Projekt - Bazy danych - dokumentacja
System zarządzania kursami i szkoleniami

Podstawy baz danych 2024/25

Jakub Zając, Stanisław Strojniak, Kamil Życzkowski

Funkcje i użytkownicy systemu

Użytkownicy bazy danych

1. **Administrator** - zarządza całym systemem, odpowiada za jego poprawne działanie. Administrator ma dostęp do wszystkich, pozostałych funkcji innych użytkowników systemu.
2. **Dyrektor szkoły** - zarządza i nadzoruje funkcjonowanie całego systemu edukacyjnego. Dyrektor szkoły ma dostęp do funkcji administratora.
3. **Pracownik sekretariatu** - dodaje webinary i kursy, generuje raporty finansowe i statystyczne, tworzy harmonogram zajęć
4. **Uczestnik** - uczestniczy w zajęciach, może korzystać z materiałów edukacyjnych (np. nagrań)
5. **Tłumacz** - tłumaczy na żywo na język polski przypisane zajęcia prowadzone w językach obcych
6. **Wykładowca** - prowadzi zajęcia, koordynuje prowadzony przedmiot, dokumentuje frekwencję uczestników, wystawia zaliczenia z modułów
7. **Prowadzący praktyki** - koordynuje praktyki, odpowiada za ich realizację i przyznanie zaliczenia praktyk
8. **Niezalogowany użytkownik** - może przeglądać listę zajęć i utworzyć konto w systemie

Funkcje systemu

Zarządzanie użytkownikami

- **Dodawanie i usuwanie wszystkich kont użytkowników**: Administrator
- **Dodawanie i usuwanie kont wykładowcy i tłumacza**: Pracownik sekretariatu
- **Założenie konta użytkownika**: Niezalogowany użytkownik
- **Usunięcie konta**: Uczestnik
- **Edycja swoich danych osobowych**: Każdy użytkownik systemu

---

Zarządzanie uprawnieniami

- **Nadawanie uprawnień użytkownikom**: Administrator
- **Przydzielenie dostępu do webinaru bez uprzedniego uiszczenia opłaty**: Dyrektor szkoły
- **Przyznanie uczestnictwa do studium/kursu bez wpłaty zaliczki lub całej kwoty na 3 dni przed rozpoczęciem kursu**: Dyrektor szkoły
- **Przydzielenie tłumacza do zajęć**: Pracownik sekretariatu

---

Obsługa finansowa i płatności

- **Generowanie linku do płatności**: System
- **Informowanie użytkownika o statusie płatności**: System
- **Zapis uczestnika na zajęcia po zaksięgowaniu płatności**: System
- **Wysyłanie informacji przypominającej o zbliżającej się płatności**: System

---

Zakupy i koszyk

- **Dodawanie produktów do koszyka**: Uczestnik
- **Tworzenie zamówienia**: Uczestnik
- **Sprawdzanie dostępności miejsc na spotkanie studyjne dla osób niezapisanych na studia**: System
- **Zapisanie się na spotkanie studyjne jako osoba nieuczestnicząca w całym studium:** Uczestnik

---

Tworzenie i zarządzanie treściami

- **Dodawanie webinaru i kursu**: Pracownik sekretariatu
- **Dodawanie nowych studiów**: Dyrektor szkoły
- **Tworzenie harmonogramu zajęć**: Pracownik sekretariatu
- **Zarządzanie harmonogramem praktyk**: Pracownik sekretariatu, Prowadzący praktyki
- **Dodawanie i modyfikacja sylabusu zajęć (przed rozpoczęciem roku akademickiego)**: Wykładowca

---

Obsługa zajęć

- **Podgląd listy zajęć, na które użytkownik jest zapisany**: Uczestnik
- **Podgląd swojej frekwencji na zajęciach**: Uczestnik
- **Podgląd swoich ocen**: Uczestnik
- **Podgląd listy zajęć do tłumaczenia na żywo**: Tłumacz
- **Podgląd listy prowadzonych zajęć**: Wykładowca
- **Podgląd listy uczestników zajęć**: Wykładowca
- **Zapisanie obecności uczestników na zajęciach**: Wykładowca
- **Rejestracja obecności na praktykach**: Prowadzący praktyki
- **Podgląd listy uczestników praktyk**: Prowadzący praktyki

- **Połączenie się do zajęć w trybie tłumacza**: Tłumacz

Obsługa webinarów i nagrań

- **Archiwizacja nagrań zajęć na platformie do webinarów i udostępnienie ich uczestnikom na okres 30 dni**: System
- **Automatyczne zaliczenie zajęć po obejrzeniu nagrania**: System
- **Skasowanie przechowywanego webinaru**: Administrator

Raportowanie i dokumentacja

- **Generowanie raportów**: Pracownik sekretariatu, Dyrektor szkoły
- **Podgląd utworzonych raportów**: Pracownik sekretariatu, Dyrektor szkoły
- **Generowanie raportu o liście osób, których przyszłe zajęcia ze sobą kolidują**: Pracownik sekretariatu
- **Generowanie raportu o dyplomach do wysłania**: Pracownik sekretariatu
- **Generowanie dyplomów**: Pracownik sekretariatu, Dyrektor szkoły
- **Wysłanie informacji do użytkownika o wysłaniu dyplomu**: System

---

Oceny i zaliczenia

- **Wystawianie zaliczeń**: Wykładowca
- **Zaliczanie studiów/kursu na podstawie obecności**: System, Wykładowca, Dyrektor szkoły

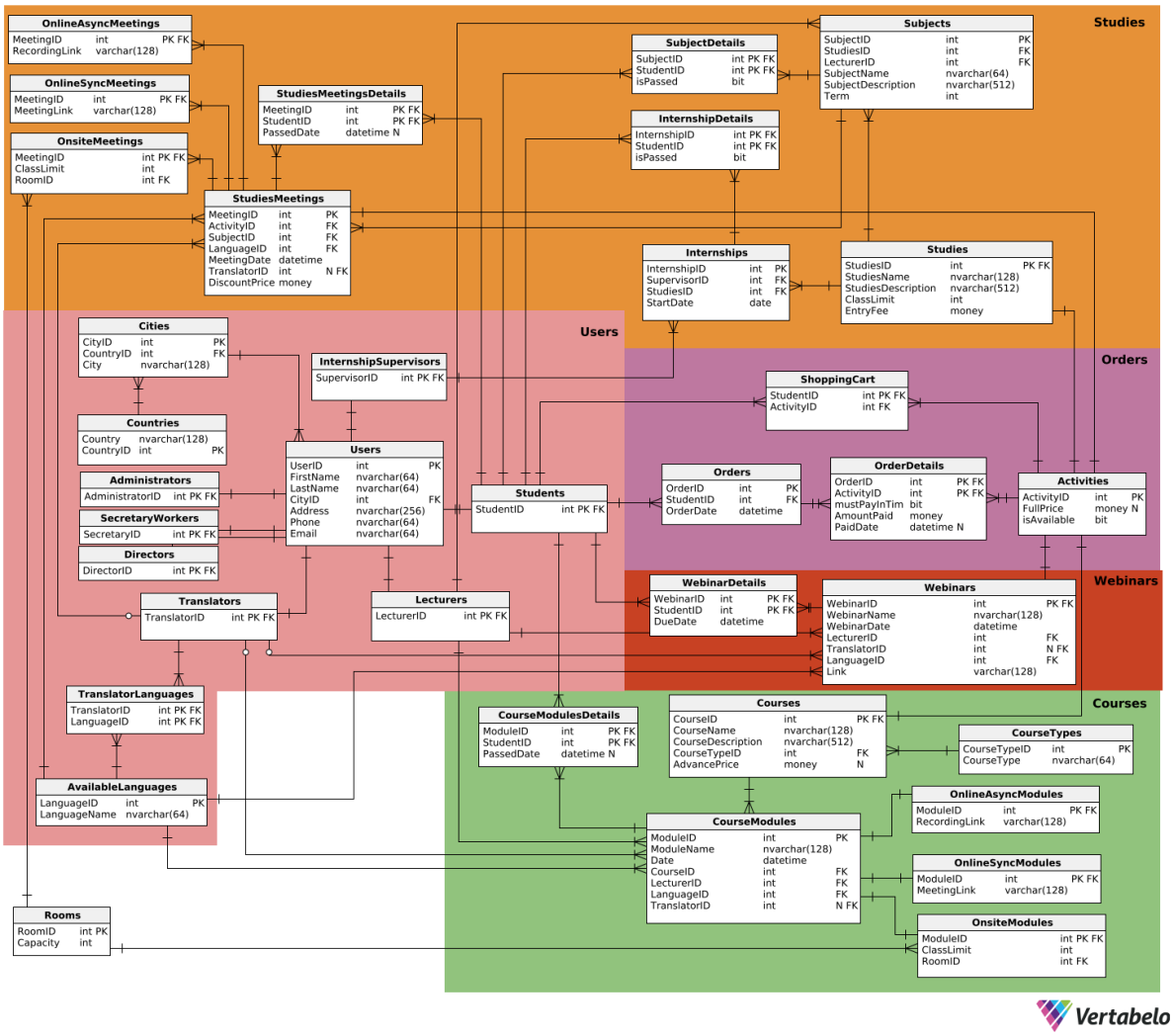
---

Dostęp do informacji

- **Przeglądanie listy kursów, webinarów i studiów**: Każdy użytkownik systemu
- **Przeglądanie sylabusa przedmiotu**: Każdy użytkownik systemu

---

Schemat bazy danych



Vertabelo

Opisy tabel

Użytkownicy

Users

Zawiera podstawowe informacje o wszystkich użytkownikach bazy danych.

- **UserID** int - klucz główny, identyfikator użytkownika
- **FirstName** nvarchar(64) - imię użytkownika
- **LastName** nvarchar(64) - nazwisko użytkownika
- **CityID** int - klucz obcy, identyfikator miasta, z którego pochodzi użytkownik
- **Address** nvarchar(256) - adres użytkownika zawierający nazwę ulicy, numer domu i kod pocztowy
- **Phone** nvarchar(64) - numer telefonu użytkownika
- **Email** nvarchar(64) unique - adres e-mail użytkownika
    - warunek: Email LIKE '%_@_%._%'

```
CREATE TABLE Users (
   UserID int  NOT NULL,
   FirstName nvarchar(64)  NOT NULL,
   LastName nvarchar(64)  NOT NULL,
   CityID int  NOT NULL,
   Address nvarchar(256)  NOT NULL,
   Phone nvarchar(64)  NOT NULL,
   Email nvarchar(64)  NOT NULL,
   CONSTRAINT uniqueEmail UNIQUE (Email),
   CONSTRAINT emailFormat CHECK ([email] LIKE '%_@_%._%'),
   CONSTRAINT Users_pk PRIMARY KEY  (UserID)
);

ALTER TABLE Users ADD CONSTRAINT Users_Cities
     FOREIGN KEY (CityID)
     REFERENCES Cities (CityID);
```

Students

Przechowuje wszystkich użytkowników będących studentami.

- **StudentID** int - klucz główny, identyfikator studenta

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

```
ALTER TABLE Students ADD CONSTRAINT Students_Users
      FOREIGN KEY (StudentID)
      REFERENCES Users (UserID);
```

Lecturers

Przechowuje wszystkich użytkowników będących wykładowcami.

- **LecturerID** int - klucz główny, klucz obcy, identyfikator wykładowcy

```
CREATE TABLE Lecturers (
   LecturerID int  NOT NULL,
   CONSTRAINT Lecturers_pk PRIMARY KEY  (LecturerID)
);

ALTER TABLE Lecturers ADD CONSTRAINT Lecturers_Users
      FOREIGN KEY (LecturerID)
      REFERENCES Users (UserID);
```

Administrators

Przechowuje wszystkich użytkowników będących administratorami.

- **AdministratorID** int - klucz główny, klucz obcy, identyfikator administratora

```
CREATE TABLE Administrators (
   AdministratorID int  NOT NULL,
   CONSTRAINT Administrators_pk PRIMARY KEY  (AdministratorID)
);

ALTER TABLE Administrators ADD CONSTRAINT Administrators_Users
      FOREIGN KEY (AdministratorID)
      REFERENCES Users (UserID);
```

SecretaryWorkers

Przechowuje wszystkich użytkowników będących pracownikami sekretariatu.

- **SecretaryID** int - klucz główny, klucz obcy, identyfikator pracownika sekretariatu

```
CREATE TABLE SecretaryWorkers (
   SecretaryID int  NOT NULL,
   CONSTRAINT SecretaryWorkers_pk PRIMARY KEY  (SecretaryID)
```

```
);

ALTER TABLE SecretaryWorkers ADD CONSTRAINT SecretaryWorkers_Users
      FOREIGN KEY (SecretaryID)
      REFERENCES Users (UserID);
```

Directors

Przechowuje wszystkich użytkowników będących dyrektorami.

- **DirectorID** int - klucz główny, klucz obcy, identyfikator dyrektora

```
CREATE TABLE Directors (
   DirectorID int  NOT NULL,
   CONSTRAINT Directors_pk PRIMARY KEY  (DirectorID)
);

ALTER TABLE Directors ADD CONSTRAINT Directors_Users
      FOREIGN KEY (DirectorID)
      REFERENCES Users (UserID);
```

InternshipSupervisors

Przechowuje wszystkich użytkowników będących prowadzącymi praktyki.

- **SupervisorID** int - klucz główny, klucz obcy, identyfikator prowadzącego praktyki

```
CREATE TABLE InternshipSupervisors (
   SupervisorID int  NOT NULL,
   CONSTRAINT InternshipSupervisors_pk PRIMARY KEY  (SupervisorID)
);

ALTER TABLE InternshipSupervisors ADD CONSTRAINT
InternshipSupervisors_Users
      FOREIGN KEY (SupervisorID)
      REFERENCES Users (UserID);
```

Translators

Przechowuje wszystkich użytkowników będących tłumaczami.

- **TranslatorID** int - klucz główny, klucz obcy, identyfikator tłumacza

```
CREATE TABLE Translators (
```

```
   TranslatorID int  NOT NULL,
   CONSTRAINT Translators_pk PRIMARY KEY  (TranslatorID)
);

ALTER TABLE Translators ADD CONSTRAINT Translators_Users
      FOREIGN KEY (TranslatorID)
      REFERENCES Users (UserID);
```

TranslatorLanguages

Zawiera informacje o tłumaczach i językach, które potrafią tłumaczyć.

- **TranslatorID** int - klucz główny, klucz obcy, identyfikator tłumacza
- **LanguageID** int - klucz główny, klucz obcy, identyfikator języka

```
CREATE TABLE TranslatorLanguages (
   TranslatorID int  NOT NULL,
   LanguageID int  NOT NULL,
   CONSTRAINT TranslatorLanguages_pk PRIMARY KEY
(TranslatorID,LanguageID)
);

ALTER TABLE TranslatorLanguages ADD CONSTRAINT
TranslatorLanguages_AvailableLanguages
      FOREIGN KEY (LanguageID)
      REFERENCES AvailableLanguages (LanguageID);

ALTER TABLE TranslatorLanguages ADD CONSTRAINT
TranslatorLanguages_Translators
      FOREIGN KEY (TranslatorID)
      REFERENCES Translators (TranslatorID);
```

AvailableLanguages

Zawiera informacje o tłumaczonych językach.

- **LanguageID** int - klucz główny, identyfikator języka
- **LanguageName** nvarchar(64) - nazwa języka

```
CREATE TABLE AvailableLanguages (
   LanguageID int  NOT NULL,
   LanguageName nvarchar(64)  NOT NULL,
   CONSTRAINT AvailableLanguages_pk PRIMARY KEY  (LanguageID)
);
```

Countries

Zawiera informacje o państwach.

- **CountryID** int - klucz główny, identyfikator kraju
- **Country** nvarchar(128) - nazwa kraju

```
CREATE TABLE Countries (
   CountryID int  NOT NULL,
   Country nvarchar(128)  NOT NULL,
   CONSTRAINT Countries_pk PRIMARY KEY  (CountryID)
);
```

Cities

Zawiera informacje o miastach.

- **CityID** int - klucz główny, identyfikator miasta
- **CountryID** int - klucz obcy, identyfikator kraju
- **City** nvarchar(128) - nazwa miasta, z którego pochodzi użytkownik

```
CREATE TABLE Cities (
   CityID int  NOT NULL,
   CountryID int  NOT NULL,
   City nvarchar(128)  NOT NULL,
   CONSTRAINT Cities_pk PRIMARY KEY  (CityID,CountryID)
);

ALTER TABLE Cities ADD CONSTRAINT Cities_Countries
      FOREIGN KEY (CountryID)
      REFERENCES Countries (CountryID);
```

Studia

Studies

Zawiera podstawowe informacje o studiach.

- **StudiesID** int - klucz główny, klucz obcy, identyfikator studiów
- **StudiesName** nvarchar(128) - nazwa studiów
- **StudiesDescription** nvarchar(512) - opis studiów
- **ClassLimit** int - limit miejsc na studiach
    - warunek: ClassLimit >= 0
- **EntryFee** money - wpisowe na studia
    - warunek: EntryFee >= 0

```
CREATE TABLE Studies (
   StudiesID int  NOT NULL,
   StudiesName nvarchar(128)  NOT NULL,
   StudiesDescription nvarchar(512)  NOT NULL,
   ClassLimit int  NOT NULL CHECK (ClassLimit >= 0),
   EntryFee money  NOT NULL CHECK (EntryFee >= 0),
   CONSTRAINT Studies_pk PRIMARY KEY  (StudiesID)
);

ALTER TABLE Studies ADD CONSTRAINT Studies_Activities
      FOREIGN KEY (StudiesID)
      REFERENCES Activities (ActivityID);
```

Subjects

Zawiera podstawowe informacje o przedmiotach na danych studiach.

- **SubjectID** int - klucz główny, identyfikator przedmiotu
- **StudiesID** int - klucz obcy, identyfikator studiów
- **LecturerID** int - klucz obcy, identyfikator prowadzącego zajęcia
- **SubjectName** nvarchar(64) - nazwa przedmiotu
- **SubjectDescription** nvarchar(512) - opis przedmiotu

```
CREATE TABLE Subjects (
   SubjectID int  NOT NULL,
   StudiesID int  NOT NULL,
   LecturerID int  NOT NULL,
   SubjectName nvarchar(64)  NOT
NULL,
   SubjectDescription
nvarchar(512)  NOT NULL,
   CONSTRAINT Subjects_pk
PRIMARY KEY  (SubjectID)
);

ALTER TABLE Subjects ADD
CONSTRAINT Subjects_Lecturers
      FOREIGN KEY (LecturerID)
      REFERENCES Lecturers
(LecturerID);

ALTER TABLE Subjects ADD
```

```
CONSTRAINT Subjects_Studies
      FOREIGN KEY (StudiesID)
      REFERENCES Studies
(StudiesID);
```

SubjectDetails

Zawiera szczegółowe informacje o przedmiotach na danych studiach.

- **SubjectID** int - klucz główny, klucz obcy, identyfikator przedmiotu
- **StudiesID** int - klucz główny, klucz obcy, identyfikator studiów
- **isPassed** bit - informacja o tym, czy student zdał przedmiot
    - wartość domyślna: 0 (jeszcze się nie odbyły)

```
CREATE TABLE SubjectDetails (
   SubjectID int  NOT NULL,
   StudentID int  NOT NULL,
   isPassed bit  NOT NULL DEFAULT 0,
   CONSTRAINT SubjectDetails_pk PRIMARY KEY  (SubjectID,StudentID)
);

ALTER TABLE SubjectDetails ADD CONSTRAINT SubjectDetails_Students
      FOREIGN KEY (StudentID)
      REFERENCES Students (StudentID);

ALTER TABLE SubjectDetails ADD CONSTRAINT SubjectDetails_Subjects
      FOREIGN KEY (SubjectID)
      REFERENCES Subjects (SubjectID);
```

StudiesMeetings

Zawiera podstawowe informacje o zajęciach na studiach.

- **MeetingID** int - klucz główny, identyfikator zajęć
- **ActivityID** int - klucz obcy, identyfikator aktywności
- **SubjectID** int - klucz obcy, identyfikator przedmiotu
- **LanguageID** int - klucz obcy, identyfikator języka, w którym będą odbywały się zajęcia
- **MeetingDate** datetime - data odbycia się zajęć
    - warunek: MeetingDate >= ‘2000-01-01’
- **TranslatorID** int - klucz obcy, identyfikator tłumacza
    - null jeśli nie jest potrzebne tłumaczenie
- **DiscountPrice** money - cena po zniżce dla uczestników studiów
    - warunek: DiscountPrice >= 0

```
CREATE TABLE StudiesMeetings (
   MeetingID int  NOT NULL,
   ActivityID int  NOT NULL,
   SubjectID int  NOT NULL,
   LanguageID int  NOT NULL,
   MeetingDate datetime  NOT NULL CHECK (MeetingDate >=
'2000-01-01'),
   TranslatorID int  NULL,
   DiscountPrice money  NULL DEFAULT NULL CHECK (DiscountPrice >=
0),
   CONSTRAINT StudiesMeetings_pk PRIMARY KEY  (MeetingID)
);

ALTER TABLE StudiesMeetings ADD CONSTRAINT
StudiesMeetings_Activities
      FOREIGN KEY (ActivityID)
      REFERENCES Activities (ActivityID);

ALTER TABLE StudiesMeetings ADD CONSTRAINT
StudiesMeetings_AvailableLanguages
      FOREIGN KEY (LanguageID)
      REFERENCES AvailableLanguages (LanguageID);

ALTER TABLE StudiesMeetings ADD CONSTRAINT
StudiesMeetings_Subjects
      FOREIGN KEY (SubjectID)
      REFERENCES Subjects (SubjectID);

ALTER TABLE StudiesMeetings ADD CONSTRAINT
StudiesMeetings_Translators
      FOREIGN KEY (TranslatorID)
      REFERENCES Translators (TranslatorID);
```

StudiesMeetingsDetails

Zawiera szczegółowe informacje o zajęciach na studiach.

- **MeetingID** int - klucz główny, klucz obcy, identyfikator zajęć
- **StudentID** int - klucz główny, klucz obcy, identyfikator studenta
- **PassedDate** datetime - data zaliczenia spotkania
    - null jeśli użytkownik jeszcze nie zaliczył zajęć
    - warunek: PassedDate >= ‘2000-01-01’

```
CREATE TABLE StudiesMeetingsDetails (
   MeetingID int  NOT NULL,
   StudentID int  NOT NULL,
   PassedDate datetime  NULL CHECK (PassedDate >= '2000-01-01'),
   CONSTRAINT StudiesMeetingsDetails_pk PRIMARY KEY
(MeetingID,StudentID)
);

ALTER TABLE StudiesMeetingsDetails ADD CONSTRAINT
StudiesMeetingsDetails_Students
      FOREIGN KEY (StudentID)
      REFERENCES Students (StudentID);

ALTER TABLE StudiesMeetingsDetails ADD CONSTRAINT
StudiesMeetingsDetails_StudiesMeetings
      FOREIGN KEY (MeetingID)
      REFERENCES StudiesMeetings (MeetingID);
```

OnlineAsyncMeetings

Zawiera szczegółowe informacje o zdalnych, asynchronicznie prowadzonych zajęciach.

- **MeetingID** int - klucz główny, klucz obcy, identyfikator zajęć
- **RecordingLink** varchar(128) - link do nagrania spotkania

```
CREATE TABLE OnlineAsyncMeetings (
   MeetingID int  NOT NULL,
   RecordingLink varchar(128)  NOT NULL,
   CONSTRAINT OnlineAsyncMeetings_pk PRIMARY KEY  (MeetingID)
);

ALTER TABLE OnlineAsyncMeetings ADD CONSTRAINT
OnlineAsyncMeeting_StudiesMeetings
      FOREIGN KEY (MeetingID)
      REFERENCES StudiesMeetings (MeetingID);
```

OnlineSyncMeetings

Zawiera szczegółowe informacje o zdalnych, synchronicznie prowadzonych zajęciach.

- **MeetingID** int - klucz główny, klucz obcy, identyfikator zajęć
- **MeetingLink** varchar(128) - link do spotkania na żywo

```
CREATE TABLE OnlineSyncMeetings (
   MeetingID int  NOT NULL,
   MeetingLink varchar(128)  NOT NULL,
   CONSTRAINT OnlineSyncMeetings_pk PRIMARY KEY  (MeetingID)
);

ALTER TABLE OnlineSyncMeetings ADD CONSTRAINT
OnlineSyncMeeting_StudiesMeetings
      FOREIGN KEY (MeetingID)
      REFERENCES StudiesMeetings (MeetingID);
```

OnsiteMeetings

Zawiera szczegółowe informacje o stacjonarnie prowadzonych zajęciach.

- **MeetingID** int - klucz główny, klucz obcy, identyfikator modułu
- **ClassLimit** int - limit osób, które mogą uczestniczyć w zajęciach
    - warunek: ClassLimit >= 0
- **RoomID** int - klucz obcy, identyfikator pomieszczenia, w którym odbędą się zajęcia

```
CREATE TABLE OnsiteMeetings (
   MeetingID int  NOT NULL,
   ClassLimit int  NOT NULL CHECK (ClassLimit >= 0),
   RoomID int  NOT NULL,
   CONSTRAINT OnsiteMeetings_pk PRIMARY KEY  (MeetingID)
);

ALTER TABLE OnsiteMeetings ADD CONSTRAINT StationaryMeeting_Rooms
      FOREIGN KEY (RoomID)
      REFERENCES Rooms (RoomID);

ALTER TABLE OnsiteMeetings ADD CONSTRAINT
StationaryMeeting_StudiesMeetings
      FOREIGN KEY (MeetingID)
      REFERENCES StudiesMeetings (MeetingID);
```

Internships

Zawiera podstawowe informacje o praktykach.

- **InternshipID** int - klucz główny, identyfikator praktyk
- **SupervisorID** int - klucz obcy, identyfikator prowadzącego praktyki
- **StudiesID** int - klucz obcy, identyfikator studiów
- **StartDate** date - data rozpoczęcia praktyk

- warunek: StartDate >= ‘2000-01-01’

```
CREATE TABLE Internships (
   InternshipID int  NOT NULL,
   SupervisorID int  NOT NULL,
   StudiesID int  NOT NULL,
   StartDate date  NOT NULL CHECK (StartDate >= '2000-01-01'),
   CONSTRAINT Internships_pk PRIMARY KEY  (InternshipID)
);

ALTER TABLE Internships ADD CONSTRAINT
Internships_InternshipSupervisors
      FOREIGN KEY (SupervisorID)
      REFERENCES InternshipSupervisors (SupervisorID);

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

InternshipDetails

Zawiera szczegółowe informacje o praktykach.

- **InternshipID** int - klucz główny, klucz obcy, identyfikator praktyk
- **StudentID** int - klucz główny, klucz obcy, identyfikator studenta
- **isPassed** bit - informacja o tym, czy student zdał praktyki
  - wartość domyślna: 0 (jeszcze się nie odbyły)

```
CREATE TABLE InternshipDetails (
   InternshipID int  NOT NULL,
   StudentID int  NOT NULL,
   isPassed bit  NOT NULL DEFAULT 0,
   CONSTRAINT InternshipDetails_pk PRIMARY KEY
(InternshipID,StudentID)
);

ALTER TABLE InternshipDetails ADD CONSTRAINT
InternshipDetails_Internships
      FOREIGN KEY (InternshipID)
      REFERENCES Internships (InternshipID);

ALTER TABLE InternshipDetails ADD CONSTRAINT
InternshipDetails_Students
```

```
      FOREIGN KEY (StudentID)
      REFERENCES Students (StudentID);
```

Webinary

Webinars

Zawiera podstawowe informacje o webinarach.

- **WebinarID** int - klucz główny, klucz obcy, identyfikator webinaru
- **WebinarName** nvarchar(128) - nazwa webinaru
- **WebinarDate** datetime - data odbycia webinaru
    - warunek: WebinarDate >= ‘2000-01-01’
- **LecturerID** int - klucz obcy, identyfikator prowadzącego webinar
- **TranslatorID** int - klucz obcy, identyfikator tłumacza
    - null jeżeli nie trzeba tłumaczyć webinaru
- **LanguageID** int - klucz obcy, identyfikator języka, w którym będzie odbywał się webinar
- **Link** varchar(128) - link do webinaru

```
CREATE TABLE Webinars (
   WebinarID int  NOT NULL,
   WebinarName nvarchar(128)  NOT NULL,
   WebinarDate datetime  NOT NULL CHECK (WebinarDate >=
'2000-01-01'),
   LanguageID int  NOT NULL,
   LecturerID int  NOT NULL,
   TranslatorID int  NULL,
   Link varchar(128)  NOT NULL,
   CONSTRAINT Webinars_pk PRIMARY KEY  (WebinarID)
);

ALTER TABLE Webinars ADD CONSTRAINT Webinars_Activities
      FOREIGN KEY (WebinarID)
      REFERENCES Activities (ActivityID);

ALTER TABLE Webinars ADD CONSTRAINT Webinars_AvailableLanguages
      FOREIGN KEY (LanguageID)
      REFERENCES AvailableLanguages (LanguageID);

ALTER TABLE Webinars ADD CONSTRAINT Webinars_Lecturers
      FOREIGN KEY (LecturerID)
      REFERENCES Lecturers (LecturerID);
```

```
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Translators
      FOREIGN KEY (TranslatorID)
      REFERENCES Translators (TranslatorID);
```

WebinarDetails

Zawiera szczegółowe informacje na temat uczestnictwa w webinarach.

- **WebinarID** int - klucz główny, klucz obcy, identyfikator webinaru
- **StudentID** int - klucz główny, klucz obcy, identyfikator użytkownika mającego dostęp do webinaru
- **DueDate** datetime - data, do której użytkownik będzie miał dostęp do nagrania webinaru
    - warunek: DueDate >= ‘2000-01-01’

```
CREATE TABLE WebinarDetails (
   WebinarID int  NOT NULL,
   StudentID int  NOT NULL,
   DueDate datetime  NOT NULL CHECK (DueDate >= '2000-01-01'),
   CONSTRAINT WebinarDetails_pk PRIMARY KEY  (WebinarID,StudentID)
);

ALTER TABLE WebinarDetails ADD CONSTRAINT WebinarDetails_Students
      FOREIGN KEY (StudentID)
      REFERENCES Students (StudentID);

ALTER TABLE WebinarDetails ADD CONSTRAINT WebinarDetails_Webinars
      FOREIGN KEY (WebinarID)
      REFERENCES Webinars (WebinarID);
```

Kursy

Courses

Zawiera podstawowe informacje o kursach.

- **CourseID** int - klucz główny, klucz obcy, identyfikator kursu
- **CourseName** nvarchar(128) - nazwa kursu
- **CourseDescription** nvarchar(512) - opis kursu
- **CourseTypeID** int - klucz obcy, identyfikator typu kursu
- **AdvancePrice** money - kwota wymaganej zaliczki przy zakupie kursu
    - null jeśli nie jest wymagana zaliczka
    - warunek: AdvancePrice>=0

```
CREATE TABLE Courses (
   CourseID int  NOT NULL,
```

```
   CourseName nvarchar(128)  NOT NULL,
   CourseDescription nvarchar(512)  NOT NULL,
   CourseTypeID int  NOT NULL,
   AdvancePrice money  NULL CHECK (AdvancePrice >= 0),
   CONSTRAINT Courses_pk PRIMARY KEY  (CourseID)
);

ALTER TABLE Courses ADD CONSTRAINT Courses_Activities
      FOREIGN KEY (CourseID)
      REFERENCES Activities (ActivityID);

ALTER TABLE Courses ADD CONSTRAINT Courses_CourseTypes
      FOREIGN KEY (CourseTypeID)
      REFERENCES CourseTypes (CourseTypeID);
```

CourseTypes

Przechowuje dostępne typy kursów.

- **CourseTypeID** int - klucz główny, identyfikator typu kursu
- **CourseType** nvarchar(64) - nazwa typu kursu

```
CREATE TABLE CourseTypes (
   CourseTypeID int  NOT NULL,
   CourseType nvarchar(64)  NOT NULL,
   CONSTRAINT CourseTypes_pk PRIMARY KEY  (CourseTypeID)
);
```

CourseModules

Zawiera podstawowe informacje o modułach kursów.

- **ModuleID** int - klucz główny, identyfikator modułu
- **ModuleName** nvarchar(128) - nazwa modułu
- **Date** datetime - data odbycia się modułu
    - warunek: Date >= ‘2000-01-01’
- **CourseID** int - klucz obcy, identyfikator kursu, w ramach którego odbywa się moduł
- **LecturerID** int - klucz obcy, identyfikator prowadzącego moduł
- **LanguageID** int - klucz obcy, identyfikator języka, w którym będzie odbywał się moduł
- **TranslatorID** int nullable - klucz obcy, identyfikator tłumacza
    - null jeśli nie jest potrzebne tłumaczenie

```
CREATE TABLE CourseModules (
```

```
   ModuleID int  NOT NULL,
   ModuleName nvarchar(128)  NOT NULL,
   Date datetime  NOT NULL CHECK (Date >= '2000-01-01'),
   CourseID int  NOT NULL,
   LecturerID int  NOT NULL,
   LanguageID int  NOT NULL,
   TranslatorID int  NULL,
   CONSTRAINT CourseModules_pk PRIMARY KEY  (ModuleID)
);

ALTER TABLE CourseModules ADD CONSTRAINT
CourseModules_AvailableLanguages
      FOREIGN KEY (LanguageID)
      REFERENCES AvailableLanguages (LanguageID);

ALTER TABLE CourseModules ADD CONSTRAINT CourseModules_Courses
      FOREIGN KEY (CourseID)
      REFERENCES Courses (CourseID);

ALTER TABLE CourseModules ADD CONSTRAINT CourseModules_Lecturers
      FOREIGN KEY (LecturerID)
      REFERENCES Lecturers (LecturerID);

ALTER TABLE CourseModules ADD CONSTRAINT CourseModules_Translators
      FOREIGN KEY (TranslatorID)
      REFERENCES Translators (TranslatorID);
```

CourseModuleDetails

Zawiera szczegółowe informacje na temat uczestnictwa w modułach.

- **ModuleID** int - klucz główny, klucz obcy, identyfikator modułu
- **StudentID** int - klucz główny, klucz obcy, użytkownik, który ma dostęp do modułu
- **PassedDate** datetime - data zaliczenia modułu
    - null jeśli użytkownik jeszcze nie zaliczył modułu
    - warunek: PassedDate >= ‘2000-01-01’

```
CREATE TABLE CourseModulesDetails (
   ModuleID int  NOT NULL,
   StudentID int  NOT NULL,
   PassedDate datetime  NULL CHECK (PassedDate >= '2000-01-01'),
   CONSTRAINT CourseModulesDetails_pk PRIMARY KEY
```

```
(ModuleID,StudentID)
);

ALTER TABLE CourseModulesDetails ADD CONSTRAINT
CourseModulesDetails_CourseModules
      FOREIGN KEY (ModuleID)
      REFERENCES CourseModules (ModuleID);

ALTER TABLE CourseModulesDetails ADD CONSTRAINT
CourseModulesDetails_Students
      FOREIGN KEY (StudentID)
      REFERENCES Students (StudentID);
```

OnlineAsyncModules

Zawiera szczegółowe informacje o zdalnych, asynchronicznie prowadzonych modułach.

- **ModuleID** int - klucz główny, klucz obcy, identyfikator modułu
- **RecordingLink** varchar(128) - link do nagrania spotkania

```
CREATE TABLE OnlineAsyncModule (
   ModuleID int  NOT NULL,
   RecordingLink varchar(128)  NOT NULL,
   CONSTRAINT OnlineAsyncModule_pk PRIMARY KEY  (ModuleID)
);

ALTER TABLE OnlineAsyncModules ADD CONSTRAINT
OnlineAsyncModule_CourseModules
      FOREIGN KEY (ModuleID)
      REFERENCES CourseModules (ModuleID);
```

OnlineSyncModules

Zawiera szczegółowe informacje o zdalnych, synchronicznie prowadzonych modułach.

- **ModuleID** int - klucz główny, klucz obcy, identyfikator modułu
- **MeetingLink** varchar(128) - link do spotkania na żywo

```
CREATE TABLE OnlineSyncModule (
   ModuleID int  NOT NULL,
   MeetingLink varchar(128)  NOT NULL,
   CONSTRAINT OnlineSyncModule_pk PRIMARY KEY  (ModuleID)
);
```

```
ALTER TABLE OnlineSyncModules ADD CONSTRAINT
OnlineSyncModule_CourseModules
      FOREIGN KEY (ModuleID)
      REFERENCES CourseModules (ModuleID);
```

OnsiteModules

Zawiera szczegółowe informacje o stacjonarnie prowadzonych modułach.

- **ModuleID** int - klucz klucz obcy, główny, identyfikator modułu
- **ClassLimit** int - limit osób, które mogą uczestniczyć w module
    - warunek: ClassLimit >= 0
- **RoomID** int - klucz obcy, identyfikator pomieszczenia, w którym odbędzie się spotkanie

```
CREATE TABLE OnsiteModules (
   ModuleID int  NOT NULL,
   ClassLimit int  NOT NULL CHECK (ClassLimit >= 0),
   RoomID int  NOT NULL,
   CONSTRAINT OnsiteModules_pk PRIMARY KEY  (ModuleID)
);

ALTER TABLE OnsiteModules ADD CONSTRAINT
StationaryModule_CourseModules
      FOREIGN KEY (ModuleID)
      REFERENCES CourseModules (ModuleID);

ALTER TABLE OnsiteModules ADD CONSTRAINT StationaryModule_Rooms
      FOREIGN KEY (RoomID)
      REFERENCES Rooms (RoomID);
```

Zamówienia

Orders

Zawiera podstawowe informacje o zamówieniach studentów.

- **OrderID** int - klucz główny, identyfikator zamówienia
- **StudentID** int - klucz obcy, identyfikator studenta
- **OrderDate** datetime - data złożenia zamówienia
    - warunek: OrderDate >= ‘2000-01-01’

```
CREATE TABLE Orders (
   OrderID int  NOT NULL,
```

```
   StudentID int  NOT NULL,
   OrderDate datetime  NOT NULL CHECK (OrderDate >= '2000-01-01'),
   CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);

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

OrderDetails

Zawiera szczegółowe informacje o zamówieniach.

- **OrderID** int - klucz główny, klucz obcy, identyfikator zamówienia
- **ActivityID** int - klucz główny, klucz obcy, identyfikator aktywności
- **mustPayInTime** bit - informacja o tym, czy student musi opłacić zamówienie w terminie, aby uzyskać dostęp do aktywności
    - wartość domyślna: 1
- **PaidDate** datetime nullable - data opłacenia zamówienia
    - wartość domyślna: null
    - warunek: PaidDate >= ‘2000-01-01’
- **AmountPaid** money - kwota, która została opłacona
    - wartość domyślna: 0
    - warunek: AmountPaid >=0

```
CREATE TABLE OrderDetails (
   OrderID int  NOT NULL,
   ActivityID int  NOT NULL,
   mustPayInTime bit  NOT NULL DEFAULT 1,
   AmountPaid money  NOT NULL DEFAULT 0 CHECK (AmountPaid >= 0),
   PaidDate datetime  NULL DEFAULT NULL CHECK (PaidDate >=
'2000-01-01'),
   CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID,ActivityID)
);

ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Activities
      FOREIGN KEY (ActivityID)
      REFERENCES Activities (ActivityID);

ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
      FOREIGN KEY (OrderID)
      REFERENCES Orders (OrderID);
```

ShoppingCart

Zawiera informacje na temat aktywności dodanych do koszyków studentów.

- **StudentID** int - klucz główny, klucz obcy, identyfikator studenta
- **ActivityID** int - klucz obcy, identyfikator aktywności

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

ALTER TABLE ShoppingCart ADD CONSTRAINT ShoppingCart_Activities
      FOREIGN KEY (ActivityID)
      REFERENCES Activities (ActivityID);

ALTER TABLE ShoppingCart ADD CONSTRAINT ShoppingCart_Students
      FOREIGN KEY (StudentID)
      REFERENCES Students (StudentID);
```

Activities

Zawiera podstawowe informacje o aktywnościach.

- **ActivityID** int - klucz główny, identyfikator aktywności
- **FullPrice** money nullable - cena zakupu danej aktywności
    - warunek: FullPrice >= 0
- **isAvailable** bit - informacja o tym, czy dany produkt jest dostępny
    - wartość domyślna: 1

```
CREATE TABLE Activities (
   ActivityID int  NOT NULL,
   FullPrice money  NULL CHECK (FullPrice >= 0),
   isAvailable bit  NOT NULL DEFAULT 1,
   CONSTRAINT Activities_pk PRIMARY KEY  (ActivityID)
);
```

Pozostałe

Rooms

Przechowuje informacje na temat dostępnych pomieszczeń.

- **RoomID** int - klucz główny, identyfikator pomieszczenia

- **Capacity** int - liczba osób, która zmieści się w sali
    - warunek: Capacity >= 0

```
CREATE TABLE Rooms (
   RoomID int  NOT NULL,
   Capacity int  NOT NULL CHECK (Capacity >= 0),
   CONSTRAINT Rooms_pk PRIMARY KEY  (RoomID)
);
```

Sposób generowania danych

Dane zostały opracowane przy użyciu różnych narzędzi i metod. Do generowania większości danych wykorzystano model językowy ChatGPT, który tworzył losowe wartości, następnie ręcznie dodawane do bazy danych. W celu wygenerowania większej liczby losowych rekordów, na podstawie zdefiniowanych schematów danych, posłużono się również serwisem Mockaroo. W wybranych przypadkach użyto skryptów w Pythonie, wygenerowanych przy pomocy ChatGPT i zmodyfikowanych, aby dopasować dane do specyficznych wymagań projektu.

Widoki

Zestawienie przychodów dla każdego wydarzenia

```
CREATE VIEW FINANCIAL_REPORT AS
SELECT W.WebinarID AS 'ID', W.WebinarName AS 'Name', 'Webinar' AS
'Type',
FullPrice*(SELECT COUNT(*) FROM OrderDetails OD
WHERE OD.ActivityID=W.WebinarID) AS 'Total Income'
FROM Webinars W
INNER JOIN Activities A ON A.ActivityID=W.WebinarID
UNION
SELECT C.CourseID AS 'ID', C.CourseName AS 'Name', 'Course' AS
'Type',
FullPrice*(SELECT COUNT(*) FROM OrderDetails OD
WHERE OD.ActivityID=C.CourseID) AS 'Total Income'
FROM Courses C
INNER JOIN Activities A ON A.ActivityID=C.CourseID
UNION
SELECT S.StudiesID AS 'ID', S.StudiesName AS 'Name', 'Study' AS
'Type',
ISNULL((A.FullPrice+EntryFee)*(SELECT COUNT(*) FROM OrderDetails
```

```
OD
WHERE OD.ActivityID=S.StudiesID) +
(SELECT TOP 1 A2.FullPrice*(SELECT TOP 1 COUNT(*) FROM
OrderDetails OD2
     WHERE OD2.ActivityID=SM.ActivityID)
     FROM StudiesMeetings SM
     INNER JOIN Subjects Sub ON SM.SubjectID = Sub.SubjectID
     INNER JOIN Activities A2 ON A2.ActivityID = SM.ActivityID
     INNER JOIN OrderDetails OD ON OD.ActivityID = SM.ActivityID
     WHERE Sub.StudiesID = S.StudiesID), 0) AS 'Total Income'
FROM Studies S
INNER JOIN Activities A ON A.ActivityID=S.StudiesID
```

Autor: Jakub Zając

Zestawienie przychodów dla każdego kursu

```
CREATE VIEW COURSES_FINANCIAL_REPORT AS
SELECT C.CourseID AS 'ID', C.CourseName AS 'Course Name',
FullPrice*(SELECT COUNT(*) FROM OrderDetails OD
WHERE OD.ActivityID=C.CourseID) AS 'Total Income'
FROM Courses C
INNER JOIN Activities A ON A.ActivityID=C.CourseID
```

Autor: Jakub Zając

Zestawienie przychodów dla każdego studium

```
CREATE VIEW STUDIES_FINANCIAL_REPORT AS
SELECT S.StudiesID AS 'ID', S.StudiesName AS 'Name', 'Study' AS
'Type',
ISNULL((A.FullPrice+EntryFee)*(SELECT COUNT(*) FROM OrderDetails
OD
WHERE OD.ActivityID=S.StudiesID) +
(SELECT A2.FullPrice*(SELECT COUNT(*) FROM OrderDetails OD2 WHERE
OD2.ActivityID=SM.ActivityID)
     FROM StudiesMeetings SM
     INNER JOIN Subjects Sub ON SM.SubjectID = Sub.SubjectID
     INNER JOIN Activities A2 ON A2.ActivityID = SM.ActivityID
     INNER JOIN OrderDetails OD ON OD.ActivityID = SM.ActivityID
     WHERE Sub.StudiesID = S.StudiesID), 0) AS 'Total Income'
FROM Studies S
INNER JOIN Activities A ON A.ActivityID=S.StudiesID
```

Autor: Jakub Zając

Zestawienie przychodów dla każdego webinaru

```
CREATE VIEW WEBINARS_FINANCIAL_REPORT AS
SELECT W.WebinarID AS 'ID', W.WebinarName AS 'Webinar Name',
FullPrice*(SELECT COUNT(*) FROM OrderDetails OD
WHERE OD.ActivityID=W.WebinarID) AS 'Total Income'
FROM Webinars W
INNER JOIN Activities A ON A.ActivityID=W.WebinarID
```

Autor: Jakub Zając

Zestawienie osób zalegających z płatnościami

```
CREATE VIEW LIST_OF_DEBTORS AS
SELECT StudentID AS 'ID', FirstName, LastName, Address, City,
Country, Phone, Email,
ISNULL((SELECT SUM(FullPrice-AmountPaid) FROM Orders O2
INNER JOIN OrderDetails OD2 ON OD2.OrderID=O2.OrderID
INNER JOIN Activities A ON A.ActivityID=OD2.ActivityID
INNER JOIN Webinars W ON W.WebinarID=A.ActivityID
WHERE O2.StudentID=S.StudentID AND mustPayInTime=0), 0) +
ISNULL((SELECT SUM(FullPrice-AmountPaid) FROM Orders O2
INNER JOIN OrderDetails OD2 ON OD2.OrderID=O2.OrderID
INNER JOIN Activities A ON A.ActivityID=OD2.ActivityID
INNER JOIN Courses C ON C.CourseID=A.ActivityID
WHERE O2.StudentID=S.StudentID AND mustPayInTime=0), 0) +
ISNULL((SELECT SUM(FullPrice+EntryFee-AmountPaid) FROM Orders O2
INNER JOIN OrderDetails OD2 ON OD2.OrderID=O2.OrderID
INNER JOIN Activities A ON A.ActivityID=OD2.ActivityID
INNER JOIN Studies St ON St.StudiesID=A.ActivityID
WHERE O2.StudentID=S.StudentID AND mustPayInTime=0), 0) +
ISNULL((SELECT SUM(FullPrice-AmountPaid) FROM Orders O2
INNER JOIN OrderDetails OD2 ON OD2.OrderID=O2.OrderID
INNER JOIN Activities A ON A.ActivityID=OD2.ActivityID
INNER JOIN StudiesMeetings SM ON SM.ActivityID=A.ActivityID
WHERE O2.StudentID=S.StudentID AND mustPayInTime=0), 0) AS 'Total
debt'
FROM Students S
INNER JOIN Users U ON U.UserID=S.StudentID
INNER JOIN Cities Cit ON U.CityID=Cit.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE StudentID IN
(SELECT StudentID FROM Orders O
```

```
INNER JOIN OrderDetails OD ON OD.OrderID=O.OrderID
INNER JOIN Activities A ON A.ActivityID=OD.ActivityID
INNER JOIN Webinars W ON W.WebinarID=A.ActivityID
WHERE AmountPaid<FullPrice AND WebinarDate < GETDATE() AND
mustPayInTime=0
UNION
SELECT StudentID FROM Orders O
INNER JOIN OrderDetails OD ON OD.OrderID=O.OrderID
INNER JOIN Activities A ON A.ActivityID=OD.ActivityID
INNER JOIN Courses C ON C.CourseID=A.ActivityID
WHERE AmountPaid<FullPrice AND
(SELECT MIN(CM.Date) FROM CourseModules CM
WHERE C.CourseID=CM.CourseID) < GETDATE() AND mustPayInTime=0
UNION
SELECT StudentID FROM Orders O
INNER JOIN OrderDetails OD ON OD.OrderID=O.OrderID
INNER JOIN Activities A ON A.ActivityID=OD.ActivityID
INNER JOIN Studies St ON St.StudiesID=A.ActivityID
WHERE AmountPaid<FullPrice+EntryFee AND
(SELECT MIN(SM.MeetingDate) FROM StudiesMeetings SM
INNER JOIN Subjects Sub ON Sub.SubjectID=SM.SubjectID
WHERE St.StudiesID=Sub.StudiesID) < GETDATE() AND mustPayInTime=0
UNION
SELECT StudentID FROM Orders O
INNER JOIN OrderDetails OD ON OD.OrderID=O.OrderID
INNER JOIN Activities A ON A.ActivityID=OD.ActivityID
INNER JOIN StudiesMeetings SM ON SM.ActivityID=A.ActivityID
WHERE AmountPaid<FullPrice AND MeetingDate < GETDATE() AND
mustPayInTime=0)
```

Autor: Jakub Zając

Zestawienie osób spóźnionych z płatnościami za studia

```
CREATE VIEW STUDENTS_LATE_WITH_PAYMENTS_FOR_STUDIES AS
SELECT StudentID AS 'ID', FirstName, LastName, Address, City,
Country, Phone, Email,
ISNULL((SELECT
(SELECT SUM(DiscountPrice) FROM StudiesMeetings SM
INNER JOIN Subjects Sub ON Sub.SubjectID=SM.SubjectID
WHERE Sub.StudiesID=A.ActivityID AND MeetingDate < DATEADD(DAY, 3,
GETDATE()))
+EntryFee-AmountPaid FROM Orders O2
```

```
INNER JOIN OrderDetails OD2 ON OD2.OrderID=O2.OrderID
INNER JOIN Activities A ON A.ActivityID=OD2.ActivityID
INNER JOIN Studies St ON St.StudiesID=A.ActivityID
WHERE O2.StudentID=S.StudentID), 0) AS 'Total debt'
FROM Students S
INNER JOIN Users U ON U.UserID=S.StudentID
INNER JOIN Cities Cit ON U.CityID=Cit.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE StudentID IN
(SELECT StudentID FROM Orders O
INNER JOIN OrderDetails OD ON OD.OrderID=O.OrderID
INNER JOIN Activities A ON A.ActivityID=OD.ActivityID
INNER JOIN Studies St ON St.StudiesID=A.ActivityID
WHERE AmountPaid<FullPrice+EntryFee AND
(SELECT MIN(SM.MeetingDate) FROM StudiesMeetings SM
INNER JOIN Subjects Sub ON Sub.SubjectID=SM.SubjectID
WHERE St.StudiesID=Sub.StudiesID) < GETDATE())
```

Autor: Jakub Zając

Widok koszyka wraz z kosztami aktywności

```
CREATE VIEW SHOPPING_CART_VIEW AS
SELECT StudentID, SC.ActivityID, FullPrice+ISNULL(EntryFee, 0) AS
'Price' FROM ShoppingCart SC
INNER JOIN Activities A ON A.ActivityID=SC.ActivityID
LEFT JOIN Studies S ON S.StudiesID=A.ActivityID
```

Autor: Jakub Zając

Zestawienie liczby osób zapisanych na przyszłe wydarzenia

```
CREATE view PeopleSignedForUpcomingEvents as
      SELECT *
      FROM PeopleSignedForUpcomingCourses
      UNION
      SELECT *
      FROM PeopleSignedForUpcomingStudiesMeetings
      UNION
      SELECT *
      FROM PeopleSignedForUpcomingWebinars
```

Autor: Stanisław Strojniak

Zestawienie liczby osób zapisanych na przyszłe spotkania studyjne

```
CREATE view PeopleSignedForUpcomingStudiesMeetings as
      SELECT sm.MeetingID
      AS ActivityID,
            s.SubjectName
      AS Name,
            Studies.StudiesName
      AS "Course/ Major Name",
            sm.MeetingDate
      AS DATE,
            COUNT(*)
      AS NumberOfStudents,
            'Study Meeting'
      AS TYPE,
            IIF(sm.MeetingID IN (SELECT om.MeetingID FROM
OnsiteMeetings om), 'Offline', 'Online') AS OnlineOrOffline
      FROM StudiesMeetings sm
            INNER JOIN Subjects s ON sm.SubjectID = s.SubjectID
            INNER JOIN StudiesMeetingsDetails smd ON sm.MeetingID =
smd.MeetingID
            INNER JOIN Studies ON s.StudiesID = Studies.StudiesID
      WHERE MeetingDate > GETDATE()
      GROUP BY sm.MeetingID, s.SubjectName, Studies.StudiesName,
sm.MeetingDate, sm.MeetingID
```

Autor: Stanisław Strojniak

Zestawienie liczby osób zapisanych na przyszłe moduły kursów

```
CREATE view PeopleSignedForUpcomingCourses as
      SELECT cm.CourseID
      AS ActivityID,
            cm.ModuleName
      AS Name,
            c.CourseName
      AS "Course/ Major Name",
            cm.Date
      AS Date,
            COUNT(*)
      AS NumberOfStudents,
            'Course'
```

```
      AS Type,
            IIF(cm.ModuleID IN (SELECT om.ModuleID FROM
OnsiteModules om), 'Offline', 'Online') AS OnlineOrOffline
      FROM CourseModules cm
            INNER JOIN CourseModulesDetails cmd ON cm.ModuleID =
cmd.ModuleID
            INNER JOIN Courses c ON cm.CourseID = c.CourseID
      WHERE cm.Date > GETDATE()
      GROUP BY cm.CourseID, cm.ModuleName, c.CourseName, cm.Date,
cm.ModuleID
```

Autor: Stanisław Strojniak

Zestawienie liczby osób zapisanych na przyszłe webinary

```
CREATE view PeopleSignedForUpcomingWebinars as
      SELECT w.WebinarID   AS ActivityID,
            w.WebinarName AS Name,
            '-'           AS "Course/ Major Name",
            w.WebinarDate AS Date,
            COUNT(*)      AS NumberOfStudents,
            'Webinar'     AS Type,
            'Online'      AS OnlineOrOffline
      FROM Webinars w
            INNER JOIN WebinarDetails wd ON w.WebinarID =
wd.WebinarID
      WHERE WebinarDate > GETDATE()
      GROUP BY w.WebinarID, w.WebinarName, w.WebinarDate
```

Autor: Stanisław Strojniak

Zestawienie frekwencji na zakończonych wydarzeniach

```
CREATE view EventsAttendanceSummary as
      SELECT *
      FROM CoursesAttendanceSummary
      UNION
      SELECT *
      FROM StudiesMeetingAttendanceSummary
```

Autor: Stanisław Strojniak

Zestawienie frekwencji na zakończonych spotkaniach studyjnych

```
CREATE view StudiesMeetingAttendanceSummary as
      SELECT smd.MeetingID
AS ID,
            Subjects.SubjectName
AS "Module/ Subject Name",
            Studies.StudiesName
AS "Course/ Major Name",
            100 * COUNT(*) / (SELECT COUNT(*)
                              FROM Students s
                                                INNER JOIN
StudiesMeetingsDetails smd2 ON s.StudentID = smd2.StudentID
                              WHERE smd.MeetingID = smd2.MeetingID)
AS 'Attendance [%]',
            'Study Meeting'
AS Type
      FROM StudiesMeetingsDetails smd
            INNER JOIN StudiesMeetings sm ON sm.MeetingID =
smd.MeetingID
            INNER JOIN Subjects ON sm.SubjectID =
Subjects.SubjectID
            INNER JOIN Studies ON Studies.StudiesID =
Subjects.StudiesID
      WHERE sm.MeetingDate < GETDATE()
      AND NOT smd.PassedDate IS NULL
      GROUP BY smd.MeetingID, Subjects.SubjectName,
Studies.StudiesName
```

Autor: Stanisław Strojniak

Zestawienie frekwencji na zakończonych modułach kursów

```
CREATE view CoursesAttendanceSummary as
      SELECT cmd.ModuleID
AS ID,
            cm.ModuleName
AS "Module/ Subject Name",
            c.CourseName
AS "Course/ Major Name",
            100 * COUNT(*) / (SELECT COUNT(*)
                              FROM Students s
```

```
                                        INNER JOIN
CourseModulesDetails cmd2 ON s.StudentID = cmd2.StudentID
                            WHERE cmd.ModuleID = cmd2.ModuleID) AS
'Attendance [%]',
           'Course'
AS Type
      FROM CourseModulesDetails cmd
           INNER JOIN CourseModules cm ON cm.ModuleID =
cmd.ModuleID
           INNER JOIN Courses c ON C.CourseID = cm.CourseID
      WHERE cm.Date < GETDATE()
      AND NOT cmd.PassedDate IS NULL
      GROUP BY cmd.ModuleID, cm.ModuleName, c.CourseName
```

Autor: Stanisław Strojniak

Pokaż dane wszystkich studentów

```
CREATE VIEW STUDENTS_DATA AS
SELECT UserID, FirstName, LastName, City, Country, Address, Phone,
Email FROM Users U
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE UserID IN
(SELECT StudentID FROM Students)
```

Autor: Jakub Zając

Pokaż dane wszystkich wykładowców

```
CREATE VIEW LECTURERS_DATA AS
SELECT UserID, FirstName, LastName, City, Country, Address, Phone,
Email FROM Users U
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE UserID IN
(SELECT LecturerID FROM Lecturers)
```

Autor: Jakub Zając

Pokaż dane wszystkich tłumaczy

```
CREATE VIEW TRANSLATORS_DATA AS
SELECT UserID, FirstName, LastName, City, Country, Address, Phone,
```

```
Email, STRING_AGG(AL.LanguageName, ', ') AS Languages FROM Users U
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
INNER JOIN Translators T ON T.TranslatorID = U.UserID
LEFT JOIN TranslatorLanguages TL ON TL.TranslatorID =
T.TranslatorID
LEFT JOIN AvailableLanguages AL ON AL.LanguageID = TL.LanguageID
GROUP BY U.UserID, U.FirstName, U.LastName, Cit.City, Co.Country,
U.Address, U.Phone, U.Email;
```

Autor: Jakub Zając

Pokaż dane wszystkich administratorów

```
CREATE VIEW ADMINISTRATORS_DATA AS
SELECT UserID, FirstName, LastName, City, Country, Address, Phone,
Email FROM Users U
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE UserID IN
(SELECT AdministratorID FROM Administrators)
```

Autor: Jakub Zając

Pokaż dane wszystkich dyrektorów

```
CREATE VIEW DIRECTORS_DATA AS
SELECT UserID, FirstName, LastName, City, Country, Address, Phone,
Email FROM Users U
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE UserID IN
(SELECT DirectorID FROM Directors)
```

Autor: Jakub Zając

Pokaż dane wszystkich pracowników sekretariatu

```
CREATE VIEW SECRETARY_WORKERS_DATA AS
SELECT UserID, FirstName, LastName, City, Country, Address, Phone,
Email FROM Users U
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE UserID IN
```

```
(SELECT SecretaryID FROM SecretaryWorkers)
```

Autor: Jakub Zając

Pokaż dane wszystkich prowadzących praktyki

```
CREATE VIEW INTERNSHIP_SUPERVISORS_DATA AS
SELECT UserID, FirstName, LastName, City, Country, Address, Phone,
Email FROM Users U
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Countries Co ON Co.CountryID=Cit.CountryID
WHERE UserID IN
(SELECT SupervisorID FROM InternshipSupervisors)
```

Autor: Jakub Zając

Lista obecności na każdym wydarzeniu

```
CREATE view AttendanceListAllEvents as
     SELECT CourseName                          AS Name,
          ModuleName                            AS Subject,
          Users.LastName + ' ' + Users.FirstName AS 'Full Name',
          'Course'                              as [Activity
type]
     FROM CourseModulesDetails
          INNER JOIN Students ON CourseModulesDetails.StudentID =
Students.StudentID
          INNER JOIN Users ON Students.StudentID = Users.UserID
          INNER JOIN CourseModules ON
CourseModulesDetails.ModuleID = CourseModules.ModuleID
          INNER JOIN Courses ON CourseModules.CourseID =
Courses.CourseID

     UNION

     SELECT StudiesName                         AS Name,
          SubjectName                           AS Subject,
          Users.LastName + ' ' + Users.FirstName AS 'Full Name',
          'Studies'                             as [Activity
type]
     FROM StudiesMeetingsDetails
          INNER JOIN Students ON StudiesMeetingsDetails.StudentID
= Students.StudentID
```

```
          INNER JOIN Users ON Students.StudentID = Users.UserID
          INNER JOIN StudiesMeetings ON
StudiesMeetingsDetails.MeetingID = StudiesMeetings.MeetingID
          INNER JOIN Subjects ON StudiesMeetings.SubjectID =
Subjects.SubjectID
          INNER JOIN Studies ON Subjects.StudiesID =
Studies.StudiesID

     UNION

     SELECT WebinarName                          AS Name,
          ''                                     as Subject,
          Users.LastName + ' ' + Users.FirstName AS 'Full Name',
          'Webinar'                              as [Activity
type]
     FROM WebinarDetails
          INNER JOIN Students ON WebinarDetails.StudentID =
Students.StudentID
          INNER JOIN Users ON Students.StudentID = Users.UserID
          INNER JOIN Webinars ON WebinarDetails.WebinarID =
Webinars.WebinarID
go
```

Autor: Kamil Życzkowski

Lista obecności na każdym spotkaniu studyjnym

```
CREATE view AttendanceListStudiesMeetings as
     SELECT StudiesName                          AS Name,
          SubjectName                            AS Subject,
          Users.LastName + ' ' + Users.FirstName AS 'Full Name',
          'Studies'                              as [Activity
type]
     FROM StudiesMeetingsDetails
          INNER JOIN Students ON StudiesMeetingsDetails.StudentID
= Students.StudentID
          INNER JOIN Users ON Students.StudentID = Users.UserID
          INNER JOIN StudiesMeetings ON
StudiesMeetingsDetails.MeetingID = StudiesMeetings.MeetingID
          INNER JOIN Subjects ON StudiesMeetings.SubjectID =
Subjects.SubjectID
          INNER JOIN Studies ON Subjects.StudiesID =
```

```
Studies.StudiesID
go
```
Autor: Kamil Życzkowski

Lista obecności na każdym module kursu

```
CREATE view AttendanceListCourseModules as
      SELECT CourseName                              AS Name,
            ModuleName                               AS Subject,
            Users.LastName + ' ' + Users.FirstName AS 'Full Name',
            'Course'                                 as [Activity
type]
      FROM CourseModulesDetails
            INNER JOIN Students ON CourseModulesDetails.StudentID =
Students.StudentID
            INNER JOIN Users ON Students.StudentID = Users.UserID
            INNER JOIN CourseModules ON
CourseModulesDetails.ModuleID = CourseModules.ModuleID
            INNER JOIN Courses ON CourseModules.CourseID =
Courses.CourseID
go
```
Autor: Kamil Życzkowski

Lista obecności na każdym webinarze

```
CREATE view AttendanceListWebinars as
      SELECT WebinarName                             AS Name,
            ''                                       as Subject,
            Users.LastName + ' ' + Users.FirstName AS 'Full Name',
            'Webinar'                                as [Activity
type]
      FROM WebinarDetails
            INNER JOIN Students ON WebinarDetails.StudentID =
Students.StudentID
            INNER JOIN Users ON Students.StudentID = Users.UserID
            INNER JOIN Webinars ON WebinarDetails.WebinarID =
Webinars.WebinarID
go
```
Autor: Kamil Życzkowski

Lista osób z kolizjami w przyszłym planie zajęć

```
CREATE VIEW PEOPLE_WITH_COLLISIONS_IN_FUTURE_PLAN AS
SELECT S.StudentID AS 'ID', FirstName, LastName, Address, City,
Phone, Email, AFE.FirstID, AFE.FirstEventID, AFE.FirstDate,
AFE2.SecondID, AFE2.SecondEventID, AFE2.SecondDate FROM Students S
INNER JOIN Users U ON U.UserID=S.StudentID
INNER JOIN Cities Cit ON Cit.CityID=U.CityID
INNER JOIN Orders O ON O.StudentID=S.StudentID
INNER JOIN OrderDetails OD ON OD.OrderID=O.OrderID
INNER JOIN OrderDetails OD2 ON OD2.OrderID=O.OrderID
INNER JOIN (
SELECT WebinarID AS 'FirstID', WebinarID AS 'FirstEventID',
WebinarDate AS 'FirstDate' FROM Webinars W
WHERE WebinarDate > GETDATE()
UNION
SELECT Sub.StudiesID AS 'FirstID', ActivityID AS 'FirstEventID',
MeetingDate AS 'FirstDate' FROM StudiesMeetings SM
INNER JOIN Subjects Sub ON SM.SubjectID=Sub.SubjectID
WHERE MeetingDate > GETDATE()
UNION
SELECT CourseID AS 'FirstID', ModuleID AS 'FirstEventID', Date AS
'FirstDate' FROM CourseModules CM
WHERE CM.Date > GETDATE()
UNION
SELECT ActivityID AS 'FirstID', null AS 'FirstEventID',
MeetingDate AS 'FirstDate' FROM StudiesMeetings SM2
WHERE MeetingDate > GETDATE()) AFE ON AFE.FirstID=OD.ActivityID
INNER JOIN (
SELECT WebinarID AS 'SecondID', WebinarID AS 'SecondEventID',
WebinarDate AS 'SecondDate' FROM Webinars W
WHERE WebinarDate > GETDATE()
UNION
SELECT Sub.StudiesID AS 'SecondID', ActivityID AS 'SecondEventID',
MeetingDate AS 'SecondDate' FROM StudiesMeetings SM
INNER JOIN Subjects Sub ON SM.SubjectID=Sub.SubjectID
WHERE MeetingDate > GETDATE()
UNION
SELECT CourseID AS 'SecondID', ModuleID AS 'SecondEventID', Date
AS 'SecondDate' FROM CourseModules CM
WHERE CM.Date > GETDATE()
UNION
```

```
SELECT ActivityID AS 'SecondID', null AS 'SecondEventID',
MeetingDate AS 'SecondDate' FROM StudiesMeetings SM2
WHERE MeetingDate > GETDATE()) AFE2 ON
AFE2.SecondID=OD2.ActivityID
WHERE NOT(AFE.FirstID=AFE2.SecondID AND
AFE.FirstEventID=AFE2.SecondEventID) AND DATEDIFF(MINUTE,
AFE.FirstDate, AFE2.SecondDate)<90 AND DATEDIFF(MINUTE,
AFE.FirstDate, AFE2.SecondDate)>0
```

Autor: Jakub Zając

Lista wszystkich przyszłych wydarzeń

```
create view UpcomingEvents as
select MeetingDate                    as Date,
           'Studies Meeting'               as EventType,
           MeetingID                       as EventID,
           Subjects.SubjectName            as Subject,
           AvailableLanguages.LanguageName as Language
     from StudiesMeetings
           inner join Subjects on StudiesMeetings.SubjectID =
Subjects.SubjectID
           inner join AvailableLanguages on
StudiesMeetings.LanguageID = AvailableLanguages.LanguageID
     where MeetingDate > getdate()

     UNION

     select Date                           as Date,
           'Course Module'                 as EventType,
           ModuleID                        as EventID,
           ModuleName                      as Subject,
           AvailableLanguages.LanguageName as Language
     from CourseModules
           inner join AvailableLanguages on
CourseModules.LanguageID = AvailableLanguages.LanguageID
           inner join Courses ON CourseModules.CourseID =
Courses.CourseID
     where Date > getdate()

     UNION
     select WebinarDate                    as Date,
```

```
           'Webinar'                          as EventType,
           WebinarID                          as EventID,
           WebinarName                        as Subject,
           AvailableLanguages.LanguageName as Language
     from Webinars
           inner join AvailableLanguages on Webinars.LanguageID =
AvailableLanguages.LanguageID
     where WebinarDate > getdate()
go
```

Autor: Kamil Życzkowski

Lista wszystkich przyszłych spotkań studyjnych

```
create view UpcomingStudiesMeetings as
select MeetingDate, MeetingID, Subjects.SubjectName,
AvailableLanguages.LanguageName
     from StudiesMeetings
           inner join Subjects on StudiesMeetings.SubjectID =
Subjects.SubjectID
           inner join AvailableLanguages on
StudiesMeetings.LanguageID = AvailableLanguages.LanguageID
     where MeetingDate > getdate()
```

Autor: Kamil Życzkowski

Lista wszystkich przyszłych modułów kursów

```
create view UpcomingCourseModules as
select Date, ModuleID, Courses.CourseName, ModuleName,
AvailableLanguages.LanguageName
     from CourseModules
           inner join AvailableLanguages on
CourseModules.LanguageID = AvailableLanguages.LanguageID
           inner join Courses ON CourseModules.CourseID =
Courses.CourseID
     where Date > getdate()
```

Autor: Kamil Życzkowski

Lista wszystkich przyszłych webinarów

```
create view UpcomingWebinars as
```

```
select WebinarDate, WebinarID, WebinarName,
AvailableLanguages.LanguageName
      from Webinars
            inner join AvailableLanguages on Webinars.LanguageID =
AvailableLanguages.LanguageID
      where WebinarDate > getdate()
```

Autor: Kamil Życzkowski

Procedury

Dodanie nowego zamówienia

```
CREATE PROCEDURE AddOrder
@StudentID INT
AS
BEGIN
      DECLARE @OrderID INT;

      INSERT INTO Orders (StudentID, OrderDate)
      VALUES (@StudentID, GETDATE());

      SET @OrderID = SCOPE_IDENTITY();

      INSERT INTO OrderDetails (OrderID, ActivityID)
      SELECT @OrderID, ActivityID FROM ShoppingCart
      WHERE StudentID = @StudentID;

      DELETE FROM ShoppingCart
      WHERE StudentID = @StudentID;

END;
```

Autor: Jakub Zając

Dodanie przedmiotu do koszyka

```
CREATE PROCEDURE AddToShoppingCart
@StudentID INT,
@ActivityID INT
AS
BEGIN
```

```
      IF NOT EXISTS (
      SELECT 1 FROM ShoppingCart
      WHERE StudentID = @StudentID AND ActivityID = @ActivityID
      )
      BEGIN
            INSERT INTO ShoppingCart(StudentID, ActivityID)
            VALUES (@StudentID, @ActivityID);
      END
END;
```

Autor: Jakub Zając

Dodanie nowego webinaru

```
CREATE PROCEDURE AddWebinar
   @WebinarName NVARCHAR(128),
   @WebinarDate DATETIME,
   @LecturerID INT,
   @TranslatorID INT,
   @LanguageID INT,
   @Link NVARCHAR(128),
   @FullPrice MONEY
AS
BEGIN
   INSERT INTO Activities(FullPrice, isAvailable)
   VALUES (@FullPrice, 1)

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

   INSERT INTO Webinars (WebinarID,WebinarName, WebinarDate,
LecturerID, TranslatorID, LanguageID, Link)
   VALUES (@ActivityID,@WebinarName, @WebinarDate, @LecturerID,
@TranslatorID, @LanguageID, @Link);
END;
```

Autor: Kamil Życzkowski

Dodanie nowego kursu

```
CREATE PROCEDURE AddCourse
   @CourseName NVARCHAR(128),
   @CourseDescription NVARCHAR(512),
```

```
   @CourseTypeID INT,
   @AdvancePrice MONEY,
   @FullPrice MONEY
AS
BEGIN
   INSERT INTO Activities(FullPrice, isAvailable)
   VALUES (@FullPrice, 1)

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

   INSERT INTO Courses
(CourseID,CourseName,CourseDescription,CourseTypeID,AdvancePrice)
   VALUES (@ActivityID,@CourseName, @CourseDescription,
@CourseTypeID, @AdvancePrice);
END;
```

Autor: Kamil Życzkowski

Dodanie nowego modułu zdalnego do kursu

```
CREATE PROCEDURE AddCourseModuleOnlineAsync
      @ModuleName NVARCHAR(128),
      @Date DATETIME,
      @CourseID INT,
      @LecturerID INT,
      @LanguageID INT,
      @TranslatorID INT,
      @RecordingLink NVARCHAR(128)
AS
BEGIN
      INSERT INTO CourseModules (ModuleName, Date, CourseID,
LecturerID, LanguageID, TranslatorID)
      VALUES (@ModuleName, @Date, @CourseID, @LecturerID,
@LanguageID, @TranslatorID);

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

      INSERT INTO OnlineAsyncModules (ModuleID, RecordingLink)
      VALUES (@LastCourseModuleID, @RecordingLink);
END;
```

Autor: Kamil Życzkowski

Dodanie nowego modułu hybrydowego do kursu

```
CREATE PROCEDURE AddCourseModuleOnlineSync
      @ModuleName NVARCHAR(128),
      @Date DATETIME,
      @CourseID INT,
      @LecturerID INT,
      @LanguageID INT,
      @TranslatorID INT,
      @MeetingLink VARCHAR(128)
AS
BEGIN
      INSERT INTO CourseModules (ModuleName, Date, CourseID,
LecturerID, LanguageID, TranslatorID)
      VALUES (@ModuleName, @Date, @CourseID, @LecturerID,
@LanguageID, @TranslatorID);

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

      INSERT INTO OnlineSyncModules (ModuleID, MeetingLink)
      VALUES (@LastCourseModuleID, @MeetingLink);
END;
```

Autor: Kamil Życzkowski

Dodanie nowego modułu stacjonarnego do kursu

```
CREATE PROCEDURE AddCourseModuleOnsite
   @ModuleName NVARCHAR(128),
   @Date DATETIME,
   @CourseID INT,
   @LecturerID INT,
   @LanguageID INT,
   @TranslatorID INT,
   @ClassLimit INT,
   @RoomID INT
AS
BEGIN
   INSERT INTO CourseModules (ModuleName, Date, CourseID,
```

```
LecturerID, LanguageID, TranslatorID)
   VALUES (@ModuleName, @Date, @CourseID, @LecturerID,
@LanguageID, @TranslatorID);

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

   INSERT INTO OnsiteModules (ModuleID, ClassLimit, RoomID)
   VALUES (@LastCourseModuleID, @ClassLimit, @RoomID);
END;
```

Autor: Kamil Życzkowski

Dodanie nowych studiów

```
CREATE PROCEDURE AddStudies @StudiesName NVARCHAR(128),
                            @StudiesDescription NVARCHAR(512),
                            @ClassLimit INT,
                            @EntryFee MONEY,
                            @FullPrice MONEY
AS
BEGIN
     INSERT INTO Activities(FullPrice, isAvailable)
     VALUES (@FullPrice, 1)

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

     INSERT INTO Studies (StudiesID, StudiesName,
StudiesDescription, ClassLimit, EntryFee)
     VALUES (@ActivityID, @StudiesName, @StudiesDescription,
@ClassLimit, @EntryFee);
END;
```

Autor: Stanisław Strojniak

Dodanie nowego przedmiotu do studiów

```
CREATE PROCEDURE AddSubject @StudiesID INT,
                            @LecturerID INT,
                            @SubjectName NVARCHAR(64),
                            @SubjectDescription NVARCHAR(512)
```

```
AS
BEGIN
	INSERT INTO Subjects (StudiesID, LecturerID, SubjectName,
SubjectDescription)
	VALUES (@StudiesID, @LecturerID, @SubjectName,
@SubjectDescription);
END;
```

Autor: Stanisław Strojniak

Dodanie nowego spotkania do przedmiotu

```
CREATE PROCEDURE AddStudyMeeting @SubjectID INT,
                                 @LanguageID INT,
                                 @MeetingDate DATETIME,
                                 @TranslatorID INT,
                                 @FullPrice MONEY,
                                 @DiscountPrice MONEY
AS
BEGIN
	INSERT INTO Activities(FullPrice, isAvailable)
	VALUES (@FullPrice, 1)

	DECLARE @MeetingActivityID INT = SCOPE_IDENTITY()

	INSERT INTO StudiesMeetings (ActivityID, SubjectID,
LanguageID, MeetingDate, TranslatorID, DiscountPrice)
	VALUES (@MeetingActivityID, @SubjectID, @LanguageID,
@MeetingDate, @TranslatorID, @DiscountPrice)

	DECLARE @StudyActivityID INT
	SELECT @StudyActivityID = a.ActivityID
	FROM Activities a
		INNER JOIN Subjects s ON s.StudiesID = a.ActivityID
	WHERE s.SubjectID = @SubjectID

	UPDATE Activities
	SET FullPrice = FullPrice + @DiscountPrice
	WHERE ActivityID = @StudyActivityID
END
GO
```

Autor: Stanisław Strojniak

Dodanie nowego stażu

```
CREATE PROCEDURE AddInternship @SupervisorID INT,
                               @StudiesID INT,
                               @StartDate DATETIME
AS
BEGIN
     INSERT INTO Internships (SupervisorID, StudiesID, StartDate)
     VALUES (@SupervisorID, @StudiesID, @StartDate);
END;
```

Autor: Stanisław Strojniak

Dodanie nowego studenta

```
CREATE PROCEDURE AddNewStudent
@FirstName NVARCHAR(64),
@LastName NVARCHAR(64),
@Country NVARCHAR(128),
@City NVARCHAR(128),
@Address NVARCHAR(256),
@Phone NVARCHAR(64),
@Email NVARCHAR(64)
AS
BEGIN
     DECLARE @CountryID INT, @CityID INT, @UserID INT;

     SELECT @CountryID = CountryID FROM Countries WHERE Country =
@Country;

     IF @CountryID IS NULL
     BEGIN
          INSERT INTO Countries(Country)
          VALUES (@Country);

          SET @CountryID = SCOPE_IDENTITY();
     END

     SELECT @CityID = CityID FROM Cities WHERE City = @City AND
CountryID = @CountryID;

     IF @CityID IS NULL
     BEGIN
```

```
            INSERT INTO Cities(CountryID, City)
            VALUES (@CountryID, @City);

            SET @CityID = SCOPE_IDENTITY();
      END

      INSERT INTO Users(FirstName, LastName, CityID, Address,
Phone, Email)
      VALUES (@FirstName, @LastName, @CityID, @Address, @Phone,
@Email);

      SET @UserID = SCOPE_IDENTITY();

      INSERT INTO Students (StudentID)
      VALUES(@UserID);
END;
```

Autor: Jakub Zając

Dodanie nowego wykładowcy

```
CREATE PROCEDURE AddNewLecturer
@FirstName NVARCHAR(64),
@LastName NVARCHAR(64),
@Country NVARCHAR(128),
@City NVARCHAR(128),
@Address NVARCHAR(256),
@Phone NVARCHAR(64),
@Email NVARCHAR(64)
AS
BEGIN
      DECLARE @CountryID INT, @CityID INT, @UserID INT;

      SELECT @CountryID = CountryID FROM Countries WHERE Country =
@Country;

      IF @CountryID IS NULL
      BEGIN
            INSERT INTO Countries(Country)
            VALUES (@Country);

            SET @CountryID = SCOPE_IDENTITY();
      END
```

```
	SELECT @CityID = CityID FROM Cities WHERE City = @City AND
CountryID = @CountryID;

	IF @CityID IS NULL
	BEGIN
		INSERT INTO Cities(CountryID, City)
		VALUES (@CountryID, @City);

		SET @CityID = SCOPE_IDENTITY();
	END

	INSERT INTO Users(FirstName, LastName, CityID, Address,
Phone, Email)
	VALUES (@FirstName, @LastName, @CityID, @Address, @Phone,
@Email);

	SET @UserID = SCOPE_IDENTITY();

	INSERT INTO Lecturers (LecturerID)
	VALUES(@UserID);
END;
```

Autor: Jakub Zając

Dodanie nowego tłumacza

```
CREATE PROCEDURE AddNewTranslator
@FirstName NVARCHAR(64),
@LastName NVARCHAR(64),
@Country NVARCHAR(128),
@City NVARCHAR(128),
@Address NVARCHAR(256),
@Phone NVARCHAR(64),
@Email NVARCHAR(64)
AS
BEGIN
	DECLARE @CountryID INT, @CityID INT, @UserID INT;

	SELECT @CountryID = CountryID FROM Countries WHERE Country =
@Country;

	IF @CountryID IS NULL
```

```
	BEGIN
		INSERT INTO Countries(Country)
		VALUES (@Country);

		SET @CountryID = SCOPE_IDENTITY();
	END

	SELECT @CityID = CityID FROM Cities WHERE City = @City AND
CountryID = @CountryID;

	IF @CityID IS NULL
	BEGIN
		INSERT INTO Cities(CountryID, City)
		VALUES (@CountryID, @City);

		SET @CityID = SCOPE_IDENTITY();
	END

	INSERT INTO Users(FirstName, LastName, CityID, Address,
Phone, Email)
	VALUES (@FirstName, @LastName, @CityID, @Address, @Phone,
@Email);

	SET @UserID = SCOPE_IDENTITY();

	INSERT INTO Translators(TranslatorID)
	VALUES(@UserID);
END;
```

Autor: Jakub Zając

Dodanie nowego języka do danego tłumacza

```
CREATE PROCEDURE AddLanguageToTranslator
@TranslatorID INT,
@Language NVARCHAR(64)
AS
BEGIN
	DECLARE @LanguageID INT;

	IF NOT EXISTS(SELECT 1 FROM Translators WHERE TranslatorID =
@TranslatorID)
	BEGIN
```

```
            RETURN;
      END

      SELECT @LanguageID = LanguageID FROM AvailableLanguages
      WHERE LanguageName = @Language;

      IF @LanguageID IS NULL
      BEGIN
            INSERT INTO AvailableLanguages(LanguageName)
            VALUES (@Language);

            SET @LanguageID = SCOPE_IDENTITY();
      END

      INSERT INTO TranslatorLanguages(TranslatorID, LanguageID)
      VALUES (@TranslatorID, @LanguageID);
END;
```

Autor: Jakub Zając

Dodanie nowego administratora

```
CREATE PROCEDURE AddNewAdministrator
@FirstName NVARCHAR(64),
@LastName NVARCHAR(64),
@Country NVARCHAR(128),
@City NVARCHAR(128),
@Address NVARCHAR(256),
@Phone NVARCHAR(64),
@Email NVARCHAR(64)
AS
BEGIN
      DECLARE @CountryID INT, @CityID INT, @UserID INT;

      SELECT @CountryID = CountryID FROM Countries WHERE Country =
@Country;

      IF @CountryID IS NULL
      BEGIN
            INSERT INTO Countries(Country)
            VALUES (@Country);

            SET @CountryID = SCOPE_IDENTITY();
```

```
	END

	SELECT @CityID = CityID FROM Cities WHERE City = @City AND
CountryID = @CountryID;

	IF @CityID IS NULL
	BEGIN
		INSERT INTO Cities(CountryID, City)
		VALUES (@CountryID, @City);

		SET @CityID = SCOPE_IDENTITY();
	END

	INSERT INTO Users(FirstName, LastName, CityID, Address,
Phone, Email)
	VALUES (@FirstName, @LastName, @CityID, @Address, @Phone,
@Email);

	SET @UserID = SCOPE_IDENTITY();

	INSERT INTO Administrators(AdministratorID)
	VALUES(@UserID);
END;
```

Autor: Jakub Zając

Dodanie nowego dyrektora

```
CREATE PROCEDURE AddNewDirector
@FirstName NVARCHAR(64),
@LastName NVARCHAR(64),
@Country NVARCHAR(128),
@City NVARCHAR(128),
@Address NVARCHAR(256),
@Phone NVARCHAR(64),
@Email NVARCHAR(64)
AS
BEGIN
	DECLARE @CountryID INT, @CityID INT, @UserID INT;

	SELECT @CountryID = CountryID FROM Countries WHERE Country =
@Country;
```

```
	IF @CountryID IS NULL
	BEGIN
		INSERT INTO Countries(Country)
		VALUES (@Country);

		SET @CountryID = SCOPE_IDENTITY();
	END

	SELECT @CityID = CityID FROM Cities WHERE City = @City AND
CountryID = @CountryID;

	IF @CityID IS NULL
	BEGIN
		INSERT INTO Cities(CountryID, City)
		VALUES (@CountryID, @City);

		SET @CityID = SCOPE_IDENTITY();
	END

	INSERT INTO Users(FirstName, LastName, CityID, Address,
Phone, Email)
	VALUES (@FirstName, @LastName, @CityID, @Address, @Phone,
@Email);

	SET @UserID = SCOPE_IDENTITY();

	INSERT INTO Directors(DirectorID)
	VALUES(@UserID);
END;
```

Autor: Jakub Zając

Dodanie nowego pracownika sekretariatu

```
CREATE PROCEDURE AddNewSecretaryWorker
@FirstName NVARCHAR(64),
@LastName NVARCHAR(64),
@Country NVARCHAR(128),
@City NVARCHAR(128),
@Address NVARCHAR(256),
@Phone NVARCHAR(64),
@Email NVARCHAR(64)
AS
```

```
BEGIN
	DECLARE @CountryID INT, @CityID INT, @UserID INT;

	SELECT @CountryID = CountryID FROM Countries WHERE Country =
@Country;

	IF @CountryID IS NULL
	BEGIN
		INSERT INTO Countries(Country)
		VALUES (@Country);

		SET @CountryID = SCOPE_IDENTITY();
	END

	SELECT @CityID = CityID FROM Cities WHERE City = @City AND
CountryID = @CountryID;

	IF @CityID IS NULL
	BEGIN
		INSERT INTO Cities(CountryID, City)
		VALUES (@CountryID, @City);

		SET @CityID = SCOPE_IDENTITY();
	END

	INSERT INTO Users(FirstName, LastName, CityID, Address,
Phone, Email)
	VALUES (@FirstName, @LastName, @CityID, @Address, @Phone,
@Email);

	SET @UserID = SCOPE_IDENTITY();

	INSERT INTO SecretaryWorkers(SecretaryID)
	VALUES(@UserID);
END;
```

Autor: Jakub Zając

Dodanie nowego prowadzącego praktyki

```
CREATE PROCEDURE AddNewInternshipSupervisor
@FirstName NVARCHAR(64),
@LastName NVARCHAR(64),
```

```
@Country NVARCHAR(128),
@City NVARCHAR(128),
@Address NVARCHAR(256),
@Phone NVARCHAR(64),
@Email NVARCHAR(64)
AS
BEGIN
	DECLARE @CountryID INT, @CityID INT, @UserID INT;

	SELECT @CountryID = CountryID FROM Countries WHERE Country =
@Country;

	IF @CountryID IS NULL
	BEGIN
		INSERT INTO Countries(Country)
		VALUES (@Country);

		SET @CountryID = SCOPE_IDENTITY();
	END

	SELECT @CityID = CityID FROM Cities WHERE City = @City AND
CountryID = @CountryID;

	IF @CityID IS NULL
	BEGIN
		INSERT INTO Cities(CountryID, City)
		VALUES (@CountryID, @City);

		SET @CityID = SCOPE_IDENTITY();
	END

	INSERT INTO Users(FirstName, LastName, CityID, Address,
Phone, Email)
	VALUES (@FirstName, @LastName, @CityID, @Address, @Phone,
@Email);

	SET @UserID = SCOPE_IDENTITY();

	INSERT INTO InternshipSupervisors(SupervisorID)
	VALUES(@UserID);
END;
```

Autor: Jakub Zając

Funkcje

Podliczenie frekwencji użytkownika na danym kursie

```
CREATE FUNCTION CourseAttendance (
      @StudentID INT,
      @CourseID INT
)
RETURNS DECIMAL(5, 2) -- 5 digits, 2 od which are after the
decimal point
AS
BEGIN
      DECLARE @TotalModules INT
      DECLARE @AttendedModules INT

      SELECT @TotalModules = COUNT(*)
      FROM CourseModules
      WHERE CourseID = @CourseID

      SELECT @AttendedModules = COUNT(*)
      FROM CourseModulesDetails cmd
      INNER JOIN CourseModules cm ON cmd.ModuleID = cm.ModuleID
      WHERE cmd.StudentID = @StudentID AND cm.CourseID = @CourseID
            AND cmd.PassedDate IS NOT NULL

      RETURN IIF(@TotalModules > 0, CAST(@AttendedModules AS
DECIMAL(5, 2)) * 100 / @TotalModules, 0)
END
```

Autor: Stanisław Strojniak

Podliczenie frekwencji użytkownika na danym przedmiocie na studiach

```
CREATE FUNCTION SubjectAttendance (
      @StudentID INT,
      @SubjectID INT
)
RETURNS DECIMAL(5, 2) -- 5 digits, 2 od which are after the
decimal point
AS
BEGIN
```

```
	DECLARE @TotalMeetings INT
	DECLARE @AttendedMeetings INT

	SELECT @TotalMeetings = COUNT(*)
	FROM StudiesMeetings
	WHERE SubjectID = @SubjectID

	SELECT @AttendedMeetings = COUNT(*)
	FROM StudiesMeetingsDetails smd
	JOIN StudiesMeetings sm ON smd.MeetingID = sm.MeetingID
	WHERE smd.StudentID = @StudentID AND sm.SubjectID =
@SubjectID
		AND smd.PassedDate IS NOT NULL

	RETURN IIF(@TotalMeetings > 0, CAST(@AttendedMeetings AS
DECIMAL(5, 2)) * 100 / @TotalMeetings, 0)
END
```

Autor: Stanisław Strojniak

Podliczenie ilości wolnych miejsc na studiach

```
CREATE FUNCTION GetStudiesAvailableSeats (@StudiesID INT)
RETURNS INT
AS
BEGIN
	DECLARE @ClassLimit INT;
	DECLARE @EnrolledCount INT;
	DECLARE @AvailableSeats INT;

	SELECT @ClassLimit = ClassLimit FROM Studies
	WHERE StudiesID = @StudiesID;

	SELECT @EnrolledCount = COUNT(*) FROM Studies S
	INNER JOIN Activities A ON A.ActivityID=S.StudiesID
	INNER JOIN OrderDetails OD ON OD.ActivityID=A.ActivityID
	INNER JOIN Orders O ON O.OrderID=OD.OrderID
	WHERE S.StudiesID = @StudiesID;

	SET @AvailableSeats = @ClassLimit - @EnrolledCount;

	RETURN @AvailableSeats;
END;
```

Autor: Jakub Zając

Podliczenie ilości wolnych miejsc na kursach

```
CREATE FUNCTION GetCourseAvailableSeats (@CourseID INT)
RETURNS INT
AS
BEGIN
      DECLARE @ClassLimit INT;
      DECLARE @EnrolledCount INT;
      DECLARE @AvailableSeats INT;

      SELECT @ClassLimit = MIN(ClassLimit) FROM OnsiteModules OM
      INNER JOIN CourseModules CM ON CM.ModuleID=OM.ModuleID
      INNER JOIN Courses C ON C.CourseID=CM.CourseID
      WHERE C.CourseID = @CourseID;

      SELECT @EnrolledCount = COUNT(*) FROM Courses C
      INNER JOIN Activities A ON A.ActivityID=C.CourseID
      INNER JOIN OrderDetails OD ON OD.ActivityID=A.ActivityID
      INNER JOIN Orders O ON O.OrderID=OD.OrderID
      WHERE C.CourseID = @CourseID;

      SET @AvailableSeats = @ClassLimit - @EnrolledCount;

      RETURN @AvailableSeats;
END;
```

Autor: Jakub Zając

Sprawdzenie czy student zaliczył praktyki

```
CREATE FUNCTION CheckInternshipPassed (@StudentID INT)
RETURNS BIT
AS
BEGIN
      DECLARE @Result BIT

      SELECT @Result = isPassed
      FROM InternshipDetails
      WHERE StudentID = @StudentID

      RETURN ISNULL(@Result, 0)
```

```
END
```

Autor: Stanisław Strojniak

Obliczenie łącznej wartości danego zamówienia

```
CREATE FUNCTION GetTotalOrderValue (@OrderID INT)
RETURNS MONEY
AS
BEGIN
      DECLARE @TotalValue MONEY;

      SELECT @TotalValue = SUM(
             CASE
                 WHEN S.StudiesID IS NOT NULL THEN (A.FullPrice +
S.EntryFee)
                 ELSE A.FullPrice
             END
            )
    FROM OrderDetails OD
    INNER JOIN Activities A ON OD.ActivityID = A.ActivityID
    LEFT JOIN Studies S ON A.ActivityID = S.StudiesID
    WHERE OD.OrderID = @OrderID;

      RETURN @TotalValue
END;
```

Autor: Jakub Zając

Obliczenie łącznej wartości koszyka

```
CREATE FUNCTION GetTotalCartValue (@StudentID INT)
RETURNS MONEY
AS
BEGIN
      DECLARE @TotalValue MONEY;

      SELECT @TotalValue = SUM(Price)
    FROM SHOPPING_CART_VIEW SC
    WHERE SC.StudentID = @StudentID;

      RETURN @TotalValue
END;
```

Autor: Jakub Zając

Zwrócenie harmonogramu danego kierunku studiów

```
CREATE FUNCTION GetScheduleForStudies (@StudiesID INT)
RETURNS TABLE
AS
RETURN
(
   SELECT
       s.StudiesName,
       sub.SubjectName,
       sm.MeetingDate,
       l.LanguageName AS Language,
       CONCAT(u.FirstName, ' ', u.LastName) AS Lecturer,
       sm.MeetingID
   FROM
       Studies AS s
   JOIN
       Subjects AS sub ON s.StudiesID = sub.StudiesID
   JOIN
       StudiesMeetings AS sm ON sub.SubjectID = sm.SubjectID
   JOIN
       AvailableLanguages AS l ON sm.LanguageID = l.LanguageID
   JOIN
       Lecturers AS lec ON sub.LecturerID = lec.LecturerID
   JOIN
       Users AS u ON lec.LecturerID = u.UserID
   WHERE
       s.StudiesID = @StudiesID
);
```

Autor: Kamil Życzkowski

Zwrócenie harmonogramu danego kierunku studiów w konkretnym semestrze

```
Routine: GetStudyMeetingsByTerm
CREATE FUNCTION GetStudyMeetingsByTerm (
     @StudiesID INT,
     @Term INT
)
RETURNS TABLE
AS
```

```
RETURN
(
      SELECT
      s.StudiesName,
      sub.SubjectName,
      sm.MeetingDate,
      l.LanguageName,
      CONCAT(u.FirstName, ' ', u.LastName) AS LecturerName,
      sub.Term
      FROM StudiesMeetings sm
      INNER JOIN Subjects sub ON sm.SubjectID = sub.SubjectID
      INNER JOIN Studies s ON sub.StudiesID = s.StudiesID
      INNER JOIN AvailableLanguages l ON sm.LanguageID =
l.LanguageID
      INNER JOIN Lecturers lec ON sub.LecturerID = lec.LecturerID
      INNER JOIN Users u ON lec.LecturerID = u.UserID
      WHERE
      s.StudiesID = @StudiesID
      AND sub.Term = @Term
)
GO
```

Autor: Stanisław Strojniak

Zwrócenie harmonogramu danego kursu

```
CREATE FUNCTION GetScheduleForCourse (@CourseID INT)
RETURNS TABLE
AS
RETURN
(
   SELECT
       c.CourseName,
       sm.Date,
       l.LanguageName AS Language,
       CONCAT(u.FirstName, ' ', u.LastName) AS Lecturer,
       sm.ModuleID
   FROM
       Courses AS c
   JOIN
       CourseModules AS sm ON c.CourseID = sm.CourseID
   JOIN
       AvailableLanguages AS l ON sm.LanguageID = l.LanguageID
```

```
    JOIN
        Lecturers AS lec ON sm.LecturerID = lec.LecturerID
    JOIN
        Users AS u ON lec.LecturerID = u.UserID
    WHERE
        c.CourseID = @CourseID
);
```

Autor: Kamil Życzkowski

Zwrócenie harmonogramu zajęć danego studenta

```
CREATE FUNCTION GetScheduleForStudent (@StudentID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        Students.StudentID,
        CONCAT(su.FirstName, ' ', su.LastName) AS Student,
        'Studies' AS [Activity Type],
        CONCAT(StudiesName, ': ', SubjectName) AS [Activity Name],
        MeetingDate AS Date,
        CONCAT(lu.FirstName, ' ', lu.LastName) AS Lecturer
    FROM
        Students
    INNER JOIN
            Users AS su ON Students.StudentID = su.UserID
    INNER JOIN
            StudiesMeetingsDetails ON Students.StudentID =
StudiesMeetingsDetails.StudentID
    INNER JOIN
            StudiesMeetings ON StudiesMeetingsDetails.MeetingID =
StudiesMeetings.MeetingID
    INNER JOIN
            Subjects ON StudiesMeetings.SubjectID =
Subjects.SubjectID
    INNER JOIN
            Studies ON Subjects.StudiesID = Studies.StudiesID
    INNER JOIN
            Lecturers ON Subjects.LecturerID = Lecturers.LecturerID
    INNER JOIN
            Users AS lu ON Lecturers.LecturerID = lu.UserID
```

```
    WHERE
        Students.StudentID = @StudentID

    UNION

    SELECT
        Students.StudentID,
        CONCAT(su.FirstName, ' ', su.LastName) AS Student,
        'Course' AS [Activity Type],
        CONCAT(CourseName, ': ', ModuleName),
        Date,
        CONCAT(lu.FirstName, ' ', lu.LastName) AS Lecturer
    FROM
        Students
    INNER JOIN
            Users AS su ON Students.StudentID = su.UserID
    INNER JOIN
            CourseModulesDetails ON Students.StudentID =
CourseModulesDetails.StudentID
    INNER JOIN
            CourseModules ON CourseModulesDetails.ModuleID =
CourseModules.ModuleID
    INNER JOIN
            Courses ON CourseModules.CourseID = Courses.CourseID
    INNER JOIN
            Lecturers ON CourseModules.LecturerID =
Lecturers.LecturerID
    INNER JOIN
            Users AS lu ON Lecturers.LecturerID = lu.UserID
    WHERE
        Students.StudentID = @StudentID

    UNION

    SELECT
        Students.StudentID,
        CONCAT(su.FirstName, ' ', su.LastName) AS Student,
        'Webinar' AS [Activity Type],
        WebinarName,
        WebinarDate,
        CONCAT(lu.FirstName, ' ', lu.LastName) AS Lecturer
    FROM
```

```
        Students
    INNER JOIN
            Users AS su ON Students.StudentID = su.UserID
    INNER JOIN
            WebinarDetails ON Students.StudentID =
WebinarDetails.StudentID
    INNER JOIN
            Webinars ON WebinarDetails.WebinarID =
Webinars.WebinarID
    INNER JOIN
            Lecturers ON su.UserID = Lecturers.LecturerID
    INNER JOIN
            Users AS lu ON Lecturers.LecturerID = lu.UserID
    WHERE
        Students.StudentID = @StudentID
);
```

Autor: Kamil Życzkowski

Triggery

Automatyczne dodanie studenta do webinaru po zakupieniu

```
CREATE TRIGGER trg_AddStudentToWebinar
      ON OrderDetails
      AFTER INSERT, UPDATE AS
BEGIN
      SET NOCOUNT ON;
      INSERT INTO WebinarDetails (WebinarID, StudentID, DueDate)
      SELECT w.WebinarID, o.StudentID, DATEADD(DAY, 30, GETDATE())
      FROM Webinars w
            INNER JOIN Inserted i ON w.WebinarID = i.ActivityID
            INNER JOIN Orders o ON o.OrderID = i.OrderID
      WHERE i.AmountPaid >= (SELECT FullPrice FROM Activities WHERE
ActivityID = i.ActivityID)
END;
```

Autor: Stanisław Strojniak

Automatyczne dodanie studenta do kursu i jego modułów po zakupieniu

```
CREATE TRIGGER trg_AddStudentToCourseModules
ON OrderDetails
AFTER INSERT,UPDATE
AS
BEGIN
  SET NOCOUNT ON;

  INSERT INTO CourseModulesDetails (ModuleID, StudentID,
PassedDate)
  SELECT
      cm.ModuleID,
      o.StudentID,
      NULL
  FROM
      CourseModules cm
  INNER JOIN
      Inserted i ON cm.CourseID = i.ActivityID
  INNER JOIN
      Orders o ON o.OrderID = i.OrderID
  WHERE i.AmountPaid >= (SELECT FullPrice FROM Activities WHERE
ActivityID = i.ActivityID) AND
dbo.GetCourseAvailableSeats(CM.CourseID) > 0;

END;
```

Autor: Kamil Życzkowski

Automatyczne dodanie studenta do studiów i spotkań studyjnych po zakupieniu

```
CREATE TRIGGER trg_AddStudentToStudies
ON OrderDetails
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    CREATE TABLE #TempMeetings (
        StudentID INT,
```

```
        MeetingID INT,
           StudiesID INT,
        CumulativeCost DECIMAL(10, 2)
    );

    INSERT INTO #TempMeetings (StudentID, MeetingID, StudiesID,
CumulativeCost)
    SELECT
        O.StudentID,
        SM.MeetingID,
           S.StudiesID,
        SUM(SM.DiscountPrice) OVER (PARTITION BY O.StudentID ORDER
BY SM.MeetingDate ROWS UNBOUNDED PRECEDING) + S.EntryFee AS
CumulativeCost
    FROM inserted i
    INNER JOIN Orders O ON O.OrderID = i.OrderID
      INNER JOIN OrderDetails OD ON OD.OrderID=O.OrderID
    INNER JOIN Activities A ON A.ActivityID = i.ActivityID
    INNER JOIN Studies S ON S.StudiesID = A.ActivityID
    INNER JOIN Subjects Sub ON Sub.StudiesID = S.StudiesID
    INNER JOIN StudiesMeetings SM ON Sub.SubjectID = SM.SubjectID
    WHERE
        i.AmountPaid >= S.EntryFee
        AND DATEADD(DAY, 3, OD.PaidDate) <= SM.MeetingDate
    ORDER BY
        SM.MeetingDate;

    INSERT INTO StudiesMeetingsDetails (StudentID, MeetingID)
    SELECT
        TM.StudentID,
        TM.MeetingID
    FROM
        #TempMeetings TM
    INNER JOIN Orders O ON O.StudentID = TM.StudentID
    WHERE TM.CumulativeCost <= (SELECT i.AmountPaid FROM inserted
i WHERE i.OrderID = O.OrderID)
        AND NOT EXISTS (
            SELECT 1
            FROM StudiesMeetingsDetails SMD
            WHERE SMD.StudentID = TM.StudentID AND SMD.MeetingID =
TM.MeetingID
        )
```

```
            AND dbo.GetStudiesAvailableSeats(TM.StudiesID) > 0;

    DROP TABLE #TempMeetings;
END;
```

Autor: Jakub Zając

Automatyczne dodanie studenta do pojedynczego spotkania studyjnego po zakupieniu

```
CREATE TRIGGER trg_AddStudentToStudyMeeting
      ON OrderDetails
      AFTER INSERT, UPDATE AS
      BEGIN
      SET NOCOUNT ON;
      INSERT INTO StudiesMeetingsDetails (MeetingID, StudentID)
      SELECT sm.MeetingID, o.StudentID
      FROM Inserted i
                  INNER JOIN Activities a ON a.ActivityID =
i.ActivityID
                  INNER JOIN StudiesMeetings sm ON sm.ActivityID =
a.ActivityID
                  INNER JOIN Orders o ON o.OrderID = i.OrderID
      WHERE i.AmountPaid >= (SELECT FullPrice FROM Activities WHERE
ActivityID = i.ActivityID)
      END;
```

Autor: Stanisław Strojniak

Uprawnienia

Administrator

```
create role administrator
grant all privileges on u_jzajac.dbo to administrator
```

Pracownik sekretariatu

```
CREATE ROLE SecretaryWorker
GRANT SELECT ON FINANCIAL_REPORT TO SecretaryWorker;
GRANT SELECT ON COURSES_FINANCIAL_REPORT TO SecretaryWorker;
```

```
GRANT SELECT ON STUDIES_FINANCIAL_REPORT TO SecretaryWorker;
GRANT SELECT ON WEBINARS_FINANCIAL_REPORT TO SecretaryWorker;
GRANT SELECT ON LIST_OF_DEBTORS TO SecretaryWorker;
GRANT SELECT ON PeopleSignedForUpcomingEvents TO SecretaryWorker;
GRANT SELECT ON PeopleSignedForUpcomingCourses TO SecretaryWorker;
GRANT SELECT ON PeopleSignedForUpcomingStudiesMeetings TO
SecretaryWorker;
GRANT SELECT ON PeopleSignedForUpcomingWebinars TO
SecretaryWorker;
GRANT SELECT ON STUDENTS_DATA TO SecretaryWorker;
GRANT SELECT ON LECTURERS_DATA TO SecretaryWorker;
GRANT SELECT ON TRANSLATORS_DATA TO SecretaryWorker;
GRANT SELECT ON SECRETARY_WORKERS_DATA TO SecretaryWorker;
GRANT SELECT ON INTERNSHIP_SUPERVISORS_DATA TO SecretaryWorker;
GRANT SELECT ON  DIRECTORS_DATA TO SecretaryWorker;
GRANT SELECT ON  ADMINISTRATORS_DATA TO SecretaryWorker;
GRANT SELECT, UPDATE, INSERT ON Students TO SecretaryWorker;
GRANT SELECT, UPDATE, INSERT ON Orders TO SecretaryWorker;
GRANT SELECT, UPDATE, INSERT ON OrderDetails TO SecretaryWorker;
```

Autor: Jakub Zając

Student

```
CREATE ROLE Student;
GRANT SELECT ON UpcomingCourseModules TO Student;
GRANT SELECT ON UpcomingStudiesMeetings TO Student;
GRANT SELECT ON UpcomingEvents TO Student;
GRANT SELECT ON UpcomingWebinars TO Student;
GRANT EXECUTE ON GetScheduleForStudent TO Student;
GRANT EXECUTE ON GetScheduleForCourse TO Student;
GRANT EXECUTE ON GetScheduleForStudies TO Student;
```

Autor: Kamil Życzkowski

Tłumacz

```
CREATE ROLE Translator
GRANT SELECT ON ALL_FUTURE_EVENTS TO Translator
GRANT SELECT ON Webinars TO Translator
GRANT SELECT ON CourseModules TO Translator
GRANT SELECT ON StudiesMeetings TO Translator
```

Autor: Jakub Zając

Wykładowca

```
CREATE ROLE Lecturer;
GRANT SELECT ON AttendanceListAllEvents TO Lecturer;
GRANT SELECT ON AttendanceListCourseModules TO Lecturer;
GRANT SELECT ON AttendanceListStudiesMeetings TO Lecturer;
GRANT SELECT ON AttendanceListWebinars TO Lecturer;
GRANT SELECT ON CoursesAttendanceSummary TO Lecturer;
GRANT SELECT ON EventsAttendanceSummary TO Lecturer;
GRANT SELECT ON StudiesMeetingAttendanceSummary TO Lecturer;
GRANT SELECT ON UpcomingCourseModules TO Lecturer;
GRANT SELECT ON UpcomingEvents TO Lecturer;
GRANT SELECT ON UpcomingStudiesMeetings TO Lecturer;
GRANT SELECT ON UpcomingWebinars TO Lecturer;
```

Autor: Kamil Życzkowski

Prowadzący praktyki

```
CREATE ROLE InternshipSupervisor
GRANT EXECUTE ON AddInternship TO InternshipSupervisor
GRANT SELECT, INSERT, UPDATE ON Internships TO
InternshipSupervisor
GRANT SELECT, INSERT, UPDATE ON InternshipDetails TO
InternshipSupervisor
GRANT EXECUTE ON dbo.CheckInternshipPassed TO
InternshipSupervisor;
```

Autor: Jakub Zając

Indeksy

Tabela Users

```
CREATE INDEX Users_CityID ON Users (CityID)
```

Tabela Cities

```
CREATE INDEX Cities_CountryID ON Cities (CountryID)
```

Tabela TranslatorLanguages

```
CREATE INDEX TranslatorLanguages_TranslatorID ON
TranslatorLanguages (TranslatorID)
CREATE INDEX TranslatorLanguages_LanguageID ON TranslatorLanguages
(LanguageID)
```

Tabela Webinars

```
CREATE INDEX Webinars_LecturerID ON Webinars (LecturerID)
CREATE INDEX Webinars_TranslatorID ON Webinars (TranslatorID)
CREATE INDEX Webinars_LanguageID ON Webinars (LanguageID)
```

Tabela WebinarDetails

```
CREATE INDEX WebinarDetails_WebinarID ON WebinarDetails
(WebinarID)
CREATE INDEX WebinarDetails_StudentID ON WebinarDetails
(StudentID)
```

Tabela Courses

```
CREATE INDEX Courses_CourseTypeID ON Courses (CourseTypeID)
```

Tabela CourseModules

```
CREATE INDEX CourseModules_CourseID ON CourseModules (CourseID)
CREATE INDEX CourseModules_LecturerID ON CourseModules
(LecturerID)
CREATE INDEX CourseModules_LanguageID ON CourseModules
(LanguageID)
CREATE INDEX CourseModules_TranslatorID ON CourseModules
(TranslatorID)
```

Tabela CourseModulesDetails

```
CREATE INDEX CourseModulesDetails_ModuleID ON CourseModulesDetails
```

```
(ModuleID)
CREATE INDEX CourseModulesDetails_StudentID ON
CourseModulesDetails (StudentID)
```

Tabela OnlineAsyncModules

```
CREATE INDEX OnlineAsyncModules_ModuleID ON OnlineAsyncModules
(ModuleID)
```

Tabela OnlineSyncModules

```
CREATE INDEX OnlineSyncModules_ModuleID ON OnlineSyncModules
(ModuleID)
```

Tabela OnsiteModules

```
CREATE INDEX OnsiteModules_ModuleID ON OnsiteModules (ModuleID)
CREATE INDEX OnsiteModules_RoomID ON OnsiteModules (RoomID)
```

Tabela Orders

```
CREATE INDEX Orders_StudentID ON Orders (StudentID)
```

Tabela OrderDetails

```
CREATE INDEX OrderDetails_OrderID ON OrderDetails (OrderID)
CREATE INDEX OrderDetails_ActivityID ON OrderDetails (ActivityID)
```

Tabela ShoppingCart

```
CREATE INDEX ShoppingCart_StudentID ON ShoppingCart (StudentID)
CREATE INDEX ShoppingCart_ActivityID ON ShoppingCart (ActivityID)
```

Tabela Subjects

```
CREATE INDEX Subjects_StudiesID ON Subjects (StudiesID)
CREATE INDEX Subjects_LecturerID ON Subjects (LecturerID)
```

Tabela Internships

```
CREATE INDEX Internships_SupervisorID ON Internships
(SupervisorID)
CREATE INDEX Internships_StudiesID ON Internships (StudiesID)
```

Tabela StudiesMeetings

```
CREATE INDEX StudiesMeetings_ActivityID ON StudiesMeetings
(ActivityID)
CREATE INDEX StudiesMeetings_SubjectID ON StudiesMeetings
(SubjectID)
CREATE INDEX StudiesMeetings_LanguageID ON StudiesMeetings
(LanguageID)
CREATE INDEX StudiesMeetings_TranslatorID ON StudiesMeetings
(TranslatorID)
```

Tabela OnlineAsyncMeetings

```
CREATE INDEX OnlineAsyncMeetings_MeetingID ON OnlineAsyncMeetings
(MeetingID)
```

Tabela OnlineSyncMeetings

```
CREATE INDEX OnlineSyncMeetings_MeetingID ON OnlineSyncMeetings
(MeetingID)
```

Tabela OnsiteMeetings

```
CREATE INDEX OnsiteMeetings_MeetingID ON OnsiteMeetings
(MeetingID)
CREATE INDEX OnsiteMeetings_RoomID ON OnsiteMeetings (RoomID)
```

Testowanie działania indeksów z użyciem optymalizatora kosztowego

Testowanie działania indeksów przeprowadziliśmy uruchamiając indeksowane zapytanie SQL wiele razy, zarówno bez indeksu, jak i z indeksem. Czas wykonania w przypadku istnienia indeksu powinien być mniejszy. Te różnice są jednak rzędu milisekund, więc dlatego dla zaprezentowania różnicy, można je wykonać wielokrotnie. Poniżej znajduje się przykład działania dla indeksu OnsiteMeetings_RoomID w przypadku wywołania 100 zapytań SELECT.

```
DECLARE @i INT=0
WHILE @i < 100
BEGIN
      SELECT * FROM OnsiteMeetings
      WHERE RoomID = 5
      SET @i = @i + 1
END;
```

Czas wykonania przed dodaniem indeksów:

	Trial 10	
Rows affected by INSERT, DELETE, or UPDATE statements	0	→
Number of SELECT statements	301	↑
Rows returned by SELECT statements	401	↑
Number of transactions	0	→
Network Statistics		
Number of server roundtrips	3	→
TDS packets sent from client	3	→
TDS packets received from server	216	↑
Bytes sent from client	402	↑
Bytes received from server	875214	↑
Time Statistics		
Client processing time	523	↑
Total execution time	619	↑
Wait time on server replies	96	↑

Czas wykonania po dodaniu indeksów:

	Trial 10	
Rows affected by INSERT, DELETE, or UPDATE statements	0	→
Number of SELECT statements	301	↑
Rows returned by SELECT statements	401	↑
Number of transactions	0	→
Network Statistics		
Number of server roundtrips	3	→
TDS packets sent from client	3	→
TDS packets received from server	205	↑
Bytes sent from client	402	↑
Bytes received from server	830894	↑
Time Statistics		
Client processing time	393	↑
Total execution time	513	↑
Wait time on server replies	120	↑

Jak można zauważyć, wykonanie 100 zapytań SELECT po dodaniu indeksów jest o około 100 milisekund szybsze.