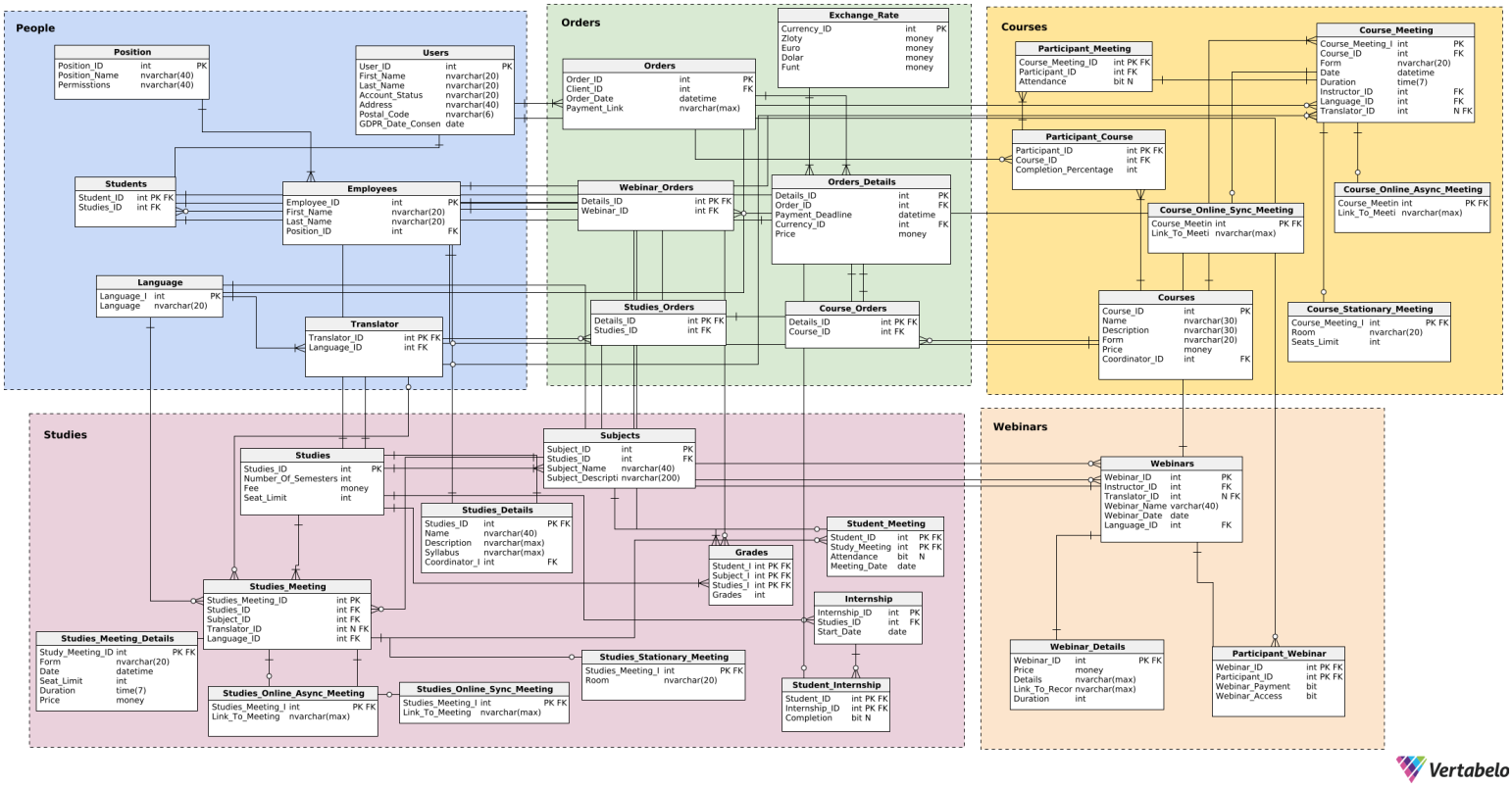
- **Pracownik administracji firmy:**
  - Układanie sylabusa studiów
  - Układanie terminarza zajęć dla poszczególnych prowadzących oraz studentów
  - Zmiana terminu wydarzenia
  - Wprowadzenie informacji o wysłanym dyplomie dla studenta
  - Generowanie raportów:
    - ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia,
    - ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach,
    - lista obecności dla każdego szkolenia z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie,
    - raport bilokacji

- **Koordynator praktyk (będący tutorem):**
  - Zaznaczenie frekwencji na praktykach
  - Wystawienie oceny końcowej z praktyk
  - Podgląd terminarza studenta (praktykanta)

Funkcje systemu

- **Funkcje systemu:**
    - Generowanie raportów:
        - Raporty finansowe – zestawienie przychodów dla każdego webinaru/kursu/studium.
        - Lista „dłużników” – osoby, które skorzystały z usług, ale nie uiściły opłat.
        - Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia (z informacją, czy wydarzenie jest stacjonarnie, czy zdalnie).
        - Ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach.
        - Lista obecności dla każdego szkolenia z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie.
        - Raport bilokacji: lista osób, które są zapisane na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo.
    - Zlicza frekwencję użytkownika na studiach/kursach/praktykach
    - Potwierdza zaliczenie modułu realizowanego online w trybie asynchronicznym
    - Po zakończeniu webinaru lub kursu online w trybie synchronicznym udostępnia nagranie użytkownikowi na 30 dni
    - Przechowuje produkty w koszyku użytkownika
    - Przechowuje informacje o dokonanych płatnościach
    - Sprawdza terminowość wpłat
    - Blokuje dostęp do kursu lub studiów, jeśli pełna opłata nie zostanie uiszczona co najmniej 3 dni przed ich rozpoczęciem.
    - Blokuje możliwość zapisania się na zajęcia, jeśli limit miejsc został osiągnięty
    - Po zatwierdzeniu wpłaty, przypisuje użytkownika do kursu/webinaru/studiów

Schemat



Vertabelo

Opisy Tabel

Activities

Tabela *Activities* w kolumnie *activity_id* zawiera id wszystkich aktywności, z których student może korzystać oraz w kolumnie *to_buy* zawiera bit informujący, czy daną aktywność można kupić. Aktywności, których nie można kupić (*Course_modules*), są dostępne w ramach większych modułów (Courses).

Warunki integralnościowe:

- Domyślnie dodawana aktywność jest oznaczana jako taka, którą można kupić (*to_buy* = 1)

```
-- Table: Activities
CREATE TABLE Activities (
    activity_id int  NOT NULL,
    to_buy bit  NOT NULL DEFAULT 1,
    CONSTRAINT Activities_pk PRIMARY KEY  (activity_id)
    CONSTRAINT Activities_activity_id CHECK ((activity_id > 0) OR
(activity_id < 100000))

);
```

Administration_employees

Tabela *Administration_employees* w kolumnie *user_id* zawiera id pracowników uczelni, z wyłączeniem tłumaczy, nauczyczycieli i studentów. W kolumnie *role_id* przechowywane jest id jego stanowiska.

```
-- Table: Administration_employees
CREATE TABLE Administration_employees (
    user_id int  NOT NULL,
    role_id int  NOT NULL,
    CONSTRAINT Administration_employees_pk PRIMARY KEY  (user_id)

);

ALTER TABLE Administration_employees ADD CONSTRAINT
Administration_employees_Roles
    FOREIGN KEY (role_id)
    REFERENCES Roles (role_id);

ALTER TABLE Administration_employees ADD CONSTRAINT
Administration_employees_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id);
```

Attendance

Tabela *Attendance* w kolumnie *meeting_id* przechowuje id spotkania, do każdego spotkania przyporządkowani są wszyscy zapisani na nie uczniowie (kolumna *student_id*). Te dwie kolumny tworzą klucz główny. Każdemu student_id na konkretnym spotkaniu przypisywany jest bit *attendance*, gdzie *1* - uczeń obecny, *0* - uczeń nieobecny.

Warunki integralnościowe:

- Domyślnie wartość w kolumnie *attendance* = 0

```
-- Table: Attendance
CREATE TABLE Attendance (
   meeting_id int  NOT NULL,
   student_id int  NOT NULL,
   attendance bit  NOT NULL DEFAULT 0,
   CONSTRAINT Attendance_pk PRIMARY KEY  (meeting_id,student_id)
);

ALTER TABLE Attendance ADD CONSTRAINT Attendence_Meetings
    FOREIGN KEY (meeting_id)
    REFERENCES Meetings (meeting_id);

ALTER TABLE Attendance ADD CONSTRAINT Attendence_Students
    FOREIGN KEY (student_id)
    REFERENCES Students (student_id);
```

City

Tabela *City* jest tabelą słownikową. W kolumnie *city_id* zawiera id miasta, w kolumnie *city* jego nazwę, a w kolumnie *country_id* id kraju w którym się znajduje.

```
-- Table: City
CREATE TABLE City (
    city_id int  NOT NULL IDENTITY(1, 1),
    city nvarchar(32)  NOT NULL,
    country_id int  NOT NULL,
    CONSTRAINT City_pk PRIMARY KEY  (city_id)
);

ALTER TABLE City ADD CONSTRAINT city_Country
    FOREIGN KEY (country_id)
    REFERENCES Country (country_id);
```

Classrooms

Tabela *Classrooms* jest tabelą słownikową. W kolumnie *classroom_id* przechowuje id sali, w kolumnie *classroom_name* nazwę sali, a w kolumnie *place_limit* limit osób w sali.

Warunki integralnościowe:

- nazwy sal (*classroom_name*) są unikalne
- limit miejsc w sali (*place_limit*) musi być większy niż 0 i mniejszy niż 1000.

```
-- Table: Classrooms
CREATE TABLE Classrooms (
    classroom_id int  NOT NULL IDENTITY(1, 1),
    classroom_name nvarchar(32)  NOT NULL,
    place_limit int  NOT NULL,
    CONSTRAINT ClassroomsClassroom_name UNIQUE (classroom_name),
    CONSTRAINT Classrooms_pk PRIMARY KEY  (classroom_id),
    CONSTRAINT Classrooms_place_limit CHECK (place_limit > 0 and
place_limit < 1000)

);
```

Country

Tabela *Country* jest tabelą słownikową. W kolumnie *country_id* przechowuje id kraju, a w kolumnie country jego nazwę.

Warunki integralnościowe:

- nazwy krajów (*country*) są unikalne

```
-- Table: Country
CREATE TABLE Country (
    country_id int  NOT NULL IDENTITY(1, 1),
    country nvarchar(32)  NOT NULL,
    CONSTRAINT CountryCountry UNIQUE (country),
    CONSTRAINT Country_pk PRIMARY KEY  (country_id)
);
```

Course_modules

Tabela *Course_modules* dla każdego modułu kursu zawiera jego *activity_id*, *course_id* oraz *module_type_id*, określającego jego typ (powiązanie z tabelą *Module_type*)

```
-- Table: Course_modules
CREATE TABLE Course_modules (
    activity_id int  NOT NULL,
    course_id int  NOT NULL,
    module_type_id smallint  NOT NULL,
    CONSTRAINT Course_modules_pk PRIMARY KEY  (activity_id)
);

ALTER TABLE Course_modules ADD CONSTRAINT Course_modules_Activities
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);

ALTER TABLE Course_modules ADD CONSTRAINT Course_modules_Courses
    FOREIGN KEY (course_id)
    REFERENCES Courses (course_id);

ALTER TABLE Course_modules ADD CONSTRAINT Course_modules_Module_type
    FOREIGN KEY (module_type_id)
    REFERENCES Module_type (module_type_id);
```

Courses

Tabela *Courses* przechowuje *activity_id*, *course_id* każdego z kursów. W kolumnie *course_name* znajduje się jego nazwa, w kolumnie *price_for_course* cena za dany kurs, *places_occupied* określa ilość osób wpisanych na kurs, *place_limit*, to limit liczby osób, które mogą zapisać się na kurs. W kolumnie *entry_fee* znajduje się zaliczka za dany kurs.

Warunki integralnościowe:

- przypisanie kolumnom wartości spełniających warunek *places_occupied* <= *place_limit* czyli liczba miejsc zajętych, musi być mniejsza lub równa limitowi miejsc lub też obie kolumny przyjmują wartość NULL (dla kursów w pełni zdalnych).
- Domyślnie liczba osób zapisanych na kurs (*places_occupied*) = 0
- *Courses_place_limit* jest większy od 0 oraz mniejszy od 1000, lub limit nie istnieje
- Cena za kurs (*price_for_course*) jest większa od 0 i mniejsza od 10 000
- Zaliczka za kurs (*entry_fee*) jest większa od 0 i mniejsza od całkowitej ceny za kurs

```
-- Table: Courses
CREATE TABLE Courses (
    course_id int  NOT NULL IDENTITY(1, 1),
    activity_id int  NOT NULL,
    course_name nvarchar(64)  NOT NULL,
    price_for_course money  NOT NULL,
    places_occupied int  NULL DEFAULT 0,
    place_limit int  NULL,
    entry_fee int  NOT NULL,
    CONSTRAINT CoursesPlaces_occupiedPlace_limit CHECK ((places_occupied
IS NULL AND place_limit IS NULL) OR (places_occupied <= place_limit)),
    CONSTRAINT Courses_pk PRIMARY KEY  (course_id),
    CONSTRAINT Courses_place_limit CHECK ((place_limit > 0 and place_limit
< 1000) or place_limit IS NULL)
    CONSTRAINT Courses_price_for_course CHECK (price_for_course > 0 and
price_for_course < 10000)
   CONSTRAINT Courses_entry_fee CHECK (entry_fee > 0 and entry_fee <
price_for_course)

);

ALTER TABLE Courses ADD CONSTRAINT Courses_Events
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);
```

Internships

Tabela *Internships* dla każdych studiów (*studies_id*) przypisuje koordynatora praktyk (*coordinator_id*)

```
-- Table: Internships
CREATE TABLE Internships (
    studies_id int  NOT NULL,
    coordinator_id int  NOT NULL,
    CONSTRAINT Internships_pk PRIMARY KEY  (studies_id)
);

ALTER TABLE Internships ADD CONSTRAINT Internships_Studies
    FOREIGN KEY (studies_id)
    REFERENCES Studies (studies_id);

ALTER TABLE Internships ADD CONSTRAINT Internships_Tutor
    FOREIGN KEY (coordinator_id)
    REFERENCES Tutor (user_id);
```

Internships_attendance

Tabela *Internships_attendance* dla każdego dnia praktyk *(internships_day*) dla każdego rodzaju studiów w tym dniu (*studies_id*) dla każdego studenta z tych studiów, który powinien uczestniczyć w tym dniu w praktykach (*student_id*) przypisuje bit (*attendance*) 1 - obecny, 0 - nieobecny. Do każdego dnia przypisana jest data (*date*).

Warunki integralnościowe:

- Wpisany dzień praktyk (*internships_day*) musi być liczbą całkowitą z zakresu 1-14
- Domyślnie obecność (*attendance*) ustawiona jest na 0 (nieobecny).
- Data dnia praktyk musi być realistyczna (między 2020 a 2025 rokiem)

```
-- Table: Internships_attendance
CREATE TABLE Internships_attendance (
    internships_day int  NOT NULL,
    studies_id int  NOT NULL,
    student_id int  NOT NULL,
    date date  NOT NULL,
    attendance bit  NOT NULL DEFAULT 0,
    CONSTRAINT Internships_attendance_pk PRIMARY KEY
(internships_day,studies_id,student_id)
    CONSTRAINT Internships_attendance_day_check CHECK (internships_day
BETWEEN 1 AND 14)
    CONSTRAINT Internships_attendance_date_check CHECK (date BETWEEN
'2020-01-01' AND '2026-01-01')

);

ALTER TABLE Internships_attendance ADD CONSTRAINT
Internships_attendance_Internships
    FOREIGN KEY (studies_id)
    REFERENCES Internships (studies_id);

ALTER TABLE Internships_attendance ADD CONSTRAINT
Internships_attendance_Students
    FOREIGN KEY (student_id)
    REFERENCES Students (student_id);
```

Languages

Tabela Languages jest tabelą słownikową. Dla każdego id języka (*language_id*) przechowuje nazwę tego języka (*language*).

Warunki integralnościowe:

- nazwy języków (*language*) są unikalne

```
-- Table: Languages
CREATE TABLE Languages (
    language_id int  NOT NULL IDENTITY(1, 1),
    language nvarchar(32)  NOT NULL,
    CONSTRAINT LanguagesLanguage UNIQUE (language),
    CONSTRAINT Languages_pk PRIMARY KEY  (language_id)
);
```

Meetings

Tabela *meetings* przechowuje id spotkania (*meeting_id*), do każdego spotkania może być przypisany język (*language*), jeśli wartość w tej kolumnie wynosi NULL, znaczy to, że spotkanie prowadzone jest w języku polskim. Do spotkania może być przypisany tłumacz, mamy tutaj informację o jego id (*translator_id*), jeśli wartość w tej kolumnie wynosi NULL, oznacza to, że do spotkania nie został przypisany tłumacz. W kolumnie *activity_id* mamy informację, aktywności o jakim id dotyczy to spotkanie.

```
-- Table: Meetings
CREATE TABLE Meetings (
    meeting_id int  NOT NULL IDENTITY(1, 1),
    translator_id int  NULL,
    language_id int  NULL,
    activity_id int  NOT NULL,
    CONSTRAINT Meetings_pk PRIMARY KEY  (meeting_id)
);

ALTER TABLE Meetings ADD CONSTRAINT Meetings_Course_modules
    FOREIGN KEY (activity_id)
    REFERENCES Course_modules (activity_id);

ALTER TABLE Meetings ADD CONSTRAINT Meetings_Study_courses
    FOREIGN KEY (activity_id)
    REFERENCES Study_courses (activity_id);

ALTER TABLE Meetings ADD CONSTRAINT Meetings_Translator
    FOREIGN KEY (translator_id,language_id)
    REFERENCES Translator (user_id,known_language_id);

ALTER TABLE Meetings ADD CONSTRAINT Meetings_Webinar
    FOREIGN KEY (activity_id)
    REFERENCES Webinar (activity_id);
```

Module_meetings

Tabela Module_meetings przechowuje informacje o przyporządkowaniu spotkań do konkretnych modułów w ramach danego kierunku studiów. Klucz główny składa się z kolumn *studies_id* oraz *module_number*, które jednoznacznie identyfikują moduł. Kolumna *meeting_id* przechowuje identyfikator konkretnego spotkania.

```
-- Table: Module_meetings
CREATE TABLE Module_meetings (
    studies_id int  NOT NULL,
    module_number int  NOT NULL,
    meeting_id int  NOT NULL,
    CONSTRAINT Module_meetings_pk PRIMARY KEY  (studies_id,module_number)
);

ALTER TABLE Module_meetings ADD CONSTRAINT Module_meetings_Meetings
    FOREIGN KEY (meeting_id)
    REFERENCES Meetings (meeting_id);

ALTER TABLE Module_meetings ADD CONSTRAINT Module_meetings_Studies_module
    FOREIGN KEY (studies_id,module_number)
    REFERENCES Studies_module (studies_id,module_number,meeting_id);
```

Module_Type

Tabela Module_type zawiera informacje o typach modułów. Kolumna *module_type_id* identyfikuje unikalnie każdy typ modułu, a kolumna *module_type* przechowuje jego nazwę.

Warunki integralnościowe:

- unikalność wartości w kolumnie *module_type*.
- *module_type_id* to liczba całkowita z zakresu 1-100

```
-- Table: Module_type
CREATE TABLE Module_type (
    module_type_id smallint  NOT NULL,
    module_type nvarchar(32)  NOT NULL,
    CONSTRAINT Module_typeModule_type UNIQUE (module_type),
    CONSTRAINT Module_type_pk PRIMARY KEY  (module_type_id)
    CONSTRAINT Module_type_type_check CHECK (module_type_id BETWEEN 1 and
100)
);
```

Online_asynchronous

Tabela Online_asynchronous przechowuje informacje o spotkaniach asynchronicznych. Kolumna *meeting_id* identyfikuje spotkanie, a *recording_URL_address* przechowuje unikalny adres URL nagrania spotkania.

Warunki integralnościowe:

- unikalność adresu URL nagrania (recording_URL_address)

```
-- Table: Online_asynchronous
CREATE TABLE Online_asynchronous (
    meeting_id int  NOT NULL,
    recording_URL_address nvarchar(64)  NOT NULL,
    CONSTRAINT Online_asynchronousRecording_URL_address UNIQUE
(recording_URL_address),
    CONSTRAINT Online_asynchronous_pk PRIMARY KEY  (meeting_id)
);

ALTER TABLE Online_asynchronous ADD CONSTRAINT
Online_asynchronous_Meetings
    FOREIGN KEY (meeting_id)
    REFERENCES Meetings (meeting_id);
```

Online_synchronous

Tabela Online_synchronous zawiera informacje o spotkaniach synchronicznych online. Kolumna *meeting_id* identyfikuje spotkanie, *start_time* oraz *end_time* określają czas jego trwania, natomiast *recording_URL_address* oraz *meeting_URL_address* przechowują odpowiednio adres URL nagrania oraz adres URL samego spotkania. Kolumna *platform_id* wskazuje platformę, na której odbywa się spotkanie.

Warunki integralnościowe:

- **Unikalność adresu URL nagrania** (`recording_URL_address`): Wartość w tej kolumnie musi być unikalna, co oznacza, że nie mogą istnieć dwa spotkania z takim samym adresem URL nagrania.
- **Unikalność adresu URL spotkania** (`meeting_URL_address`): Adres URL spotkania musi być unikalny, co oznacza, że każda platforma spotkania powinna mieć unikalny URL.
- **Czas rozpoczęcia musi być wcześniejszy niż czas zakończenia** (`start_time < end_time`): Spotkanie musi zacząć się przed jego zakończeniem.
- **Zakres dat dla `start_time`**: Czas rozpoczęcia spotkania musi mieścić się w przedziale od 1 stycznia 2020 roku do 1 stycznia 2026 roku.
- **Zakres dat dla `end_time`**: Czas zakończenia spotkania musi mieścić się w przedziale od 1 stycznia 2020 roku do 1 stycznia 2026 roku.
- **Max czas trwania spotkania** (`DATEDIFF(HOUR, start_time, end_time) <= 3`): Ten warunek zapewnia, że różnica między czasem rozpoczęcia (`start_time`) a czasem zakończenia (`end_time`) spotkania nie może wynosić więcej niż 3 godziny. Funkcja `DATEDIFF(HOUR, start_time, end_time)` oblicza różnicę w godzinach, a warunek zapewnia, że ta różnica nie przekroczy 3 godzin.

```
-- Table: Online_synchronous
CREATE TABLE Online_synchronous (
    meeting_id int  NOT NULL,
    start_time datetime  NOT NULL,
    end_time datetime  NOT NULL,
    recording_URL_address nvarchar(64)  NOT NULL,
    meeting_URL_address nvarchar(64)  NOT NULL,
    platform_id int  NOT NULL,
    CONSTRAINT Online_synchronousRecording_URL_address UNIQUE
(recording_URL_address),
    CONSTRAINT Online_synchronousMeeting_URL_address UNIQUE
(meeting_URL_address),
    CONSTRAINT Online_synchronousStart_timeEnd_time CHECK (start_time <
end_time),
    CONSTRAINT Online_synchronousStart_time CHECK (start_time BETWEEN
'2020-01-01' AND '2026-01-01'),
```

```
    CONSTRAINT Online_synchronousEnd_time CHECK (end_time BETWEEN
'2020-01-01' AND '2026-01-01'),
    CONSTRAINT Online_synchronousMax_duration CHECK (DATEDIFF(HOUR,
start_time, end_time) <= 3),
    CONSTRAINT Online_synchronous_pk PRIMARY KEY  (meeting_id)
);

ALTER TABLE Online_synchronous ADD CONSTRAINT Online_synchronous_Meetings
    FOREIGN KEY (meeting_id)
    REFERENCES Meetings (meeting_id);

ALTER TABLE Online_synchronous ADD CONSTRAINT Online_synchronous_Platforms
    FOREIGN KEY (platform_id)
    REFERENCES Platforms (platform_id);
```

Order_details

Tabela Order_details przechowuje szczegółowe informacje o zamówieniach. Klucz główny składa się z kombinacji kolumn *order_id*, *activity_id* oraz *description*, które jednoznacznie identyfikują zamówienie. Dodatkowo kolumna *price* przechowuje cenę, a *payment_date* wskazuje datę płatności (jeśli została zrealizowana).

Warunki integralnościowe:

- Cena zamówienia (`price`) musi być w przedziale od 0 do 5000. Dzięki temu kontrolujemy, by ceny nie wychodziły poza akceptowalny zakres.
- Data płatności (`payment_date`) musi być mniejsza lub równa bieżącej dacie. Oznacza to, że płatność nie może być zaplanowana na przyszłość. Jeśli płatność nie została zrealizowana, data może pozostać pusta (`NULL`).

```
-- Table: Order_details
CREATE TABLE Order_details (
    order_id int  NOT NULL,
    activity_id int  NOT NULL,
    description nvarchar(64)  NOT NULL,
    price money  NOT NULL,
    payment_date datetime  NULL,
    CONSTRAINT Order_details_pk PRIMARY KEY
(order_id,activity_id,description)
),
    CONSTRAINT chk_price CHECK (price >= 0 AND price <= 5000),
    CONSTRAINT chk_payment_date CHECK (payment_date <= GETDATE() OR
payment_date IS NULL);


ALTER TABLE Order_details ADD CONSTRAINT Order_details_Activities
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);

ALTER TABLE Order_details ADD CONSTRAINT Order_details_Orders
    FOREIGN KEY (order_id)
    REFERENCES Orders (order_id);
```

Orders

Tabela Orders przechowuje informacje o zamówieniach składanych przez studentów. Kolumna *order_id* identyfikuje zamówienie, kolumna *student_id* wskazuje studenta, który je złożył, a *order_date* przechowuje datę złożenia zamówienia.
Warunki integralnościowe:

- Kolumna `order_date` ma domyślną wartość ustawioną na bieżącą datę i godzinę w momencie wstawiania rekordu. Dzięki temu, jeśli nie zostanie określona wartość dla tej kolumny, automatycznie przyjmuje wartość aktualnej daty i godziny.
- Ograniczenie dla kolumny `order_date`, które zapewnia, że data zamówienia będzie pomiędzy 1 stycznia 2020 a bieżącą datą (czyli nie będzie starsza niż 1 stycznia 2020, ani późniejsza niż dzisiaj). Wartość `GETDATE()` zwraca aktualną datę i godzinę w momencie wykonywania zapytania.

```
-- Table: Orders
CREATE TABLE Orders (
    order_id int  NOT NULL IDENTITY(1, 1),
    student_id int  NOT NULL,
    order_date datetime  NOT NULL DEFAULT GETDATE(),
    CONSTRAINT Orders_pk PRIMARY KEY  (order_id),
    CONSTRAINT chk_order_date CHECK (order_date >= '2020-01-01' AND
order_date <= GETDATE())
);

ALTER TABLE Orders ADD CONSTRAINT Orders_Students
    FOREIGN KEY (student_id)
    REFERENCES Students (student_id);
```

Platforms

Tabela Platforms przechowuje informacje o platformach wykorzystywanych podczas spotkań online. Kolumna *platform_id* identyfikuje platformę, a kolumna *platform_URL_address* przechowuje jej unikalny adres URL.

Warunki integralnościowe:

- unikalność adresu URL platformy (*platform_URL_address*)

```
-- Table: Platforms
CREATE TABLE Platforms (
    platform_id int  NOT NULL IDENTITY(1, 1),
    platform_URL_address nvarchar(64)  NOT NULL,
    CONSTRAINT PlatformsPlatform_URL_address UNIQUE
(platform_URL_address),
    CONSTRAINT Platforms_pk PRIMARY KEY  (platform_id)
);
```

Roles

Tabela Roles przechowuje informacje o rolach przypisanych użytkownikom systemu. Kolumna *role_id* identyfikuje unikalnie rolę, natomiast *role_name* zawiera jej nazwę.

Warunki integralnościowe:

- unikalność nazwy roli (*role_name*)

```
-- Table: Roles
CREATE TABLE Roles (
    role_id int  NOT NULL IDENTITY(1, 1),
    role_name nvarchar(32)  NOT NULL,
    CONSTRAINT RolesRole_name UNIQUE (role_name)
    CONSTRAINT Roles_pk PRIMARY KEY  (role_id)
);
```

Semesters

Tabela Semesters zawiera informacje o semestrach akademickich. Kolumna *semester* identyfikuje numer semestru, a *start_date* oraz *end_date* określają jego czas trwania.

Warunki integralnościowe:

- Ograniczenie, które zapewnia, że data rozpoczęcia semestru (`start_date`) musi być wcześniejsza niż data zakończenia semestru (`end_date`). To gwarantuje logiczną poprawność danych.
- Ograniczenie, które zapewnia, że numer semestru (`semester`) mieści się w przedziale od 1 do 10. Określa to zakres semestrów, które mogą występować w systemie (np. od 1 do 10 semestru).
- Ograniczenie, które ustala, że data rozpoczęcia semestru musi być pomiędzy 1 stycznia 2020 a 1 stycznia 2026. Zapewnia to, że wszystkie semestry są w odpowiednim zakresie czasowym.
- Ograniczenie, które ustala, że data zakończenia semestru musi być pomiędzy 1 stycznia 2020 a 1 stycznia 2026. Zapewnia to, że wszystkie semestry są w odpowiednim zakresie czasowym.

```
-- Table: Semesters
CREATE TABLE Semesters (
    semester int  NOT NULL,
    start_date date  NOT NULL,
    end_date date  NOT NULL,
    CONSTRAINT SemestersStart_timeEnd_time CHECK (start_date < end_date),
    CONSTRAINT Semesters_pk PRIMARY KEY  (semester),
    CONSTRAINT chk_semester_id CHECK (semester BETWEEN 1 AND 10),
    CONSTRAINT chk_start_date CHECK (start_date BETWEEN '2020-01-01' AND
'2026-01-01'),
    CONSTRAINT chk_end_date CHECK (end_date BETWEEN '2020-01-01' AND
'2026-01-01')
);
```

Shopping_cart

Tabela Shopping_cart przechowuje informacje o aktywnościach dodanych przez studentów do koszyka. Kolumna *student_id* identyfikuje studenta, a *activity_id* wskazuje aktywność dodaną do koszyka. Klucz główny składa się z kolumny *student_id* oraz *activity_id*.

```
-- Table: Shopping_cart
CREATE TABLE Shopping_cart (
    student_id int  NOT NULL,
    activity_id int  NOT NULL,
    CONSTRAINT Shopping_cart_pk PRIMARY KEY  (student_id, activity_id)
);

ALTER TABLE Shopping_cart ADD CONSTRAINT Shopping_cart_Activities
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);

ALTER TABLE Shopping_cart ADD CONSTRAINT Shopping_cart_Students
    FOREIGN KEY (student_id)
    REFERENCES Students (student_id);
```

Stationary

Tabela *Stationary* w kolumnie *meeting_id* zawiera id spotkań (zajęć) odbywających się stacjonarnie. W kolumnach start_time i end_time zawiera datę i godzinę rozpoczęcia i zakończenia danego spotkania. W kolumnie classroom_id zawiera id klasy w której odbywa się spotkanie. Kolumna places_occupied zawiera informację o liczbie zajętych miejsc.

Warunki integralnościowe:

- Ograniczenie `CHECK`, które zapewnia, że czas rozpoczęcia spotkania (`start_time`) jest wcześniejszy niż czas zakończenia (`end_time`).
- Ograniczenie `CHECK`, które zapewnia, że data rozpoczęcia spotkania (`start_time`) mieści się w przedziale od 1 stycznia 2020 do 1 stycznia 2026.
- Ograniczenie `CHECK`, które zapewnia, że data zakończenia spotkania (`end_time`) mieści się w przedziale od 1 stycznia 2020 do 1 stycznia 2026.
- Ograniczenie `CHECK`, które zapewnia, że spotkanie nie trwa dłużej niż 3 godziny. Wykorzystujemy funkcję `DATEDIFF` do obliczenia różnicy czasu w godzinach pomiędzy `start_time` i `end_time`.
- Ograniczenie `CHECK`, które zapewnia, że liczba zajętych miejsc (`places_occupied`) mieści się w przedziale od 0 do 120.
- Domyślna wartość `places_occupied` jest ustawiona na 0, jeśli nie zostanie podana wartość przy wstawianiu nowego rekordu.

```
-- Table: Stationary
CREATE TABLE Stationary (
    meeting_id int  NOT NULL,
    start_time datetime  NOT NULL,
    end_time datetime  NOT NULL,
    classroom_id int  NOT NULL,
    places_occupied int  NOT NULL DEFAULT 0,
    CONSTRAINT StationaryStart_timeEnd_time CHECK (start_time < end_time),
    CONSTRAINT chk_start_time_stationary CHECK (start_time BETWEEN
'2020-01-01' AND '2026-01-01'),
    CONSTRAINT chk_end_time_stationary CHECK (end_time BETWEEN
'2020-01-01' AND '2026-01-01'),
    CONSTRAINT chk_duration_stationary CHECK (DATEDIFF(HOUR, start_time,
end_time) <= 3),
    CONSTRAINT chk_places_occupied_stationary CHECK (places_occupied
BETWEEN 0 AND 120),
    CONSTRAINT Stationary_pk PRIMARY KEY  (meeting_id)
);

ALTER TABLE Stationary ADD CONSTRAINT Stationary_Classrooms
    FOREIGN KEY (classroom_id)
```

```
    REFERENCES Classrooms (classroom_id);

ALTER TABLE Stationary ADD CONSTRAINT Stationary_Meetings
    FOREIGN KEY (meeting_id)
    REFERENCES Meetings (meeting_id);
```

Student_activities

Tabela *Student_activities* w kolumnie *student_id* zawiera id wszystkich studenta, a *activity_id* zawiera aktywność do której przypisany jest dany student.

Warunki integralnościowe:

- Ocena (*grade*) jest jedną z wartości (2.0, 3.0, 3.5, 4.0, 4.5, 5.0) lub IS NULL jeśli nie dotyczy ona danego activity_id, bądź nie jest jeszcze wystawiona.

```
-- Table: Student_activities
CREATE TABLE Student_activities (
   student_id int  NOT NULL,
   activity_id int  NOT NULL,
   grade float(2)  NULL,
   CONSTRAINT Student_activities_grade_check CHECK (grade IN (2.0, 3.0,
3.5, 4.0, 4.5, 5.0) or grade IS NULL)

   CONSTRAINT Student_activities_pk PRIMARY KEY  (activity_id,student_id)
);


ALTER TABLE Student_activities ADD CONSTRAINT Student_activities_Events
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);

ALTER TABLE Student_activities ADD CONSTRAINT Student_activities_Students
    FOREIGN KEY (student_id)
    REFERENCES Students (student_id);
```

Students

Tabela *Students* kolumna *student_id* zawiera id wszystkich studentów tj. klientów firmy.

```
-- Table: Students
CREATE TABLE Students (
    student_id int  NOT NULL,
    CONSTRAINT Students_pk PRIMARY KEY  (student_id)
);

ALTER TABLE Students ADD CONSTRAINT Users_Students
    FOREIGN KEY (student_id)
    REFERENCES Users (user_id);
```

Studies

Tabela *Studies* w kolumnie *studies_id* zawiera unikalne id danego kierunku studiów. W kolumnie *activity_id* zawiera id aktywności, którą są studia. W kolumnie *price_for_studies_module* znajduje się cena za jeden zjazd w ramach danego kierunku studiów, a w kolumnie *entry_fee* znajduje się wartość wpisowego. Kolumna *places_ocuppied* zawiera informację ilu studentów zapisało się już na dany zjazd, kolumna *place_limit* informuje jaka jest maksymalna liczba miejsc. Kolumna *semester* jest liczbą oznaczającą, który semestr studiów zawiera tabela.

Warunki integralnościowe:

- *Gwarantuje, że kolumny `places_occupied` i `place_limit` spełniają jeden z dwóch warunków:*
    - *Obie kolumny są puste (`NULL`), co jest dopuszczalne, lub*
    - *Liczba miejsc zajętych (`places_occupied`) nie przekracza limitu miejsc (`place_limit`).*
- Ogranicza wartość kolumny `price_for_studies_module` (cena za zjazd) do przedziału większego niż 0, ale nieprzekraczającego 1000.
- Ogranicza wartość wpisowego (`entry_fee`) do przedziału większego niż 0, ale mniejszego niż cena za zjazd (`price_for_studies_module`).
- Gwarantuje, że wartość `places_occupied` (liczba studentów zapisanych) mieści się w przedziale od 0 do wartości kolumny `place_limit` (maksymalny limit miejsc).
- Ogranicza wartość `place_limit` (limit miejsc) do przedziału większego niż 0, ale mniejszego niż 100.
- Zapewnia, że wartość kolumny `semester` mieści się w przedziale od 1 do 10 (reprezentującym semestry studiów).
- Wartość `places_occupied` domyślnie wynosi 0, jeśli nie zostanie podana przy wstawianiu rekordu.

```sql
-- Table: Studies
CREATE TABLE Studies (
    studies_id int  NOT NULL IDENTITY(1, 1),
    activity_id int  NOT NULL,
    price_for_studies_module money  NOT NULL,
    entry_fee money  NOT NULL,
    places_occupied int  NULL DEFAULT 0,
    place_limit int  NULL,
    semester int  NOT NULL,
    CONSTRAINT StudiesPlaces_occupiedPlace_limit CHECK ((places_occupied
IS NULL AND place_limit IS NULL) OR (places_occupied <= place_limit)),
    CONSTRAINT StudiesSemester CHECK ( 1<= semester AND semester <= 10),
    CONSTRAINT StudiesPrice CHECK (price_for_studies_module > 0 AND
price_for_studies_module <= 1000),
    CONSTRAINT StudiesEntryFee CHECK (entry_fee > 0 AND entry_fee <
price_for_studies_module),
    CONSTRAINT StudiesPlacesOccupied CHECK (places_occupied BETWEEN 0 AND
place_limit),
    CONSTRAINT StudiesPlaceLimit CHECK (place_limit > 0 AND place_limit <
100),
    CONSTRAINT Studies_pk PRIMARY KEY  (studies_id)
);

ALTER TABLE Studies ADD CONSTRAINT Studies_Events
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);

ALTER TABLE Studies ADD CONSTRAINT Studies_Semesters
    FOREIGN KEY (semester)
    REFERENCES Semesters (semester);
```

Studies_module (zjazdy)

Tabela *Studies_modules* w kolumnie *studies_id* zawiera id kierunku studiów, którego dany zjazd jest częścią. Kolumna *module_number* zawiera liczbę oznaczającą, która jest to zjazd w ramach danego kierunku studiów. Kolumny *module_start* i *module_end* zawierają daty rozpoczęcia i zakończenia danego zjazdu. Kolumna *module_type_id* zawiera id które jest powiązane z daną formą spotkań (np. stacjonarne). Kolumna *places_ocuppied* zawiera informację ilu studentów zapisało się już na dany zjazd, kolumna *place_limit* informuje jaka jest maksymalna liczba miejsc.

Warunki integralnościowe:

- Liczba zajętych miejsc (`places_occupied`) nie może być ujemna i musi być mniejsza lub równa wartości maksymalnej liczby miejsc (`place_limit`).
- Każdy zjazd ma przypisany numer (`module_number`), który musi mieścić się w zakresie od 1 do 7, co pozwala na jednoznaczną identyfikację zjazdów w ramach danego kierunku studiów.
- Data rozpoczęcia zjazdu (`module_start`) i data zakończenia (`module_end`) muszą należeć do przedziału od 1 stycznia 2020 roku do 1 stycznia 2026 roku. Ponadto data zakończenia zjazdu musi być późniejsza niż data jego rozpoczęcia.
- Limit miejsc (`place_limit`) musi być większy od 0 i nie może przekraczać 10,000. Zapewnia to sensowne wartości dotyczące pojemności danego zjazdu.
- Kombinacja identyfikatora kierunku studiów (`studies_id`) oraz numeru zjazdu (`module_number`) musi być unikalna. Zapewnia to brak powtarzających się wpisów dla tego samego zjazdu w ramach danego kierunku.
- Domyślną wartością dla liczby zajętych miejsc (`places_occupied`) jest 0, co oznacza, że początkowo żaden student nie jest zapisany na zjazd.

```
-- Table: Studies_module
CREATE TABLE Studies_module (
    studies_id int  NOT NULL,
    module_number int  NOT NULL,
    module_start date  NOT NULL,
    module_end date  NOT NULL,
    module_type_id smallint  NOT NULL,
    places_occupied int  NOT NULL DEFAULT 0,
    place_limit int  NOT NULL,
    CONSTRAINT Studies_modulePlaces_ocupiedPlace_limit CHECK
(places_occupied <= place_limit),
    CONSTRAINT chk_module_number CHECK (module_number BETWEEN 1 AND 7),
    CONSTRAINT chk_module_start CHECK (module_start BETWEEN '2020-01-01'
AND '2026-01-01'),
    CONSTRAINT chk_module_end CHECK (module_end BETWEEN '2020-01-01' AND
'2026-01-01'),
    CONSTRAINT chk_module_dates CHECK (module_end > module_start),
```

```
    CONSTRAINT chk_place_limit CHECK (place_limit > 0 AND place_limit <=
10000),
    CONSTRAINT chk_places_occupied_limit CHECK (places_occupied <=
place_limit AND places_occupied >= 0),
    CONSTRAINT Studies_module_pk PRIMARY KEY  (studies_id,module_number)
);

ALTER TABLE Studies_module ADD CONSTRAINT Table_51_Studies
    FOREIGN KEY (studies_id)
    REFERENCES Studies (studies_id);

ALTER TABLE Studies_module ADD CONSTRAINT Studies_module_Module_type
    FOREIGN KEY (module_type_id)
    REFERENCES Module_type (module_type_id);
```

Study_courses

Tabela *Study_courses* w kolumnie *activity_id* zawiera id aktywności, którą jest kurs na studiach. W kolumnie *tutor_id* zawiera id prowadzącego kurs. W kolumnie *study_course_id* zawiera id nazwy kursu na studiach. W kolumnie *syllabus* zawiera program danego kursu na studiach. Kolumna *studies_id* zawiera id kierunku studiów, którego dany kurs jest częścią. Kolumna *price_for_study_course_meeting* zawiera cenę za jedno spotkanie w ramach danego kursu, dla osoby z zewnątrz.

Warunki integralnościowe:

- Cena za jedno spotkanie w ramach kursu musi być większa niż 0 i mniejsza niż 500. To oznacza, że wartość tej kolumny jest ograniczona do przedziału od 0 do 500 (nie wliczając tych wartości).

```
-- Table: Study_courses
CREATE TABLE Study_courses (
    activity_id int  NOT NULL,
    studies_id int  NOT NULL,
    tutor_id int  NOT NULL,
    study_course_id int  NOT NULL,
    syllabus nvarchar(max)  NOT NULL,
    price_for_study_course_meeting money  NOT NULL,
    CONSTRAINT Study_courses_pk PRIMARY KEY  (activity_id),
    CONSTRAINT Study_courses_price CHECK (price_for_study_course_meeting >
0 AND price_for_study_course_meeting < 500)

);

ALTER TABLE Study_courses ADD CONSTRAINT Study_courses_Activity
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);

ALTER TABLE Study_courses ADD CONSTRAINT Study_courses_Studies
    FOREIGN KEY (studies_id)
    REFERENCES Studies (studies_id);

ALTER TABLE Study_courses ADD CONSTRAINT Study_courses_Study_courses_names
    FOREIGN KEY (study_course_id)
    REFERENCES Study_courses_names (study_course_id);

ALTER TABLE Study_courses ADD CONSTRAINT Study_courses_Tutor
    FOREIGN KEY (tutor_id)
    REFERENCES Tutor (user_id);
```

Study_courses_names

Tabela *Study_course_names* w kolumnie *activity_id* zawiera id aktywności, którą jest webinar, w kolumnie *webinar_name* zawiera nazwę danego webinaru.

Warunki integralnościowe:

- niepowtarzalność wartości w kolumnie *course_names.*

```
-- Table: Study_courses_names
CREATE TABLE Study_courses_names (
    study_course_id int  NOT NULL IDENTITY(1, 1),
    course_name nvarchar(64)  NOT NULL,
    CONSTRAINT Study_courses_namesCourse_name UNIQUE (course_name),
    CONSTRAINT Study_courses_names_pk PRIMARY KEY  (study_course_id)
);
```

Translator

Tabela *Translator* w kolumnie *user_id* zawiera unikalne id użytkownika, który jest tłumaczem. W kolumnie *known_language_id* zawiera id języka (JEDEN TŁUMACZ MA PRZYPISANY JEDEN JĘZYK) z którego tłumaczy na język polski.

```
-- Table: Translator
CREATE TABLE Translator (
    user_id int  NOT NULL,
    known_language_id int  NOT NULL,
    CONSTRAINT Translator_pk PRIMARY KEY  (user_id,known_language_id)
);

ALTER TABLE Translator ADD CONSTRAINT Translator_Languages
    FOREIGN KEY (known_language_id)
    REFERENCES Languages (language_id);

ALTER TABLE Translator ADD CONSTRAINT Coordinator_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id);
```

Tutor

Tabela *Tutor* w kolumnie *user_id* zawiera unikalne id użytkownika, który jest tutorem. W kolumnie *internship_coordinator* zawiera informację w postaci bitu, czy dany tutor jest koordynatorem praktyk(1 - jest, 0 - nie jest).

Warunki integralnościowe:

- Domyślną wartością dla kolumny `internship_coordinator` jest 0, co oznacza, że domyślnie tutor nie jest koordynatorem praktyk, jeśli nie podano innej wartości.

```
-- Table: Tutor
CREATE TABLE Tutor (
    user_id int  NOT NULL,
    internship_coordinator bit  NOT NULL DEFAULT 0,
    CONSTRAINT Tutor_pk PRIMARY KEY  (user_id)
);

ALTER TABLE Tutor ADD CONSTRAINT Trainer_Users
    FOREIGN KEY (user_id)
    REFERENCES Users (user_id);
```

Users

Tabela *Users* w kolumnie *user_id* zawiera unikalne id każdego zarejestrowanego użytkownika bazy danych. W kolumnach *first_name* i *last_name* zawiera odpowiednia imię i nazwisko użytkownika. W kolumnie *email_address* zawiera adres email użytkownika. W kolumnie *password* zawiera hasło użytkownika w formie zakodowanej. W kolejnych kolumnach znajdują dane adresu zamieszkania: w kolumnie *street* - ulica, w kolumnie *zip_code* - kod pocztowy, w kolumnie *city_id* - id danego miasta.

Warunki integralnościowe:

- niepowtarzalność wartości kolumny email_address oraz odpowiednia postać wartości tej kolumny: ‘(minimum jeden znak)@(minimum jeden znak).(minimum jeden znak)’.

```
-- Table: Users
CREATE TABLE Users (
    user_id int  NOT NULL IDENTITY(1, 1),
    first_name nvarchar(32)  NOT NULL,
    last_name nvarchar(32)  NOT NULL,
    email_address nvarchar(320)  NOT NULL,
    password nvarchar(32)  NOT NULL,
    street nvarchar(64)  NOT NULL,
    zip_code nvarchar(8)  NOT NULL,
    city_id int  NOT NULL,
    CONSTRAINT UsersEmail_address UNIQUE (email_address),
    CONSTRAINT UserEmail_address2 CHECK (email_address LIKE '_%@_%._%'),
    CONSTRAINT Users_pk PRIMARY KEY  (user_id)
);

ALTER TABLE Users ADD CONSTRAINT Users_City
    FOREIGN KEY (city_id)
    REFERENCES City (city_id)
```

Webinar

Tabela *Webinar* w kolumnie *activity_id* zawiera id aktywności, którą jest webinar, w kolumnie *webinar_name* zawiera nazwę danego webinaru, w kolumnie *price_for_webinar* zawiera cenę danego webinaru.

Warunki integralnościowe:

- **Domyślna wartość**: Kolumna `price_for_webinar` ma domyślną wartość `0`. Oznacza to, że jeśli użytkownik nie poda ceny, automatycznie zostanie ustawiona na `0`.
- **Zakres ceny**: Cena za webinar, jeśli została określona (czyli jeśli nie jest równa `0`), musi być większa lub równa 0 i mniejsza niż 200.

```
-- Table: Webinar
CREATE TABLE Webinar (
    activity_id int  NOT NULL,
    webinar_name nvarchar(64)  NOT NULL,
    price_for_webinar money  NOT NULL DEFAULT 0,
    CONSTRAINT chk_price_for_webinar CHECK (price_for_webinar >= 0 AND
price_for_webinar < 200),
    CONSTRAINT Webinar_pk PRIMARY KEY  (activity_id)
);

ALTER TABLE Webinar ADD CONSTRAINT Webinar_Events
    FOREIGN KEY (activity_id)
    REFERENCES Activities (activity_id);
```

Widoki

Administration_roles_view

Widok *Administration_roles_view* wyświetla informacje o wszystkich pracownikach administracji. W kolumnach *first_name*, *last_name* i *email_address* znajdują się kolejno imię, nazwisko i adres email pracownika administracji. W kolumnie *role_name* znajduje się jego rola.

```
CREATE VIEW Administration_roles_view AS
SELECT
    u.first_name,
    u.last_name,
    u.email_address,
    r.role_name
FROM Administration_employees ae
JOIN Users u ON ae.user_id = u.user_id
JOIN Roles r ON ae.role_id = r.role_id;
```

Translators_languages_view

Widok *Translators_languages_view* wyświetla informacje o wszystkich tłumaczach. W kolumnach *first_name*, *last_name* i *email_address* znajdują się kolejno imię, nazwisko i adres email tłumaczach. W kolumnie *language* znajduje się nazwa języka z którego może tłumaczyć.

```
CREATE VIEW Translators_languages_view AS
SELECT
    u.first_name,
    u.last_name,
    u.email_address,
    l.language
FROM Translator t
JOIN Users u ON t.user_id = u.user_id
JOIN Languages l ON t.known_language_id = l.language_id;
```

Tutor_studies_date_view

Widok *Tutor_studies_data_view* wyświetla informacje o wszystkich tutorach odpowiedzialnych za przedmioty na studiach. W kolumnach user_id, *first_name*, *last_name* i *email_address* znajdują się kolejno id, imię, nazwisko i adres email tutora. W kolumnie *studies_id* znajduje się id studiów. W kolumnach *studies_name* i *course_name* znajdują się kolejno nazwa kierunku studiów i nazwa przedmiotu. W kolumnie *semester* znajduje się numer semestru na którym tutor prowadzi dane zajęcia.

```
CREATE VIEW Tutor_studies_date_view AS
SELECT DISTINCT
    u.user_id,
    u.first_name,
    u.last_name,
    u.email_address,
    s.studies_id,
    s.studies_name,
    scn.course_name,
    s.semester
FROM Users u
JOIN Tutor t ON u.user_id = t.user_id
JOIN Study_courses sc ON t.user_id = sc.tutor_id
JOIN Studies s ON sc.studies_id = s.studies_id
JOIN Study_courses_names scn ON sc.study_course_id = scn.study_course_id;
```

Tutor_internships_data_view

Widok *Tutor_internships_data_view* wyświetla informacje o wszystkich tutorach odpowiedzialnych za praktyki na studiach. W kolumnach user_id, *first_name*, *last_name* i *email_address* znajdują się kolejno id, imię, nazwisko i adres email tutora. W kolumnie *studies_id* znajduje się id studiów. W kolumnie *studies_name* znajduje się nazwa kierunku studiów. W kolumnie *semester* znajduje się numer semestru na którym tutor odpowiedzialny jest za praktyki

.

```
CREATE VIEW Tutor_internships_data_view AS
SELECT DISTINCT
    u.user_id,
    u.first_name,
    u.last_name,
    u.email_address,
    s.studies_id,
    s.studies_name,
    s.semester
FROM Users u
JOIN Tutor t ON u.user_id = t.user_id
JOIN Internships i ON t.user_id = i.coordinator_id
JOIN Studies s ON i.studies_id = s.studies_id;
```

Cities_countries_view

Widok *Cities_countries_view* w kolumnach *city*, *country* wyświetla wszystkie miasta oraz państwa, w których one leży, zamieszkałe przez przynajmniej jednego użytkownika systemu.

```
CREATE VIEW Cities_countries_view AS
SELECT DISTINCT
    ci.city,
    co.country
FROM Users u
JOIN City ci ON u.city_id = ci.city_id
JOIN Country co ON ci.country_id = co.country_id;
```

Low_attendance_students_view

Widok *Low_attendance_students_view* wyświetla informacje o wszystkich studentach, którzy na ten moment mają niewystarczający procent obecności do zaliczenia swojej aktywności. (poniżej 80%). W kolumnie *student_id* znajdują się id studentów, w kolumnie *activity_id* id aktywności, a w kolumnie *activity_name* nazwy tych aktywności (nazwy kierunków studiów lub nazwy kursów).

```
CREATE VIEW Low_attendance_students_view AS
SELECT
    a.student_id,
    s.studies_name as activity_name,
    s.activity_id,
    SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) * 100.0 / COUNT(*)
as attendance_percentage
FROM Attendance a
JOIN Meetings m ON m.meeting_id = a.meeting_id
JOIN Study_courses sc
ON m.activity_id = sc.activity_id
JOIN Studies s ON s.studies_id = sc.studies_id
GROUP BY a.student_id, s.studies_name, s.activity_id
HAVING SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) * 100.0 /
COUNT(*) < 80


UNION


SELECT
    a.student_id,
    c.course_name as activity_name,
    c.activity_id,
    SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) * 100.0 / COUNT(*)
as attendance_percentage
FROM Attendance a
JOIN Meetings m ON m.meeting_id = a.meeting_id
JOIN Course_modules cm ON m.activity_id = cm.activity_id
JOIN Courses c ON cm.course_id = c.course_id
GROUP BY a.student_id, c.course_name, c.activity_id
HAVING SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) * 100.0 /
COUNT(*) < 80
```

Activities_to_buy_view

Widok `Activities_to_buy_view` wyświetla identyfikatory aktywności (activity_id) możliwych do kupienia oraz ich nazwy (activity_name). Widok obejmuje aktywności, które mają flagę `to_buy` ustawioną na 1 oraz dla których istnieją przyszłe spotkania powiązane z tymi aktywnościami. Widok sprawdza daty rozpoczęcia spotkań w tabelach `Stationary` i `Online_synchronous`, aby uwzględnić tylko te aktywności, których spotkania jeszcze się nie rozpoczęły (start_time ≥ GETDATE()).

```
CREATE VIEW Activities_to_buy_view AS
SELECT
    a.activity_id,
    COALESCE(c.course_name, w.webinar_name, s.studies_name,
scn.course_name) AS activity_name
FROM Activities a
LEFT JOIN Courses c
    ON a.activity_id = c.activity_id
LEFT JOIN Webinar w
    ON a.activity_id = w.activity_id
LEFT JOIN Studies s
    ON a.activity_id = s.activity_id
LEFT JOIN Study_courses sc
    ON a.activity_id = sc.activity_id
LEFT JOIN Study_courses_names scn
    ON scn.study_course_id = sc.study_course_id
LEFT JOIN Meetings m
    ON a.activity_id = m.activity_id
LEFT JOIN Stationary st
    ON m.meeting_id = st.meeting_id
LEFT JOIN Online_synchronous os
    ON m.meeting_id = os.meeting_id
WHERE a.to_buy = 1
  AND (
        (st.start_time >= CAST(GETDATE() AS DATETIME)) OR
        (os.start_time >= CAST(GETDATE() AS DATETIME))
      );
```

Past_Full_Stationary_Meetings_View

Widok **Past_Full_Stationary_Meetings_View** wyświetla identyfikatory spotkań stacjonarnych (`meeting_id`), które odbyły się w przeszłości i w których wszystkie miejsca w sali były zajęte. Widok sprawdza, czy liczba zajętych miejsc (`places_occupied`) była równa limitowi miejsc w sali (`place_limit`), oraz filtruje wydarzenia na podstawie daty ich rozpoczęcia (`start_time <= GETDATE()`).

```
CREATE VIEW Past_Full_stationary_meetings_view AS
SELECT
    s.meeting_id
FROM Stationary s
JOIN Classrooms c ON s.classroom_id = c.classroom_id
WHERE s.places_occupied = c.place_limit
    AND s.start_time <= GETDATE();
```

Future_Full_Stationary_Meetings_View

Widok **Future_Full_Stationary_Meetings_View** wyświetla identyfikatory spotkań stacjonarnych (`meeting_id`), które odbędą się w przyszłości i w których wszystkie miejsca w sali są już zajęte. Widok sprawdza, czy liczba zajętych miejsc (`places_occupied`) jest równa limitowi miejsc w sali (`place_limit`), oraz filtruje wydarzenia na podstawie daty ich rozpoczęcia (`start_time > GETDATE()`).

```
CREATE VIEW Future_Full_stationary_meetings_View AS
SELECT
    s.meeting_id
FROM Stationary s
JOIN Classrooms c ON s.classroom_id = c.classroom_id
WHERE s.places_occupied = c.place_limit
    AND s.start_time > GETDATE();
```

Perfect_Attendance_Students_View

Widok **Perfect_Attendance_Students_View** wyświetla informacje o studentach, którzy mają 100% frekwencję na zajęciach. W kolumnie `student_id` znajduje się identyfikator studenta. Widok grupuje dane według identyfikatora studenta i sprawdza, czy każdy z nich był obecny na wszystkich zajęciach.

```
CREATE VIEW Perfect_Attendance_Students_View AS
SELECT DISTINCT
    s.student_id
FROM Students s
JOIN Attendance a ON s.student_id = a.student_id
GROUP BY s.student_id
HAVING MIN(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) = 1;
```

Ever_absent_students_view

Widok *Ever_absent_students_view* wyświetla informacje o wszystkich studentach którzy byli kiedykolwiek nieobecni na zajęciach. W kolumnach user_id, *first_name*, *last_name* znajdują się kolejno id, imię i nazwisko studenta.

```
CREATE VIEW Ever_absent_students_view AS
SELECT DISTINCT
    s.student_id,
    u.first_name,
    u.last_name
FROM Students s
JOIN Users u ON u.user_id=s.student_id
JOIN Attendance a ON s.student_id = a.student_id
WHERE a.attendance = 0;
```

Unassigned_tutors_view

Widok *Unasigned_tutors_view* wyświetla informacje o wszystkich tutorach którzy nie prowadzą żadnych zajęć i nie są odpowiedzialni za żadne praktyki. W kolumnach user_id, *first_name*, *last_name* znajdują się kolejno id, imię i nazwisko tutora.

```
CREATE VIEW Unassigned_tutors_view AS
SELECT
      t.user_id,
      u.first_name,
      u.last_name
FROM Tutor t
JOIN Users u ON t.user_id = u.user_id
WHERE NOT EXISTS
(
    SELECT 1 FROM Study_courses sc WHERE sc.tutor_id = t.user_id
    UNION
    SELECT 1 FROM Internships i WHERE i.coordinator_id = t.user_id
);
```

Unassigned_translators_view

Widok *Unasigned_tutors_view* wyświetla informacje o wszystkich tłumaczach którzy nie tłumaczą żadnych zajęć. W kolumnach user_id, *first_name*, *last_name* znajdują się kolejno id, imię i nazwisko tlumacza.

```
CREATE VIEW Unassigned_translators_view AS
SELECT DISTINCT
      t.user_id,
      u.first_name,
      u.last_name
FROM Translator t
JOIN Users u ON t.user_id = u.user_id
WHERE NOT EXISTS
(
    SELECT 1 FROM Meetings m WHERE m.translator_id = t.user_id
);
```

Upcoming_webinars_view

Widok *Upcoming_webinars_view* wyświetla informacje o wszystkich webinarach, które dopiero się odbędą. W kolumnach *activity_id* znajduje się id aktywności danego webinaru. W kolumnie *webinar_name* znajduje się nazwa danego webinaru. W kolumnach *start_time* i *end_time* znajdują się kolejno data wraz z godziną rozpoczęcia i zakończenia webinaru.

```
CREATE VIEW Upcoming_webinars_view AS
SELECT
    w.activity_id,
    w.webinar_name,
    os.start_time,
    os.end_time
FROM Webinar w
JOIN Meetings m ON w.activity_id = m.activity_id
JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
WHERE os.start_time > GETDATE();
```

Non_purchasing_students_view

Widok *Non_purchasing_users* wyświetla studentów (*user_id*), którzy jeszcze nic nie kupili.

```
CREATE VIEW Non_purchasing_users_view AS
SELECT u.*
FROM Users u
LEFT JOIN Orders o ON u.user_id = o.student_id
WHERE o.order_id IS NULL;
```

Users_addresses_view

Widok *Student_addresses_view* wyświetla imię (first_name), nazwisko (last_name), email (email_address), adres (street, zip_code, city, country) każdego użytkownika systemu.

```
CREATE VIEW Users_addresses_view AS
SELECT
    u.first_name,
    u.last_name,
    u.email_address,
    u.street,
    u.zip_code,
    c.city,
    co.country
FROM Users u
JOIN City c ON u.city_id = c.city_id
JOIN Country co ON c.country_id = co.country_id;
```

Meeting_details_view

Widok *Meeting_details_view* wypisuje dla każdego *meeting_id* pełne informacje o nim - *meeting_id, translator_id, language_id, activity_id* oraz informacje z poszczególnych tabel w zależności od typu spotkania*:*
- *start_time, end_time, recording_URL_address, platform_URL_address* dla spotkań Online_synchronous,
- *recording_URL_address* dla Online_asynchronous
- *start_time, end_time, classroom_name* dla Stationary

```
CREATE VIEW Meeting_details_view AS
SELECT
    m.*,
    os.start_time AS online_sync_start,
    os.end_time AS online_sync_end,
    os.recording_URL_address AS online_sync_recording,
    p.platform_URL_address,
    oa.recording_URL_address AS async_recording,
    s.start_time AS stationary_start,
    s.end_time AS stationary_end,
    c.classroom_name
FROM Meetings m
LEFT JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
LEFT JOIN Platforms p ON os.platform_id = p.platform_id
LEFT JOIN Online_asynchronous oa ON m.meeting_id = oa.meeting_id
LEFT JOIN Stationary s ON m.meeting_id = s.meeting_id
LEFT JOIN Classrooms c ON s.classroom_id = c.classroom_id;
```

Course_Time_Frames_View

Widok **Course_Time_Frames_View** prezentuje ramy czasowe każdego kursu. W kolumnach `course_id` i `course_name` znajdują się odpowiednio identyfikator i nazwa kursu. W kolumnach `course_start` i `course_end` znajdują się daty rozpoczęcia i zakończenia kursu, wyliczane na podstawie najwcześniejszego i najpóźniejszego terminu spotkań powiązanych z danym kursem.

```
CREATE VIEW Course_Time_Frames_View AS
SELECT
    c.course_id,
    c.course_name,
    MIN(os.start_time) AS course_start,
    MAX(os.end_time) AS course_end
FROM Courses c
JOIN Course_modules cm ON c.course_id = cm.course_id
JOIN Meetings m ON cm.activity_id = m.activity_id
JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
GROUP BY c.course_id, c.course_name;
```

Activity_Revenue_View

Widok **Activity_Revenue_View** zestawia przychody generowane przez różne rodzaje aktywności: webinary, kursy i kierunki studiów. W kolumnie `activity_type` znajduje się typ aktywności (np. Webinar, Course, Studies). W kolumnie `name` wyświetlana jest nazwa aktywności, a w kolumnie `total_revenue` łączna kwota przychodów związanych z daną aktywnością.

```
CREATE VIEW Activity_Revenue_View AS
SELECT
    'Webinar' AS activity_type,
    w.webinar_name AS name,
    COALESCE(SUM(od.price), 0) AS total_revenue
FROM Webinar w
LEFT JOIN Order_details od ON w.activity_id = od.activity_id
GROUP BY w.activity_id, w.webinar_name
UNION ALL
SELECT
    'Course' AS activity_type,
    c.course_name,
    COALESCE(SUM(od.price), 0) AS total_revenue
FROM Courses c
LEFT JOIN Order_details od ON c.activity_id = od.activity_id
GROUP BY c.activity_id, c.course_name
UNION ALL
SELECT
    'Studies' AS activity_type,
    s.studies_name,
    COALESCE(SUM(od.price), 0) AS total_revenue
FROM Studies s
LEFT JOIN Order_details od ON s.activity_id = od.activity_id
GROUP BY s.activity_id, s.studies_name;
```

Unpaid_Activities_View

Widok **Unpaid_Activities_View** wyświetla listę osób, które skorzystały z usług, ale nie uiściły opłat. W kolumnach `user_id`, `first_name` i `last_name` znajdują się odpowiednio identyfikator użytkownika, jego imię i nazwisko. W kolumnach `activity_id` i `price` znajdują się identyfikator aktywności oraz należna kwota. Widok uwzględnia tylko zamówienia bez daty płatności.

```
CREATE VIEW Unpaid_Activities_View AS
SELECT
    u.user_id,
    u.first_name,
    u.last_name,
    od.activity_id,
    od.price
FROM Users u
JOIN Orders o ON u.user_id = o.student_id
JOIN Order_details od ON o.order_id = od.order_id
WHERE od.payment_date IS NULL;
```

Future_Event_Registrations_View

Widok **Future_Event_Registrations_View** przedstawia raport dotyczący liczby zapisanych osób na przyszłe wydarzenia. W kolumnie `event_type` znajduje się typ wydarzenia (Stationary, Online, Asynchronous). W kolumnie `registered_students` widnieje liczba zapisanych uczestników, a w kolumnie `event_start` data rozpoczęcia wydarzenia.

```
CREATE VIEW Future_Event_Registrations_View AS
SELECT
    CASE
        WHEN s.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN os.meeting_id IS NOT NULL THEN 'Online'
        ELSE 'Asynchronous'
    END AS event_type,
    COUNT(DISTINCT sa.student_id) AS registered_students,
    COALESCE(os.start_time, s.start_time) AS event_start
FROM Student_activities sa
JOIN Meetings m ON sa.activity_id = m.activity_id
LEFT JOIN Stationary s ON m.meeting_id = s.meeting_id
LEFT JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
WHERE COALESCE(os.start_time, s.start_time) > GETDATE()
GROUP BY
    CASE
        WHEN s.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN os.meeting_id IS NOT NULL THEN 'Online'
        ELSE 'Asynchronous'
    END,
    COALESCE(os.start_time, s.start_time);
```

Completed_Event_Attendance_View

Widok **Completed_Event_Attendance_View** wyświetla informacje o frekwencji na zakończonych już wydarzeniach. W kolumnie `meeting_id` znajduje się identyfikator spotkania. W kolumnie `total_students` liczba uczestników, którzy byli zapisani na wydarzenie. W kolumnie `attended_students` widnieje liczba obecnych uczestników, a w kolumnie `attendance_percentage` procent obecności.

```
CREATE VIEW Completed_Event_Attendance_View AS
SELECT
    m.meeting_id,
    COUNT(DISTINCT a.student_id) AS total_students,
    SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) AS
attended_students,
    CAST(SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) AS FLOAT) /
    COUNT(DISTINCT a.student_id) * 100 AS attendance_percentage
FROM Meetings m
JOIN Attendance a ON m.meeting_id = a.meeting_id
LEFT JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
LEFT JOIN Stationary s ON m.meeting_id = s.meeting_id
WHERE COALESCE(os.end_time, s.end_time) < GETDATE()
GROUP BY m.meeting_id;
```

Time_Conflicts_View

Widok **Time_Conflicts_View** prezentuje listę studentów zapisanych na przynajmniej dwa przyszłe wydarzenia, które kolidują ze sobą czasowo. W kolumnie `student_id` znajduje się identyfikator studenta. W kolumnach `meeting1_id` i `meeting2_id` znajdują się identyfikatory kolidujących spotkań. W kolumnach `meeting1_start`, `meeting1_end`, `meeting2_start` i `meeting2_end` widnieją czasy rozpoczęcia i zakończenia tych spotkań.

```
CREATE VIEW Time_Conflicts_View AS
SELECT DISTINCT
    sa1.student_id,
    m1.meeting_id AS meeting1_id,
    m2.meeting_id AS meeting2_id,
    os1.start_time AS meeting1_start,
    os1.end_time AS meeting1_end,
    os2.start_time AS meeting2_start,
    os2.end_time AS meeting2_end
FROM Student_activities sa1
JOIN Student_activities sa2 ON sa1.student_id = sa2.student_id
JOIN Meetings m1 ON sa1.activity_id = m1.activity_id
JOIN Meetings m2 ON sa2.activity_id = m2.activity_id
JOIN Online_synchronous os1 ON m1.meeting_id = os1.meeting_id
JOIN Online_synchronous os2 ON m2.meeting_id = os2.meeting_id
WHERE
    m1.meeting_id < m2.meeting_id
    AND os1.start_time < os2.end_time
    AND os2.start_time < os1.end_time
    AND os1.start_time > GETDATE();
```

Procedury

Widoki z parametrem

getStudentsByCountry

Procedura *getStudentsByCountry* z parametrem *CountryName*, którym jest nazwa kraju, wyświetla informacje o wszystkich studentach, którzy pochodzą z kraju *CountryName*.
W kolumnach *student_id*, *first_name* i *last_name* znajdują się kolejno id, imię i nazwisko tych studentów.

```
CREATE PROCEDURE getStudentsByCountry
    @CountryName nvarchar(32)
AS
BEGIN
    SELECT DISTINCT
        s.student_id,
        u.first_name,
        u.last_name
    FROM Students s
    JOIN Users u ON s.student_id = u.user_id
    JOIN City c ON u.city_id = c.city_id
    JOIN Country co ON c.country_id = co.country_id
    WHERE co.country = @CountryName;
END;
```

getTutorSchedule

Procedura *getTutorSchedule* z parametrem *TutorId*, którym jest id tutora, wyświetla plan tutora którego id wynosi *TutorId*.
W kolumnie *meeting_id* znajduje się id spotkania, które prowadzi. W kolumnach *start_time* i *end_time* znajdują się kolejno daty wraz z godziną w których kolejno rozpoczyna i kończy zajęcia.

```
CREATE PROCEDURE getTutorSchedule
    @TutorId int
AS
BEGIN
    SELECT
        m.meeting_id,
        COALESCE(os.start_time, s.start_time) AS start_time,
        COALESCE(os.end_time, s.end_time) AS end_time
    FROM Study_courses sc
    JOIN Meetings m ON sc.activity_id = m.activity_id
    LEFT JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
    LEFT JOIN Stationary s ON m.meeting_id = s.meeting_id
    WHERE sc.tutor_id = @TutorId
    ORDER BY COALESCE(os.start_time, s.start_time);
END;
```

getTranslatorSchedule

Procedura *getTutorSchedule* z parametrem *TranslatorId*, którym jest id tłumacza, wyświetla plan tłumacza którego id wynosi *TranslatorId*.
W kolumnie *meeting_id* znajduje się id spotkania, które tłumaczy. W kolumnach *start_time* i *end_time* znajdują się kolejno daty wraz z godziną w których kolejno rozpoczyna i kończy zajęcia. W kolumnie *language* znajduje się nazwa języka z którego tłumaczy.

```
CREATE PROCEDURE getTranslatorSchedule
    @TranslatorId int
AS
BEGIN
    SELECT
        m.meeting_id,
        COALESCE(os.start_time, s.start_time) AS start_time,
        COALESCE(os.end_time, s.end_time) AS end_time,
        l.language
    FROM Meetings m
    JOIN Translator t ON m.translator_id = t.user_id
    JOIN Languages l ON t.known_language_id = l.language_id
    LEFT JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
    LEFT JOIN Stationary s ON m.meeting_id = s.meeting_id
    WHERE t.user_id = @TranslatorId
    ORDER BY COALESCE(os.start_time, s.start_time);
END;
```

getMeetingsByLanguage

Procedura *getMeetingsByLanguage* z parametrem *LanguageName*, którym jest nazwa języka, wyświetla wszytskie spotkania tłumaczone z języka *LanguageName* na polski.
W kolumnie *meeting_id* znajduje się id spotkania. W kolumnach *start_time* i *end_time* znajdują się kolejno daty wraz z godziną w których kolejno rozpoczyna i kończy się spotkanie.

```
CREATE PROCEDURE getMeetingsByLanguage
    @LanguageName nvarchar(32)
AS
BEGIN
    SELECT
        m.meeting_id,
        COALESCE(os.start_time, s.start_time) AS start_time,
        COALESCE(os.end_time, s.end_time) AS end_time
    FROM Meetings m
    JOIN Translator t ON m.translator_id = t.user_id
    JOIN Languages l ON t.known_language_id = l.language_id
    LEFT JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
    LEFT JOIN Stationary s ON m.meeting_id = s.meeting_id
    WHERE l.language = @LanguageName;
END;
```

getStudentAttendance

Procedura *getStudentAttendance* z parametrem *StudentId* wypisuje dla studenta o zadanym *student_id* liczbę wszystkich spotkań w kolumnie *activity_id*, na które jest zapisany, a w kolumnie *attended_meetings* liczbę spotkań na których ten student był obecny. Dla każdej aktywności wypisuje jej nazwę (*activity_name*).

```
CREATE PROCEDURE getStudentAttendance
    @StudentId int
AS
BEGIN
    SELECT
        m.activity_id, COALESCE(co.course_name, scn.course_name,
w.webinar_name) as activity_name,
        COUNT(*) AS total_meetings,
        SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) AS
attended_meetings
    FROM Meetings m
    JOIN Attendance a ON m.meeting_id = a.meeting_id
    LEFT JOIN Course_modules as cm
    on m.activity_id = cm.activity_id
    LEFT JOIN Study_courses as sc
    on m.activity_id = sc.activity_id
    LEFT JOIN Study_courses_names scn
    on sc.study_course_id = scn.study_course_id
    LEFT JOIN Courses as co
    on cm.course_id = co.course_id
    LEFT JOIN Webinar as w
    on m.activity_id = w.activity_id
    WHERE a.student_id = @StudentId
    GROUP BY m.activity_id, co.course_name, scn.course_name,
w.webinar_name;
END;
```

getStudentActivityAttendance

Procedura *getStudentActivityAttendance* z parametrami *StudentId* oraz *ActivityId* wypisuje dla studenta o zadanym *student_id* liczbę wszystkich spotkań w aktywności o zadanym *activity_id*, a w kolumnie *attended_meetings* liczbę spotkań na których ten student był obecny. Wypisuje także nazwę tej aktywności (*activity_name*).

```
CREATE PROCEDURE getStudentActivityAttendance
    @StudentId int,
    @ActivityId int
AS
BEGIN
    SELECT
        m.activity_id, COALESCE(co.course_name, scn.course_name,
w.webinar_name) as activity_name,
        COUNT(*) AS total_meetings,
        SUM(CASE WHEN a.attendance = 1 THEN 1 ELSE 0 END) AS
attended_meetings
    FROM Meetings m
    JOIN Attendance a ON m.meeting_id = a.meeting_id
    LEFT JOIN Course_modules as cm
    on m.activity_id = cm.activity_id
    LEFT JOIN Study_courses as sc
    on m.activity_id = sc.activity_id
    LEFT JOIN Study_courses_names scn
    on sc.study_course_id = scn.study_course_id
    LEFT JOIN Courses as co
    on cm.course_id = co.course_id
    LEFT JOIN Webinar as w
    on m.activity_id = w.activity_id
    WHERE a.student_id = @StudentId AND m.activity_id = @ActivityId
    GROUP BY m.activity_id, co.course_name, scn.course_name,
w.webinar_name;
END;
```

getStudiesWithCourse

Procedura *getStudiesWithCourse* z parametrem StudyCourseId dla zadanego *study_course_id* (który jest nazwą przedmiotu) wypisuje nazwy kierunków studiów (*studies_name*), ich id (*studies_id*) oraz semestr (*semester*) na którym realizowany jest ten przedmiot (*study_course_id*)

```
CREATE PROCEDURE getStudiesWithCourse
    @StudyCourseId int
AS
BEGIN
    SELECT DISTINCT
        s.studies_id,
        s.studies_name,
        s.semester
    FROM Studies s
    JOIN Study_courses sc ON s.studies_id = sc.studies_id
    WHERE sc.study_course_id = @StudyCourseId;
END;
```

getStudentPurchase

Procedura *getStudentPurchases* z parametrem *StudentId* wyświetla dla zadanego *student_id* wszystkie jego dotychczasowe zakupy. Kolejno: *order_id*, *activity_id*, *description*, *price*, *payment_date*.

```
CREATE PROCEDURE getStudentPurchases
    @StudentId int
AS
BEGIN
    SELECT
        o.order_id,
        od.activity_id,
        od.description,
        od.price,
        od.payment_date
    FROM Orders o
    JOIN Order_details od ON o.order_id = od.order_id
    WHERE o.student_id = @StudentId;
END;
```

getStudentCart

Procedura **GetStudentCart** z parametrem `@StudentId`, którym jest id studenta, wyświetla zawartość koszyka studenta o podanym id.
W kolumnie `activity_id` znajduje się id aktywności (kursu, webinaru lub studiów).
W kolumnie `activity_name` znajduje się nazwa tej aktywności.

```
CREATE PROCEDURE getStudentCart
    @StudentId int
AS
BEGIN
    SELECT
        sc.activity_id,
        COALESCE(c.course_name, w.webinar_name, s.studies_name,
stcn.course_name) as activity_name
    FROM Shopping_cart sc
    LEFT JOIN Courses c ON sc.activity_id = c.activity_id
    LEFT JOIN Webinar w ON sc.activity_id = w.activity_id
    LEFT JOIN Studies s ON sc.activity_id = s.activity_id
    LEFT JOIN Study_courses stc ON sc.activity_id = stc.activity_id
    LEFT JOIN Study_courses_names stcn ON stc.study_course_id =
stcn.study_course_id
    WHERE sc.student_id = @StudentId;
END;
```

getStudiesCourses

Procedura **GetStudiesCourses** z parametrem `@StudiesId`, którym jest id studiów, wyświetla przedmioty realizowane na studiach o podanym id.
W kolumnie `study_course_id` znajduje się id przedmiotu.
W kolumnie `course_name` znajduje się nazwa przedmiotu.
W kolumnie `tutor_name` znajduje się imię i nazwisko prowadzącego przedmiot.

```
CREATE PROCEDURE getStudiesCourses
    @StudiesId int
AS
BEGIN
    SELECT
        sc.study_course_id,
        scn.course_name,
        u.first_name + ' ' + u.last_name AS tutor_name
    FROM Study_courses sc
    JOIN Study_courses_names scn ON sc.study_course_id =
scn.study_course_id
    JOIN Users u ON sc.tutor_id = u.user_id
    WHERE sc.studies_id = @StudiesId;
END;
```

getCourseModules

Procedura **GetCourseModules** z parametrem `@CourseId`, którym jest id kursu, wyświetla moduły związane z kursem o podanym id.
W kolumnie `module_type_id` znajduje się id typu modułu.
W kolumnie `module_type` znajduje się nazwa typu modułu.
W kolumnie `meetings_count` znajduje się liczba spotkań przypisanych do modułu.

```
CREATE PROCEDURE getCourseModules
    @CourseId int
AS
BEGIN
    SELECT
        cm.module_type_id,
        mt.module_type,
        COUNT(m.meeting_id) AS meetings_count
    FROM Course_modules cm
    JOIN Module_type mt ON cm.module_type_id = mt.module_type_id
    LEFT JOIN Meetings m ON cm.activity_id = m.activity_id
    WHERE cm.course_id = @CourseId
    GROUP BY cm.module_type_id, mt.module_type;
END;
```

getStudiesSemesterDates

Procedura **GetStudiesSemesterDates** z parametrem `@StudiesId`, którym jest id studiów, wyświetla daty rozpoczęcia i zakończenia semestru dla studiów o podanym id.
W kolumnie `semester_start` znajduje się data rozpoczęcia semestru.
W kolumnie `semester_end` znajduje się data zakończenia semestru.

```
CREATE PROCEDURE getStudiesSemesterDates
    @StudiesId int
AS
BEGIN
    SELECT
        s.start_date AS semester_start,
        s.end_date AS semester_end
    FROM Studies st
    JOIN Semesters s ON st.semester = s.semester
    WHERE st.studies_id = @StudiesId;
END;
```

getMeetingAttendance

Procedura **GetMeetingAttendance** z parametrem `@MeetingId`, którym jest id spotkania, wyświetla listę obecności na spotkaniu o podanym id.
W kolumnie `meeting_id` znajduje się id spotkania.
W kolumnie `meeting_date` znajduje się data spotkania.
W kolumnach `first_name` i `last_name` znajdują się kolejno imię i nazwisko uczestnika.
W kolumnie `attendance` znajduje się informacja, czy uczestnik był obecny (`1` dla obecnych, `0` dla nieobecnych).

```
CREATE PROCEDURE GetMeetingAttendance
    @MeetingId int
AS
BEGIN
    SELECT
        m.meeting_id,
        COALESCE(os.start_time, s.start_time) AS meeting_date,
        u.first_name,
        u.last_name,
        a.attendance
    FROM Meetings m
    JOIN Attendance a ON m.meeting_id = a.meeting_id
    JOIN Users u ON a.student_id = u.user_id
    LEFT JOIN Online_synchronous os ON m.meeting_id = os.meeting_id
    LEFT JOIN Stationary s ON m.meeting_id = s.meeting_id
    WHERE m.meeting_id = @MeetingId
    ORDER BY u.last_name, u.first_name;
END;
```

Zarządzanie użytkownikami

changeUserPassword

Procedura *changeUserPassword* zmienia hasło danego użytkownika *@user_id* na *@new_password.*

```
CREATE PROCEDURE changeUserPassword
    @user_id INT,
    @new_password NVARCHAR(32)
AS
BEGIN
    UPDATE Users
    SET password = @new_password
    WHERE user_id = @user_id
END;
```

Zarządzanie koszykiem

addToShoppingCart

Procedura *addToShoppingCart* dodaje do koszyka należącego do *@student_id* aktywność o zadanym *@activity_id*.

```
CREATE PROCEDURE addToShoppingCart
    @student_id INT,
    @activity_id INT
AS
BEGIN
    INSERT INTO Shopping_cart (student_id, activity_id)
    VALUES (@student_id, @activity_id)
END;
```

removeFromShoppingCart

Procedura *removeFromShoppingCart* usuwa z koszyka należącego do *@student_id* aktywność o zadanym *@activity_id*.

```
CREATE PROCEDURE removeFromShoppingCart
    @student_id INT,
    @activity_id INT
AS
BEGIN
    DELETE FROM Shopping_cart
    WHERE student_id = @student_id AND activity_id = @activity_id
END;
```

createOrderFromCart

Procedura *createOrderFromCart* przenosi dane z koszyka studenta o zadanym @student_id na wpisy z aktualną datą do tabel *Orders* oraz *Order_details* oraz usuwa te dane z koszyka.

```
CREATE PROCEDURE createOrderFromCart
    @student_id INT
AS
BEGIN
    DECLARE @order_id INT
    DECLARE @current_date DATETIME = GETDATE()

    -- Create new order
    INSERT INTO Orders (student_id, order_date)
    VALUES (@student_id, @current_date)

    SET @order_id = SCOPE_IDENTITY()

    -- Move items from cart to order details
    INSERT INTO Order_details (order_id, activity_id, description, price)
    SELECT
        @order_id,
        sc.activity_id,
        COALESCE(c.course_name, w.webinar_name, s.studies_name,
'Activity'),
        COALESCE(c.price_for_course, w.price_for_webinar,
s.price_for_studies_module, 0)
    FROM Shopping_cart sc
    LEFT JOIN Courses c ON sc.activity_id = c.activity_id
    LEFT JOIN Webinar w ON sc.activity_id = w.activity_id
    LEFT JOIN Studies s ON sc.activity_id = s.activity_id
    WHERE sc.student_id = @student_id

    -- Clear the cart
    DELETE FROM Shopping_cart
    WHERE student_id = @student_id
END;
```

Zarządzanie ocenami

updateGrade

Procedura *updateGrade* dla danego *@student_id* oraz *@activity_id* modyfikuje aktualną ocenę na daną ocenę (*@grade*).

```
CREATE PROCEDURE updateGrade
    @student_id INT,
    @activity_id INT,
    @grade FLOAT
AS
BEGIN
    UPDATE Student_activities
    SET grade = @grade
    WHERE student_id = @student_id AND activity_id = @activity_id
END;
```

removeGrade

Procedura *removeGrade* dla danego *@student_id* oraz *@activity_id* usuwa aktualną ocenę (*@grade*).

```
CREATE PROCEDURE removeGrade
    @student_id INT,
    @activity_id INT
AS
BEGIN
    UPDATE Student_activities
    SET grade = NULL
    WHERE student_id = @student_id AND activity_id = @activity_id
END;
```

Zarządzanie obecnością

addAttendance

Procedura *addAttendance* przypisuje dla danego spotkania (*@meeting_id*) studenta (*@student_id*) i zaznacza czy był on obecny (bit *@attendance*).

```
CREATE PROCEDURE addAttendance
    @meeting_id INT,
    @student_id INT,
    @attendance BIT
AS
BEGIN
    INSERT INTO Attendance (meeting_id, student_id, attendance)
    VALUES (@meeting_id, @student_id, @attendance)
END;
```

updateAttendance

Procedura updateAttendance aktualizuje dla danego studenta (*@student_id*) na danym spotkaniu (*@meeting_id*) jego obecność (bit *@attendance*) na zadaną.

```
CREATE PROCEDURE updateAttendance
    @meeting_id INT,
    @student_id INT,
    @attendance BIT
AS
BEGIN
    UPDATE Attendance
    SET attendance = @attendance
    WHERE meeting_id = @meeting_id AND student_id = @student_id
END;
```

removeAttendance

Procedura *removeAttendance* usuwa wpis danego studenta (*@student_id*) oraz jego obenność (bit *@attendance*) dla danego spotkania (*@meeting_id*)

```
CREATE PROCEDURE removeAttendance
    @meeting_id INT,
    @student_id INT
AS
BEGIN
    DELETE FROM Attendance
    WHERE meeting_id = @meeting_id AND student_id = @student_id
END;
```

Zarządzanie kursami

AddCourseWithModules

Procedura *addCourse* dodaje do bazy kurs z zadanymi parametrami, jednocześnie dodając wpis do tabeli *Activities* z danym dla tego kursu *@activity_id*.

```
CREATE PROCEDURE AddCourseWithModules
    @course_activity_id INT,
    @course_name NVARCHAR(64),
    @price_for_course MONEY,
    @place_limit INT,
    @entry_fee INT,
    @module1_type_id SMALLINT,
    @module1_activity_id INT,
    @module2_type_id SMALLINT,
    @module2_activity_id INT,
    @module3_type_id SMALLINT,
    @module3_activity_id INT,
    @module4_type_id SMALLINT,
    @module4_activity_id INT
AS
BEGIN

        BEGIN TRANSACTION
            DECLARE @course_id INT

            -- Create main activity for the course
            INSERT INTO Activities (activity_id, to_buy)
            VALUES (@course_activity_id, 1)

            -- Create activities for additional modules
            INSERT INTO Activities (activity_id, to_buy)
            VALUES (@module1_activity_id, 0)

            INSERT INTO Activities (activity_id, to_buy)
            VALUES (@module2_activity_id, 0)

            INSERT INTO Activities (activity_id, to_buy)
            VALUES (@module3_activity_id, 0)

            INSERT INTO Activities (activity_id, to_buy)
            VALUES (@module4_activity_id, 0)
```

```
-- Create the course
INSERT INTO Courses (
    activity_id,
    course_name,
    price_for_course,
    place_limit,
    entry_fee,
    places_occupied
)
VALUES (
    @course_activity_id,
    @course_name,
    @price_for_course,
    @place_limit,
    @entry_fee,
    0
)
SET @course_id = SCOPE_IDENTITY()

-- Create course modules
-- Module 1
INSERT INTO Course_modules (
    activity_id,
    course_id,
    module_type_id
)
VALUES (
    @module1_activity_id,
    @course_id,
    @module1_type_id
)

-- Module 2
INSERT INTO Course_modules (
    activity_id,
    course_id,
    module_type_id
)
VALUES (
    @module2_activity_id,
    @course_id,
    @module2_type_id
)
```

```
        -- Module 3
        INSERT INTO Course_modules (
            activity_id,
            course_id,
            module_type_id
        )
        VALUES (
            @module3_activity_id,
            @course_id,
            @module3_type_id
        )

        -- Module 4
        INSERT INTO Course_modules (
            activity_id,
            course_id,
            module_type_id
        )
        VALUES (
            @module4_activity_id,
            @course_id,
            @module4_type_id
        )
END;
```

updateCourse

Procedura *updateCourse* dla kursu o danym *@course_id* aktualizuje dane na te przekazane w parametrach.

```
CREATE PROCEDURE updateCourse
    @course_id INT,
    @course_name NVARCHAR(64),
    @price_for_course MONEY,
    @place_limit INT,
    @entry_fee INT
AS
BEGIN
    UPDATE Courses
    SET course_name = @course_name,
        price_for_course = @price_for_course,
        place_limit = @place_limit,
        entry_fee = @entry_fee
    WHERE course_id = @course_id
END;
```

deleteCourse

Procedura *deleteCourse* usuwa kurs o zadanym *@course_id* z tabeli *Courses* oraz pobiera jego *activity_id* i usuwa je z tabeli *Activities*.

```
CREATE PROCEDURE deleteCourse
    @course_id INT
AS
BEGIN
    DECLARE @activity_id INT

    SELECT @activity_id = activity_id FROM Courses WHERE course_id =
@course_id

    DELETE FROM Course_modules WHERE course_id = @course_id
    DELETE FROM Courses WHERE course_id = @course_id
    DELETE FROM Activities WHERE activity_id = @activity_id
END;
```

Zarządzanie webinarami

addWebinar

Procedura *addWebinar* dodaje webinar o zadanych parametrach.

```
CREATE PROCEDURE addWebinar
    @webinar_name NVARCHAR(64),
    @price_for_webinar MONEY,
    @activity_id INT
AS
BEGIN

    INSERT INTO Activities (activity_id, to_buy)
    VALUES (@activity_id, 1)


    INSERT INTO Webinar (activity_id, webinar_name, price_for_webinar)
    VALUES (@activity_id, @webinar_name, @price_for_webinar)
END;
```

updateWebinar

Procedura *updateWebinar* dla webiranu o danym *@activity_id* aktualizuje jego dane na te zadane.

```
CREATE PROCEDURE updateWebinar
    @activity_id INT,
    @webinar_name NVARCHAR(64),
    @price_for_webinar MONEY
AS
BEGIN
    UPDATE Webinar
    SET webinar_name = @webinar_name,
        price_for_webinar = @price_for_webinar
    WHERE activity_id = @activity_id
END;
```

deleteWebinar

Procedura *deleteWebinar* usuwa z bazy webinar o zadanym *@activity_id*.

```
CREATE PROCEDURE deleteWebinar
    @activity_id INT
AS
BEGIN
    DELETE FROM Webinar WHERE activity_id = @activity_id
    DELETE FROM Activities WHERE activity_id = @activity_id
END;
```

Zarządzanie studiami

addStudies

Procedura *addStudies* dodaje do tabeli *Studies* studia o zadanych parametrach oraz do tabeli *Activities* ich *@activity_id*.

```
CREATE PROCEDURE addStudies
    @studies_name NVARCHAR(64),
    @price_for_module MONEY,
    @entry_fee MONEY,
    @place_limit INT,
    @semester INT,
    @activity_id INT
AS
BEGIN
    DECLARE @studies_id INT

    INSERT INTO Activities (activity_id, to_buy)
    VALUES (@activity_id, 1)


    INSERT INTO Studies (
        activity_id, studies_name, price_for_studies_module,
        entry_fee, place_limit, places_occupied, semester
    )
    VALUES (
        @activity_id, @studies_name, @price_for_module,
        @entry_fee, @place_limit, 0, @semester
    )
END;
```

updateStudies

Procedura *updateStudies* dla studiów o danym *@studies_id* aktualizuje dane na te zadane.

```
CREATE PROCEDURE updateStudies
    @studies_id INT,
    @studies_name NVARCHAR(64),
    @price_for_module MONEY,
    @entry_fee MONEY,
    @place_limit INT
AS
BEGIN
    UPDATE Studies
    SET studies_name = @studies_name,
        price_for_studies_module = @price_for_module,
        entry_fee = @entry_fee,
        place_limit = @place_limit
    WHERE studies_id = @studies_id
END;
```

deleteStudies

Procedura deleteStudies usuwa studia o zadanym *@study_id* z tabeli *Studies* oraz pobiera jego *activity_id* i usuwa je z tabeli *Activities*.

```
CREATE PROCEDURE deleteStudies
    @studies_id INT
AS
BEGIN
    DECLARE @activity_id INT

    SELECT @activity_id = activity_id FROM Studies WHERE studies_id =
@studies_id

    DELETE FROM Study_courses WHERE studies_id = @studies_id
    DELETE FROM Studies WHERE studies_id = @studies_id
    DELETE FROM Activities WHERE activity_id = @activity_id
END;
```

Procedury zarządzania spotkaniami

AddStationaryMeeting

Procedura `AddStationaryMeeting` z parametrami `@activity_id`, `@translator_id` (opcjonalnie), `@language_id` (opcjonalnie), `@start_time`, `@end_time`, `@classroom_id` i `@places_occupied` (bazowo 0), dodaje nowe spotkanie stacjonarne do tabel Meetings i Stationary. Zapisuje dane o przypisanej aktywności, tłumaczu, języku, czasie rozpoczęci i zakończenia spotkania, id sali, liczbie zajętych miejsc.

```
CREATE PROCEDURE AddStationaryMeeting
    @activity_id INT,
    @translator_id INT = NULL,
    @language_id INT = NULL,
    @start_time DATETIME,
    @end_time DATETIME,
    @classroom_id INT,
    @places_occupied INT = 0
AS
BEGIN

    DECLARE @meeting_id INT

    INSERT INTO Meetings (activity_id, translator_id, language_id)
    VALUES (@activity_id, @translator_id, @language_id)

    SET @meeting_id = SCOPE_IDENTITY()

    INSERT INTO Stationary (
        meeting_id,
        start_time,
        end_time,
        classroom_id,
        places_occupied
    )
    VALUES (
        @meeting_id,
        @start_time,
        @end_time,
        @classroom_id,
        @places_occupied
    )

    SELECT @meeting_id AS MeetingId
END;
```

UpdateStationaryMeeting

Procedura `UpdateStationaryMeeting` z parametrami `@meeting_id`, `@translator_id` (opcjonalnie), `@language_id` (opcjonalnie), `@start_time`, `@end_time`, `@classroom_id` i `@places_occupied` (bazowo 0), modyfikuje dane spotkania stacjonarnego o danym `@meeting_id` w tabelach Meetings i Stationary. Zapisuje dane o tłumaczu, języku, czasie rozpoczęci i zakończenia spotkania, id sali, liczbie zajętych miejsc.

```
CREATE PROCEDURE UpdateStationaryMeeting
    @meeting_id INT,
    @translator_id INT = NULL,
    @language_id INT = NULL,
    @start_time DATETIME,
    @end_time DATETIME,
    @classroom_id INT,
    @places_occupied INT
AS
BEGIN
            UPDATE Meetings
            SET translator_id = @translator_id,
                language_id = @language_id
            WHERE meeting_id = @meeting_id

            UPDATE Stationary
            SET start_time = @start_time,
                end_time = @end_time,
                classroom_id = @classroom_id,
                places_occupied = @places_occupied
            WHERE meeting_id = @meeting_id
END;
```

AddOnlineSynchronousMeeting

Procedura `AddOnlineSynchronousMeeting` z parametrami `@activity_id`, `@translator_id` (opcjonalnie), `@language_id` (opcjonalnie), `@start_time`, `@end_time`, `@recording_URL_address` i `@platform_id`, dodaje nowe spotkanie online synchroniczne do tabel Meetings i Online_synchronous. Zapisuje dane o przypisanej aktywności, tłumaczu, języku, czasie rozpoczęci i zakończenia spotkania, adresie nagrania, adresie platformy.

```
CREATE PROCEDURE AddOnlineSynchronousMeeting
    @activity_id INT,
    @translator_id INT = NULL,
    @language_id INT = NULL,
    @start_time DATETIME,
    @end_time DATETIME,
    @recording_URL_address NVARCHAR(64),
    @platform_id INT
AS
BEGIN
    DECLARE @meeting_id INT

    INSERT INTO Meetings (activity_id, translator_id, language_id)
    VALUES (@activity_id, @translator_id, @language_id)

    SET @meeting_id = SCOPE_IDENTITY()

    INSERT INTO Online_synchronous (
        meeting_id,
        start_time,
        end_time,
        recording_URL_address,
        platform_id
    )
    VALUES (
        @meeting_id,
        @start_time,
        @end_time,
        @recording_URL_address,
        @platform_id
    )

    SELECT @meeting_id AS MeetingId
END;
```

UpdateOnlineSynchronousMeeting

Procedura `UpdateOnlineSynchronousMeeting` z parametrami `@meeting_id`, `@translator_id` (opcjonalnie), `@language_id` (opcjonalnie), `@start_time`, `@end_time`, `@recording_URL_address` i `@platform_id`, modyfikuje dane spotkania online synchronicznego o danym `@meeting_id` w tabelach Meetings i Online_synchronous. Zapisuje dane o tłumaczu, języku, czasie rozpoczęci i zakończenia spotkania, adresie nagrania, adresie platformy.

```
CREATE PROCEDURE UpdateOnlineSynchronousMeeting
    @meeting_id INT,
    @translator_id INT = NULL,
    @language_id INT = NULL,
    @start_time DATETIME,
    @end_time DATETIME,
    @recording_URL_address NVARCHAR(64),
    @platform_id INT
AS
BEGIN
            UPDATE Meetings
            SET translator_id = @translator_id,
                language_id = @language_id
            WHERE meeting_id = @meeting_id

            UPDATE Online_synchronous
            SET start_time = @start_time,
                end_time = @end_time,
                recording_URL_address = @recording_URL_address,
                platform_id = @platform_id
            WHERE meeting_id = @meeting_id
END;
```

AddOnlineAsynchronousMeeting

Procedura `AddOnlineAsynchronousMeeting` z parametrami `@activity_id`, `@translator_id` (opcjonalnie), `@language_id` (opcjonalnie), `@recording_URL_address`, dodaje nowe spotkanie online asynchroniczne do tabel Meetings i Online_asynchronous. Zapisuje dane o przypisanej aktywności, tłumaczu, języku i adresie nagrania.

```
CREATE PROCEDURE AddOnlineAsynchronousMeeting
    @activity_id INT,
    @translator_id INT = NULL,
    @language_id INT = NULL,
    @recording_URL_address NVARCHAR(64)
```

```
AS
BEGIN
            DECLARE @meeting_id INT

            INSERT INTO Meetings (activity_id, translator_id, language_id)
            VALUES (@activity_id, @translator_id, @language_id)

            SET @meeting_id = SCOPE_IDENTITY()

            INSERT INTO Online_asynchronous (
                meeting_id,
                recording_URL_address
            )
            VALUES (
                @meeting_id,
                @recording_URL_address
            )

            SELECT @meeting_id AS MeetingId
END;
```

UpdateOnlineAsynchronousMeeting

Procedura `AddOnlineAsynchronousMeeting` z parametrami `@activity_id`, `@translator_id` (opcjonalnie), `@language_id` (opcjonalnie), `@recording_URL_address`, modyfikuje dane o spotkaniu online asynchronicznym o danym `@meeting_id` w tabelach Meetings i Online_asynchronous. Zapisuje dane o tłumaczu, języku i adresie nagrania.

```
CREATE PROCEDURE UpdateOnlineAsynchronousMeeting
    @meeting_id INT,
    @translator_id INT = NULL,
    @language_id INT = NULL,
    @recording_URL_address NVARCHAR(64)
AS
BEGIN
            UPDATE Meetings
            SET translator_id = @translator_id,
                language_id = @language_id
            WHERE meeting_id = @meeting_id

            UPDATE Online_asynchronous
            SET recording_URL_address = @recording_URL_address
            WHERE meeting_id = @meeting_id
```

```
END;
```

DeleteMeeting

Procedura `DeleteMeeting` z parametrem `@meeting_id` usuwa spotkanie o podanym id z tabeli Meetings i odpowiadający jemu rekord w jednej z tabel Stationary, Online_synchronous lub online_asynchronous

```
CREATE PROCEDURE DeleteMeeting
    @meeting_id INT
AS
BEGIN
    DELETE FROM Meetings WHERE meeting_id = @meeting_id
    DELETE FROM Stationary WHERE meeting_id = @meeting_id
    DELETE FROM Online_synchronous WHERE meeting_id = @meeting_id
    DELETE FROM Online_asynchronous WHERE meeting_id = @meeting_id
END;
```

Procedury zarządzania studentami

addStudent

Procedura `AddStudent` z parametrami dotyczącymi danych osobowych i adresowych dodaje nowego użytkownika do tabeli `Users`, a następnie przypisuje go jako studenta w tabeli `Students`.

```
CREATE PROCEDURE AddStudent
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @password NVARCHAR(32),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT
AS
BEGIN
    DECLARE @user_id INT
    INSERT INTO Users (
        first_name, last_name, email_address,
        password, street, zip_code, city_id
    )
```

```
    VALUES (
        @first_name, @last_name, @email_address,
        @password, @street, @zip_code, @city_id
    )
    SET @user_id = SCOPE_IDENTITY()
    INSERT INTO Students (student_id)
    VALUES (@user_id)
END;
```

updateStudent

Procedura `UpdateStudent` z parametrami dotyczącymi danych osobowych i adresowych aktualizuje dane studenta w tabeli `Users` na podstawie jego id.

```
CREATE PROCEDURE UpdateStudent
    @student_id INT,
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT
AS
BEGIN
    UPDATE Users
    SET first_name = @first_name,
        last_name = @last_name,
        email_address = @email_address,
        street = @street,
        zip_code = @zip_code,
        city_id = @city_id
    WHERE user_id = @student_id
END;
```

deleteStudent

Procedura `DeleteStudent` z parametrem `@student_id` usuwa studenta z tabeli `Students`, a także odpowiadającego mu użytkownika z tabeli `Users`.

```
CREATE PROCEDURE DeleteStudent
    @student_id INT
AS
BEGIN
```

```
    DELETE FROM Students WHERE student_id = @student_id
    DELETE FROM Users WHERE user_id = @student_id
END;
```

Procedury zarządzania korepetytorami

addTutor

Procedura `AddTutor` dodaje nowego użytkownika do tabeli `Users`, a następnie rejestruje go jako tutora w tabeli `Tutor`. Uwzględnia informację, czy pełni funkcję koordynatora praktyk.

```
CREATE PROCEDURE AddTutor
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @password NVARCHAR(32),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT,
    @internship_coordinator BIT
AS
BEGIN
    DECLARE @user_id INT

    INSERT INTO Users (
        first_name, last_name, email_address,
        password, street, zip_code, city_id
    )
    VALUES (
        @first_name, @last_name, @email_address,
        @password, @street, @zip_code, @city_id
    )

    SET @user_id = SCOPE_IDENTITY()

    INSERT INTO Tutor (user_id, internship_coordinator)
    VALUES (@user_id, @internship_coordinator)
END;
```

updateTutor

Procedura `UpdateTutor` aktualizuje dane tutora w tabeli `Users` oraz jego rolę w tabeli `Tutor` na podstawie id użytkownika.

```
CREATE PROCEDURE UpdateTutor
    @tutor_id INT,
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT,
    @internship_coordinator BIT
AS
BEGIN
    UPDATE Users
    SET first_name = @first_name,
        last_name = @last_name,
        email_address = @email_address,
        street = @street,
        zip_code = @zip_code,
        city_id = @city_id
    WHERE user_id = @tutor_id

    UPDATE Tutor
    SET internship_coordinator = @internship_coordinator
    WHERE user_id = @tutor_id
END;
```

deleteTutor

Procedura `DeleteTutor` usuwa tutora z tabeli `Tutor` oraz odpowiadającego mu użytkownika z tabeli `Users`.

```
CREATE PROCEDURE DeleteTutor
    @tutor_id INT
AS
BEGIN
    DELETE FROM Tutor WHERE user_id = @tutor_id
```

```
    DELETE FROM Users WHERE user_id = @tutor_id
END;
```

Procedury zarządzania tłumaczami

addTranslator

Procedura `AddTranslator` dodaje nowego użytkownika do tabeli `Users`, a następnie przypisuje go jako tłumacza w tabeli `Translator`, uwzględniając id języka, który zna.

```
CREATE PROCEDURE AddTranslator
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @password NVARCHAR(32),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT,
    @known_language_id INT
AS
BEGIN
    DECLARE @user_id INT

    INSERT INTO Users (
        first_name, last_name, email_address,
        password, street, zip_code, city_id
    )
    VALUES (
        @first_name, @last_name, @email_address,
        @password, @street, @zip_code, @city_id
    )

    SET @user_id = SCOPE_IDENTITY()

    INSERT INTO Translator (user_id, known_language_id)
    VALUES (@user_id, @known_language_id)
END;
```

updateTranslator

Procedura `UpdateTranslator` aktualizuje dane osobowe i adresowe tłumacza w tabeli `Users` na podstawie jego id.

```
CREATE PROCEDURE UpdateTranslator
    @translator_id INT,
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT
AS
BEGIN
    UPDATE Users
    SET first_name = @first_name,
        last_name = @last_name,
        email_address = @email_address,
        street = @street,
        zip_code = @zip_code,
        city_id = @city_id
    WHERE user_id = @translator_id
END;
```

deleteTranslator

Procedura `DeleteTranslator` usuwa tłumacza z tabeli `Translator` oraz odpowiadającego mu użytkownika z tabeli `Users`.

```
CREATE PROCEDURE DeleteTranslator
    @translator_id INT
AS
BEGIN
    DELETE FROM Translator WHERE user_id = @translator_id
    DELETE FROM Users WHERE user_id = @translator_id
END;
```

Procedury zarządzania pracownikami administracyjnymi

addAdminEmployee

Procedura `AddAdminEmployee` dodaje nowego użytkownika do tabeli `Users`, a następnie rejestruje go jako pracownika administracji w tabeli `Administration_employees`, przypisując mu odpowiednią rolę.

```
CREATE PROCEDURE AddAdminEmployee
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @password NVARCHAR(32),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT,
    @role_id INT
AS
BEGIN
    DECLARE @user_id INT

    INSERT INTO Users (
        first_name, last_name, email_address,
        password, street, zip_code, city_id
    )
    VALUES (
        @first_name, @last_name, @email_address,
        @password, @street, @zip_code, @city_id
)

    SET @user_id = SCOPE_IDENTITY()

    INSERT INTO Administration_employees (user_id, role_id)
    VALUES (@user_id, @role_id)
END;
```

updateAdminEmployee

Procedura `UpdateAdminEmployee` aktualizuje dane osobowe i adresowe pracownika administracji w tabeli `Users` oraz jego rolę w tabeli `Administration_employees`.

```
CREATE PROCEDURE UpdateAdminEmployee
    @admin_id INT,
    @first_name NVARCHAR(32),
    @last_name NVARCHAR(32),
    @email_address NVARCHAR(320),
    @street NVARCHAR(64),
    @zip_code NVARCHAR(8),
    @city_id INT,
    @role_id INT
AS
BEGIN
    UPDATE Users
    SET first_name = @first_name,
        last_name = @last_name,
        email_address = @email_address,
        street = @street,
        zip_code = @zip_code,
        city_id = @city_id
    WHERE user_id = @admin_id

    UPDATE Administration_employees
    SET role_id = @role_id
    WHERE user_id = @admin_id
END;
```

deleteAdminEmployee

Procedura `DeleteAdminEmployee` usuwa pracownika administracji z tabeli `Administration_employees` oraz odpowiadającego mu użytkownika z tabeli `Users`.

```
CREATE PROCEDURE DeleteAdminEmployee
    @admin_id INT
AS
BEGIN
    DELETE FROM Administration_employees WHERE user_id = @admin_id
    DELETE FROM Users WHERE user_id = @admin_id
END;
```

Procedury przeglądania ocen

viewStudentGrades

Procedura `ViewStudentGrades` z parametrem `@student_id` wyświetla oceny studenta dla wszystkich przypisanych mu aktywności. W wynikach znajdują się nazwa aktywności, id aktywności oraz ocena.

```
CREATE PROCEDURE ViewStudentGrades
    @student_id INT
AS
BEGIN
    SELECT
        sa.activity_id,
        COALESCE(c.course_name, w.webinar_name, s.studies_name,
'Activity') as activity_name,
        sa.grade
    FROM Student_activities sa
    LEFT JOIN Courses c ON sa.activity_id = c.activity_id
    LEFT JOIN Webinar w ON sa.activity_id = w.activity_id
    LEFT JOIN Studies s ON sa.activity_id = s.activity_id
    WHERE sa.student_id = @student_id
    AND sa.grade IS NOT NULL
    ORDER BY sa.grade DESC
END;
```

getStudentGradeAverage

Procedura `GetStudentGradeAverage` z parametrem `@student_id` oblicza średnią ocen studenta, liczbę ocen oraz zwraca jego id.

```
CREATE PROCEDURE GetStudentGradeAverage
    @student_id INT
AS
BEGIN
    SELECT
        @student_id as student_id,
        AVG(grade) as average_grade,
        COUNT(grade) as total_grades
    FROM Student_activities
    WHERE student_id = @student_id
    AND grade IS NOT NULL
END;
```

Zarządzanie szczegółami zamówienia

updateOrderDetails

Procedura `UpdateOrderDetails` aktualizuje szczegóły zamówienia w tabeli `Order_details` na podstawie id zamówienia i aktywności. Umożliwia zmianę opisu, ceny i daty płatności.

```
CREATE PROCEDURE UpdateOrderDetails
    @order_id INT,
    @activity_id INT,
    @description NVARCHAR(64),
    @price MONEY,
    @payment_date DATETIME = NULL
AS
BEGIN
    UPDATE Order_details
    SET description = @description,
        price = @price,
        payment_date = @payment_date
    WHERE order_id = @order_id
    AND activity_id = @activity_id
END;
```

Zadanie z aktywności studenckiej

assignStudentActivity

Procedura `AssignStudentActivity` przypisuje studenta do aktywności, dodając odpowiedni wpis do tabeli `Student_activities`.

```
CREATE PROCEDURE AssignStudentActivity
    @student_id INT,
    @activity_id INT
AS
BEGIN
    INSERT INTO Student_activities (student_id, activity_id)
    VALUES (@student_id, @activity_id)
END;
```

Funkcje

getStudentTotalCosts

Funkcja *getStudentTotalCost* z argumentem *StudentId*, którym jest id studenta, zwraca sumę wszystkich płatności poniesionych przez niego.

```
CREATE FUNCTION getStudentTotalCosts
(
    @StudentId int
)
RETURNS money
AS
BEGIN
    DECLARE @TotalCost money;
    SELECT @TotalCost = SUM(od.price)
    FROM Orders o
    JOIN Order_details od ON o.order_id = od.order_id
    WHERE o.student_id = @StudentId;
    RETURN ISNULL(@TotalCost, 0);
END;
```

getActivityType

Funkcja *getActivityType* z argumentem *ActivityId*, którym jest id aktywności, zwraca czym jest dana aktywność (typ aktywności). *Activity_id* może należeć do kursu (‘Course’), modułu kursu (‘Course module’), webinaru (‘Webinar’), studiów (‘Studies’), przedmiotu na studiach (*Study course*). Zwraca ‘No data’ gdy *activity_id* nie przypisano typu aktywności.

```
CREATE FUNCTION getActivityType
(
    @ActivityId int
)
RETURNS nvarchar(32)
AS
BEGIN
    DECLARE @ActivityType nvarchar(32);

    IF EXISTS (SELECT 1 FROM Courses WHERE activity_id = @ActivityId)
        SET @ActivityType = 'Course';
    ELSE IF EXISTS (SELECT 1 FROM Webinar WHERE activity_id = @ActivityId)
        SET @ActivityType = 'Webinar';
    ELSE IF EXISTS (SELECT 1 FROM Studies WHERE activity_id = @ActivityId)
        SET @ActivityType = 'Studies';
    ELSE IF EXISTS (SELECT 1 FROM Study_courses WHERE activity_id =
@ActivityId)
        SET @ActivityType = 'Study course';
    ELSE IF EXISTS (SELECT 1 FROM Course_modules WHERE activity_id =
@ActivityId)
        SET @ActivityType = 'Course module';
    ELSE
        SET @ActivityType = 'No data';

    RETURN @ActivityType;
END;
```

getPurchaseCount

Funkcja *getPurchaseCount* z argumentem *ActivitytId*, którym jest id aktywności, zwraca liczbę osób, która zakupiła aktywność o zadanym *activity_id.*

```
CREATE FUNCTION getPurchaseCount
(
    @ActivityId int
)
RETURNS int
AS
BEGIN
    DECLARE @Count int;
    SELECT @Count = COUNT(*)
    FROM Student_activities
    WHERE activity_id = @ActivityId;
    RETURN @Count;
END;
```

getModuleTypeCount

Funkcja **GetModuleTypeCount** z parametrem `@ModuleTypeId`, którym jest id typu modułu, zwraca liczbę modułów kursu (`Course_modules`) o podanym typie.
Wartość zwracana to liczba (`int`) modułów o zadanym `module_type_id`.

```
CREATE FUNCTION getModuleTypeCount
(
    @ModuleTypeId smallint
)
RETURNS int
AS
BEGIN
    DECLARE @Count int;
    SELECT @Count = COUNT(*)
    FROM Course_modules
    WHERE module_type_id = @ModuleTypeId;
    RETURN @Count;
END;
```

hasAssignedClasses

Funkcja **HasAssignedClasses** z parametrem `@UserId`, którym jest id użytkownika (tutora lub tłumacza), zwraca wartość logiczną (`bit`), wskazującą, czy użytkownik ma przypisane zajęcia. Funkcja zwraca `1`, jeśli użytkownik prowadzi co najmniej jeden przedmiot, praktyki lub zajęcia tłumaczeniowe, lub `0`, jeśli nie ma przypisanych żadnych zajęć.

```
CREATE FUNCTION hasAssignedClasses
(
    @UserId int
)
RETURNS bit
AS
BEGIN
    DECLARE @HasClasses bit = 0;

    IF EXISTS (
        SELECT 1
        FROM Study_courses
        WHERE tutor_id = @UserId
        UNION
        SELECT 1
        FROM Internships
        WHERE coordinator_id = @UserId
        UNION
        SELECT 1
        FROM Meetings
        WHERE translator_id = @UserId
    )
        SET @HasClasses = 1;

    RETURN @HasClasses;
END;
```

Triggery

deleteConnectedWithMeeting

Trigger *deleteConectedWithMeeting* podczas próby usunięcia rekordu z tablicy *Meetings*, usunie wszystkie rekordy zależne od niego, to jest posiadające to samo *meeting_id* i na końcu rekord, który chcieliśmy usunąć.

```
CREATE TRIGGER deleteConnectedWithMeeting
ON Meetings
INSTEAD OF DELETE
AS
BEGIN
    DELETE FROM Attendance WHERE meeting_id IN (SELECT meeting_id FROM
deleted);
    DELETE FROM Online_synchronous WHERE meeting_id IN (SELECT meeting_id
FROM deleted);
    DELETE FROM Online_asynchronous WHERE meeting_id IN (SELECT meeting_id
FROM deleted);
    DELETE FROM Stationary WHERE meeting_id IN (SELECT meeting_id FROM
deleted);
    DELETE FROM Module_meetings WHERE meeting_id IN (SELECT meeting_id
FROM deleted);
    DELETE FROM Meetings WHERE meeting_id IN (SELECT meeting_id FROM
deleted);
END;
```

validateActivityPurchase

Trigger ValidateActivityPurchase jest uruchamiany przy próbie dodania nowej pozycji do tabeli `Shopping_cart`.

Jeśli `activity_id` odpowiada aktywności, której wartość `to_buy` wynosi `0`, trigger zapobiega wstawieniu takiego rekordu i zgłasza błąd o treści *"Cannot add non-purchasable activity to cart"*.

Dla aktywności, które są oznaczone jako dostępne do zakupu (`to_buy = 1`), rekord jest wstawiany poprawnie.

```
CREATE TRIGGER ValidateActivityPurchase
ON Shopping_cart
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Activities a ON i.activity_id = a.activity_id
        WHERE a.to_buy = 0
    )
    BEGIN
        RAISERROR ('Cannot add non-purchasable activity to cart', 16, 1);
        RETURN;
    END

    INSERT INTO Shopping_cart (student_id, activity_id)
    SELECT i.student_id, a.activity_id
    FROM inserted i
    JOIN Activities a ON i.activity_id = a.activity_id
    WHERE a.to_buy = 1;
END;
```

validateOrderDetails

Trigger ValidateOrderDetails jest uruchamiany przy próbie dodania nowego rekordu do tabeli `Order_details`.
Jeśli `activity_id` odnosi się do aktywności z `to_buy = 0`, trigger blokuje operację i zgłasza błąd o treści *"Cannot add non-purchasable activity to order"*.
W przypadku aktywności oznaczonych jako dostępne do zakupu (`to_buy = 1`), rekord jest wstawiany poprawnie do tabeli.

```
CREATE TRIGGER validateOrderDetails
ON Order_details
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Activities a ON i.activity_id = a.activity_id
        WHERE a.to_buy = 0
    )
    BEGIN
        RAISERROR ('Cannot add non-purchasable activity to order', 16, 1);
        RETURN;
    END

    INSERT INTO Order_details (order_id, activity_id, description, price,
payment_date)
    SELECT order_id, a.activity_id, description, price, payment_date
    FROM inserted i
    JOIN Activities a ON i.activity_id = a.activity_id
    WHERE a.to_buy = 1;
END;
```

AddToStudentActivities

Trigger *AddToStudentActivities* podczas zakupu aktywności przez studenta (dodaniu do tabeli *Order_details* rekordu z określonym *student_id* oraz *activity_id*) dodaje do tabeli *Student_activities* rekord o tym samym *student_id* oraz *activity_id.*

```
CREATE TRIGGER AddToStudentActivities
ON Order_details
AFTER INSERT
AS
BEGIN
    INSERT INTO Student_activities (student_id, activity_id)
    SELECT o.student_id, i.activity_id
    FROM inserted i
    JOIN Orders o ON i.order_id = o.order_id
    WHERE NOT EXISTS (
        SELECT 1
        FROM Student_activities sa
        WHERE sa.student_id = o.student_id
        AND sa.activity_id = i.activity_id
    );
END;
```

Indeksy

Indeksy dla Wszystkich kluczy głównych utworzone zostały automatycznie przy tworzeniu tabel. Dodajemy indeksy dla wyszkich kluczy obcych.

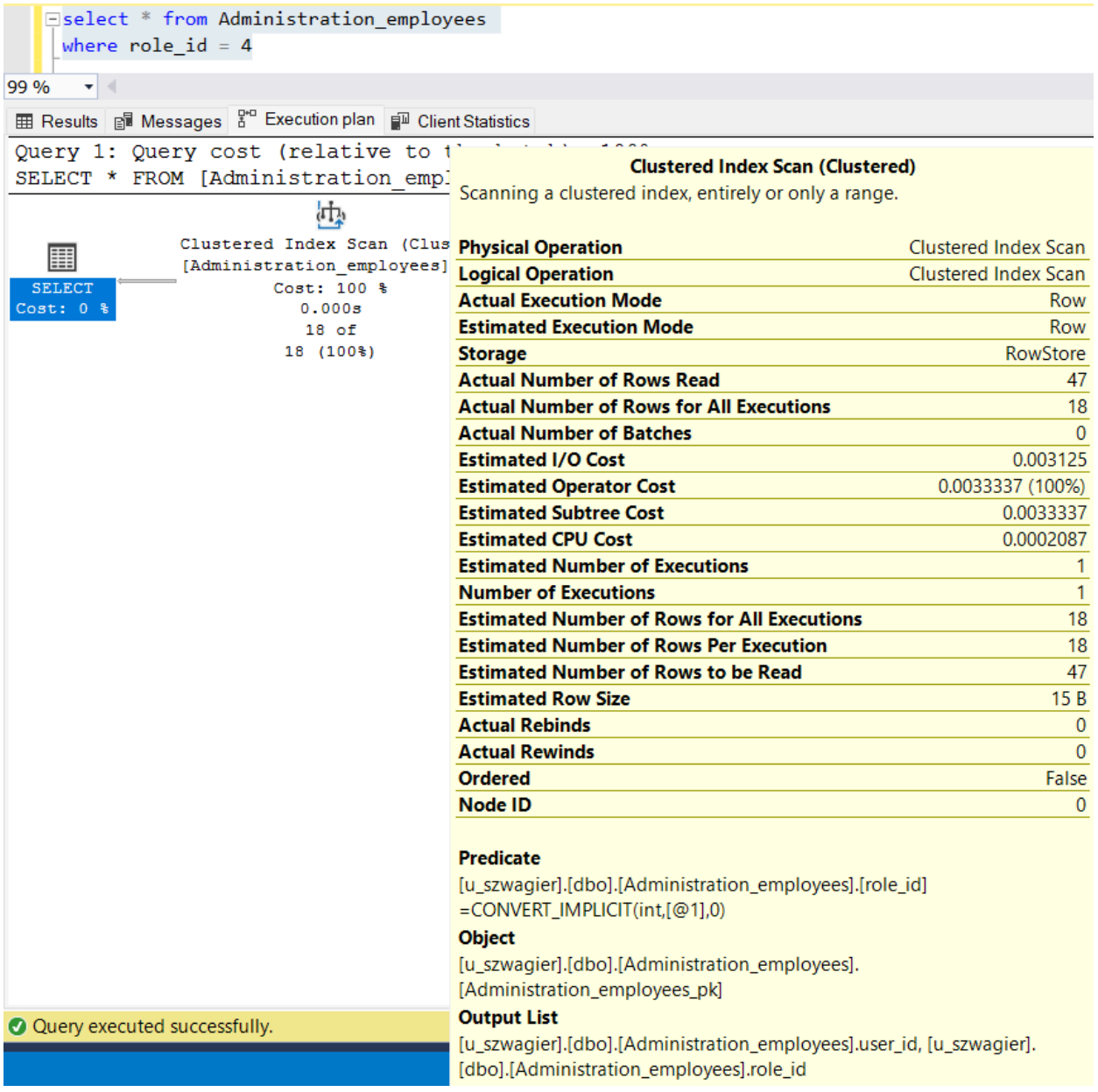
```
CREATE INDEX AdministrationEmployees_RoleID ON Administration_employees
(role_id);

CREATE INDEX AdministrationEmployees_UserID ON Administration_employees
(user_id);

CREATE INDEX Attendance_MeetingID ON Attendance (meeting_id);

CREATE INDEX Attendance_StudentID ON Attendance (student_id);

CREATE INDEX City_CountryID ON City (country_id);

CREATE INDEX CourseModules_ActivityID ON Course_modules (activity_id);

CREATE INDEX CourseModules_CourseID ON Course_modules (course_id);

CREATE INDEX CourseModules_ModuleTypeID ON Course_modules
(module_type_id);

CREATE INDEX Courses_ActivityID ON Courses (activity_id);

CREATE INDEX Internships_StudiesID ON Internships (studies_id);

CREATE INDEX Internships_CoordinatorID ON Internships (coordinator_id);

CREATE INDEX InternshipsAttendance_StudiesID ON Internships_attendance
(studies_id);

CREATE INDEX InternshipsAttendance_StudentID ON Internships_attendance
(student_id);

CREATE INDEX Meetings_ActivityID ON Meetings (activity_id);

CREATE INDEX Meetings_Translator ON Meetings (translator_id, activity_id);

CREATE INDEX ModuleMeetings_MeetingID ON Module_meetings (meeting_id);
```

```
CREATE INDEX ModuleMeetings_StudiesModule ON Module_meetings (studies_id,
module_number);

CREATE INDEX OnlineAsynchronoues_MeetingID ON Online_asynchronous
(meeting_id);

CREATE INDEX OnlineSynchronoues_MeetingID ON Online_synchronous
(meeting_id);

CREATE INDEX OnlineSynchronoues_PlatformID ON Online_synchronous
(platform_id);

CREATE INDEX OrderDetails_ActivityID ON Order_details (activity_id);

CREATE INDEX OrderDetails_OrderID ON Order_details (order_id);

CREATE INDEX Orders_StudentID ON Orders (student_id);

CREATE INDEX ShoppingCart_ActivityID ON Shopping_cart (activity_id);

CREATE INDEX ShoppingCart_StudentID ON Shopping_cart (student_id);

CREATE INDEX Stationary_ClassroomID ON Stationary (classroom_id);

CREATE INDEX Stationary_MeetingID ON Stationary (meeting_id);

CREATE INDEX StudentActivities_ActivityID ON Student_activities
(activity_id);

CREATE INDEX StudentActivities_StudentID ON Student_activities
(student_id);

CREATE INDEX Students_studentID ON Students (student_id);

CREATE INDEX Studies_ActivityID ON Studies (activity_id);

CREATE INDEX Studies_Semester ON Studies (semester);

CREATE INDEX StudiesModule_StudiesID ON Studies_module (studies_id);

CREATE INDEX StudiesModule_ModuleTypeID ON Studies_module
(module_type_id);
```

```
CREATE INDEX StudyCourses_ActivityID ON Study_courses (activity_id);

CREATE INDEX StudyCourses_StudiesID ON Study_courses (studies_id);

CREATE INDEX StudyCourses_StudyCourseID ON Study_courses
(study_course_id);

CREATE INDEX StudyCourses_TutorID ON Study_courses (tutor_id);

CREATE INDEX Translator_KnownLanguageID ON Translator (known_language_id);

CREATE INDEX Translator_UserID ON Translator (user_id);

CREATE INDEX Tutor_UserID ON Tutor (user_id);

CREATE INDEX Users_CityID ON Users (city_id);

CREATE INDEX Webinar_ActivityID ON Webinar (activity_id);
```

Sprawdzenie działania przykładowego indeksu

Pierwszy select bez dodanego indeksu, drugi po dodaniu indeksu na kolumnę role_id.

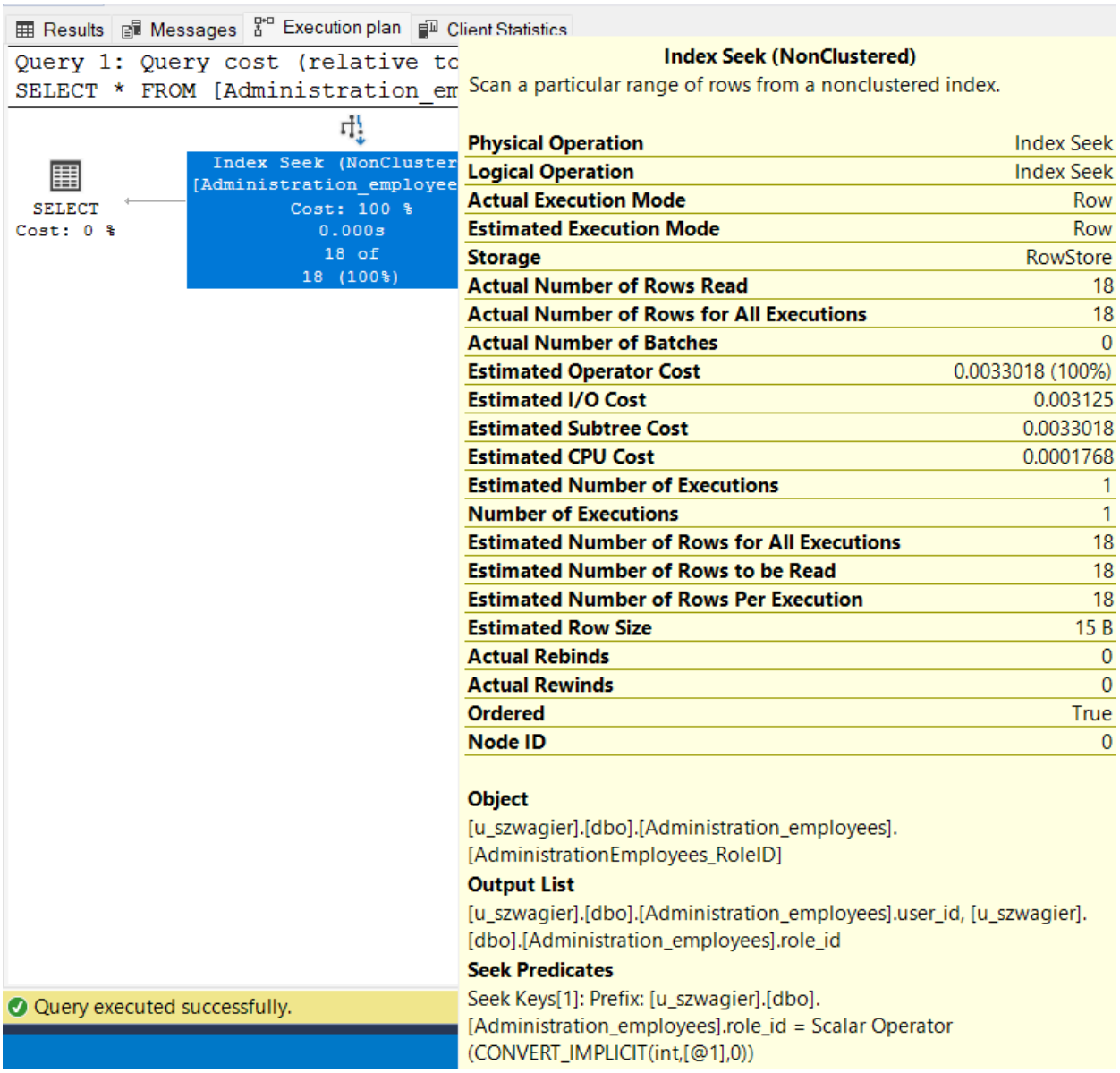


```
select * from Administration_employees
where role_id = 4
```

99 %

Results | Messages | Execution plan | Client Statistics

	Trial 6	
Client Execution Time	10:46:26	
Query Profile Statistics		
Number of INSERT, DELETE and UPDATE statements	0	→
Rows affected by INSERT, DELETE, or UPDATE statements	0	→
Number of SELECT statements	3	→
Rows returned by SELECT statements	19	→
Number of transactions	0	→
Network Statistics		
Number of server roundtrips	3	→
TDS packets sent from client	3	→
TDS packets received from server	5	→
Bytes sent from client	290	→
Bytes received from server	9952	→
Time Statistics		
Client processing time	81	↑
Total execution time	91	↑
Wait time on server replies	10	↓


Results
Messages
Execution plan
Client Statistics
Query 1: Query cost (relative to
SELECT * FROM [Administration_em
SELECT
Cost: 0 %
Index Seek (NonCluster
[Administration_employee
Cost: 100 %
0.000s
18 of
18 (100%)
Query executed successfully.
Index Seek (NonClustered)
Scan a particular range of rows from a nonclustered index.
Physical Operation Index Seek
Logical Operation Index Seek
Actual Execution Mode Row
Estimated Execution Mode Row
Storage RowStore
Actual Number of Rows Read 18
Actual Number of Rows for All Executions 18
Actual Number of Batches 0
Estimated Operator Cost 0.0033018 (100%)
Estimated I/O Cost 0.003125
Estimated Subtree Cost 0.0033018
Estimated CPU Cost 0.0001768
Estimated Number of Executions 1
Number of Executions 1
Estimated Number of Rows for All Executions 18
Estimated Number of Rows to be Read 18
Estimated Number of Rows Per Execution 18
Estimated Row Size 15 B
Actual Rebinds 0
Actual Rewinds 0
Ordered True
Node ID 0
Object
[u_szwagier].[dbo].[Administration_employees].
[AdministrationEmployees_RoleID]
Output List
[u_szwagier].[dbo].[Administration_employees].user_id, [u_szwagier].
[dbo].[Administration_employees].role_id
Seek Predicates
Seek Keys[1]: Prefix: [u_szwagier].[dbo].
[Administration_employees].role_id = Scalar Operator
(CONVERT_IMPLICIT(int,[@1],0))

```
select * from Administration_employees
where role_id = 4
```

99 %

Results | Messages | Execution plan | Client Statistics

	Trial 2
Client Execution Time	10:30:58
Query Profile Statistics	
Number of INSERT, DELETE and UPDATE statements	0
Rows affected by INSERT, DELETE, or UPDATE statements	0
Number of SELECT statements	3
Rows returned by SELECT statements	19
Number of transactions	0
Network Statistics	
Number of server roundtrips	3
TDS packets sent from client	3
TDS packets received from server	5
Bytes sent from client	290
Bytes received from server	9968
Time Statistics	
Client processing time	19
Total execution time	42
Wait time on server replies	23

Role i uprawnienia

```
--Guest
GRANT SELECT ON Courses TO guest;
GRANT SELECT ON Studies TO guest;
GRANT SELECT ON Webinar TO guest;
GRANT EXECUTE ON addStudent TO guest;

--Student
GRANT SELECT ON Courses TO student;
GRANT SELECT ON Course_modules TO student;
GRANT SELECT ON Studies TO student;
GRANT SELECT ON Study_courses TO student;
GRANT SELECT ON Webinar TO student;
GRANT EXECUTE ON addToShoppingCart TO student;
GRANT EXECUTE ON createOrderFromCart TO student;
```

```
GRANT EXECUTE ON getStudentAttendance TO student;
GRANT SELECT ON Tutor_studies_date_view TO student;

--Tutor
GRANT EXECUTE ON updateGrade TO tutor;
GRANT EXECUTE ON removeGrade TO tutor;
GRANT EXECUTE ON addAttendance TO tutor;
GRANT EXECUTE ON getTutorSchedule TO tutor;
GRANT EXECUTE ON AddStationaryMeeting TO tutor;
GRANT EXECUTE ON AddOnlineAsynchronousMeeting TO tutor;
GRANT EXECUTE ON AddOnlineSynchronousMeeting TO tutor;
GRANT EXECUTE ON updateAttendance TO tutor;
GRANT SELECT ON Translators_languages_view TO tutor;

--Admin
GRANT ALL ON Activities TO admin;
GRANT ALL ON Administration_employees TO admin;
GRANT ALL ON Attendance TO admin;
GRANT ALL ON City TO admin;
GRANT ALL ON Classrooms TO admin;
GRANT ALL ON Country TO admin;
GRANT ALL ON Course_modules TO admin;
GRANT ALL ON Courses TO admin;
GRANT ALL ON Internships TO admin;
GRANT ALL ON Internships_attendance TO admin;
GRANT ALL ON Languages TO admin;
GRANT ALL ON Meetings TO admin;
GRANT ALL ON Module_meetings TO admin;
GRANT ALL ON Module_Type TO admin;
GRANT ALL ON Online_asynchronous TO admin;
GRANT ALL ON Online_synchronous TO admin;
GRANT ALL ON Order_details TO admin;
GRANT ALL ON Orders TO admin;
GRANT ALL ON Platforms TO admin;
GRANT ALL ON Roles TO admin;
GRANT ALL ON Semesters TO admin;
GRANT ALL ON Shopping_cart TO admin;
GRANT ALL ON Stationary TO admin;
GRANT ALL ON Student_activities TO admin;
GRANT ALL ON Students TO admin;
GRANT ALL ON Studies TO admin;
GRANT ALL ON Studies_module TO admin;
GRANT ALL ON Study_courses TO admin;
```

```
GRANT ALL ON Study_courses_names TO admin;
GRANT ALL ON Translator TO admin;
GRANT ALL ON Tutor TO admin;
GRANT ALL ON Users TO admin;
GRANT ALL ON Webinar TO admin;
GRANT EXECUTE ON getStudentsByCountry TO admin;
GRANT EXECUTE ON getTutorSchedule TO admin;
GRANT EXECUTE ON getTranslatorSchedule TO admin;
GRANT EXECUTE ON getMeetingsByLanguage TO admin;
GRANT EXECUTE ON getStudentAttendance TO admin;
GRANT EXECUTE ON getStudentActivityAttendance TO admin;
GRANT EXECUTE ON getStudiesWithCourse TO admin;
GRANT EXECUTE ON getStudentCart TO admin;
GRANT EXECUTE ON getStudiesCourses TO admin;
GRANT EXECUTE ON getCourseModules TO admin;
GRANT EXECUTE ON getStudiesSemesterDates TO admin;
GRANT EXECUTE ON getMeetingAttendance TO admin;
GRANT EXECUTE ON changeUserPassword TO admin;
GRANT EXECUTE ON addToShoppingCart TO admin;
GRANT EXECUTE ON removeFromShoppingCart TO admin;
GRANT EXECUTE ON createOrderFromCart TO admin;
GRANT EXECUTE ON updateGrade TO admin;
GRANT EXECUTE ON removeGrade TO admin;
GRANT EXECUTE ON viewStudentGrades TO admin;
GRANT EXECUTE ON getStudentGradeAverage TO admin;
GRANT EXECUTE ON addAttendance TO admin;
GRANT EXECUTE ON updateAttendance TO admin;
GRANT EXECUTE ON removeAttendance TO admin;
GRANT EXECUTE ON AddCourseWithModules TO admin;
GRANT EXECUTE ON updateCourse TO admin;
GRANT EXECUTE ON deleteCourse TO admin;
GRANT EXECUTE ON addWebinar TO admin;
GRANT EXECUTE ON updateWebinar TO admin;
GRANT EXECUTE ON deleteWebinar TO admin;
GRANT EXECUTE ON addStudies TO admin;
GRANT EXECUTE ON updateStudies TO admin;
GRANT EXECUTE ON deleteStudies TO admin;
GRANT EXECUTE ON AddStationaryMeeting TO admin;
GRANT EXECUTE ON UpdateStationaryMeeting TO admin;
GRANT EXECUTE ON AddOnlineSynchronousMeeting TO admin;
GRANT EXECUTE ON UpdateOnlineSynchronousMeeting TO admin;
GRANT EXECUTE ON AddOnlineAsynchronousMeeting TO admin;
GRANT EXECUTE ON UpdateOnlineAsynchronousMeeting TO admin;
```

```
GRANT EXECUTE ON DeleteMeeting TO admin;
GRANT EXECUTE ON addStudent TO admin;
GRANT EXECUTE ON updateStudent TO admin;
GRANT EXECUTE ON deleteStudent TO admin;
GRANT EXECUTE ON addTutor TO admin;
GRANT EXECUTE ON updateTutor TO admin;
GRANT EXECUTE ON deleteTutor TO admin;
GRANT EXECUTE ON addTranslator TO admin;
GRANT EXECUTE ON updateTranslator TO admin;
GRANT EXECUTE ON deleteTranslator TO admin;
GRANT EXECUTE ON addAdminEmployee TO admin;
GRANT EXECUTE ON updateAdminEmployee TO admin;
GRANT EXECUTE ON deleteAdminEmployee TO admin;
GRANT EXECUTE ON updateOrderDetails TO admin;
GRANT EXECUTE ON assignStudentActivity TO admin;
GRANT EXECUTE ON getStudentTotalCosts TO admin;
GRANT EXECUTE ON getActivityType TO admin;
GRANT EXECUTE ON getPurchaseCount TO admin;
GRANT EXECUTE ON getModuleTypeCount TO admin;
GRANT EXECUTE ON hasAssignedClasses TO admin;

--Accountant
GRANT SELECT ON Activity_Revenue_View TO accountant;
GRANT SELECT ON Unpaid_Activities_View TO accountant;


--Headmaster
GRANT SELECT ON Tutor_studies_date_view TO headmaster;
GRANT EXECUTE ON viewStudentGrades TO headmaster;
GRANT EXECUTE ON AssignStudentActivity TO headmaster;
GRANT SELECT ON Activity_Revenue_View TO headmaster;
GRANT SELECT ON Unpaid_Activities_View TO headmaster;
GRANT SELECT ON Future_Event_Registrations_View TO headmaster;
GRANT SELECT ON Completed_Event_Attendance_View TO headmaster;
GRANT SELECT ON Time_Conflicts_View TO headmaster;
GRANT EXECUTE ON getStudentActivityAttendance TO headmaster;
GRANT EXECUTE ON getMeetingAttendance TO headmaster;

--Administration Employee
GRANT EXECUTE ON updateStudies TO administration_employee;
GRANT ALL ON Courses TO administration_employee;
GRANT ALL ON Course_modules TO administration_employee;
GRANT ALL ON Webinar TO administration_employee;
```

```
GRANT ALL ON Studies TO administration_employee;
GRANT ALL ON Study_courses TO administration_employee;
GRANT EXECUTE ON AddCourseWithModules TO administration_employee;
GRANT EXECUTE ON updateCourse TO administration_employee;
GRANT EXECUTE ON deleteCourse TO administration_employee;
GRANT EXECUTE ON AddWebinar TO administration_employee;
GRANT EXECUTE ON updateWebinar TO administration_employee;
GRANT EXECUTE ON deleteWebinar TO administration_employee;
GRANT EXECUTE ON AddStudies TO administration_employee;
GRANT EXECUTE ON updateStudies TO administration_employee;
GRANT EXECUTE ON deleteStudies TO administration_employee;
GRANT EXECUTE ON AddStationaryMeeting TO administration_employee;
GRANT EXECUTE ON UpdateStationaryMeeting TO administration_employee;
GRANT EXECUTE ON AddOnlineSynchronousMeeting TO administration_employee;
GRANT EXECUTE ON UpdateOnlineSynchronousMeeting TO
administration_employee;
GRANT EXECUTE ON AddOnlineAsynchronousMeeting TO administration_employee;
GRANT EXECUTE ON UpdateOnlineAsynchronousMeeting TO
administration_employee;
GRANT EXECUTE ON DeleteMeeting TO administration_employee;
GRANT SELECT ON assignStudentActivity TO administration_employee;
GRANT SELECT ON Future_Event_Registrations_View TO
administration_employee;
GRANT SELECT ON Completed_Event_Attendance_View TO
administration_employee;

--Internship Coordinator
GRANT EXECUTE ON addAttendance TO internship_coordinator;
GRANT EXECUTE ON updateGrade TO internship_coordinator;
GRANT SELECT ON Tutor_studies_date_view TO internship_coordinator;
```

Generowanie danych

Na początku, część danych była generowana za pomocą poniższego kodu. Dane nie były generowane za jednym razem i plik był wielokrotnie edytowany, aby generować dane do jednej tabeli w kilku rzutach. W niektórych przypadkach dane były dodawane ręcznie np. do Study_course_names.

```
from faker import Faker
import random
import json
import numpy as np

NUMBER=10

# Initialize Faker
fake = Faker("pl_PL")

# Helper function to generate realistic data for tables
def generate_data():
    user_id_student=[]
    user_tutor_id=[]
    user_translator_id=[]
    usre_admin_id=[]
    # for i in range(3,4201):
    #     rand=random.randint(0,99)
    #     if rand<93:
    #         user_id_student.append(i)
    #     elif rand<97:
    #         user_tutor_id.append(i)
    #     elif rand<99:
    #         user_translator_id.append(i)
    #     else:
    #         usre_admin_id.append(i)
    for i in range(4201,5001):
        rand=random.randint(0,99)
        if rand<95:
            user_id_student.append(i)
        else:
            user_tutor_id.append(i)


    # Data structure
    data = {}
```

```
    activ=[]
    for i in range(400):
        activ

    # Table: Activities
    data["Activities"] = [
        {"activity_id": i, "to_buy": 0} for i in range(31001, 0)
    ]

    # Table: Administration_employees
    data["Administration_employees"] = [
        {"user_id": usre_admin_id[i], "role_id": random.randint(3, 4)} for
i in range(0, len(usre_admin_id))
    ]



    # Table: City
    #cities=["Warszawa", "Kraków", "Łódź", "Wrocław", "Poznań", "Gdańsk",
"Szczecin", "Bydgoszcz", "Lublin", "Katowice",
 #"Białystok", "Gdynia", "Częstochowa", "Radom", "Toruń", "Kielce",
"Zielona Góra", "Olsztyn", "Rzeszów",
 #"Opole", "Legnica", "Koszalin", "Tarnów", "Chorzów", "Nowy Sącz",
"Słupsk", "Świętochłowice", "Jaworzno",
 #"Płock", "Siemianowice Śląskie", "Stalowa Wola", "Zabrze", "Mysłowice",
"Elbląg", "Gorzów Wielkopolski",
 #"Gniezno", "Radomsko", "Przemyśl", "Lubań", "Kalisz", "Piotrków
Trybunalski", "Siedlce", "Świdnica",
 #"Piła", "Pabianice", "Kutno", "Włocławek", "Skarżysko-Kamienna",
"Bielsko-Biała", "Lubliniec", "Świecie"]
    #cities=["Paryż", "Marsylia", "Lyon", "Tuluza", "Nicea", "Nantes",
"Strasburg", "Montpellier", "Bordeaux", "Lille"]
    cities=["Berlin", "Monachium", "Hamburg", "Frankfurt", "Stuttgart",
"Düsseldorf", "Dortmund", "Essen", "Leipzig", "Bremen"]
    #cities=["Ułan Bator", "Erdenet", "Darhan", "Mörön", "Sühbaatar",
"Ölgii", "Choibalsan", "Altai", "Zuunmod", "Baganuur"]
    #cities=["Kijów", "Charków", "Odessa", "Dnipro", "Zaporoże",
"Chersoń", "Donieck", "Łuck", "Krzywy Róg"]


    data["City"] = [
        {"city": cities[i], "country_id": 2} for i in
range(0,0)#len(cities))
```

```
    ]

    # Table: Classrooms
    data["Classrooms"] = [
        {"classroom_name": f"{i}", "place_limit": random.randint(60, 100)}
for i in range(401,  401)
    ]

    # Table: Country
    #data["Country"] = [
    #    {"country": country} for country in ["Polska", "Niemcy",
"Francja", "Mongolia", "Ukraina"]
    #]

    # Table: Course_modules
    data["Course_modules"] = [
        {"activity_id": i, "course_id": 1+(i-31000-1)//4,
"module_type_id": random.randint(1, 4)} for i in range(31001,   0)
    ]

    # Table: Courses
    data["Courses"] = [
        {
            "activity_id": i,
            "course_name": fake.word() + " " + fake.word(),
            "price_for_course": random.randint(100, 500),
            "places_occupied": random.randint(0, 30),
            "place_limit": random.randint(30, 40),
            "entry_fee": random.randint(50, 200),
        }
        for i in range(300001, 30021)
    ]

    # Table: Internships
    data["Internships"] = [
        {"studies_id": i, "coordinator_id": random.randint(1, 3)} for i in
range(1, 1)
    ]

    # Table: Internships_attendance


    lang=["niemiecki","francuski","mongolski","ukraiński"]
```

```
    # Table: Languages
    data["Languages"] = [
        {"language": lang[_-1]} for _ in range(1, 1)
    ]

    # Table: Meetings
    data["Meetings"] = [
        {"translator_id": "NULL", "language_id": "NULL", "activity_id": _}
        for _ in range(11001, 11097)
    ]

    # Table: Module_meetings
    data["Module_meetings"] = [
        {"studies_id": random.randint(1, 5), "module_number":
random.randint(1, 5), "meeting_id": random.randint(1, 1)}
        for _ in range(1, 1)
    ]

    # Table: Module_type
    data["Module_type"] = [
        {"module_type": f"Module {i}"} for i in range(1, 1)
    ]

    # Table: Platforms
    data["Platforms"] = [
        {"platform_URL_address":
"www.youtube.com/"+str(random.randint(123456799,999999999))} for _ in
range(1, 1)
    ]

    # Table: Roles
    role=["dyrektor","księgowy","pracownik administracji"]
    data["Roles"] = [
        {"role_name": role[i-1]} for i in range(1, 1)
    ]

    # Table: Semesters
    data["Semesters"] = [
        {"semester": i, "start_date": fake.date(), "end_date":
fake.date()}
        for i in range(1, 1)
    ]
```

```
    # Table: Shopping_cart
    data["Shopping_cart"] = [
        {"student_id": random.randint(1, 5001), "activity_id":
random.randint(10000, 32000)}
        for _ in range(1, 1)
    ]

    # Table: Student_activities
    t=[1589, 2253, 687, 2894, 215, 710, 2207, 192, 856, 2107, 3040, 4955,
1735, 2276, 2399, 3481, 3727, 4168, 1689, 3232, 610, 169, 1105, 2502,
3996, 464, 4188, 3335, 4586, 607, 1271, 1812, 2814, 3976, 3212, 4537,
4709, 1440, 4686, 836, 730, 1228, 2751, 736, 1068, 3060, 3793, 3770, 4835,
1609, 3601, 4271, 876, 1045, 2373, 4294, 899, 4440, 4749, 375, 2542, 2628,
3624, 2233, 2688, 3020, 3315, 4311, 916, 3816, 272, 1108, 2977, 4895,
1672, 2170, 3166, 3687, 4706, 770, 4397, 4663, 2150, 4142, 1194, 3899, 6,
4334, 4832, 3730, 292, 1629, 3538, 590, 2754, 2997, 3252, 4918, 441, 982,
3146, 2213, 3295, 1921, 3249, 3913, 355, 1234, 1898, 2562, 4128, 1065,
1729, 4500, 424, 3080, 2980, 4105, 3126, 4059, 1506, 4351, 2124, 673,
2665, 650, 773, 1832, 4915, 3398, 2957, 942, 2167, 2416, 4580, 152, 3982,
3541, 3790, 4623, 444, 985, 2419,
521, 2728, 3129, 4434, 4148, 4935, 753, 1062, 4062, 4394, 1360, 1254,
2333, 4603, 713, 1208, 1795, 2834, 3332, 2525, 1125, 1792, 1815, 3524,
2791, 776, 1145, 4563, 255, 3587, 43, 4371, 2270, 3893, 4975, 3544, 4626,
1772, 2313, 2854, 3395, 4477, 3865, 3796, 1094, 2345, 3009, 4337, 779,
1117, 138, 802, 1466, 2963, 948, 1071, 1489, 2153, 2840, 576, 1904, 2886,
3719, 3573, 4011, 4111, 4947, 4552, 2929, 4921, 699, 1861, 2906, 3570,
4068, 722, 4801, 2196, 2737, 158, 656, 1320, 1389, 530, 2030, 848, 4629,
805, 2797, 4718, 928, 3507, 2279, 2820, 2943, 3135, 2448, 3530, 2342, 410,
1406, 218, 719, 1300, 1841, 4503, 4941, 178, 3593, 4589, 2574, 1343, 3115,
4781,
 931, 1429, 1715, 3636, 2013, 2511, 4177, 4675, 198, 3444, 3510, 2800,
2159, 4320, 1844, 29, 908, 2236, 2531, 4738, 3610, 4005, 1990, 221, 716,
1549, 616, 1403, 1944, 2485, 3026, 1695, 2408, 3902, 4400, 1054, 1887,
3069, 3733, 3092, 3384, 4466, 4735, 2757, 4025, 4715, 2700, 4274, 3590,
52, 885, 1134, 2216, 2657, 427, 3344, 1137, 450, 258, 1615, 2156, 1638,
3630, 1784, 3112, 3653, 78, 32, 905, 4987, 3175, 4254, 1160, 4795, 3321,
4108, 12, 4755, 1529, 2242, 3736, 4732, 4469,
 1827, 1017, 4844, 1750, 3078, 230, 2932, 4598, 1455, 1996, 2291, 1896,
2142, 2978, 3224, 4057, 2042, 3370, 3911, 4452, 4698, 1412, 58, 3808, 81,
1953, 38, 207, 2663, 3290, 4913, 3098, 3596, 1332, 1624, 2165, 4870, 2703,
4910, 3868, 1853, 3436, 456, 997, 1120, 18, 4346, 1312, 2331, 3659, 519,
2185, 3519, 3762, 4996, 2139, 3221, 4060, 124, 911, 1993, 2534, 1813,
2809, 4495, 602, 2789, 61, 3997, 433, 3350, 4761, 41, 1037, 1535, 4369,
```

```
4867, 3977, 4973, 496, 2162, 2660, 3393, 1518, 2205, 2869, 3748, 144,
2351, 3679, 2892, 4220, 4389, 2374, 3579, 3410, 313, 2723, 708, 2577, 21,
4684, 1246, 2477, 1077, 4827, 1438, 2600, 3851, 4515, 1787, 2451, 3951,
4492, 4615,
2990, 4487, 2472, 2890, 3554, 219, 2426, 2721, 660, 906, 1739, 1052, 2675,
1301, 1842, 73, 1155, 368, 534, 4613, 2598, 2575, 511, 1765, 3216, 3282,
3780, 4633, 3431, 4513, 2698, 2947, 434, 242, 1075, 683, 4384, 4925, 1865,
491, 1573, 285, 617, 3534, 4530, 594, 3488, 3511, 4507, 116, 4361, 139,
2154, 2323, 1759, 2841, 1951, 3279, 1264, 1891, 2887, 4902, 1350, 4158,
1845, 2386, 202, 700, 4922, 1911, 3903, 33, 2452, 4616, 3969, 1241, 3405,
2260, 2758, 3385, 743, 3056, 3448, 637, 1135, 1178, 1327, 3491, 451, 4032,
4696, 1161, 4742, 1138, 282, 1453, 2781, 79, 328, 1407, 720, 3614, 3863,
4201, 1307, 3176, 3514, 4842, 912, 2240, 3322, 2140, 1530, 4258, 2761,
4470, 205, 13, 1430, 2555, 3053, 2661, 3302, 3388, 4029, 4527, 2512, 4178,
4676,  4175, 454, 2446, 4367, 1141, 3010, 408, 3528, 3611, 1596, 3657,
4653, 122, 1450, 4112, 2678, 1745, 1991, 517, 1364, 3285, 3783, 2074,
2031, 4610, 3305, 580, 2572, 3826, 4069, 2054, 4759, 1121, 2117, 4928,
4238, 1384, 1925, 3634, 2701, 3199, 345, 4132, 1044, 1708, 2395, 4410,
2564, 1539, 311, 1854, 334, 1585, 2495, 2913, 357, 1021, 1685, 1808, 1393,
4061, 2000, 4997, 2687, 4015, 4748, 1293, 1459, 1957, 1436, 583, 2833,
3500, 560, 460, 1124, 2790, 483, 4141, 752, 603, 775, 2767, 3308, 2710,
3600, 1087, 1665, 3351, 3792, 3643, 4084, 3935, 4476, 546, 2229, 3225,
4436, 1496, 955, 268, 563, 4098, 4891, 1751, 2083, 1064, 3417, 1648, 3812,
1127, 3119, 108, 1104, 414, 915, 4161, 437, 4851, 1147, 3311, 3354, 2080,
3162, 3789, 1774, 3895, 2664, 4914, 543, 2209, 4373, 4871, 2856, 4479,
4330, 394, 935, 3640, 4722, 1399, 2063, 2727, 4934, 4078, 4247, 2773,
3437, 4888, 2627, 1940, 4393, 4642, 1794, 1645, 666, 812, 1107, 1894,
1602, 105, 769, 4639, 4854, 4831, 4868, 2604, 1914, 2996, 1273, 1373,
1622, 3537, 3786, 981, 1479, 3686, 248, 689, 1771, 2312, 3394, 2870, 2893,
4104, 4768, 3039, 3062, 4390, 623, 2544, 2329, 3411, 1396, 1642, 2392,
1356, 2584, 4994, 2976, 4058, 2289, 1768, 1247, 2501, 1482, 3952, 1811,
1439, 2521, 4144, 3042, 4665, 400, 2023, 792, 2458, 2956, 643, 686, 4473,
380, 2633, 403, 1067, 1282, 572, 1900, 2441, 257, 2295, 1405, 672, 31,
1654, 1551, 1800, 3466, 3715, 2487, 3028, 3815, 2925, 3420, 3861, 4402,
4648, 2218, 154, 818, 4445, 134, 2049, 1485, 2149, 1193, 303, 2567, 2759,
695, 3941, 1136, 987, 1428, 1920, 3002, 3835, 1279, 3984, 2467, 3463,
4459, 237, 3440, 4173, 3154, 4482, 970, 924, 469, 1903, 2676, 2922, 3672,
4004, 5000, 68, 609, 4064, 2381, 174, 1943, 131, 3048, 4044, 1837, 1296,
821, 1319, 1362, 2593, 1488, 4714, 2009, 3005, 3091, 2072, 2613, 194,
3981, 1425, 4671, 3738, 4279, 48, 2301, 1614, 2255, 2232, 3798, 4462,
2155, 4147, 3506, 3752, 3111, 3529, 4193, 4857, 4980, 1050, 2378, 612,
1737, 3065, 4608, 4001, 658, 217, 466, 297, 3214, 3380, 2739, 3237, 509,
4923, 1863, 632, 3549, 4047, 2524, 4688, 2175, 3257, 3921, 4588, 1906,
```

```
4070, 1465, 2212, 340, 3337, 1322, 2404, 1030, 1671, 3795, 901, 3772,
3841, 4837, 2822, 4814, 1047, 1634, 4296, 4960,
 3254, 51, 2215, 3881, 2364, 2756, 4771, 1305, 3558, 4222, 618, 4909,
2184, 2848, 4932, 1643, 3220, 2284, 4594, 2138, 2038, 2825, 269, 1992,
1079, 1577, 2241, 2622, 3263, 3927, 1789, 1912, 2453, 2118, 3243, 103,
2012, 2994, 395, 638, 2161, 2018, 4425, 2951, 3492, 4033, 1185, 2158,
4883, 4196, 3578, 1394, 3114, 1099, 4016, 1202, 3323, 4989, 4597, 2828,
4451, 1554, 4471, 4969, 747, 2762, 2244, 3781, 4491, 14, 4514, 578, 2742,
1076, 4322, 3495, 3346, 4428, 1852, 2516, 1806, 2393, 1706, 1829, 2370,
3034, 3698, 432, 850, 1019,
    ]
    data["Student_activities"] = [
        {"student_id": _, "activity_id": random.randint(20001, 20101)}
        for _ in []#t
    ]

    # Table: Students
    data["Students"] = [
        {"student_id": user_id_student[i]} for i in range(0,
0)#len(user_id_student))
    ]

    # Table: Studies
    data["Studies"] = [
        {
            "studies_id": i,
            "activity_id": random.randint(1, 5),
            "price_for_studies_module": random.randint(100, 500),
            "entry_fee": random.randint(50, 200),
            "places_occupied": random.randint(0, 30),
            "place_limit": random.randint(20, 40),
            "semester": random.randint(1, 10),
        }
        for i in range(1, 1)
    ]

    # Table: Studies_module
    data["Studies_module"] = [
        {
            "studies_id": random.randint(1, 5),
            "module_number": random.randint(1, 5),
            "module_start": fake.date(),
            "module_end": fake.date(),
```

```
            "module_type_id": random.randint(1, 3),
            "places_ocuppied": random.randint(0, 30),
            "place_limit": random.randint(20, 40),
        }
        for _ in range(1, 1)
    ]

    # Table: Study_courses
    data["Study_courses"] = [
        {
            "activity_id": i,
            "studies_id": 1+(i-11000-1)//6,
            "tutor_id": 66,
            "study_course_id": random.randint(1, 5),
            "syllabus": fake.text(),
            "price_for_study_course_meeting": random.randint(100, 120),
        }
        for i in range(11001, 11)
    ]



    # Table: Translator
    data["Translator"] = [
        {"user_id": user_translator_id[i], "known_language_id":
random.randint(1, 4)}
        for i in range(0, 0)#len(user_translator_id))
    ]

    data["Tutor"] = [
        {
            "user_id": user_tutor_id[i],
            "internship_coordinator": 0
        } for i in range(0,0)# len(user_tutor_id))
    ]
    #NUMBER=10

    fn=[]
    ln=[]
    ea=[]
    zc=[]
    ci=[]
    NUMBER=0
    for i in range(0,NUMBER):
```

```
        fn.append(fake.first_name())
        ln.append(fake.last_name())

ea.append(fn[i][:min(3,len(fn[i]))].lower()+ln[i][:min(5,len(ln[i]))].lowe
r()+str(np.random.randint(0,9))+str(np.random.randint(0,99))+"@gmail.com")
        ci.append(str(random.randint(52,90)))
        if len(ci[i])==1:

zc.append("0"+ci[i]+"-"+str(random.randint(1,9))+str(random.randint(1,9))+
str(random.randint(1,9)))
        else:

zc.append(ci[i]+"-"+str(random.randint(1,9))+str(random.randint(1,9))+str(
random.randint(1,9)))

    data["Users"] = [
        {
            "first_name": fn[j],
            "last_name": ln[j],
            "email_address": ea[j],
            "password": fake.password(),
            "street": fake.street_address(),
            "zip_code": zc[j],
            "city_id": ci[j]
        } for j in range(0, NUMBER)
    ]

    data["Webinar"] = [
        {
            "activity_id": i,
            "webinar_name": fake.word().capitalize(),
            "price_for_webinar": random.randint(50, 300)
        } for i in range(20001, 20000)
    ]

    return data

# Generate SQL INSERT statements for each table
def generate_sql_insert(data):
    sql_statements = []

    for table, records in data.items():
        for record in records:
```

```
            columns = ", ".join(record.keys())
            values = ", ".join([repr(value) for value in record.values()])
            sql_statement = f"INSERT INTO {table} ({columns}) VALUES
({values});"
            sql_statements.append(sql_statement)

    return sql_statements

# Generate the data
generated_data = generate_data()

# Generate SQL insert statements
sql_statements = generate_sql_insert(generated_data)

# Write to a .sql file
with open("generated_data.sql", "w") as f:
    f.write("\n".join(sql_statements))

print("SQL data has been written to generated_data.sql")
```

Część danych była generowana lub edytowana w kodzie SQL, aby zachować integralność między danymi

```
UPDATE Stationary
SET places_occupied = (select sm.places_occupied from Studies_module sm
                       inner join Module_meetings as mm
                       on sm.studies_id=mm.studies_id
                       inner join Meetings as m
                       on m.meeting_id=mm.meeting_id
                       inner join Stationary as s
                       on s.meeting_id=m.meeting_id
                       where s.places_occupied<sm.places_occupied)
WHERE places_occupied<(select sm.places_occupied from Studies_module sm
                       inner join Module_meetings as mm
                       on sm.studies_id=mm.studies_id
                       inner join Meetings as m
                       on m.meeting_id=mm.meeting_id
                       inner join Stationary as s
                       on s.meeting_id=m.meeting_id
                       where s.places_occupied<sm.places_occupied);
```

```
DECLARE @StartDate AS datetime;
DECLARE @EndDate AS datetime;
SELECT @StartDate = '2023-01-01 12:00:00',
       @EndDate   = '2025-12-31 14:00:00';


insert into stationary
(meeting_id,start_time,end_time,classroom_id,places_occupied)
select m.meeting_id,
                (SELECT DATEADD(DAY,
RAND(meeting_id*99999)*(1+DATEDIFF(DAY, @StartDate,
@EndDate)),@StartDate))+round(RAND(m.meeting_id*999999)*5,0)*0.083333333,
                (SELECT DATEADD(DAY,
RAND(meeting_id*99999)*(1+DATEDIFF(DAY, @StartDate,
@EndDate)),@StartDate))+round(RAND(m.meeting_id*999999)*5,0)*0.083333333+0
.083333333
                ,round(RAND(m.meeting_id*999999)*79,0)+1,
                                                        (select
count(student_id) from attendance a
                                                        inner join
Meetings mm on mm.meeting_id=a.meeting_id
```

```
                                                        where
mm.meeting_id=m.meeting_id)
from meetings m
inner join Course_modules c
on c.activity_id=m.activity_id
where c.module_type_id=2 and ((m.meeting_id-m.meeting_id%80)/80)%2 =1
```

```
insert into stationary
(meeting_id,start_time,end_time,classroom_id,places_occupied)
select m.meeting_id, cast(module_end as
datetime)+round(RAND(m.meeting_id*999999)*5,0)*0.083333333++0.083333333*4,
                cast(module_end as
datetime)+round(RAND(m.meeting_id*999999)*5,0)*0.083333333+0.083333333*5
                ,round(RAND(m.meeting_id*999999)*79,0)+1,
                                                        (select
count(student_id) from attendance a
                                                        inner join
Meetings mm on mm.meeting_id=a.meeting_id
                                                        where
mm.meeting_id=m.meeting_id) as 'places occ'

from meetings m
inner join Study_courses w
on w.activity_id=m.activity_id
inner join Module_meetings as mo
on mo.meeting_id=m.meeting_id
inner join Studies_module as sm
on sm.studies_id=mo.studies_id and sm.module_number=mo.module_number
where w.activity_id%2=0
```

```
DECLARE @StartDate AS datetime;
DECLARE @EndDate AS datetime;
SELECT @StartDate = '2023-01-01 12:00:00',
       @EndDate   = '2025-12-31 14:00:00';
insert into Online_synchronous
(meeting_id,start_time,end_time,recording_URL_address,platform_id)
```

```
select m.meeting_id, (SELECT DATEADD(DAY,
RAND(meeting_id*99999)*(1+DATEDIFF(DAY, @StartDate,
@EndDate)),@StartDate))+round(RAND(m.meeting_id*999999)*5,0)*0.083333333,
                (SELECT DATEADD(DAY,
RAND(meeting_id*99999)*(1+DATEDIFF(DAY, @StartDate,
@EndDate)),@StartDate))+round(RAND(m.meeting_id*999999)*5,0)*0.083333333+0
.083333333
                ,'www.youtube.com/'+(select cast(
ROUND(RAND(m.meeting_id*99999)*(899999)+99999,0) as nvarchar))
                ,round(rand(m.meeting_id*999999)*(4)+6,0)
from meetings m
inner join Course_modules c
on c.activity_id=m.activity_id
where c.module_type_id=3
```

```
DECLARE @StartDate AS datetime;
DECLARE @EndDate AS datetime;
SELECT @StartDate = '2023-01-01 12:00:00',
       @EndDate   = '2025-12-31 14:00:00';



insert into Online_synchronous
(meeting_id,start_time,end_time,recording_URL_address,platform_id)
select meeting_id, (SELECT DATEADD(DAY,
RAND(meeting_id*99999)*(1+DATEDIFF(DAY, @StartDate,
@EndDate)),@StartDate)),(SELECT DATEADD(DAY,
RAND(meeting_id*99999)*(1+DATEDIFF(DAY, @StartDate, @EndDate)),@StartDate)
AS 'SalesDate')+(0.083333333)
                ,'www.youtube.com/'+(select cast(
ROUND(RAND(meeting_id*99999)*(899999)+99999,0) as nvarchar))
                ,round(rand(meeting_id*999999)*(4)+6,0)
from meetings m
inner join webinar w
on w.activity_id=m.activity_id
```

```
DECLARE @StartDate AS datetime;
DECLARE @EndDate AS datetime;
```

```
SELECT @StartDate = '2023-01-01 12:00:00',
       @EndDate   = '2025-12-31 14:00:00';


insert into Online_synchronous
(meeting_id,start_time,end_time,recording_URL_address,platform_id)
select m.meeting_id, cast(module_end as
datetime)+round(RAND(m.meeting_id*999999)*5,0)*0.083333333++0.083333333*4,
cast(module_end as
datetime)+round(RAND(m.meeting_id*999999)*5,0)*0.083333333+0.083333333*5
                ,'www.youtube.com/'+(select cast(
ROUND(RAND(m.meeting_id*99999)*(899999)+99999,0) as nvarchar))
                ,round(rand(m.meeting_id*999999)*(4)+6,0)
from meetings m
inner join Study_courses w
on w.activity_id=m.activity_id
inner join Module_meetings as mo
on mo.meeting_id=m.meeting_id
inner join Studies_module as sm
on sm.studies_id=mo.studies_id and sm.module_number=mo.module_number
where w.activity_id%2=1
```

```
DECLARE @StartDate AS datetime;
DECLARE @EndDate AS datetime;
SELECT @StartDate = '2023-01-01 12:00:00',
       @EndDate   = '2025-12-31 14:00:00';
insert into Online_asynchronous (meeting_id,recording_URL_address)
select m.meeting_id,'www.youtube.com/'+(select cast(
ROUND(RAND(m.meeting_id*99999)*(899999)+99999,0) as nvarchar))
from meetings m
inner join Course_modules c
on c.activity_id=m.activity_id
where c.module_type_id=4
```

```
DECLARE @StartDate AS datetime;
DECLARE @EndDate AS datetime;
SELECT @StartDate = '2023-01-01 12:00:00',
       @EndDate   = '2025-12-31 14:00:00';
```

```
insert into attendance (meeting_id,student_id,attendance)
select m.meeting_id,a.student_id,(SELECT CASE
    WHEN RAND(CHECKSUM(NEWID())) < 0.99 THEN 1
    ELSE 0
END AS RandomValue) from Course_modules cm
inner join Meetings as m
on m.activity_id=cm.activity_id
inner join Courses c
on c.course_id=cm.course_id
inner join Student_activities as a
on a.activity_id=c.activity_id
```

```
insert into attendance (meeting_id,student_id,attendance)
select m.meeting_id,a.student_id,(SELECT CASE
    WHEN RAND(CHECKSUM(NEWID())) < 0.99 THEN 1
    ELSE 0
END AS RandomValue) from Module_meetings mm
inner join Meetings as m
on m.meeting_id=mm.meeting_id
inner join study_courses as c
on c.activity_id=m.activity_id
inner join studies s
on s.studies_id=c.studies_id
inner join Student_activities as a
on a.activity_id=s.activity_id
where module_number=7 and cast(m.meeting_id as nvarchar)+'
'+cast(a.student_id as nvarchar) not in (select cast(meeting_id as
nvarchar)+' '+cast(student_id as nvarchar) from attendance)
order by 1,2
```

```
DECLARE @StartDate AS datetime;
DECLARE @EndDate AS datetime;
SELECT @StartDate = '2023-01-01 12:00:00', -- Date Format - MM/DD/YYYY
       @EndDate   = '2024-11-30 14:00:00';

WITH CombinedData AS (
    SELECT sa.student_id AS a,sa.activity_id as e,c.price_for_course as f,
           MIN(start_time) - RAND(sa.student_id) * 5 AS b
```

```
    FROM Student_activities AS sa
    INNER JOIN Courses AS c ON c.activity_id = sa.activity_id
    INNER JOIN Course_modules AS cm ON cm.course_id = c.course_id
    INNER JOIN Meetings AS m ON m.activity_id = cm.activity_id
    INNER JOIN Stationary AS s ON s.meeting_id = m.meeting_id
    GROUP BY sa.student_id,sa.activity_id,c.price_for_course

    UNION ALL

    SELECT sa.student_id AS a, sa.activity_id as e,c.price_for_course as
f,
           MIN(start_time) - RAND(sa.student_id) * 5 AS b
    FROM Student_activities AS sa
    INNER JOIN Courses AS c ON c.activity_id = sa.activity_id
    INNER JOIN Course_modules AS cm ON cm.course_id = c.course_id
    INNER JOIN Meetings AS m ON m.activity_id = cm.activity_id
    INNER JOIN Online_synchronous AS s ON s.meeting_id = m.meeting_id
    GROUP BY sa.student_id,sa.activity_id,c.price_for_course

    UNION ALL

    SELECT sa.student_id AS a, sa.activity_id as e,c.price_for_course as
f,
           (SELECT DATEADD(DAY, RAND(sa.student_id * 99999) * (1 +
DATEDIFF(DAY, @StartDate, @EndDate)), @StartDate)) AS b
    FROM Student_activities AS sa
    INNER JOIN Courses AS c ON c.activity_id = sa.activity_id
    INNER JOIN Course_modules AS cm ON cm.course_id = c.course_id
    INNER JOIN Meetings AS m ON m.activity_id = cm.activity_id
    INNER JOIN Online_asynchronous AS s ON s.meeting_id = m.meeting_id
    GROUP BY sa.student_id,sa.activity_id,c.price_for_course
)
insert Orders (student_id,order_date)
SELECT
       a,
       MIN(b) AS MinB
FROM CombinedData
GROUP BY a,e,f;
```

```
DECLARE @StartDate AS datetime;
```

```
DECLARE @EndDate AS datetime;
SELECT @StartDate = '2023-01-01 12:00:00',
       @EndDate   = '2024-11-30 14:00:00';

WITH CombinedData AS (
    SELECT sa.student_id AS a,sa.activity_id as e,c.price_for_course as f,
           MIN(start_time) - RAND(sa.student_id) * 5 AS b
    FROM Student_activities AS sa
    INNER JOIN Courses AS c ON c.activity_id = sa.activity_id
    INNER JOIN Course_modules AS cm ON cm.course_id = c.course_id
    INNER JOIN Meetings AS m ON m.activity_id = cm.activity_id
    INNER JOIN Stationary AS s ON s.meeting_id = m.meeting_id
    GROUP BY sa.student_id,sa.activity_id,c.price_for_course

    UNION ALL

    SELECT sa.student_id AS a, sa.activity_id as e,c.price_for_course as
f,
           MIN(start_time) - RAND(sa.student_id) * 5 AS b
    FROM Student_activities AS sa
    INNER JOIN Courses AS c ON c.activity_id = sa.activity_id
    INNER JOIN Course_modules AS cm ON cm.course_id = c.course_id
    INNER JOIN Meetings AS m ON m.activity_id = cm.activity_id
    INNER JOIN Online_synchronous AS s ON s.meeting_id = m.meeting_id
    GROUP BY sa.student_id,sa.activity_id,c.price_for_course

    UNION ALL

    SELECT sa.student_id AS a, sa.activity_id as e,c.price_for_course as
f,
           (SELECT DATEADD(DAY, RAND(sa.student_id * 99999) * (1 +
DATEDIFF(DAY, @StartDate, @EndDate)), @StartDate)) AS b
    FROM Student_activities AS sa
    INNER JOIN Courses AS c ON c.activity_id = sa.activity_id
    INNER JOIN Course_modules AS cm ON cm.course_id = c.course_id
    INNER JOIN Meetings AS m ON m.activity_id = cm.activity_id
    INNER JOIN Online_asynchronous AS s ON s.meeting_id = m.meeting_id
    GROUP BY sa.student_id,sa.activity_id,c.price_for_course
)
insert Order_details (order_id,activity_id,description,price,payment_date)
SELECT ROW_NUMBER() OVER (ORDER BY a, MIN(b)) AS RowIndex, -- Numeracja
wierszy
       e,'FULL PAYMENT',f,
```

```
       MIN(b) AS MinB
FROM CombinedData
GROUP BY a,e,f;
```