

Podstawy Baz Danych Mini Projekt

XX.XX.2024

Krzysztof Chmielewski

Szymon Migas

Wiktor Sędzimir,

Informatyka AGH, rok II

Użyte programy:

Generowanie schematu bazy danych: Vertabelo

Generowanie danych - Python3, Geonames API

Użyte biblioteki:

- **pyodbc**

Geonames API - uzyskiwanie informacji o krajach oraz miastach.

Do wygenerowania przykładowych danych do bazy danych wykorzystano język python oraz bibliotekę *pyodbc* do nawiązania połączenia z serwerem bazodanowym. Dla każdej tabeli napisano osobną funkcję wypełniającą tabelę zadaną liczbą rekordów. Dane o istniejących państwach zostały pobrane z wykorzystaniem *Geonames API*, natomiast dla pozostałych tabel kluczowe dane (takie jak nazwa kursu, nazwa studiów czy przedmiotu na studiach) były umieszczane w plikach *.json*, które następnie przygotowany skrypt w pythonie wczytywał i uzupełniał o losowe wartości wygenerowane z wykorzystaniem biblioteki *random*. Aby zachować integrację pomiędzy danymi, dane które miały uzupełniać kolumny będące kluczami obcymi czytano z już istniejących i wypełnionych tabel, do których odwoływały się dane klucze obce, a następnie wybierano losowe wartości z tych kolumn zgodnie z warunkami integralności. Kluczowe więc było zachowanie odpowiedniej kolejności podczas generowania danych.

Użytkownicy systemu

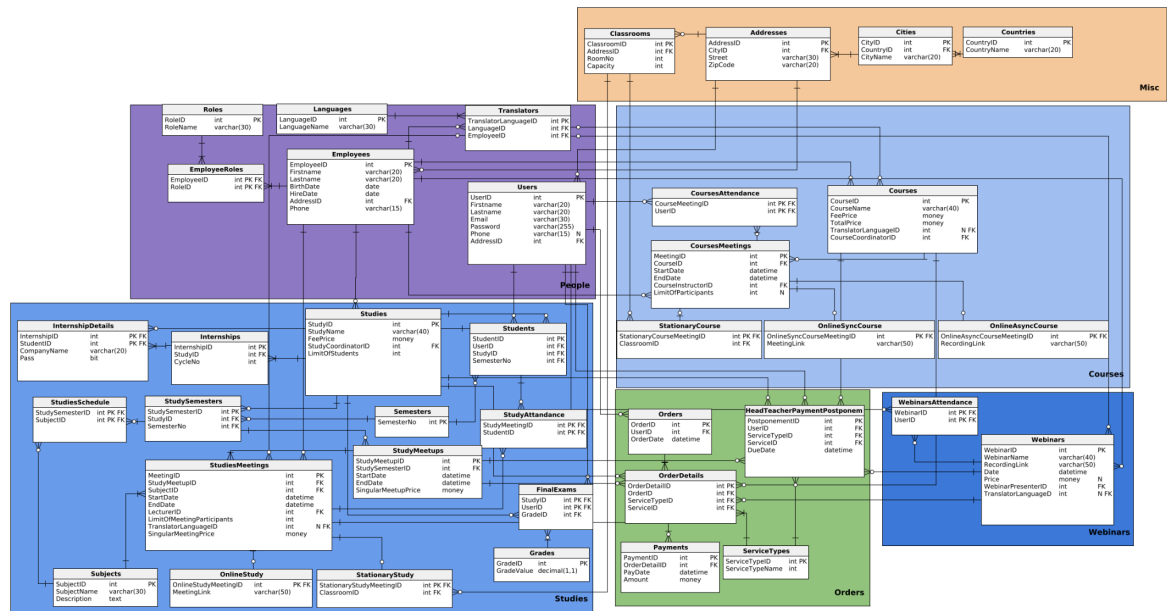
- Administrator systemu
- Dyrektor Szkoły
- Koordynator kursu
- Koordynator studiów
- Prowadzący kursu
- Prowadzący webinaru
- Wykładowca
- Pracownik administracyjny
- Tłumacz
- Uczestnik
- Student
- Osoba niezarejestrowana

Propozycja funkcji realizowanych przez system i użytkowników

- Zarządzanie kontami i uprawnieniami
 - Utworzenie konta użytkownika - **Osoba niezarejestrowana**
 - Usunięcie konta użytkownika - **Uczestnik**
 - Utworzenie i usunięcie konta Koordynatora - **Pracownik administracyjny**
 - Utworzenie i usunięcie konta Prowadzącego - **Pracownik administracyjny**
 - Utworzenie i usunięcie konta Pracownika administracyjnego - **Dyrektor Szkoły**
 - Utworzenie i usunięcie konta Dyrektora Szkoły - **Administrator Systemu**
- Zarządzanie dostępnością usług
 - Umożliwienie przeglądania dostępnych usług dowolnej osobie posiadającej i nie posiadającej konta - **System**
 - Blokowanie dostępu do usług (w chwili nieuiszczenia opłaty przez użytkownika w danym terminie) - **System**
 - Weryfikacja możliwości zapisu na dane wydarzenie (np. blokada możliwości zapisu jednego użytkownika dwa razy na to samo wydarzenie) - **System**
- Zarządzanie usługami
 - Dodawanie, modyfikowanie oraz usuwanie studium - **Koordynator studiów**
 - Dodawanie, modyfikowanie oraz usuwanie webinarów - **Koordynator webinarów**
 - Dodawanie, modyfikowanie oraz usuwanie kursów - **Koordynator kursu**
 - Dodawania sylabusu dla danego studium przed jego rozpoczęciem - **Koordynator studiów**
 - Uniemożliwienie modyfikacji sylabusu - **System**
 - Tworzenie oraz modyfikacja harmonogramu spotkań studiów - **Koordynator studiów**
 - Umożliwia odrabianie zajęć, pod warunkiem uczestnictwa w zajęciach komercyjnych o zbliżonej tematyce - **Prowadzący**
- Zarządzanie dostępem do nagrań webinarów
 - Weryfikacja dostępu do nagrań - czy użytkownik posiada konto oraz czy zapłacił za nagranie (w przypadku płatnych webinarów) - **System**
 - Umożliwienie dostępu do nagrań na okres 30 dni - **System**
- Zarządzanie frekwencją i statusem zaliczenia
 - Rejestrowanie obecności dla kursów, studiów - **Prowadzący**

- Zaliczanie modułu prowadzonego online asynchronicznie - **System**
- Limitowanie liczby osób na danym wydarzeniu - **System**
- Raporty
 - Generowanie raportu przychodów generowanych przez dany webinar, kurs, studium - **Pracownik administracyjny**
 - Generowanie raportu osób, które nie zapłaciły za usługi, ale z nich korzystały - **Pracownik administracyjny**
 - Generowanie raportu o zakupie usług przez użytkownika oraz statusie ich zapłacenia, wraz z datą zatwierdzenia płatności - **Pracownik administracyjny, Uczestnik**
 - Generowanie raportu bilokacji - tworzenie listy osób, które są zapisane na wydarzenia kolidujące ze sobą czasowo - **Pracownik administracyjny**
 - Generowanie raportu dla każdego szkolenia z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie - **Pracownik administracyjny**
 - Generowanie raportu osób zapisanych na przyszłe zajęcia z informacją o tym czy są to zajęcia online czy nie - **Pracownik administracyjny**
- Płatności
 - Tworzenie koszyków zakupów dla użytkowników - **System**
 - Dodawanie i usuwanie elementów z koszyka zakupów - **Uczestnik**
 - Generowanie linków do płatności z informacją zwrotną o powodzeniu/niepowodzeniu operacji - **System**
 - W przypadku studiów wymaganie wpłaty wpisowego oraz wpłaty za dany zjazd - **System**
 - Obsługa wyjątku jakim jest odraczanie wpłaty - **Dyrektor Szkoły**

Schemat bazy danych



Opis tabel

Lokalizacje

Countries (wzorzec)

Tabela *Country* jest tabelą słownikową przechowującą informacje o wszystkich istniejących państwach:

- **CountryID** (klucz główny) [NOT NULL][int] - unikalne ID identyfikujące dany kraj
- **CountryName** [NOT NULL][varchar(50)]- nazwa danego kraju

Warunki integralności:

- CountryName jest unikatowe

Kod SQL:

```
CREATE TABLE Countries (  
    CountryID int NOT NULL,  
    CountryName varchar(50) NOT NULL,  
    CONSTRAINT unique_country_name UNIQUE (CountryName),  
    CONSTRAINT Countries_pk PRIMARY KEY (CountryID)  
);
```

Cities

Tabela *Cities* jest tabelą słownikową przechowującą informacje o miastach znajdujących się w danych państwie:

- **CityID** (klucz główny) [NOT NULL][int] - unikalne ID identyfikujące kombinację miasta z państwem
- **CountryID** (klucz obcy do **CountryID** z tabeli *Countries*) [NOT NULL][int] - ID danego państwa
- **CityName** [NOT NULL][varchar(50)] - nazwa danego miasta

Warunki integralności:

- Kombinacja (CountryID, CityName) musi być unikatowa

Kod SQL:

```
CREATE TABLE Cities (  
    CityID int NOT NULL,  
    CountryID int NOT NULL,  
    CityName varchar(50) NOT NULL,  
    CONSTRAINT unique_city_country_combination UNIQUE  
    (CountryID, CityName),  
    CONSTRAINT Cities_pk PRIMARY KEY (CityID)  
);
```

Addresses

Tabela *Addresses* przechowuje informacje o adresach pracowników, użytkowników oraz sal, w których odbywają się zajęcia:

- **AddressID** (klucz główny) [NOT NULL][int]- unikalne ID identyfikujące adres
- **CityID** (klucz obcy do **CityID** z tabeli *Cities*) [NOT NULL][int] - ID danego miasta
- **Street** [NOT NULL][varchar(20)] - nazwa ulicy
- **ZipCode** [NOT NULL][varchar(20)] - kod pocztowy danej lokalizacji

Kod SQL:

```
CREATE TABLE Addresses (  
    AddressID int NOT NULL,  
    CityID int NOT NULL,  
    Street varchar(30) NOT NULL,  
    ZipCode varchar(20) NOT NULL,  
    CONSTRAINT Addresses_pk PRIMARY KEY (AddressID)  
);
```


Classrooms

Tabela *Classrooms* przechowuje informacje o klasach, w których odbywają się zajęcia stacjonarne:

- **ClassroomID** (klucz główny) [NOT NULL][int] - unikalne ID identyfikujące klasę
- **AddressID** (klucz obcy do **AddressID** z tabeli *Addresses*) [NOT NULL][int] - ID adresu, pod którym znajduje się dana klasa
- **RoomNo** [NOT NULL][int] - numer klasy
- **Capacity** [NOT NULL][int] - pojemność klasy, jako liczba dostępnych miejsc w klasie

Warunki integralności:

- Capacity jest większe od 0
- RoomNo jest większe od 0

Kod SQL:

```
CREATE TABLE Classrooms (  
    ClassroomID int NOT NULL,  
    AddressID int NOT NULL,  
    RoomNo int NOT NULL,  
    Capacity int NOT NULL,  
    CONSTRAINT CapacityGtZero CHECK (Capacity > 0),  
    CONSTRAINT RoomNoGtZero CHECK (RoomNo > 0),  
    CONSTRAINT Classrooms_pk PRIMARY KEY (ClassroomID)  
);
```

Ludzie

Roles

Tabela *Roles* przechowuje informacje stanowiskach jakie mogą pełnić pracownicy w firmie klienta:

- **RoleID** (klucz główny) [NOT NULL] [int] - unikalne ID identyfikujące rolę
- **RoleName** [NOT NULL] [varchar(30)] - nazwa stanowiska w firmie

Kod SQL:

```
CREATE TABLE Roles (  
    RoleID int NOT NULL,  
    RoleName varchar(30) NOT NULL,  
    CONSTRAINT Roles_pk PRIMARY KEY (RoleID)  
);
```

Employees

Tabela *Employees* przechowuje informacje o pracownikach firmy klienta:

- **EmployeeID** (klucz główny) [NOT NULL] [int] - unikalne ID identyfikujące danego pracownika
- **Firstname** [NOT NULL] [varchar(20)] - imię pracownika
- **Lastname** [NOT NULL] [varchar(20)] - nazwisko pracownika
- **BirthDate** [NOT NULL] [date] - data urodzenia pracownika
- **HireDate** [NOT NULL] [date] - data zatrudnienia pracownika
- **AddressID** [NOT NULL] [int] (klucz obcy do **AddressID** z tabeli *Addresses*) - adres zamieszkania pracownika
- **Phone** [NOT NULL] [varchar(15)] - numer telefonu pracownika

Warunki integralności:

- BirthDate musi być nie wcześniejsze niż 1 stycznia 1900 roku i nie późniejsze niż obecna data
- HireDate musi być nie wcześniejsze niż BirthDate i nie późniejsza niż obecna data

Kod SQL:

```
CREATE TABLE Employees (  
    EmployeeID int NOT NULL,  
    Firstname varchar(20) NOT NULL,  
    Lastname varchar(20) NOT NULL,  
    BirthDate date NOT NULL,  
    HireDate date NOT NULL,  
    AddressID int NOT NULL,  
    Phone varchar(15) NOT NULL,  
    CONSTRAINT ValidBirthDate CHECK (BirthDate >= '1900-01-01'  
and BirthDate <= GETDATE()),  
    CONSTRAINT ValidHireDate CHECK (HireDate > BirthDate and  
HireDate <= GETDATE()),  
    CONSTRAINT Employees_pk PRIMARY KEY (EmployeeID)  
);
```

EmployeeRoles

Tabela *EmployeeRoles* przechowuje informacje o rolach stanowiskach pełnionych przez pracowników jako kombinacja *pracownik-stanowisko*.

- **EmployeeID** [NOT NULL] [int] (klucz obcy do **EmployeeID** z tabeli *Employees* i jednocześnie element klucza głównego) - ID pracownika
- **RoleID** [NOT NULL] [int] (klucz obcy do **RoleID** z tabeli *Roles* i jednocześnie element klucza głównego) - ID stanowiska

Languages

Tabela *Languages* jest tabelą słownikową przechowującą informacje o możliwych językach.

- **LanguageID** (klucz główny) [NOT NULL] [int] - unikalne ID identyfikujące dany język
- **LanguageName** [NOT NULL] [varchar(30)] - nazwa danego języka

Warunki integralności

- LanguageName musi być unikatowe

Kod SQL:

```
CREATE TABLE Languages (  
    LanguageID int NOT NULL,  
    LanguageName varchar(30) NOT NULL,  
    CONSTRAINT LanguagesIsUnique UNIQUE (LanguageName),  
    CONSTRAINT Languages_pk PRIMARY KEY (LanguageID)  
);
```

Translators

Tabela *Translators* przechowuje informację o wszystkich osobach zatrudnionych jako tłumacz, jako kombinacja *język-osoba*:

- **TranslatorLanguageID** (klucz główny) [NOT NULL] [int] - identyfikator połączenia tłumacza z językiem
- **LanguageID** (klucz obcy do **LanguageID** z tabeli *Languages*) [NOT NULL] [int] - ID konkretnego języka dostępnego w szkole
- **EmployeeID** (klucz obcy do **EmployeeID** z tabeli *Employees*) [NOT NULL] [int] - ID konkretnej zatrudnionej przez szkołę osoby

Kod SQL:

```
CREATE TABLE Translators (  
    TranslatorLanguageID int NOT NULL,  
    LanguageID int NOT NULL,  
    EmployeeID int NOT NULL,  
    CONSTRAINT Translators_pk PRIMARY KEY  
    (TranslatorLanguageID)  
);
```

Users

Tabela *Users* przechowuje informację o zarejestrowanych użytkownikach, czyli posiadających konto.


- **UserID** (klucz główny) [NOT NULL] [int] - unikalne ID identyfikujące użytkownika
- **Firstname** [NOT NULL] [varchar(20)] - imię użytkownika
- **Lastname** [NOT NULL] [varchar(20)] - nazwisko użytkownika
- **Email** [NOT NULL] [varchar(50)] - adres email użytkownika stanowiący jednocześnie login dla konta
- **Password** [NOT NULL] [varchar(255)] - hasło użytkownika przechowywane w zaszyfrowanej postaci
- **Phone** (nullable) [varchar(15)] - numer telefonu użytkownika
- **AddressID** (klucz obcy do **AddressID** z tabeli *Addresses*) [NOT NULL] [int] - adres użytkownika; na ten adres zostanie wysłany certyfikat w przypadku pozytywnego ukończenia szkolenia

Warunki integralności:

- Email musi składać się z następującej sekwencji: co najmniej jednego znaku, "@", co najmniej dwóch znaków, ".", co najmniej dwóch znaków.
- Email jest unikatowy
- Numer telefonu jest unikatowy

Kod SQL:

```
CREATE TABLE Users (  
    UserID int NOT NULL,  
    Firstname varchar(20) NOT NULL,  
    Lastname varchar(20) NOT NULL,  
    Email varchar(50) NOT NULL,  
    Password varchar(255) NOT NULL,  
    Phone varchar(15) NULL,  
    AddressID int NOT NULL,  
    CONSTRAINT UniqueEmail UNIQUE (Email),  
    CONSTRAINT UniquePhone UNIQUE (Phone),  
    CONSTRAINT ValidEmail CHECK (Email LIKE '%_@__%.__%'),
```



```
CONSTRAINT Users_pk PRIMARY KEY (UserID)  
);
```

Kursy

Courses

Tabela *Courses* zawiera ogólne informacje o danym kursie:

- **CourseID** (klucz główny) [NOT NULL] [int] - unikatowy identyfikator kursu
- **CourseName** [NOT NULL] [varchar(40)] - nazwa danego kursu
- **FeePrice** [NOT NULL] [money] - zawiera informację jaka jest kwota zaliczki za dany kurs
- **TotalPrice** [NOT NULL] [money] - zawiera informację jaki jest cały koszt danego kursu
- **TranslatorLanguageID** (klucz obcy do **TranslatorLanguageID** z tabeli *Translators*) (nullable) [int] - ID konkretnego tłumacza związanego z danym językiem, który tłumaczy (kolumna jest nullable, ponieważ kurs prowadzony w języku polskim nie wymaga tłumacza)
- **CourseCoordinatorID** (klucz obcy do **EmployeeID** z tabeli *Employees*) [NOT NULL] [int] - ID konkretnego pracownika, który jest koordynatorem danego kursu

Warunki integralności:

- CourseName jest unikatowe
- FeePrice jest większe od 0
- TotalPrice jest większe od FeePrice

Kod SQL:

```
CREATE TABLE Courses (
    CourseID int NOT NULL,
    CourseName varchar(40) NOT NULL,
    FeePrice money NOT NULL,
    TotalPrice money NOT NULL,
    TranslatorLanguageID int NULL,
    CourseCoordinatorID int NOT NULL,
    CONSTRAINT FeePriceLtTotalPrice CHECK (FeePrice <
TotalPrice),
    CONSTRAINT FeePriceGtZero CHECK (FeePrice > 0),
    CONSTRAINT Courses_pk PRIMARY KEY (CourseID));
```


CoursesAttendance

Tabela *CoursesAttendance* pozwala wyznaczyć obecność danego użytkownika na danym kursie. W ten sposób można pobrać listę obecności użytkowników na danym kursie.

- **CourseMeetingID** (element klucza głównego i klucz obcy do **CourseMeetingID** z tabeli *CoursesMeetings*) [NOT NULL] [int] - unikatowe ID konkretnego spotkania odbywającego się w ramach danego kursu
- **UserID** (element klucza głównego i klucz obcy do **UserID** z tabeli *Users*) [NOT NULL] [int] - ID użytkownika, który uczestniczył w danym spotkaniu

Kod SQL:

```
CREATE TABLE CoursesAttendance (
    CourseMeetingID int NOT NULL,
    UserID int NOT NULL,
    CONSTRAINT CoursesAttendance_pk PRIMARY KEY
    (CourseMeetingID,UserID)
);
```

CoursesMeetings

Tabela *CoursesMeetings* przechowuje informacje o spotkaniach odbywających się w ramach konkretnego kursu.

- **MeetingID** (klucz główny) [NOT NULL] [int] - unikatowe ID spotkania
- **CourseID** (klucz obcy do **CourseID** z tabeli *Courses*) [NOT NULL] [int] - ID kursu, w ramach którego jest realizowane spotkanie
- **StartDate** [NOT NULL] [datetime] - data i czas rozpoczęcia danego spotkania
- **EndDate** [NOT NULL] [datetime] - data i czas końca danego spotkania
- **CourseInstructorID** (klucz obcy do **EmployeeID** z tabeli *Employees*) [NOT NULL] [int] - ID konkretnego pracownika, który jest instruktorem na danym spotkaniu
- **LimitOfParticipants** (nullable) [int] - limit uczestników na danym kursie

Warunki integralności:

- StartDate jest wcześniejsze od EndDate
- LimitOfParticipants większe od 0

Kod SQL:

```
CREATE TABLE CoursesMeetings (  
    MeetingID int NOT NULL,  
    CourseID int NOT NULL,  
    StartDate datetime NOT NULL,  
    EndDate datetime NOT NULL,  
    CourseInstructorID int NOT NULL,  
    LimitOfParticipants int NULL,  
    CONSTRAINT LimitOfParticipantsGtZero CHECK  
(LimitOfParticipants > 0),  
    CONSTRAINT EndDateGtStartDateCM CHECK (EndDate >  
StartDate),  
    CONSTRAINT CoursesMeetings_pk PRIMARY KEY (MeetingID)  
);
```



StationaryCourse

Tabela *StationaryCourse* przechowuje informacje o stacjonarnych spotkaniach dla kursów. Każdy rekord z tej tabeli odpowiada dokładnie jednemu rekordowi z tabeli *CourseMeetings*.

- **StationaryCourseMeetingID** (klucz główny i klucz obcy z tabeli *CourseMeetings*) [NOT NULL] [int] - identyfikator stacjonarnego spotkania
- **ClassroomID** (klucz obcy do **ClassroomID** z tabeli *Classrooms*) [NOT NULL] [int] - identyfikator pomieszczenia, w którym odbywa się dane spotkanie

Kod SQL:

```
CREATE TABLE StationaryCourse (  
    StationaryCourseMeetingID int NOT NULL,  
    ClassroomID int NOT NULL,  
    CONSTRAINT StationaryCourse_pk PRIMARY KEY  
    (StationaryCourseMeetingID)  
);
```

OnlineSyncCourse

Tabela *OnlineSyncCourse* przechowuje informacje o synchronicznych spotkaniach online dla kursów. Każdy rekord z tej tabeli odpowiada dokładnie jednemu rekordowi z tabeli *CourseMeetings*.

- **OnlineSyncCourseMeetingID** (klucz główny i klucz obcy do **MeetingID** z tabeli *CourseMeetings*) [NOT NULL] [int] - identyfikator spotkania online synchronicznego
- **MeetingLink** [NOT NULL] [varchar(100)] - link do spotkania online

Kod SQL:

```
CREATE TABLE OnlineSyncCourse (  
    OnlineSyncCourseMeetingID int NOT NULL,  
    MeetingLink varchar(100) NOT NULL,  
    CONSTRAINT OnlineSyncCourse_pk PRIMARY KEY  
    (OnlineSyncCourseMeetingID)  
);
```

OnlineAsyncCourse

Tabela *OnlineAsyncCourse* przechowuje informacje o asynchronicznych spotkaniach online dla kursów. Każdy rekord z tej tabeli odpowiada dokładnie jednemu rekordowi z tabeli *CourseMeetings*.

- **OnlineAsyncCourseMeetingID** (klucz główny i klucz obcy do **MeetingID** z tabeli *CourseMeetings*) [NOT NULL] [int] - identyfikator spotkania online asynchronicznego
- **RecordingLink** [NOT NULL] [varchar(100)] - link do nagrania ze spotkania

Kod SQL:

```
CREATE TABLE OnlineAsyncCourse (  
    OnlineAsyncCourseMeetingID int NOT NULL,  
    RecordingLink varchar(100) NOT NULL,  
    CONSTRAINT OnlineAsyncCourse_pk PRIMARY KEY  
    (OnlineAsyncCourseMeetingID)  
);
```

Webinary

Webinars

Tabela *Webinars* przechowuje wszystkie informacje dotyczące danego webinaru


- **WebinarID** (klucz główny) [NOT NULL][int] - unikatowy identyfikator danego webinaru
- **WebinarName** [NOT NULL][varchar(80)] - nazwa danego webinaru
- **RecordingLink** [NOT NULL][varchar(100)] - link do spotkania danego webinaru
- **Date** [NOT NULL][datetime] - data i godzina rozpoczęcia webinaru
- **Price** (nullable) [NULL][money] - kwota do zapłaty za dany webinar, w przypadku wartości null oznacza, że jest to webinar darmowy
- **WebinarPresenterID** (klucz obcy do **EmployeeID** z tabeli *Employees*) [NOT NULL][int]- identyfikator pracownika, który prowadzi dany webinar
- **TranslatorLanguageID** (nullable, klucz obcy) [NULL][int]- ID konkretnego tłumacza związanego z danym językiem, który tłumaczy (kolumna jest nullable, ponieważ webinar prowadzony w języku polskim nie wymaga tłumacza)

Warunki integralności:

- WebinarName jest unikatowe
- RecordingLink jest unikatowe
- Price jest większe od 0

Kod SQL:

```
CREATE TABLE Webinars (  
    WebinarID int NOT NULL,  
    WebinarName varchar(80) NOT NULL,  
    RecordingLink varchar(100) NOT NULL,  
    Date datetime NOT NULL,  
    Price money NULL,  
    WebinarPresenterID int NOT NULL,  
    TranslatorLanguageID int NULL,  
    CONSTRAINT WebinarNameUnique UNIQUE (WebinarName),  
    CONSTRAINT RecordingLinkUnique UNIQUE (RecordingLink),
```



```
CONSTRAINT WebinarPriceGtZero CHECK (Price > 0),  
CONSTRAINT Webinars_pk PRIMARY KEY (WebinarID)  
);
```


WebinarsAttendance

Tabela *WebinarsAttendance* a zawiera informację o parach webinar-użytkownik, jeśli para znajduje się w tej tabeli to znaczy, że dany użytkownik był obecny na danym webinarze.

- **WebinarID** (klucz obcy do **WebinarID** z tabeli *Webinars* i jednocześnie element klucza głównego) [NOT NULL][int] - identyfikator konkretnego webinaru
- **UserID** (klucz obcy do **UserID** z tabeli *Users* i jednocześnie element klucza głównego) [NOT NULL][int] - identyfikator konkretnego użytkownika

Kod SQL:

```
CREATE TABLE WebinarsAttendance (  
    WebinarID int NOT NULL,  
    UserID int NOT NULL,  
    CONSTRAINT WebinarsAttendance_pk PRIMARY KEY  
    (WebinarID,UserID)  
);
```

Studia

Semesters

Tabela *Semesters* jest tabelą słownikową zawierającą informacje o wszystkich numerach semestrów.

- **SemesterNo** (klucz główny) [NOT NULL][int] - numer semestru

Warunki integralności:

- **SemestrNo** musi być większe lub równe 1

Kod SQL:

```
CREATE TABLE Semesters (  
    SemesterNo int NOT NULL,  
    CONSTRAINT ValidSemester CHECK (SemesterNo >= 1),  
    CONSTRAINT Semesters_pk PRIMARY KEY (SemesterNo)  
);
```

Students

Tabela *Students* przechowuje informacje o studentach danych studiów na danym semestrze.

- **StudentID** (klucz główny) [NOT NULL][int] - unikalne ID identyfikujące studenta
- **UserID** (klucz obcy do **UserID** z tabeli *Users*) [NOT NULL][int] - ID użytkownika będącego studentem
- **StudyID** (klucz obcy do **StudyID** z tabeli *Studies*) [NOT NULL][int] - ID studiów, na które jest zapisany dany student
- **SemesterNo** (klucz obcy do **SemesterNo** z tabeli *Semesters*) [NOT NULL][int] - numer semestru, na którym studiuje dany student

Kod SQL:

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

Studies

Tabela *Studies* przechowuje informacje o studiach.

- **StudyID** (klucz główny) [NOT NULL] [int] - unikalne ID studiów
- **StudyName** [NOT NULL] [varchar(40)] - nazwa danych studiów
- **FeePrice** [NOT NULL] [money] - wartość zaliczki za dane studia (domyślnie 100)
- **StudyCoordinatorID** (klucz obcy do **EmployeeID** z tabeli *Employees*) [NOT NULL] [int] - ID koordynatora studiów
- **LimitOfStudents** [NOT NULL] [int] - ograniczenie na liczbę studentów, którzy mogą być zapisani na dane studia

Warunki integralności:

- StudyName musi być unikatowe
- LimitOfStudents musi być większe od 0
- FeePrice musi być większe od 0

Kod SQL:

```
CREATE TABLE Studies (  
    StudyID int NOT NULL,  
    StudyName varchar(40) NOT NULL,  
    FeePrice money NOT NULL DEFAULT 100,  
    StudyCoordinatorID int NOT NULL,  
    LimitOfStudents int NOT NULL,  
    CONSTRAINT UniqueStudyName UNIQUE (StudyName),  
    CONSTRAINT LimitOfStudentsGtZero CHECK (LimitOfStudents >  
0),  
    CONSTRAINT StudiesFeePriceGtZero CHECK (FeePrice > 0)  
    CONSTRAINT Studies_pk PRIMARY KEY (StudyID)  
);
```

Grades

Tabela *Grades* zawiera informację o ocenach, które znajdują się w regulaminie studiów.

- **GradeID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator oceny
- **GradeValue** [NOT NULL] [decimal(1,1)] - Wartość liczbową oceny podana w postaci *decimal(1,1)*

Warunki integralności:

- Ocena nie może, być mniejsza niż 2.0 oraz większa niż 5.0

Kod SQL:

```
CREATE TABLE Grades (  
    GradeID int NOT NULL,  
    GradeValue decimal(1,1) NOT NULL,  
    CONSTRAINT GradeValueGt2Lt5 CHECK (GradeValue between 2  
and 5),  
    CONSTRAINT Grades_pk PRIMARY KEY (GradeID)  
);
```

FinalExams

Tabela *FinalExams* zawiera informację o ocenach z egzaminów końcowych na studiach uzyskanych przez studentów.

- **StudyID** (klucz główny i obcy z tabeli *Studies*) [NOT NULL] [int] - Identyfikator studiów z których egzamin końcowy był pisany
- **StudentID** (klucz główny i obcy z tabeli *Students*) [NOT NULL] [int] - Identyfikator studenta, który pisał egzamin końcowy
- **GradeID** (klucz obcy z tabeli *Grades*) [NOT NULL] [int] - Identyfikator oceny jaka została uzyskana przez studenta z egzaminu końcowego

Kod SQL:

```
CREATE TABLE FinalExams (  
    StudyID int NOT NULL,  
    StudentID int NOT NULL,  
    GradeID int NOT NULL,  
    CONSTRAINT FinalExams_pk PRIMARY KEY (StudyID, StudentID)  
);
```

Subjects

Tabela *Subjects* Przechowuje informacje o przedmiotach, które znajdują się w programie uczelni.

- **SubjectID** (klucz główny) [NOT NULL] [int] - Unikalny identyfikator przedmiotu
- **SubjectName** [NOT NULL] [varchar(30)] - Nazwa przedmiotu
- **Description** [NOT NULL] [text] - Opis przedmiotu

Kod SQL:

```
CREATE TABLE Subjects (  
    SubjectID int NOT NULL,  
    SubjectName varchar(30) NOT NULL,  
    Description text NOT NULL,  
    CONSTRAINT Subjects_pk PRIMARY KEY (SubjectID)  
);
```

StudySemesters

Tabela *StudySemesters* zawiera informację o wszystkich semestrach rozumianych jako połączenie numeru semestru oraz danych studiów.

- **StudySemesterID** (klucz główny) [NOT NULL] [int] - Unikalny identyfikator semestru studiów
- **StudyID** (klucz obcy **StudyID** z tabeli *Studies*) [NOT NULL] [int] - Identyfikator studiów
- **SemesterNo** (klucz obcy **SemesterNo** z tabeli *Semesters*) [NOT NULL] [int] - Numer semestru.

Kod SQL:

```
CREATE TABLE StudySemesters (  
    StudySemesterID int NOT NULL,  
    StudyID int NOT NULL,  
    SemesterNo int NOT NULL,  
    CONSTRAINT StudySemesters_pk PRIMARY KEY  
    (StudySemesterID)  
);
```

StudiesSchedule

Tabela *StudiesSchedule* zawierająca informację, o tym jakie przedmioty znajdują się na poszczególnych semestrach

- **StudySemesterID** (klucz główny i obcy z tabeli *StudySemesters*) [NOT NULL] [int] - Identyfikator semestru danych studiów
- **SubjectID** (klucz główny i obcy z tabeli *Subjects*) [NOT NULL] [int] - Identyfikator przedmiotu, z którego zajęcia będą się odbywały w ramach danego semestru

Kod SQL:

```
CREATE TABLE StudiesSchedule (  
    StudySemesterID int NOT NULL,  
    SubjectID int NOT NULL,  
    CONSTRAINT StudiesSchedule_pk PRIMARY KEY  
    (StudySemesterID, SubjectID)  
);
```

StudyMeetups

Tabela *StudyMeetups* przechowująca informacje o zjazdach w ramach danego semestru danych studiów.

- **StudyMeetupID** (klucz główny) [NOT NULL] [int] - Unikalny identyfikator zjazdu
- **StudySemesterID** (klucz obcy z tabeli *StudySemesters*) [NOT NULL] [int] - Identyfikator semestru, którego zjazd dotyczy.
- **StartDate** [NOT NULL] [datetime] - Data rozpoczęcia zjazdu
- **EndDate** [NOT NULL] [datetime] - Data zakończenia zjazdu
- **Price** [NOT NULL] [money] - Cena, którą musi zapłacić student przed każdym zjazdem, aby wziąć udział w zajęciach w ramach tego zjazdu.
- **ExtraPrice** [NOT NULL] [money] - Cena, którą musi zapłacić osoba z zewnątrz, aby wziąć udział w zajęciach w ramach tego zjazdu.

Warunki integralności:

- Data zakończenia zjazdu musi być późniejsza od daty jego początku.
- domyślna cena dla osoby z zewnątrz wynosi 250 zł.
- Price musi być większa od 0.
- ExtraPrice musi być większa od 0.

Kod SQL:

```
CREATE TABLE StudyMeetups (
    StudyMeetupID int NOT NULL,
    StudySemesterID int NOT NULL,
    StartDate datetime NOT NULL,
    EndDate datetime NOT NULL,
    Price money NOT NULL DEFAULT 250,
    ExtraPrice money NOT NULL,
    CONSTRAINT EndDateGtStartDate CHECK (EndDate > StartDate),
    CONSTRAINT PriceGt0 CHECK (Price > 0),
    CONSTRAINT ExtraPriceGt0 CHECK (ExtraPrice > 0),
    CONSTRAINT StudyMeetups_pk PRIMARY KEY (StudyMeetupID)
);
```


StudyMeetings

Tabela *StudyMeetings* przechowuje informację o zajęciach odbywających się w ramach studiów.

- **MeetingID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator spotkania studyjnego
- **StudyMeetupID** (klucz obcy z tabeli *StudyMeetups*) [NOT NULL] [int] - identyfikator zjazdu studyjnego
- **SubjectID** (klucz obcy z tabeli *Subjects*) [NOT NULL] [int] - identyfikator przedmiotu
- **StartDate** [NOT NULL] [datetime] - data rozpoczęcia zajęć
- **EndDate** [NOT NULL] [datetime] - data zakończenia zajęć
- **LecturerID** (klucz obcy z tabeli *Employees*) [NOT NULL] [int] - identyfikator prowadzącego zajęcia
- **LimitOfMeetingParticipants** [NOT NULL] [int] - limit uczniów na zajęciach
- **TranslatorLanguageID** (nullable, klucz obcy z tabeli *Translators*) [int]- identyfikator tłumacza, jeżeli jest on przypisany do zajęć

Warunki integralności:

- Data zakończenia spotkania musi być późniejsza od daty jego początku.
- Limit osób na spotkaniu musi być większy od 0

Kod SQL:

```
CREATE TABLE StudiesMeetings (  
    MeetingID int NOT NULL,  
    StudyMeetupID int NOT NULL,  
    SubjectID int NOT NULL,  
    StartDate datetime NOT NULL,  
    EndDate datetime NOT NULL,  
    LecturerID int NOT NULL,  
    LimitOfMeetingParticipants int NOT NULL,  
    TranslatorLanguageID int NULL,  
    CONSTRAINT StudyMeetingDates CHECK (StartDate < EndDate),  
    CONSTRAINT StudyParticipantsMoreThan0 CHECK  
(LimitOfMeetingParticipants > 0),  
    CONSTRAINT StudiesMeetings_pk PRIMARY KEY (MeetingID));
```

StudyAttendance

Tabela *StudyAttendance* odnotowuje obecność studentów na danych zajęciach

- **StudyMeetingID** (klucz główny, klucz obcy z tabeli *StudyMeetings*) [NOT NULL] [int] - identyfikator zajęć
- **StudentID** (klucz główny, klucz obcy z tabeli *Students*) [NOT NULL] [int] - identyfikator studenta, który pojawił się na zajęciach.

Kod SQL:

```
CREATE TABLE StudyAttendance (  
    StudyMeetingID int NOT NULL,  
    StudentID int NOT NULL,  
    CONSTRAINT StudyAttendance_pk PRIMARY KEY  
    (StudyMeetingID, StudentID)  
);
```

OnlineStudy

Tabela *OnlineStudy* przechowuje informację o zajęciach odbywających się zdalnie.

- **OnlineStudyMeetingID** (klucz główny, klucz obcy z tabeli *StudyMeetings*) [NOT NULL] [int] - identyfikator zajęć odbywających się zdalnie
- **MeetingLink** [NOT NULL] [varchar(100)] - Link do spotkania online.

Warunki integralności:

- Link musi mieć formę: *https://[...]* lub *http://[...]*

Kod SQL:

```
CREATE TABLE OnlineStudy (  
    OnlineStudyMeetingID int NOT NULL,  
    MeetingLink varchar(100) NOT NULL,  
    CONSTRAINT CheckMeetingLink CHECK (MeetingLink LIKE  
'http://% ' OR MeetingLink LIKE 'https://% '),  
    CONSTRAINT OnlineStudy_pk PRIMARY KEY  
    (OnlineStudyMeetingID)  
);
```

StationaryStudy

Tabela *StationaryStudy* przechowuje informację o zajęciach odbywających się stacjonarnie.

- **StationaryStudyMeetingID** (klucz główny, klucz obcy z tabeli *StudyMeetings*) [NOT NULL] [int] - identyfikator zajęć odbywających się stacjonarnie
- **ClassroomID** (klucz obcy do tabeli *Classrooms*) [NOT NULL] [int] - identyfikator sali, w której odbywa się zajęcie

Kod SQL:

```
CREATE TABLE StationaryStudy (  
    StationaryStudyMeetingID int NOT NULL,  
    ClassroomID int NOT NULL,  
    CONSTRAINT StationaryStudy_pk PRIMARY KEY  
    (StationaryStudyMeetingID)  
);
```

Internships

Tabela *Internships* przechowuje informację o praktykach, które odbywają się w ramach studiów.

- **InternshipID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator praktyk
- **StudyID** (klucz obcy z tabeli *Studies*) [NOT NULL] [int] - identyfikator studiów na których odbywają się dane praktyki.
- **CycleNo** [NOT NULL] [int] - numer praktyk, które odbywają się na danym studium.

Warunki integralności:

- Numer praktyk musi być większy od 0

Kod SQL:

```
CREATE TABLE Internships (  
    InternshipID int NOT NULL,  
    StudyID int NOT NULL,  
    CycleNo int NOT NULL,  
    CONSTRAINT CheckCycleNo CHECK NOT FOR REPLICATION (CycleNo  
in (1,2)),  
    CONSTRAINT Internships_pk PRIMARY KEY (InternshipID)  
);
```

InternshipDetails

Tabela *InternshipDetails* zawiera informację o praktykach odbywanych przez studentów (w ramach jakich praktyk do jakiej firmy poszedł dany student)

- **InternshipID** (klucz główny, klucz obcy z tabeli *Internships*) [NOT NULL] [int] - identyfikator praktyk
- **StudentID** (klucz główny, klucz obcy z tabeli *Students*) [NOT NULL] [int] - identyfikator studenta, który odbywa praktyki
- **CompanyName** [NOT NULL] [varchar(20)] - nazwa firmy w której praktyki studenta były odbywane.
- **Pass** [NOT NULL] [bit] - Czy student zaliczył praktyki, informacja o tym przychodzi od firmy, w której praktyki były odbyte (domyślnie 0).

Kod SQL:

```
CREATE TABLE InternshipDetails (  
    InternshipID int NOT NULL,  
    StudentID int NOT NULL,  
    CompanyName varchar(20) NOT NULL,  
    Pass bit NOT NULL DEFAULT 0,  
    CONSTRAINT InternshipDetails_pk PRIMARY KEY  
    (InternshipID, StudentID)  
);
```

Zamówienia

ServiceTypes

Tabela *ServiceType* zawiera informacje o typach usług, jakie można zakupić na uczelni są to: *studia*, *pojedyncze spotkania na studiach*, *webinary* oraz *kursy*.

- **ServiceTypeID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator typu serwisu oferowanego przez uczelnie.
- **ServiceTypeName** [NOT NULL] [varchar(20)] - nazwa oferowanego typu serwisu np: *webinar*

warunki integralności:

- Nazwa typu serwisu musi być unikatowa

Kod SQL:

```
CREATE TABLE ServiceTypes (  
    ServiceTypeID int NOT NULL,  
    ServiceTypeName varchar(20) NOT NULL,  
    CONSTRAINT ServiceNameUnique UNIQUE (ServiceTypeName),  
    CONSTRAINT ServiceTypes_pk PRIMARY KEY (ServiceTypeID)  
);
```

Orders

Tabela *Orders* przechowuje informację o zamówieniach złożonych przez klientów.

- **OrderID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator zamówienia
- **UserID** (klucz obcy z tabeli *Users*) [NOT NULL] [int] - identyfikator użytkownika, który złożył dane zamówienie
- **OrderDate** [NOT NULL] [datetime] - data złożenia zamówienia

Kod SQL:

```
CREATE TABLE Orders (  
    OrderID int NOT NULL,  
    UserID int NOT NULL,  
    OrderDate datetime NOT NULL,  
    CONSTRAINT Orders_pk PRIMARY KEY (OrderID));
```

OrderDetails

Tabela *OrderDetails* zawiera rozszerzone informacje o zamówieniu

- **OrderDetailID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator dodatkowych informacji
- **OrderID** (klucz obcy z tabeli *Orders*) [NOT NULL] [int] - identyfikator zamówienia, którego dodatkowe informacje są przechowywane
- **ServiceTypeID** (klucz obcy z tabeli *ServiceTypesID*) [NOT NULL] [int] - identyfikator typu serwisu
- **ServiceID** (klucz obcy z tabel *Studies*, *StudyMeetups*, *Courses*, *Webinars*) [NOT NULL] [int] - identyfikator usługi w danym typie.

Kod SQL:

```
CREATE TABLE OrderDetails (  
    OrderDetailID int NOT NULL,  
    OrderID int NOT NULL,  
    ServiceTypeID int NOT NULL,  
    ServiceID int NOT NULL,  
    CONSTRAINT OrderDetails_pk PRIMARY KEY (OrderDetailID)  
);
```

Payments

Tabela *Payments* zawiera informacje o zatwierdzonych płatnościach.

- **PaymentID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator płatności
- **OrderDetailID** (klucz obcy z tabeli *OrderDetails*) [NOT NULL] [int] - identyfikator szczegółów zamówienia, którego płatność dotyczy.
- **PayDate** [NOT NULL] [datetime] - data zaksięgowania płatności w systemie zewnętrznym
- **Amount** [NOT NULL] [money] - kwota płatności

warunki integralności:

- kwota płatności musi być większa od 0.
- data płatności nie może być z przyszłości

Kod SQL:

```
CREATE TABLE Payments (  
    PaymentID int NOT NULL,  
    OrderDetailID int NOT NULL,  
    PayDate datetime NOT NULL,  
    Amount money NOT NULL,  
    CONSTRAINT AmountGt0 CHECK (Amount > 0),  
    CONSTRAINT PayDateCheck CHECK (PayDate < GETDATE()),  
    CONSTRAINT Payments_pk PRIMARY KEY (PaymentID)  
);
```


HeadTeacherPaymentPostponements

Tabela *HeadTeacherPaymentPostponements* zawiera zawieszenia płatności za usługi, wystawiane przez dyrektora.

- **PostponementID** (klucz główny) [NOT NULL] [int] - unikalny identyfikator odroczenia płatności
- **UserID** (klucz obcy z tabeli *Users*) [NOT NULL] [int] - identyfikator użytkownika, którego płatność została odroczone
- **ServiceTypeID** (klucz obcy z tabeli *ServiceTypes*) [NOT NULL] [int] - identyfikator typu serwisu
- **ServiceID** (klucz obcy z tabel *Studies*, *StudyMeetups*, *Courses*, *Webinars*) [NOT NULL] [int] - identyfikator usługi w danym typie.
- **DueDate** [NOT NULL] [datetime] - data określająca, do kiedy zawieszono płatność.

Kod SQL:

```
CREATE TABLE HeadTeacherPaymentPostponements (
    PostponementID int NOT NULL,
    UserID int NOT NULL,
    ServiceTypeID int NOT NULL,
    ServiceID int NOT NULL,
    DueDate datetime NOT NULL,
    CONSTRAINT HeadTeacherPaymentPostponements_pk PRIMARY KEY
    (PostponementID)
);
```

Widoki

Raporty finansowe

Autor: Wiktor Sędzimir

Przychody za kursy

Autor: Wiktor Sędzimir

Widok *CoursesIncome* zawiera zestawienie przychodów za poszczególne kursy.

```
create view CoursesIncome as
    select
        CourseID,
        CourseName,
        isnull(sum(Amount), 0) as TotalIncome
    from Payments
    inner join OrderDetails
        on Payments.OrderDetailID = OrderDetails.OrderDetailID
    inner join ServiceTypes
        on OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID
    right join Courses
        on ServiceID = CourseID and ServiceTypeName = 'Course'
    group by CourseID, CourseName;
```

Przychody za webinary

Autor: Wiktor Sędzimir

Widok *WebinarsIncome* zawiera zestawienie przychodów za poszczególne webinary.

```
create view WebinarsIncome as
    select
        WebinarID,
        WebinarName,
        isnull(sum(Amount), 0) as TotalIncome
    from Payments
    inner join OrderDetails
        on Payments.OrderDetailID = OrderDetails.OrderDetailID
    inner join ServiceTypes
        on OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID
    right join Webinars
        on ServiceID = WebinarID and ServiceTypeName =
'Webinar'
    group by WebinarID, WebinarName;
```

Przychody za studia

Autor: Wiktor Sędzimir

Widok *StudiesIncome* zawiera zestawienie przychodów za poszczególne Studia.

```
create view StudyIncome as
    with StudyFeeIncome as (
        select
            StudyID,
            isnull(sum(Amount), 0) as TotalFeeIncome
        from Payments
        inner join OrderDetails
            on Payments.OrderDetailID =
OrderDetails.OrderDetailID
        inner join ServiceTypes
            on OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID
        right join Studies
            on ServiceID = StudyID and ServiceTypeName =
'Studies'
        group by StudyID
    ), StudyMeetupsIncome as (
        select
            StudyID,
            isnull(sum(Amount), 0) as TotalMeetupsIncome
        from Payments
        inner join OrderDetails
            on Payments.OrderDetailID =
OrderDetails.OrderDetailID
        inner join ServiceTypes
            on OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID
```

```
        inner join StudyMeetups
            on ServiceID = StudyMeetupID
        right join StudySemesters
            on StudyMeetups.StudySemesterID =
StudySemesters.StudySemesterID and ServiceTypeName = 'Studies
Meetup'
        group by StudyID
    )
select
    Studies.StudyID,
    StudyName,
    TotalFeeIncome + TotalMeetupsIncome as TotalIncome
from StudyFeeIncome
inner join StudyMeetupsIncome
    on StudyFeeIncome.StudyID = StudyMeetupsIncome.StudyID
inner join Studies
    on StudyFeeIncome.StudyID = Studies.StudyID and
StudyMeetupsIncome.StudyID = Studies.StudyID;
```

Przychody za wszystkie usługi

Autor: Szymon Migas

Widok *SummaryIncome* zawiera zbiorcze zestawienie przychodów za wszystkie usługi.

```
create view SummaryIncome as
    select
        'Course' as ServiceType,
        CourseID as ServiceID,
        CourseName as ServiceName,
        TotalIncome
    from CoursesIncome
    union
    select
        'Study' as ServiceType,
        StudyID as ServiceID,
        StudyName as ServiceName,
        TotalIncome
    from StudyIncome
    union
    select
        'Webinar' as ServiceType,
        WebinarID as ServiceID,
        WebinarName as ServiceName,
        TotalIncome
    from WebinarsIncome;
```

Raporty dłużników

Autor: Wiktor Sędzimir

Dłużnicy za kursy

Autor: Wiktor Sędzimir

Widok *CoursesDebtors* zawiera zestawienie dłużników wraz z wysokością długu dla kursów. Jako dłużnika traktujemy osobę, która zamówiła kurs, otrzymała odroczenie od dyrektora, natomiast nie zapłaciła pełnej kwoty.

```
create view CoursesDebtors as

    with UserCoursePayments as (

        select

            UserID,

            ServiceID,

            sum(Amount) as TotalAmountPaid

        from Payments

        right join OrderDetails

            on Payments.OrderDetailID =

OrderDetails.OrderDetailID

        inner join ServiceTypes

            on OrderDetails.ServiceTypeID =

ServiceTypes.ServiceTypeID and ServiceTypeName = 'Course'

        inner join Orders

            on Orders.OrderID = OrderDetails.OrderID

        group by UserID, ServiceID

    ), UsersThatPaidFullPriceForCourse as (

        select

            UserID,
```

```
        CourseID,  
        FeePrice,  
        TotalPrice,  
        TotalAmountPaid  
from UserCoursePayments  
inner join Courses  
        on ServiceID = CourseID  
where TotalAmountPaid >= Courses.TotalPrice  
, CoursesAttendanceWithCourseID as (  
    select  
        distinct  
        UserID,  
        MeetingID,  
        CourseID  
    from CoursesAttendance  
    inner join CoursesMeetings  
        on CoursesAttendance.CourseMeetingID =  
CoursesMeetings.MeetingID  
, Debtors as (  
    select  
        distinct  
        cawcid.UserID,  
        CourseID,  
        DueDate as HeadTeacherPostponementDueDate  
    from CoursesAttendanceWithCourseID as cawcid
```



```
        inner join HeadTeacherPaymentPostponements
            on cawcid.CourseID =
HeadTeacherPaymentPostponements.ServiceID
            and cawcid.UserID =
HeadTeacherPaymentPostponements.UserID
        inner join ServiceTypes
            on ServiceTypes.ServiceTypeID =
HeadTeacherPaymentPostponements.ServiceTypeID
            and ServiceTypeName = 'Course'
        where not exists (
            select 1
            from UsersThatPaidFullPriceForCourse
            where UsersThatPaidFullPriceForCourse.UserID =
cawcid.UserID
            and UsersThatPaidFullPriceForCourse.CourseID =
cawcid.CourseID
        )
        and getdate() > DueDate
    )
select
    Debtors.UserID,
    Firstname,
    Lastname,
    Courses.CourseID,
    CourseName,
    Email,
```

```
    Phone,
    TotalPrice,
    TotalAmountPaid,
    TotalPrice - isnull(TotalAmountPaid, 0) as Debt,
    HeadTeacherPostponementDueDate
from Debtors
left join UserCoursePayments
    on Debtors.UserID = UserCoursePayments.UserID and
Debtors.CourseID = UserCoursePayments.ServiceID
inner join Courses
    on Debtors.CourseID = Courses.CourseID
inner join Users
    on Debtors.UserID = Users.UserID;
```

Dłużnicy za studia

Autor: Wiktor Sędzimir

Widok *StudentDebtors* zawiera zestawienie dłużników za Studia. Jako dłużnika traktujemy studenta, który zamówił studia, otrzymał odroczenie od dyrektora, natomiast nie zapłacili pełnej kwoty.

```
create view StudentDebtors as
    with UserMeetupPayments as (
        select
            UserID,
            ServiceID as StudyMeetupID,
            sum(Amount) as TotalAmountPaid
        from Payments
        right join OrderDetails
            on Payments.OrderDetailID =
OrderDetails.OrderDetailID
        inner join ServiceTypes
            on OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID and ServiceTypeName = 'Studies
Meetup'
        inner join Orders
            on Orders.OrderID = OrderDetails.OrderID
        group by UserID, ServiceID
    ), UsersThatPaidFullPriceForMeetup as (
        select
            UserID,
            StudyMeetups.StudyMeetupID,
            Price,
```

```
TotalAmountPaid,
    StudyID
from UserMeetupPayments
    inner join StudyMeetups
        on UserMeetupPayments.StudyMeetupID =
StudyMeetups.StudyMeetupID
    inner join StudySemesters
        on StudyMeetups.StudySemesterID =
StudySemesters.StudySemesterID
where UserMeetupPayments.TotalAmountPaid >=
StudyMeetups.Price
    and exists (
        select
            distinct
                Orders.OrderID
        from OrderDetails
            inner join Orders
                on Orders.OrderID = OrderDetails.OrderID
            inner join ServiceTypes
                on OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID and ServiceTypeName = 'Studies
Meetup'
        where Orders.UserID =
UserMeetupPayments.UserID
            and OrderDetails.ServiceID =
UserMeetupPayments.StudyMeetupID
        intersect
```

```
        select
            distinct
                Orders.OrderID
        from OrderDetails
        inner join Orders
            on Orders.OrderID = OrderDetails.OrderID
        inner join ServiceTypes
            on OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID and ServiceTypeName = 'Studies'
        where Orders.UserID =
UserMeetupPayments.UserID
            and OrderDetails.ServiceID =
StudySemesters.StudyID
    )
), StudiesMeetingsAttendanceWithMeetupsID as (
    select
        distinct
            UserID,
            StudyMeetups.StudyMeetupID
        from StudyAttendance
        inner join StudiesMeetings
            on StudyAttendance.StudyMeetingID =
StudiesMeetings.MeetingID
        inner join StudyMeetups
            on StudyMeetups.StudyMeetupID =
StudiesMeetings.StudyMeetupID
```

```
        inner join Students
            on Students.StudentID = StudyAttendance.StudentID
    ), Debtors as (
        select
            distinct
                StudiesMeetingsAttendanceWithMeetupsID.UserID,
                StudyMeetupID,
                DueDate as HeadTeacherPostponementDueDate
        from StudiesMeetingsAttendanceWithMeetupsID
        inner join HeadTeacherPaymentPostponements
            on HeadTeacherPaymentPostponements.UserID =
StudiesMeetingsAttendanceWithMeetupsID.UserID
            and HeadTeacherPaymentPostponements.ServiceID
= StudiesMeetingsAttendanceWithMeetupsID.StudyMeetupID
        inner join ServiceTypes
            on ServiceTypes.ServiceTypeID =
HeadTeacherPaymentPostponements.ServiceTypeID
            and ServiceTypeName = 'Studies Meetup'
        where not exists (
            select 1
            from UsersThatPaidFullPriceForMeetup
            where UsersThatPaidFullPriceForMeetup.UserID =
StudiesMeetingsAttendanceWithMeetupsID.UserID
            and
UsersThatPaidFullPriceForMeetup.StudyMeetupID =
StudiesMeetingsAttendanceWithMeetupsID.StudyMeetupID
```

```
)  
    and DueDate < getdate()  
)  
  
select  
    Debtors.UserID,  
    Debtors.StudyMeetupID,  
    Firstname,  
    Lastname,  
    Email,  
    Phone,  
    Price,  
    TotalAmountPaid,  
    StudyMeetups.Price - isnull(TotalAmountPaid, 0) as  
Debt  
    , HeadTeacherPostponementDueDate  
from Debtors  
left join UserMeetupPayments  
    on Debtors.UserID = UserMeetupPayments.UserID and  
Debtors.StudyMeetupID = UserMeetupPayments.StudyMeetupID  
inner join StudyMeetups  
    on Debtors.StudyMeetupID = StudyMeetups.StudyMeetupID  
inner join Users  
    on Debtors.UserID = Users.UserID;
```

Raporty wydarzeń

Autor: Szymon Migas

Osoby mogące wziąć udział w webinarze

Autor: Szymon Migas

Widok *UsersAuthorizedForWebinar* zawiera zestawienie webinarów oraz wszystkich użytkowników, którzy spełniają warunki, aby wziąć w nim udział.

```
create view UsersAuthorizedForWebinar as
select WebinarID, O.UserID
from Payments
    inner join dbo.OrderDetails OD on
Payments.OrderDetailID = OD.OrderDetailID
    inner join dbo.Orders O on OD.OrderID = O.OrderID
    inner join dbo.ServiceTypes ST on OD.ServiceTypeID =
ST.ServiceTypeID
    left join dbo.Webinars W on W.WebinarID =
OD.ServiceID
where ServiceTypeName = 'Webinar'
    and PayDate < W.Date
```

Miejsca dostępne na wydarzeniu

Autor: Szymon Migas

Widok przedstawia maksymalną ilość osób, które mogą pojawić się na kursie, lub być zapisane na studia

```
CREATE VIEW AvailableSpots AS
SELECT 'Course' as Service ,C.CourseID as ServiceID,
MIN(LimitOfParticipants) as Limit FROM Courses C
INNER JOIN dbo.CoursesMeetings CM on C.CourseID = CM.CourseID
group by C.CourseID
UNION
SELECT 'Studies',StudyID, LimitOfStudents FROM Studies S
```


Osoby mogące wziąć udział w kursie

Autor: Szymon Migas

Widok *UsersAuthorizedForCourseMeeting* zawiera zestawienie spotkań na kursy oraz wszystkich użytkowników spełniających warunki, aby wziąć w nim udział

```
create view UsersAuthorizedForCourseMeeting as
with users_course_payments as
    (select O.UserID, OD.ServiceID, sum(Amount) paid,
max(PayDate) maxPaymentDate
    from Payments
        inner join dbo.OrderDetails OD
            on Payments.OrderDetailID =
OD.OrderDetailID and ServiceTypeID = 5
        inner join dbo.Orders O on OD.OrderID =
O.OrderID
    group by O.UserID, OD.ServiceID)
select MeetingID, UserID
from users_course_payments U
    inner join dbo.Courses C on U.ServiceID = C.CourseID
    left join dbo.CoursesMeetings CM on C.CourseID =
CM.CourseID
where MeetingID is not null
    and DATEDIFF(hour, maxPaymentDate, StartDate) > 72
    and paid >= TotalPrice
union
select CM2.MeetingID, UserID
from HeadTeacherPaymentPostponements H
```

```
        inner join dbo.CoursesMeetings CM2 on CM2.CourseID =  
ServiceID  
where H.ServiceTypeID = 5
```

Osoby mogące wziąć udział w zajęciach na studiach

Autor: Szymon Migas

Widok *UsersAuthorizedForStudyMeeting* zawiera zestawienie zajęć na studiach wraz z wszystkimi studentami, którzy mogą wziąć udział w danym wydarzeniu.

```
create view UsersAuthorizedForStudyMeeting as  
select MeetingID, O.UserID  
from Payments  
        inner join dbo.OrderDetails OD on Payments.OrderDetailID  
= OD.OrderDetailID  
        inner join dbo.Orders O on OD.OrderID = O.OrderID  
        inner join dbo.ServiceTypes ST on OD.ServiceTypeID =  
ST.ServiceTypeID  
        left join dbo.StudiesMeetings SM on SM.StudyMeetupID =  
OD.ServiceID  
where ServiceTypeName = 'Studies Meetup'  
and PayDate < SM.StartDate  
union  
select SM2.MeetingID, UserID  
from HeadTeacherPaymentPostponements  
        inner join ServiceTypes ST2 on ST2.ServiceTypeName =  
'Studies Meetup'  
        inner join dbo.StudiesMeetings SM2 on SM2.StudyMeetupID =  
ServiceID
```

Osoby mogące wziąć udział w zjazdach

Autor: Szymon Migas

Widok *UsersAuthorizedForStudyMeetup* zawiera zestawienie zjazdów na studiach wraz z wszystkimi studentami, którzy mogą wziąć udział w danym zjeździe

```
create view UsersAuthorizedForStudyMeetup as
select S.StudyID, StudyMeetupID, UserID from Studies S
inner join StudySemesters SS on S.StudyID = SS.StudyID
inner join StudyMeetups SM on SS.StudySemesterID = SM.StudySemesterID
inner join OrderDetails OD on SM.StudyMeetupID = OD.ServiceID
inner join dbo.ServiceTypes ST on OD.ServiceTypeID = ST.ServiceTypeID
inner join dbo.Orders O on OD.OrderID = O.OrderID
inner join Payments P on OD.OrderDetailID = P.OrderDetailID
where ServiceTypeName = 'Studies Meetup'
and UserID IN (SELECT UserID FROM Students)
and not Amount < SM.Price
and not StartDate < P.PayDate
```

Liczba osób dopuszczonych do webinaru

Autor: Szymon Migas

Widok *NumberOfUsersAuthorizedForWebinar* ukazuje liczbę użytkowników, którzy mogą być na danym webinarze

```
create view NumberOfUsersAuthorizedForWebinar as
select WebinarID, count(*) [Users]
from UsersAuthorizedForWebinar
group by WebinarID
```

Liczba osób dopuszczonych do spotkania na kursie

Autor: Szymon Migas

Widok *NumberOfUsersAuthorizedForCourseMeeting* ukazuje liczbę użytkowników, którzy mogą być na danym spotkaniu w ramach kursu

```
create view NumberOfUsersAuthorizedForCourseMeeting as
select MeetingID, count(*) as [Users]
from UsersAuthorizedForCourseMeeting
group by MeetingID
```

Liczba osób dopuszczonych do zajęć na studiach

Autor: Szymon Migas

Widok *NumberOfUsersAuthorizedForStudyMeeting* ukazuje liczbę studentów, którzy mogą być na danych zajęciach w ramach zjazdu.

```
create view NumberOfUsersAuthorizedForStudyMeeting as
select MeetingID, count(*) [Users] from
UsersAuthorizedForStudyMeeting
group by MeetingID
```

Liczba osób dopuszczonych do zjazdu na studiach

Autor: Szymon Migas

Widok *NumberOfUsersAuthorizedForStudyMeetup* ukazuje liczbę studentów, którzy mogą być na danym zjeździe

```
create view NumberOfUsersAuthorizedForStudyMeetup as
select StudyID, StudyMeetupID, count(*) cnt from
UsersAuthorizedForStudyMeetup
group by StudyID, StudyMeetupID
```

Raport obecności na webinarach

Autor: Szymon Migas

Widok *WebinarAttendance* zawiera raport dla każdego webinaru, pokazujący ilość osób, którzy byli obecni na webinarze w stosunku do osób, które mogły się na nim znaleźć, oraz procentową wartość tego stosunku.

```

create view WebinarAttendance as
with users_on_webinar
    as(select WebinarID, count(*) [Users]
        from WebinarsAttendance
        group by WebinarID)
select
    NU.WebinarID,
    WebinarName,
    cast(UW.Users as varchar) + '/' + cast(NU.Users as varchar) as
[Users on webinar],
    FORMAT(UW.Users/convert(decimal(7,2),NU.Users), 'P') as Percentage
from NumberOfUsersAuthorizedForWebinar NU
inner join users_on_webinar UW on UW.WebinarID = NU.WebinarID
inner join Webinars W on W.WebinarID = NU.WebinarID
where Date < GETDATE()

```

Raport obecności na spotkaniach do kursów

Autor: Szymon Migas

Widok *CourseAttendance* zawiera raport dla każdego spotkania w ramach kursów, pokazujący ilość osób, którzy byli obecni na spotkaniu w stosunku do osób, które mogły się na nim znaleźć, oraz procentową wartość tego stosunku.

```

create view CourseAttendance as
with users_on_course_meeting as
    (select CA.CourseMeetingID, count(*) [Users]
        from CoursesAttendance CA
        group by CA.CourseMeetingID)
select NUM.MeetingID as
'CourseMeetingID',
    cast(UOM.Users as varchar) + '/' + cast(NUM.Users as varchar)
[Users on course meeting],
    FORMAT(UOM.Users / cast(NUM.Users as decimal(7, 2)), 'P') as
[Percentage]

```

```

from NumberOfUsersAuthorizedForCourseMeeting NUM
    inner join users_on_course_meeting UOM on UOM.CourseMeetingID
= NUM.MeetingID
    inner join CoursesMeetings CM on CM.MeetingID = NUM.MeetingID
where StartDate < GETDATE()

```

Raport obecności na spotkaniach w ramach studiów

Autor: Szymon Migas

Widok *StudyAttendance* zawiera raport dla każdego spotkania w ramach studiów, pokazujący ilość osób, którzy byli obecni na spotkaniu w stosunku do osób, które mogły się na nim znaleźć, oraz procentową wartość tego stosunku.

```

create view StudyAttendance as
with users_on_meeting as
    (select SA.StudyMeetingID, count(*) as [User on meeting]
    from StudyAttendance SA
    group by SA.StudyMeetingID)
select NU.MeetingID,
    cast([User on meeting] as varchar) + '/' + cast(NU.Users as
varchar) as [Users Present],
    FORMAT((select count(*)
    from StudyAttendance SA
    where SA.StudyMeetingID = NU.MeetingID) /
convert(decimal(7, 2), NU.Users), 'P') as [Percentege]
from NumberOfUsersAuthorizedForStudyMeeting NU
    inner join users_on_meeting UOM on UOM.StudyMeetingID =
NU.MeetingID
    inner join StudiesMeetings SM on NU.MeetingID = SM.MeetingID
where StartDate < GETDATE()

```

Raport obecności

Autor: Szymon Migas

Widok *AttendanceReport* zawiera raport dla każdego typu wydarzeń odbywających się w szkole oraz średnią procentową obecność na spotkaniach w ramach danego typu wydarzenia.

```
create view AttendanceReport as
select
    'Webinars' as ServiceType,
    Format(avg(cast(replace(Percentage, '%', '') as decimal(7, 2))) /
100, 'P') as AverageAttendance
from WebinarAttendance

union select 'Study Meetings' as ServiceType,
    Format(avg(cast(replace(Percentage, '%', '') as decimal(7, 2))) /
100, 'P') as AverageAttendance
from StudyAttendance

union select 'Course Meetings' as Servicetype,
    Format(avg(cast(replace(Percentage, '%', '') as decimal(7, 2))) /
100, 'P') as AverageAttendance
from CourseAttendance
```

Raport dotyczący liczby zapisanych osób na przyszłe wydarzenia

Autor: Szymon Migas

Widok *FutureActivitiesReport* ukazuje zestawienie wszystkich przyszłych zajęć prowadzonych przez szkołę, wraz z informacją, czy odbędą się one zdalnie, czy też stacjonarnie oraz liczbą użytkowników na nie zapisanych.

```
create view FutureActivitiesReport as
with users_on_activities as (select 'Webinar'
as [Service Type],
                                W.WebinarName
as ServiceName,
                                (SELECT ServiceTypeID
FROM ServiceTypes
where ServiceTypeName =
'Webinar') as ServiceTypeID,
                                UW.WebinarID
as ServiceID,
                                'Remote'
as Place,
                                UW.Users
as [Users signed in],
                                Date
as [Date of activity]
                                from
NumberOfUsersAuthorizedForWebinar UW
                                inner join Webinars W on
UW.WebinarID = W.WebinarID
union
select 'Study Meeting',
StudyName,
(SELECT ServiceTypeID
FROM ServiceTypes
```



```

                                where ServiceTypeName =
'Sudies Meeting'),
                                US.MeetingID,
                                iif(US.MeetingID in (select
OnlineStudyMeetingID from OnlineStudy), 'Remote',
                                'On-site'),
                                US.Users,
                                SM.StartDate
                                from
NumberOfUsersAuthorizedForStudyMeeting US
                                inner join StudiesMeetings
SM on SM.MeetingID = US.MeetingID
                                inner join StudyMeetups SM2
on SM2.StudySemesterID = SM.StudyMeetupID
                                inner join StudySemesters SS
on SM2.StudySemesterID = SS.StudySemesterID
                                inner join Studies S ON
S.StudyID = SS.StudyID
                                union
                                select 'Course Meeting',
                                CourseName,
                                (SELECT ServiceTypeID
FROM ServiceTypes
                                where ServiceTypeName =
'Course'),
                                UC.MeetingID,
                                iif(UC.MeetingID in (select
StationaryCourseMeetingID from StationaryCourse),
                                'On-site', 'Remote'),
                                UC.Users,
                                CM.StartDate
                                from
NumberOfUsersAuthorizedForCourseMeeting UC
                                inner join CoursesMeetings
CM on CM.MeetingID = UC.MeetingID
                                inner join Courses on
CM.CourseID = Courses.CourseID)
select *
from users_on_activities UOA
where UOA.[Date of activity] > GETDATE()

```

Raport bilokacji

Autor: Krzysztof Chmielewski

Raport Bilokacji

Widok *BilocationReport* przedstawia informację o użytkownikach zapisanych na dwa lub więcej spotkań odbywających się w tym samym czasie

```
create view BilocationReport as
with futureWebinarsList as (
    select
        u.UserID as UID,
        w.WebinarID as EventID,
        'Webinar' as EventType,
        w.Date as StartDate,
        dateadd(hour, 2, w.Date) as EndDate
    from Users as u
    inner join Orders as o on o.UserID = u.UserID
    inner join OrderDetails as od on o.OrderID = od.OrderID
    inner join dbo.ServiceTypes ST on od.ServiceTypeID =
ST.ServiceTypeID and ServiceTypeName = 'Webinar'
    inner join webinars as w on w.WebinarID = od.ServiceID
    where w.Date > getdate()),
futureCourseList as (
    select
        u.UserID as UID,
        cm.MeetingID as EventID,
        'Course Meeting' as EventType,
        cm.StartDate as StartDate,
```

```

        cm.EndDate as EndDate
    from Users as u
    inner join Orders as o on o.UserID = u.UserID
    inner join OrderDetails as od on o.OrderID = od.OrderID
    inner join dbo.ServiceTypes ST on od.ServiceTypeID =
ST.ServiceTypeID and ServiceTypeName = 'Webinar'
    inner join CoursesMeetings as cm on cm.CourseID = od.ServiceID
    where cm.StartDate > getdate()
),
futureStudyList as (
    select
        u.UserID as UID,
        m.MeetingID as EventID,
        'Study Meeting' as EventType,
        m.StartDate as StartDate,
        m.EndDate as EndDate
    from Users as u
    inner join students as s on s.UserID = u.UserID
    inner join StudySemesters as ss on s.StudyID = ss.StudyID
    inner join dbo.StudyMeetups SM on ss.StudySemesterID =
SM.StudySemesterID
    inner join dbo.StudiesMeetings M on SM.StudyMeetupID =
M.StudyMeetupID
    where m.StartDate > getdate()
),
allFutureEvents as (
    select * from futureWebinarsList
    union
    select * from futureCourseList
    union
    select * from futureStudyList

```

```
),
overlappingEvents as (
    select
        e1.UID, u.Firstname, u.Lastname,
        e1.EventID as [1st Event ID],
        e1.EventType as [1st Event Type],
        e2.EventID as [2nd Event ID],
        e2.EventType as [2nd Event Type],
        e1.StartDate as [1st Event Start],
        e1.EndDate as [1st Event End],
        e2.StartDate as [2nd Event Start],
        e2.EndDate as [2nd Event End]
    from allFutureEvents e1
    inner join allFutureEvents e2 on e1.UID = e2.UID
        and e1.EventID < e2.EventID -- unikanie duplikatów i łączenia
eventów samych ze sobą
        and e1.StartDate < e2.EndDate
        and e1.EndDate > e2.StartDate -- warunek czasowy kolizji
eventów
    inner join Users as u on e1.UID = u.UserID
) select distinct * from overlappingEvents;
```

Raporty obecności

Autor: Krzysztof Chmielewski

Obecność na kursach

Autor: Krzysztof Chmielewski

Widok *CourseMeetingPresense* Przedstawia obecność użytkowników na spotkaniach w ramach kursów

```
create view CourseMeetingPresence as (  
select distinct cm.MeetingID, cm.StartDate, u.Firstname, u.Lastname,  
u.UserID,  
        iif(u.UserID in (select ca.userid from CoursesAttendance as ca  
where ca.CourseMeetingID = cm.MeetingID), 'Present', 'Absent') as  
Presence  
        from CoursesMeetings as cm  
        inner join OrderDetails as od on od.ServiceID = cm.CourseID  
        inner join orders as o on o.OrderID = od.OrderID  
        inner join users as u on u.UserID = o.UserID  
        inner join dbo.ServiceTypes ST on od.ServiceTypeID =  
ST.ServiceTypeID and st.ServiceTypeName = 'Course');
```

Obecność na spotkaniach w ramach studiów

Autor: Krzysztof Chmielewski

Widok *StudyMeetingPresense* Przedstawia obecność użytkowników na spotkaniach studyjnych

```
create view StudyMeetingPresence as (  
select distinct sm.MeetingID, sm.StartDate ,s2.StudentID,  
u.Firstname, u.Lastname,
```

```

        iif(s2.StudentID in (select sa.StudentID from StudyAttendance
as sa where sa.StudyMeetingID = sm.MeetingID), 'Present', 'Absent')
as Presence

        from StudiesMeetings as sm

        inner join dbo.StudyMeetups S on sm.StudyMeetupID =
S.StudyMeetupID

        inner join dbo.StudySemesters SS on S.StudySemesterID =
SS.StudySemesterID

        inner join dbo.Students S2 on ss.StudyID = S2.StudyID

        inner join users as u on u.UserID = s2.UserID);

```

Obecność na webinarach

Autor: Krzysztof Chmielewski

Widok *WebinarMeetingPresense* Przedstawia obecność użytkowników na webinarach

```

create view WebinarMeetingPresence as (

select distinct w.webinarID, w.Date, u.Firstname, u.Lastname,
u.UserID,

        iif(u.UserID in (select wa.userid from dbo.WebinarsAttendance
as wa where wa.WebinarID = w.WebinarID), 'Present', 'Absent') as
Presence

        from Webinars as w

        inner join OrderDetails as od on od.ServiceID = w.WebinarID

        inner join orders as o on o.OrderID = od.OrderID

        inner join users as u on u.UserID = o.UserID

        inner join dbo.ServiceTypes ST on od.ServiceTypeID =
ST.ServiceTypeID and st.ServiceTypeName = 'Webinar');

```

Organizacja wydarzeń

Autor: Wiktor Sędzimir

Organizacja spotkań w ramach kursów

Autor: Wiktor Sędzimir

Widok *CoursesMeetingsOrganization* zawiera szczegółowe informacje o organizacji spotkań w ramach poszczególnych kursów.

```
create view CoursesMeetingsOrganization as
    with ClassroomsAddresses as (
        select
            ClassroomID,
            CountryName + ', ' +
            CityName + ', ' +
            Street + ' ' +
            ZipCode + ', ' +
            'classroom no. ' + cast(RoomNo as VARCHAR) as
FullAddress
        from Classrooms
        inner join Addresses
            on Classrooms.AddressID = Addresses.AddressID
        inner join Cities on
            Cities.CityID = Addresses.CityID
        inner join Countries
            on Cities.CountryID = Countries.CountryID
    )
    select
        Courses.CourseID,
        Courses.CourseName,
        case
            when FullAddress is not null then 'Stationary'
```

```

        when MeetingLink is not null then 'Online
Synchronous'

        when RecordingLink is not null then 'Online
Asynchronous'

    end as Form,

    coalesce(FullAddress, MeetingLink, RecordingLink) as
Location,

    MeetingID,

    CoursesMeetings.StartDate as MeetingStartDate,

    CoursesMeetings.EndDate as MeetingEndDate
from CoursesMeetings
left join StationaryCourse
    on StationaryCourseMeetingID = MeetingID
left join OnlineSyncCourse
    on OnlineSyncCourseMeetingID = MeetingID
left join OnlineAsyncCourse
    on OnlineAsyncCourseMeetingID = MeetingID
inner join Courses
    on Courses.CourseID = CoursesMeetings.CourseID
inner join CoursesTypes
    on Courses.CourseID = CoursesTypes.CourseID
left join ClassroomsAddresses
    on StationaryCourse.ClassroomID =
ClassroomsAddresses.ClassroomID;

```

Typy kursów ze względu na rodzaje spotkań

Autor: Wiktor Sędzimir

Widok *CoursesTypes* zawiera informacje o typie kursu ze względu na formę spotkań (*Stacjonarny*, *Online Synchroniczny*, *Online Asynchroniczny*, *Hybrydowy*).

```
create view CoursesTypes as
    with CourseMeetingsCategoryCount as (
        select
            Courses.CourseID,
            CourseName,
            count(StationaryCourseMeetingID) as
StationaryMeetingsCount,
            count(OnlineSyncCourseMeetingID) as
OnlineSyncMeetingsCount,
            count(OnlineAsyncCourseMeetingID) as
OnlineAsyncMeetingsCount
        from CoursesMeetings
        left join StationaryCourse
            on StationaryCourseMeetingID = MeetingID
        left join OnlineSyncCourse
            on OnlineSyncCourseMeetingID = MeetingID
        left join OnlineAsyncCourse
            on OnlineAsyncCourseMeetingID = MeetingID
        inner join Courses
            on Courses.CourseID = CoursesMeetings.CourseID
        group by Courses.CourseID, CourseName
    )
    select
        CourseID,
```

```

        CourseName,

        case

            when OnlineSyncMeetingsCount +
OnlineAsyncMeetingsCount = 0 then 'Stationary'

            when OnlineAsyncMeetingsCount +
StationaryMeetingsCount = 0 then 'Online Synchronous'

            when OnlineSyncMeetingsCount +
StationaryMeetingsCount = 0 then 'Online Asynchronous'

            else 'Hybrid'

        end as CourseType,

        StationaryMeetingsCount,

        OnlineSyncMeetingsCount,

        OnlineAsyncMeetingsCount

    from CourseMeetingsCategoryCount;

```

Organizacja spotkań w ramach studiów

Autor: Wiktor Sędzimir

Widok *StudiesMeetingsOrganization* zawiera szczegółowe informacje o organizacji spotkań w ramach poszczególnych studiów.

```

create view StudiesMeetingsOrganization as

    with ClassroomsAddresses as (

        select

            ClassroomID,

            CountryName + ', ' +

            CityName + ', ' +

            Street + ' ' +

            ZipCode + ', ' +

```

```

        'classroom no. ' + cast(RoomNo as VARCHAR) as
FullAddress

    from Classrooms

    inner join Addresses

        on Classrooms.AddressID = Addresses.AddressID

    inner join Cities on

        Cities.CityID = Addresses.CityID

    inner join Countries

        on Cities.CountryID = Countries.CountryID

)

select

    Studies.StudyID,

    StudyName,

    SemesterNo,

    SubjectName,

    iif(MeetingLink is not null, 'Online', 'Stationary')
as Form,

    iif(MeetingLink is not null, MeetingLink, FullAddress)
as Location,

    StudiesMeetings.StudyMeetupID,

    StudyMeetups.StartDate as MeetupStartDate,

    StudyMeetups.EndDate as MeetupEndDate,

    MeetingID,

    StudiesMeetings.StartDate as MeetingStartDate,

    StudiesMeetings.EndDate as MeetingEndDate

from StudyMeetups

```

```

        inner join StudySemesters
            on StudyMeetups.StudySemesterID =
StudySemesters.StudySemesterID

        inner join Studies
            on Studies.StudyID = StudySemesters.StudyID

        inner join StudiesMeetings
            on StudyMeetups.StudyMeetingID =
StudiesMeetings.StudyMeetingID

        inner join Subjects
            on Subjects.SubjectID = StudiesMeetings.SubjectID

        left join StationaryStudy
            on StationaryStudyMeetingID = MeetingID

        left join OnlineStudy
            on OnlineStudyMeetingID = MeetingID

        left join ClassroomsAddresses
            on StationaryStudy.ClassroomID =
ClassroomsAddresses.ClassroomID;

```

Informację o dyplomach do wygenerowania

Autor: Szymon Migas

Widok ten służy do otrzymania informacji o użytkownikach oraz ich ocenach ze studiów potrzebne do wygenerowania dyplomów ukończenia studiów

```

create view DiplomaInfo
as
select Firstname, Lastname, STU.StudyName,
    (SELECT TOP 1 [MEETUP END DATE] from GetStudyTimetable(FE.StudyID) order by
1 desc) as 'Graduation Date',
    A.Street, A.ZipCode, C.CityName, C2.CountryName

```

```

from Students S
inner join FinalExams FE on FE.StudentID = S.StudentID
inner join dbo.Grades G on FE.GradeID = G.GradeID
inner join Studies STU on FE.StudyID = STU.StudyID
inner join Users U on S.UserID = U.UserID
inner join Addresses A on A.AddressID = U.AddressID
inner join dbo.Cities C on C.CityID = A.CityID
inner join dbo.Countries C2 on C2.CountryID = C.CountryID
where GradeValue >= 3.0
and(S.StudentID in
    (select StudentID from Internships
     inner join dbo.InternshipDetails ID on Internships.InternshipID = ID.InternshipID
     where Internships.StudyID = FE.StudyID and ID.Pass = 1
     group by StudyID, StudentID
     having count(*) = 2))

```

Informację o wydarzeniach w ofercie uczelni

Autor: Szymon Migas

Widok ten stanowi zestawienie oferowanych przez uczelnie produktów, które jest w stanie zamówić klient.

```

Create view ServiceOffer as
select 'Webinar' as ServiceType,
       WebinarName as ServiceName,
       Date as Date,
       Price as Price,
       null as [Extra/Total Price]
from Webinars
where Date > GETDATE()
UNION
select 'Study Meetup',
       StudyName,

```

```

        StartDate,
        Price,
        ExtraPrice
from StudyMeetups
        inner join dbo.StudySemesters SS on
StudyMeetups.StudySemesterID = SS.StudySemesterID
        inner join Studies S on SS.StudyID = S.StudyID
where StartDate > GETDATE()
UNION
select
        'Course',
        CourseName,
        (SELECT TOP 1 StartDate FROM CoursesMeetings CM where C.CourseID
= CM.CourseID order by StartDate),
        FeePrice,
        TotalPrice
from Courses C
where (SELECT TOP 1 StartDate FROM CoursesMeetings CM where
C.CourseID = CM.CourseID order by StartDate) > GETDATE()

```

Procedury

Dodawanie Pracowników

Autor: Wiktor Sędzimir

Procedura *AddEmployee* służy do dodawania nowego pracownika.

```

CREATE PROCEDURE AddEmployee
    @Firstname VARCHAR(20),
    @Lastname VARCHAR(20),
    @BirthDate DATE,
    @HireDate DATE,

```

```
@CountryName VARCHAR(50),  
@CityName VARCHAR(50),  
@Street VARCHAR(30),  
@ZipCode VARCHAR(20),  
@Phone VARCHAR(15)  
AS  
BEGIN  
    SET NOCOUNT ON;  
  
    DECLARE @EmployeeID INT = (SELECT MAX(EmployeeID) + 1 FROM  
Employees);  
  
    IF NOT EXISTS(SELECT 1 FROM Countries WHERE CountryName =  
@CountryName)  
    BEGIN  
        RAISERROR('Given country does not exists.', 16, 1);  
        RETURN;  
    END;  
  
    IF EXISTS(SELECT 1 FROM Employees WHERE Phone = @Phone)  
    BEGIN  
        RAISERROR('Given phone number is not unique.', 16, 1);  
        RETURN;  
    END;
```

```
IF @BirthDate > @HireDate

BEGIN

    RAISERROR('Birth date cannot be later than hire
date.', 16, 1);

    RETURN;

END;


IF @BirthDate < '1900-01-01'

BEGIN

    RAISERROR('Birth date cannot be earlier than
1900-01-01.', 16, 1);

    RETURN;

END;


IF @HireDate > GETDATE()

BEGIN

    RAISERROR('Hire date cannot be later than current
date.', 16, 1);

    RETURN;

END;


DECLARE @CountryID INT = (SELECT CountryID FROM Countries
WHERE CountryName = @CountryName);


IF NOT EXISTS (

    SELECT 1
```



```
        FROM Cities

        INNER JOIN Countries

            ON Cities.CountryID = Countries.CountryID

        WHERE CityName = @CityName AND CountryName =
@CountryName

    )

    BEGIN

        EXEC AddCity

            @CityName = @CityName,

            @CountryID = @CountryID;

    END;

    DECLARE @CityID INT = (SELECT CityID FROM Cities WHERE
CityName = @CityName);

    IF NOT EXISTS(

        SELECT 1

        FROM Addresses

        WHERE ZipCode = @ZipCode AND CityID = @CityID

    )

    BEGIN

        EXEC AddAddress

            @CityID = @CityID,

            @Street = @Street,

            @ZipCode = @ZipCode
```

```

END;

DECLARE @AddressID INT = (SELECT AddressID FROM Addresses
WHERE ZipCode = @ZipCode AND Street = @Street AND CityID =
@CityID);

INSERT INTO Employees (EmployeeID, Firstname, Lastname,
BirthDate, HireDate, AddressID, Phone)

VALUES (@EmployeeID, @Firstname, @Lastname, @BirthDate,
@HireDate, @AddressID, @Phone);

PRINT 'Employee added successfully.';

END;

```

Dodawanie roli do pracownika

Autor: Wiktor Sędzimir

Procedura *AddEmployeeRole* służy do dodania nowej roli do istniejącego pracownika.

```

CREATE PROCEDURE AddEmployeeRole

    @EmployeeID INT,

    @RoleID INT

AS

BEGIN

    SET NOCOUNT ON;

    IF NOT EXISTS(SELECT 1 from Employees where EmployeeID =
@EmployeeID)

    BEGIN

        RAISERROR('Given employee does not exist', 16, 1);
    
```

```
        RETURN;

    END

    IF NOT EXISTS(SELECT 1 from Roles where RoleID = @RoleID)
    BEGIN
        RAISERROR('Given role does not exist', 16, 1);

        RETURN;
    END

    IF EXISTS(SELECT 1 from EmployeeRoles where EmployeeID =
@EmployeeID AND RoleID = @RoleID)
    BEGIN
        RAISERROR('Given employee already has given a role',
16, 1);

        RETURN;
    END

    INSERT INTO EmployeeRoles (EmployeeID, RoleID)
    VALUES (@EmployeeID, @RoleID);

    PRINT 'Employee Role pair added correctly';
END;
```

Dodawanie języka do tłumacza

Autor: Wiktor Sędzimir

Procedura *AddTranslatorLanguage* służy do dodania nowego języka do istniejącego tłumacza.

```
CREATE PROCEDURE AddTranslatorLanguage
    @EmployeeID INT,
    @LanguageID INT
AS
BEGIN

    SET NOCOUNT ON;

    DECLARE @TranslatorLanguageID INT = (SELECT
MAX(TranslatorLanguageID) + 1 FROM Translators);

    IF NOT EXISTS(SELECT 1 from Employees where EmployeeID =
@EmployeeID)
    BEGIN
        RAISERROR('Given employee does not exist', 16, 1);
        RETURN;
    END

    IF NOT EXISTS(SELECT 1 from Languages where LanguageID =
@LanguageID)
    BEGIN
        RAISERROR('Given language does not exist', 16, 1);
        RETURN;
    END

    IF NOT EXISTS (
```

```
        SELECT
            1
        FROM EmployeeRoles
        INNER JOIN Roles
            ON Roles.RoleID = EmployeeRoles.RoleID
        WHERE RoleName = 'Translator' AND EmployeeID =
@EmployeeID
    )
    BEGIN
        RAISERROR('Given employee does not have a translator
role', 16, 1);
        RETURN;
    END

    IF EXISTS(SELECT 1 from Translators where EmployeeID =
@EmployeeID AND LanguageID = @LanguageID)
    BEGIN
        RAISERROR('Given employee is already a translator for
given language', 16, 1);
        RETURN;
    END

    INSERT INTO Translators (TranslatorLanguageID, LanguageID,
EmployeeID)
        VALUES (@TranslatorLanguageID, @LanguageID, @EmployeeID);
```

```
PRINT 'Translator added correctly';  
END;
```

Dodawanie użytkownika

Autor: Wiktor Sędzimir

Procedura *AddUser* służy do dodania nowego użytkownika.

```
CREATE PROCEDURE AddUser  
  
    @Firstname VARCHAR(20),  
  
    @Lastname VARCHAR(20),  
  
    @Email VARCHAR(50),  
  
    @Password VARCHAR(255),  
  
    @Phone VARCHAR(15),  
  
    @CountryName VARCHAR(50),  
  
    @CityName VARCHAR(50),  
  
    @Street VARCHAR(30),  
  
    @ZipCode VARCHAR(20)  
  
AS  
  
BEGIN  
  
    SET NOCOUNT ON;  
  
  
    DECLARE @UserID INT = (SELECT MAX(UserID) + 1 FROM Users);  
  
  
    IF EXISTS(SELECT 1 FROM Users WHERE Email = @Email)  
  
    BEGIN  
  
        RAISERROR('Given email is not unique.', 16, 1);  
  
        RETURN;  
  
    END
```

```
END;

IF @Email NOT LIKE '%_@__%.__%'
BEGIN
    RAISERROR('Given email is not valid.', 16, 1);
    RETURN;
END;

IF EXISTS(SELECT 1 FROM Users WHERE Phone = @Phone)
BEGIN
    RAISERROR('Given phone number is not unique.', 16, 1);
    RETURN;
END;

DECLARE @CountryID INT = (SELECT CountryID FROM Countries
WHERE CountryName = @CountryName);

IF NOT EXISTS(
    SELECT 1
    FROM Cities
    INNER JOIN Countries
        ON Cities.CountryID = Countries.CountryID
    WHERE CityName = @CityName AND CountryName =
@CountryName
)
```

```
BEGIN

    EXEC AddCity

        @CityName = @CityName,

        @CountryID = @CountryID;

END;


DECLARE @CityID INT = (SELECT CityID FROM Cities WHERE
CityName = @CityName);


IF NOT EXISTS(

    SELECT 1

    FROM Addresses

    WHERE ZipCode = @ZipCode AND CityID = @CityID

)

BEGIN

    EXEC AddAddress

        @CityID = @CityID,

        @Street = @Street,

        @ZipCode = @ZipCode

END;


DECLARE @AddressID INT = (SELECT AddressID FROM Addresses
WHERE ZipCode = @ZipCode AND Street = @Street AND CityID =
@CityID);
```



```

    INSERT INTO Users (UserID, Firstname, Lastname, Email,
Password, Phone, AddressID)

    VALUES (@UserID, @Firstname, @Lastname, @Email, @Password,
@Phone, @AddressID);

    PRINT 'User added successfully.';

END;

```

Dodawanie kursu

Autor: Wiktor Sędzimir

Procedura *AddCourse* służy do dodania nowego kursu.

```

CREATE PROCEDURE AddCourse

    @CourseName VARCHAR(40),

    @FeePrice MONEY,

    @TotalPrice MONEY,

    @TranslatorLanguageID INT,

    @CourseCoordinatorID INT

AS

BEGIN

    SET NOCOUNT ON;

    DECLARE @CourseID INT = (SELECT MAX(CourseID) + 1 FROM
Courses);

    BEGIN TRY

        IF NOT @FeePrice > 0

            BEGIN

```

```
        RAISERROR('Fee price must be greater than 0.', 16,
1);

    END;

    IF NOT @TotalPrice > @FeePrice
    BEGIN
        RAISERROR('Total price must be greater than fee
price.', 16, 1);
    END;

    IF @TranslatorLanguageID IS NOT NULL AND NOT
EXISTS(SELECT 1 FROM Translators WHERE TranslatorLanguageID =
@TranslatorLanguageID)
    BEGIN
        RAISERROR('Given translator language pair does not
exist.', 16, 1);
    END;

    IF NOT EXISTS(
        SELECT 1
        FROM EmployeeRoles
        INNER JOIN Roles
            ON Roles.RoleID = EmployeeRoles.RoleID
        WHERE RoleName = 'Course Coordinator' AND
EmployeeID = @CourseCoordinatorID
    )
```

```

        BEGIN

            RAISERROR('Given course coordinator does not
exist.', 16, 1);

        END;

        INSERT INTO Courses (CourseID, CourseName, FeePrice,
TotalPrice, TranslatorLanguageID, CourseCoordinatorID)

        VALUES (@CourseID, @CourseName, @FeePrice,
@TotalPrice, @TranslatorLanguageID, @CourseCoordinatorID);

        PRINT 'Course added successfully.';

    END TRY

    BEGIN CATCH

        DECLARE @ErrorMsg NVARCHAR(4000), @ErrorSeverity INT,
@ErrorState INT;

        SELECT @ErrorMsg = ERROR_MESSAGE(), @ErrorSeverity =
ERROR_SEVERITY(), @ErrorState = ERROR_STATE();

        RAISERROR(@ErrorMsg, @ErrorSeverity, @ErrorState);

    END CATCH

END;

```

Dodawanie spotkania stacjonarnego w ramach kursu

Autor: Wiktor Sędzimir

Procedura *AddStationaryCourseMeeting* służy do dodania nowego spotkania stacjonarnego w ramach danego kursu.

```

CREATE PROCEDURE AddStationaryCourseMeeting

    @CourseID INT,

    @StartDate DATETIME,

```

```
@EndDate DATETIME,

@CourseInstructorID INT,

@LimitOfParticipants INT,

@ClassroomID INT

AS

BEGIN

    SET NOCOUNT ON;

    DECLARE @StationaryCourseMeetingID INT = (SELECT
MAX(MeetingID) + 1 FROM CoursesMeetings);

    IF NOT EXISTS(SELECT 1 FROM Courses WHERE CourseID =
@CourseID)

    BEGIN

        RAISERROR('Given course does not exist.', 16, 1);

        RETURN;

    END;

    IF @StartDate < GETDATE()

    BEGIN

        RAISERROR('Date cannot be in the past.', 16, 1);

        RETURN;

    END;

    IF @StartDate >= @EndDate
```

```
BEGIN

    RAISERROR('Start date must be earlier than end date.',
16, 1);

    RETURN;

END;

IF @LimitOfParticipants <= 0

BEGIN

    RAISERROR('Limit of participants must be greater than
0.', 16, 1);

    RETURN;

END;

IF @LimitOfParticipants > (SELECT Capacity FROM Classrooms
WHERE ClassroomID = @ClassroomID)

BEGIN

    RAISERROR('Limit of participants cannot be greater
than classroom capacity.', 16, 1);

    RETURN;

END;

IF NOT EXISTS (

    SELECT 1

    FROM EmployeeRoles

    INNER JOIN Roles

        ON EmployeeRoles.RoleID = Roles.RoleID
```

```
        WHERE RoleName = 'Course Instructor' AND EmployeeID =
@CourseInstructorID
    )
    BEGIN
        RAISERROR('Given course instructor does not exist.',
16, 1);
        RETURN;
    END;

    IF NOT EXISTS(SELECT 1 FROM Classrooms WHERE ClassroomID =
@ClassroomID)
    BEGIN
        RAISERROR('Given classroom does not exist.', 16, 1);
        RETURN;
    END;

    IF EXISTS(
        SELECT 1
        FROM CoursesMeetings
        WHERE CourseID = @CourseID AND StartDate = @StartDate
AND EndDate = @EndDate
    )
    BEGIN
        RAISERROR('Given course meeting already exists.', 16,
1);
        RETURN;
    END;
```

```

END;

INSERT INTO CoursesMeetings (MeetingID, CourseID,
StartDate, EndDate, CourseInstructorID, LimitOfParticipants)
VALUES (@StationaryCourseMeetingID, @CourseID, @StartDate,
@EndDate, @CourseInstructorID, @LimitOfParticipants);

INSERT INTO StationaryCourse (StationaryCourseMeetingID,
ClassroomID)
VALUES (@StationaryCourseMeetingID, @ClassroomID);

PRINT 'Stationary course meeting added successfully.';
END;

```

Dodawanie spotkania online synchronicznego w ramach kursu

Autor: Wiktor Sędzimir

Procedura *AddOnlineSyncCourseMeeting* służy do dodania nowego spotkania online synchronicznego w ramach danego kursu.

```

CREATE PROCEDURE AddOnlineSyncCourseMeeting
    @CourseID INT,
    @StartDate DATETIME,
    @EndDate DATETIME,
    @CourseInstructorID INT,
    @LimitOfParticipants INT = NULL,
    @MeetingLink VARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

```

```
    DECLARE @OnlineSyncCourseMeetingID INT = (SELECT
MAX(MeetingID) + 1 FROM CoursesMeetings);

    IF NOT EXISTS(SELECT 1 FROM Courses WHERE CourseID =
@CourseID)

    BEGIN

        RAISERROR('Given course does not exist.', 16, 1);

        RETURN;

    END;

    IF @StartDate < GETDATE()

    BEGIN

        RAISERROR('Date cannot be in the past.', 16, 1);

        RETURN;

    END;

    IF @StartDate >= @EndDate

    BEGIN

        RAISERROR('Start date must be earlier than end date.',
16, 1);

        RETURN;

    END;

    IF @LimitOfParticipants IS NOT NULL AND
@LimitOfParticipants <= 0
```



```
BEGIN

    RAISERROR('Limit of participants must be greater than
0.', 16, 1);

    RETURN;

END;

IF NOT EXISTS (

    SELECT 1

    FROM EmployeeRoles

    INNER JOIN Roles

        ON EmployeeRoles.RoleID = Roles.RoleID

    WHERE RoleName = 'Course Instructor' AND EmployeeID =
@CourseInstructorID

)

BEGIN

    RAISERROR('Given course instructor does not exist.',
16, 1);

    RETURN;

END;

IF EXISTS (

    SELECT 1

    FROM CoursesMeetings

    WHERE CourseID = @CourseID AND StartDate = @StartDate
AND EndDate = @EndDate

)
```

```

BEGIN

    RAISERROR('Given course meeting already exists.', 16,
1);

    RETURN;

END;

INSERT INTO CoursesMeetings (MeetingID, CourseID,
StartDate, EndDate, CourseInstructorID, LimitOfParticipants)
VALUES (@OnlineSyncCourseMeetingID, @CourseID, @StartDate,
@EndDate, @CourseInstructorID, @LimitOfParticipants);

INSERT INTO OnlineSyncCourse (OnlineSyncCourseMeetingID,
MeetingLink)
VALUES (@OnlineSyncCourseMeetingID, @MeetingLink);

PRINT 'Online synchronous course meeting added
successfully.';
END;

```

Dodawanie spotkania online asynchronicznego w ramach kursu

Autor: Wiktor Sędzimir

Procedura *AddOnlineAsyncCourseMeeting* służy do dodania nowego spotkania online asynchronicznego w ramach danego kursu.

```

CREATE PROCEDURE AddOnlineAsyncCourseMeeting

    @CourseID INT,

    @StartDate DATETIME,

    @EndDate DATETIME,

    @CourseInstructorID INT,

```

```
@LimitOfParticipants INT = NULL,  
@RecordingLink VARCHAR(100)  
AS  
BEGIN  
    SET NOCOUNT ON;  
  
    DECLARE @OnlineAsyncCourseMeetingID INT = (SELECT  
MAX(MeetingID) + 1 FROM CoursesMeetings);  
  
    IF NOT EXISTS(SELECT 1 FROM Courses WHERE CourseID =  
@CourseID)  
    BEGIN  
        RAISERROR('Given course does not exist.', 16, 1);  
        RETURN;  
    END;  
  
    IF @StartDate < GETDATE()  
    BEGIN  
        RAISERROR('Date cannot be in the past.', 16, 1);  
        RETURN;  
    END;  
  
    IF @StartDate >= @EndDate  
    BEGIN  
        RAISERROR('Start date must be earlier than end date.',  
16, 1);
```

```
        RETURN;

    END;

    IF @LimitOfParticipants IS NOT NULL AND
@LimitOfParticipants <= 0

    BEGIN

        RAISERROR('Limit of participants must be greater than
0.', 16, 1);

        RETURN;

    END;

    IF NOT EXISTS (

        SELECT 1

        FROM EmployeeRoles

        INNER JOIN Roles

            ON EmployeeRoles.RoleID = Roles.RoleID

        WHERE RoleName = 'Course Instructor' AND EmployeeID =
@CourseInstructorID

    )

    BEGIN

        RAISERROR('Given course instructor does not exist.',
16, 1);

        RETURN;

    END;

    IF EXISTS (
```

```
        SELECT 1
        FROM CoursesMeetings
        WHERE CourseID = @CourseID AND StartDate = @StartDate
        AND EndDate = @EndDate
    )
    BEGIN
        RAISERROR('Given course meeting already exists.', 16,
1);

        RETURN;
    END;

    INSERT INTO CoursesMeetings (MeetingID, CourseID,
StartDate, EndDate, CourseInstructorID, LimitOfParticipants)
    VALUES (@OnlineAsyncCourseMeetingID, @CourseID,
@StartDate, @EndDate, @CourseInstructorID,
@LimitOfParticipants);

    INSERT INTO OnlineAsyncCourse (OnlineAsyncCourseMeetingID,
RecordingLink)
    VALUES (@OnlineAsyncCourseMeetingID, @RecordingLink);

    PRINT 'Online asynchronous course meeting added
successfully.';
END;
```

Dodawanie obecności na spotkaniu kursu dla danego użytkownika

Autor: Wiktor Sędzimir

Procedura *AddCourseMeetingAttendance* służy do rejestrowania obecności danego użytkownika na spotkaniu w ramach kursu.

```
CREATE PROCEDURE AddCourseMeetingAttendance
    @CourseMeetingID INT,
    @UserID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS(SELECT 1 FROM CoursesMeetings WHERE
MeetingID = @CourseMeetingID)
    BEGIN
        RAISERROR('Given course meeting does not exist.', 16,
1);

        RETURN;
    END;

    IF NOT EXISTS(SELECT 1 FROM Users WHERE UserID = @UserID)
    BEGIN
        RAISERROR('Given user does not exist.', 16, 1);

        RETURN;
    END;
```

```
    IF NOT EXISTS (SELECT 1 FROM
UsersAuthorizedForCourseMeeting WHERE MeetingID =
@CourseMeetingID AND UserID = @UserID)

    BEGIN

        RAISERROR('Given user is not authorized for given
course meeting.', 16, 1);

        RETURN;

    END;

    IF EXISTS (SELECT 1 FROM CoursesAttendance WHERE
CourseMeetingID = @CourseMeetingID AND UserID = @UserID)

    BEGIN

        RAISERROR ('User is already on the list', 16, 1);

        RETURN;

    END;

    INSERT INTO CoursesAttendance (CourseMeetingID, UserID)
VALUES (@CourseMeetingID, @UserID);

    PRINT 'User added to the course attendance list';

END;
```

Dodawanie oceny z egzaminu końcowego studenta

Autor: Krzysztof Chmielewski

Procedura *AddFinalExam* dodaje rekord do tabeli *FinalExams*, służy do wpisywania oceny z końcowego egzaminu studenta zapisanego na dane studia.

```
CREATE PROCEDURE AddFinalExam
    @StudyID int,
    @StudentID int,
    @GradeID int
AS
BEGIN
    SET NOCOUNT ON

    --handling study ID exceptions
    IF @StudyID IS NULL
        BEGIN
            RAISERROR ('Study ID cannot be null', 16, 1)

            RETURN

        END

    IF @StudyID NOT IN (SELECT StudyID FROM Studies)
        BEGIN
            RAISERROR ('There is no study with that ID', 16, 1)

            RETURN

        END

    --handling student ID exceptions
    IF @StudentID IS NULL
        BEGIN
            RAISERROR ('Student ID cannot be null', 16, 1)
```



```
        RETURN

    END

    IF @StudentID NOT IN (SELECT StudentID FROM Students)

        BEGIN

            RAISERROR ('There is no student with that ID', 16, 1)

            RETURN

        END

        IF @StudyID != (SELECT StudyID FROM Students WHERE StudentID =
@StudentID)

            BEGIN

                RAISERROR ('This student is not on given studies', 16,
1)

                RETURN

            END

--handling grade ID exceptions

    IF @GradeID IS NULL

        BEGIN

            RAISERROR ('Grade ID cannot be null', 16, 1)

            RETURN

        END

    IF @GradeID NOT IN (SELECT GradeID FROM Grades)

        BEGIN

            RAISERROR ('There is no grade with that ID', 16, 1)

            RETURN

        END

--adding correct values to the table
```

```
INSERT INTO FinalExams (StudyID, StudentID, GradeID)
VALUES (@StudyID, @StudentID, @GradeID)

PRINT 'Final exam added successfully'

END
```

Dodawanie rekordu praktyk dla danego studenta

Autor: Krzysztof Chmielewski

Procedura *AddInternshipDetail* dodaje rekord do tabeli *InternshipDetails*, służy do wpisywania studenta na praktyki do danej firmy w ramach konkretnego cyklu praktyk

```
CREATE PROCEDURE AddInternshipDetail
    @InternshipID int,
    @StudentID int,
    @CompanyName varchar(20),
    @Pass bit
AS
BEGIN
    SET NOCOUNT ON

    --handling internship id exceptions
    IF @InternshipID IS NULL
    BEGIN
        RAISERROR ('Internship ID cannot be null', 16, 1)

        RETURN
    END

    IF @InternshipID NOT IN (SELECT InternshipID FROM Internships)
    BEGIN
        RAISERROR ('There is no internship with that ID', 16,
1)

        RETURN
    END

    --handling student id exceptions
    IF @StudentID IS NULL
```

```
BEGIN

    RAISERROR ('Student ID cannot be null', 16, 1)

    RETURN

END

IF @StudentID NOT IN (SELECT StudentID FROM Students)

BEGIN

    RAISERROR ('There is no student with that ID', 16, 1)

    RETURN

END

IF (SELECT COUNT(*) FROM InternshipDetails WHERE StudentID =
@StudentID) = 2

BEGIN

    RAISERROR ('This student is already registered for
both internships', 16, 1)

    RETURN

END

IF (SELECT COUNT(*) FROM InternshipDetails WHERE StudentID =
@StudentID) = 1 AND

    @InternshipID IN (SELECT InternshipID FROM
InternshipDetails WHERE StudentID = @StudentID)

BEGIN

    RAISERROR ('This student is already registered for
internship with that ID', 16, 1)

    RETURN

END

--handling company name exceptions

IF @CompanyName IS NULL

BEGIN
```

```
        RAISERROR ('Company name cannot be null', 16, 1)

        RETURN

    END

    IF @StudentID NOT IN (SELECT StudentID FROM Students)

        BEGIN

            RAISERROR ('There is no student with that ID', 16, 1)

            RETURN

        END

        INSERT INTO InternshipDetails (InternshipID, StudentID,
CompanyName, Pass)

        VALUES (@InternshipID, @StudentID, @CompanyName, @Pass)

        PRINT 'Internship detail added successfully'

    END
```

Dodawanie spotkania online dla studiów

Autor: Krzysztof Chmielewski

Procedura *AddOnlineStudyMeeting* dodaje rekord do tabeli *OnlineStudyMeetings*, służy do wpisywania spotkania online i wszystkich jego detali.

```
CREATE PROCEDURE AddOnlineStudyMeeting
    @StudyMeetupID int,
    @SubjectID int,
    @StartDate datetime,
    @EndDate datetime,
    @LecturerID int,
    @LimitOfMeetingParticipants int,
    @TranslatorLanguageID int,
    @MeetingLink varchar(100)
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @MeetingID int = (SELECT MAX(MeetingID) + 1 FROM
StudiesMeetings)

    --handling study meetup exceptions
    IF @StudyMeetupID IS NULL
        BEGIN
            RAISERROR ('Study meetup ID cannot be null', 16 ,1)

            RETURN
        END

    IF @StudyMeetupID NOT IN (SELECT StudyMeetupID FROM
StudyMeetups)
```

```

        BEGIN
            RAISERROR ('There is no study meetup with that ID',
16, 1)

            RETURN

        END

--handling subjects exceptions
IF @SubjectID IS NULL
    BEGIN
        RAISERROR ('Subject ID cannot be null', 16 ,1)

        RETURN

    END

IF @SubjectID NOT IN (SELECT SubjectID FROM Subjects)
    BEGIN
        RAISERROR ('There is no subject with that ID', 16, 1)

        RETURN

    END

IF @SubjectID NOT IN (
    SELECT DISTINCT sch.SubjectID FROM StudiesSchedule AS sch
        INNER JOIN StudyMeetups AS sm ON sm.StudySemesterID =
sch.StudySemesterID
        WHERE sm.StudyMeetupID = @StudyMeetupID)
    BEGIN
        RAISERROR ('This subject is not in the schedule of
study and semester on which given meetup is', 16, 1)

        RETURN

    END

```

```
--handling dates exceptions

IF @StartDate IS NULL OR @EndDate IS NULL

    BEGIN

        RAISERROR ('Date cannot be null', 16, 1)

        RETURN

    END

IF @StartDate < GETDATE()

    BEGIN

        RAISERROR ('Cannot add meeting with past datetime',
16, 1)

        RETURN

    END

IF @EndDate <= @StartDate

    BEGIN

        RAISERROR ('End date must be further in time than
start date', 16, 1)

        RETURN

    END

    DECLARE @MeetupStartDate datetime = (SELECT StartDate FROM
StudyMeetups WHERE StudyMeetupID = @StudyMeetupID)

    DECLARE @MeetupEndDate datetime = (SELECT EndDate FROM
StudyMeetups WHERE StudyMeetupID = @StudyMeetupID)

    IF @StartDate < @MeetupStartDate OR @StartDate >=
@MeetupEndDate OR

        @EndDate > @MeetupEndDate OR @EndDate <= @MeetupStartDate

        BEGIN
```



```

        RAISERROR ('Given dates are out of scope of meetup
dates', 16, 1)

        RETURN

    END

    IF @StartDate > DATEADD(HOUR, -1, @MeetupEndDate)

        BEGIN

            RAISERROR ('Start date must be at least an hour
before meetup end date', 16, 1)

            RETURN

        END

    IF DATEDIFF(MINUTE, @StartDate, @EndDate) != 45

        BEGIN

            RAISERROR ('Meeting must have a duration of exactly
45 minutes', 16, 1)

            RETURN

        END

    IF EXISTS(SELECT 1 FROM StudiesMeetings WHERE StudyMeetupID =
@StudyMeetupID AND (@StartDate < EndDate AND DATEADD(MINUTE, 15,
@EndDate) > StartDate))

        BEGIN

            RAISERROR ('There already is a meeting that overlaps
with the given time', 16, 1)

            RETURN

        --handling lecturer exceptions

        IF @LecturerID IS NULL

            BEGIN

                RAISERROR ('Lecturer ID cannot be null', 16, 1)

                RETURN

```

```

        END

        IF @LecturerID NOT IN (

            SELECT e.EmployeeID FROM Employees AS e

                INNER JOIN EmployeeRoles AS er ON e.EmployeeID =
er.EmployeeID

                INNER JOIN Roles AS r ON er.RoleID = r.RoleID

                WHERE r.RoleName = 'Lecturer')

        BEGIN

            RAISERROR ('Given employee is not a lecturer', 16, 1)

            RETURN

        END

        --handling limit of participants exceptions

        IF @LimitOfMeetingParticipants IS NULL

            BEGIN

                RAISERROR ('Limit of participants cannot be null', 16,
1)

                RETURN

            END

        IF @LimitOfMeetingParticipants <= 0

            BEGIN

                RAISERROR ('Limit of participants must be greater than
0', 16, 1)

                RETURN

            END

        IF @LimitOfMeetingParticipants <= (

            SELECT s.LimitOfStudents FROM Studies AS s

```

```

        INNER JOIN StudySemesters AS ss ON s.StudyID =
ss.StudyID

        INNER JOIN StudyMeetups AS sm ON ss.StudySemesterID =
sm.StudySemesterID

        WHERE sm.StudyMeetupID = @StudyMeetupID

    BEGIN

        RAISERROR ('Limit of participants must be at least of
value determined by studies on which given meeting is supposed to
be', 16 , 1)

        RETURN

    END

--handling translator language id exceptions

    IF @TranslatorLanguageID NOT IN (SELECT TranslatorLanguageID
FROM Translators)

    BEGIN

        RAISERROR ('There is no translator language pair with
that ID', 16, 1)

        RETURN

    END

--handling meeting link exceptions

    IF @MeetingLink IS NULL

    BEGIN

        RAISERROR ('Meeting link cannot be null', 16, 1)

        RETURN

    END

    IF @MeetingLink NOT LIKE
'https://skibidischool.pl/online-study/%'

```

```
BEGIN

    RAISERROR ('Invalid address of meeting link', 16, 1)

    RETURN

END

--adding correct values to the tables

INSERT INTO StudiesMeetings (MeetingID, StudyMeetupID,
SubjectID, StartDate, EndDate, LecturerID,
LimitOfMeetingParticipants, TranslatorLanguageID)

VALUES (@MeetingID, @StudyMeetupID, @SubjectID, @StartDate,
@EndDate, @LecturerID, @LimitOfMeetingParticipants,
@TranslatorLanguageID)

INSERT INTO OnlineStudy (OnlineStudyMeetingID, MeetingLink)

VALUES (@MeetingID, @MeetingLink)

PRINT 'Online study meeting added successfully'

END
```

Dodawanie spotkania stacjonarnego dla studiów

Autor: Krzysztof Chmielewski

Procedura *AddStationaryStudyMeeting* dodaje rekord do tabeli *StationaryStudyMeetings*, służy do wpisywania spotkania stacjonarnego i wszystkich jego detali.

```
CREATE PROCEDURE AddStationaryStudyMeeting

    @StudyMeetupID int,

    @SubjectID int,

    @StartDate datetime,

    @EndDate datetime,

    @LecturerID int,

    @LimitOfMeetingParticipants int,

    @TranslatorLanguageID int,

    @ClassroomID int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @MeetingID int = (SELECT MAX(MeetingID) + 1 FROM
StudiesMeetings)

    --handling study meetup exceptions
    IF @StudyMeetupID IS NULL
        BEGIN
            RAISERROR ('Study meetup ID cannot be null', 16 ,1)

            RETURN
        END

    IF @StudyMeetupID NOT IN (SELECT StudyMeetupID FROM
StudyMeetups)
```

```

        BEGIN
            RAISERROR ('There is no study meetup with that ID',
16, 1)

            RETURN

        END

--handling subjects exceptions
IF @SubjectID IS NULL
    BEGIN
        RAISERROR ('Subject ID cannot be null', 16 ,1)

        RETURN

    END

IF @SubjectID NOT IN (SELECT SubjectID FROM Subjects)
    BEGIN
        RAISERROR ('There is no subject with that ID', 16, 1)

        RETURN

    END

IF @SubjectID NOT IN (
    SELECT DISTINCT sch.SubjectID FROM StudiesSchedule AS sch
        INNER JOIN StudyMeetups AS sm ON sm.StudySemesterID =
sch.StudySemesterID
        WHERE sm.StudyMeetupID = @StudyMeetupID)
    BEGIN
        RAISERROR ('This subject is not in the schedule of
study and semester on which given meetup is', 16, 1)

        RETURN

    END

```

```

--handling dates exceptions

IF @StartDate IS NULL OR @EndDate IS NULL

    BEGIN

        RAISERROR ('Date cannot be null', 16, 1)

        RETURN

    END

IF @StartDate < GETDATE()

    BEGIN

        RAISERROR ('Cannot add meetup with past datetime', 16,
1)

        RETURN

    END

IF @EndDate <= @StartDate

    BEGIN

        RAISERROR ('End date must be further in time than
start date', 16, 1)

        RETURN

    END

    DECLARE @MeetupStartDate datetime = (SELECT StartDate FROM
StudyMeetups WHERE StudyMeetupID = @StudyMeetupID)

    DECLARE @MeetupEndDate datetime = (SELECT EndDate FROM
StudyMeetups WHERE StudyMeetupID = @StudyMeetupID)

    IF @StartDate < @MeetupStartDate OR @StartDate >=
@MeetupEndDate OR

        @EndDate > @MeetupEndDate OR @EndDate <= @MeetupStartDate

        BEGIN

```

```

        RAISERROR ('Given dates are out of scope of meetup
dates', 16, 1)

        RETURN

    END

    IF @StartDate > DATEADD(HOUR, -1, @MeetupEndDate)

        BEGIN

            RAISERROR ('Start date must be at least an hour
before meetup end date', 16, 1)

            RETURN

        END

    IF DATEDIFF(MINUTE, @StartDate, @EndDate) != 45

        BEGIN

            RAISERROR ('Meeting must have a duration of exactly
45 minutes', 16, 1)

            RETURN

        END

    IF EXISTS(SELECT 1 FROM StudiesMeetings WHERE StudyMeetupID =
@StudyMeetupID AND (@StartDate < EndDate AND DATEADD(MINUTE, 15,
@EndDate) > StartDate))

        BEGIN

            RAISERROR ('There already is a meeting that overlaps
with the given time', 16, 1)

            RETURN

        END

    --handling lecturer exceptions

    IF @LecturerID IS NULL

        BEGIN

            RAISERROR ('Lecturer ID cannot be null', 16, 1)

```



```

        END

        IF @LecturerID NOT IN (

            SELECT e.EmployeeID FROM Employees AS e

                INNER JOIN EmployeeRoles AS er ON e.EmployeeID =
er.EmployeeID

                INNER JOIN Roles AS r ON er.RoleID = r.RoleID

                WHERE r.RoleName = 'Lecturer')

        BEGIN

            RAISERROR ('Given employee is not a lecturer', 16, 1)

            RETURN

        END

        --handling limit of participants exceptions

        IF @LimitOfMeetingParticipants IS NULL

            BEGIN

                RAISERROR ('Limit of participants cannot be null', 16,
1)

                RETURN

            END

        IF @LimitOfMeetingParticipants <= 0

            BEGIN

                RAISERROR ('Limit of participants must be greater than
0', 16, 1)

                RETURN

            END

        IF @LimitOfMeetingParticipants <= (

            SELECT s.LimitOfStudents FROM Studies AS s

```

```

        INNER JOIN StudySemesters AS ss ON s.StudyID =
ss.StudyID

        INNER JOIN StudyMeetups AS sm ON ss.StudySemesterID =
sm.StudySemesterID

        WHERE sm.StudyMeetupID = @StudyMeetupID)

    BEGIN

        RAISERROR ('Limit of participants must be at least of
value determined by studies on which given meeting is supposed to
be', 16 , 1)

        RETURN

    END

--handling translator language id exceptions

    IF @TranslatorLanguageID NOT IN (SELECT TranslatorLanguageID
FROM Translators)

    BEGIN

        RAISERROR ('There is no translator language pair with
that ID', 16, 1)

        RETURN

    END

--handling classroom exceptions

    IF @ClassroomID IS NULL

    BEGIN

        RAISERROR ('Classroom ID cannot be null', 16, 1)

        RETURN

    END

    IF @ClassroomID NOT IN (SELECT ClassroomID FROM Classrooms)

    BEGIN

```

```
        RAISERROR ('There is no classroom with that ID', 16,
1)

        RETURN

    END

    --adding correct values to the tables

    INSERT INTO StudiesMeetings (MeetingID, StudyMeetupID,
SubjectID, StartDate, EndDate, LecturerID,
LimitOfMeetingParticipants, TranslatorLanguageID)

        VALUES (@MeetingID, @StudyMeetupID, @SubjectID, @StartDate,
@EndDate, @LecturerID, @LimitOfMeetingParticipants,
@TranslatorLanguageID)

    INSERT INTO StationaryStudy (StationaryStudyMeetingID,
ClassroomID)

        VALUES (@MeetingID, @ClassroomID)

    PRINT 'Stationary study meeting added successfully'

END
```

Dodawanie elementu do słownika (semestr na studiach) - (przedmiot)

Autor: Krzysztof Chmielewski

Procedura *AddStudiesSchedule* dodaje rekord do tabeli *StudiesSchedule*, służy do wpisywania rekordu do tabeli słownikowej, która przechowuje dane o tym jaki przedmiot jest na danym semestrze na konkretnych studiach.

```
CREATE PROCEDURE AddStudiesSchedule

    @StudySemesterID int,

    @SubjectID int
AS
BEGIN
    SET NOCOUNT ON

    --handling study semester exceptions

    IF @StudySemesterID IS NULL
    BEGIN
        RAISERROR ('Study schedule ID cannot be null', 16, 1)

        RETURN
    END

    IF @StudySemesterID NOT IN (SELECT StudySemesterID FROM
StudySemesters)
    BEGIN
        RAISERROR ('There is no study semester with that ID',
16, 1)

        RETURN
    END

    --handling subject exceptions

    IF @SubjectID IS NULL
    BEGIN
```

```
        RAISERROR ('Subject ID cannot be null', 16, 1)

        RETURN

    END

    IF @SubjectID NOT IN (SELECT SubjectID FROM Subjects)

        BEGIN

            RAISERROR ('There is no subject with that ID', 16, 1)

            RETURN

        END

    --adding correct values to the table

    INSERT INTO StudiesSchedule (StudySemesterID, SubjectID)

    VALUES (@StudySemesterID, @SubjectID)

    PRINT 'Studies schedule added successfully'

END
```

Dodawanie elementu do słownika (spotkanie studenckie) - (student)

Autor: Krzysztof Chmielewski

Procedura *AddStudyAttendance* dodaje rekord do tabeli *StudyAttendance*, służy do wpisywania rekordu do tabeli słownikowej, która przechowuje dane o studentach, którzy byli obecni na konkretnych spotkaniach.

```
CREATE PROCEDURE AddStudyAttendance
    @StudyMeetingID int,
    @StudentID int
AS
BEGIN
    SET NOCOUNT ON

    --handling study meeting id exceptions
    IF @StudyMeetingID IS NULL
    BEGIN
        RAISERROR ('Study meeting ID cannot be null', 16, 1)
        RETURN
    END

    IF @StudyMeetingID NOT IN (SELECT MeetingID FROM
StudiesMeetings)
    BEGIN
        RAISERROR ('There is no study meeting with that ID',
16, 1)
        RETURN
    END

    --handling student id exceptions
    IF @StudentID IS NULL
    BEGIN
```

```

        RAISERROR ('Student ID cannot be null', 16, 1)

        RETURN

    END

    IF @StudentID NOT IN (SELECT StudentID FROM Students)

        BEGIN

            RAISERROR ('There is no student with that ID', 16, 1)

            RETURN

        END

        --check if this student is on the list of all students that
        should be on this meeting

        IF @StudentID NOT IN (

            select distinct s2.StudentID from StudiesMeetings as sm

                inner join dbo.StudyMeetups S on sm.StudyMeetupID =
S.StudyMeetupID

                inner join dbo.StudySemesters SS on S.StudySemesterID
= SS.StudySemesterID

                inner join dbo.Students S2 on ss.StudyID = S2.StudyID

            where sm.MeetingID = @StudyMeetingID)

            BEGIN

                RAISERROR ('This student is not supposed to be on this
meeting', 16, 1)

                RETURN

            END

            INSERT INTO StudyAttendance (StudyMeetingID, StudentID)

            VALUES (@StudyMeetingID, @StudentID)

            PRINT 'Student attendance added successfully'

        END

```

Dodawanie zjazdu studenckiego

Autor: Krzysztof Chmielewski

Procedura *AddStudyMeetup* dodaje rekord do tabeli *StudyMeetups*, służy do wpisywania zjazdu studenckiego oraz jego detali.

```
CREATE PROCEDURE AddStudyMeetup
    @StudySemesterID int,
    @StartDate datetime,
    @EndDate datetime,
    @Price money,
    @ExtraPrice money
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @StudyMeetupID int = (SELECT MAX(StudyMeetupID) + 1
FROM StudyMeetups)

    --handling study semester exceptions
    IF @StudySemesterID IS NULL
        BEGIN
            RAISERROR ('Study schedule ID cannot be null', 16, 1)
            RETURN
        END

    IF @StudySemesterID NOT IN (SELECT StudySemesterID FROM
StudySemesters)
        BEGIN
            RAISERROR ('There is no study semester with that ID',
16, 1)

            RETURN
        END
```



```

        END

        --handling dates exceptions

        IF @StartDate IS NULL OR @EndDate IS NULL

            BEGIN

                RAISERROR ('Date cannot be null', 16, 1)

                RETURN

            END

        IF @StartDate < GETDATE()

            BEGIN

                RAISERROR ('Cannot add meetup with past datetime', 16,
1)

                RETURN

            END

        IF @EndDate <= @StartDate

            BEGIN

                RAISERROR ('End date must be further in time than
start date', 16, 1)

                RETURN

            END

        --handling price exceptions

        IF @Price IS NULL OR @ExtraPrice IS NULL

            BEGIN

                RAISERROR ('Price cannot be null', 16, 1)

                RETURN

            END

        IF @Price <= 0 OR @ExtraPrice <= 0

```

```
BEGIN

    RAISERROR ('Price must be greater than 0', 16, 1)

    RETURN

END

--adding correct values to the table

INSERT INTO StudyMeetups (StudyMeetupID, StudySemesterID,
StartDate, EndDate, Price, ExtraPrice)

VALUES (@StudyMeetupID, @StudySemesterID, @StartDate,
@EndDate, @Price,@ExtraPrice)

PRINT 'Study meetup added successfully'

END
```

Dodawanie przedmiotu

Autor: Krzysztof Chmielewski

Procedura *AddSubject* dodaje rekord do tabeli Subjects, służy do wpisywania przedmiotu.

```
CREATE PROCEDURE AddSubject
    @SubjectName varchar(30),
    @Description text
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @SubjectID int = (SELECT MAX(SubjectID) + 1 FROM
Subjects)

    --handling study name exceptions
    IF @SubjectName IS NULL
        BEGIN
            RAISERROR ('Subject name cannot be null', 16, 1)

            RETURN
        END

    IF @SubjectName IN (SELECT SubjectName FROM Subjects)
        BEGIN
            RAISERROR ('There already exists subject with that
name', 16, 1)

            RETURN
        END

    --handling description exceptions
    IF @Description IS NULL
```

```
BEGIN

    RAISERROR ('Description cannot be null', 16, 1)

    RETURN

END

--adding correct values to the table

INSERT INTO Subjects (SubjectID, SubjectName, Description)

VALUES (@SubjectID, @SubjectName, @Description)

PRINT 'Subject added successfully'

END
```

Dodawanie studiów

Autor: Krzysztof Chmielewski

Procedura *AddStudy* dodaje rekord do tabeli *Studies*, służy do dodawania studiów.

```
CREATE PROCEDURE AddStudy
    @StudyName varchar(40),
    @FeePrice money,
    @StudyCoordinator int,
    @LimitOfStudents int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @StudyID int = (SELECT MAX(StudyID) + 1 FROM Studies)

    --handling study name exceptions
    IF @StudyName IS NULL
        BEGIN
            RAISERROR ('Study name cannot be null', 16, 1)

            RETURN
        END

    IF @StudyName IN (SELECT StudyName FROM Studies)
        BEGIN
            RAISERROR ('There already exists study with that
name', 16, 1)

            RETURN
        END

    --handling fee price exceptions
    IF @FeePrice IS NULL
```

```

        BEGIN
            RAISERROR ('Fee price cannot be null', 16, 1)

            RETURN

        END

    IF @FeePrice < 0

        BEGIN
            RAISERROR ('Invalid value: Fee price must be
non-negative value', 16, 1)

            RETURN

        END

    IF @FeePrice > 10000

        BEGIN
            RAISERROR ('Invalid value: Fee price cannot exceed
10000', 16, 1)

            RETURN

        END

    --handling study coordinator exceptions

    IF @StudyCoordinator IS NULL

        BEGIN
            RAISERROR ('Study coordinator cannot be null', 16, 1)

            RETURN

        END

    IF @StudyCoordinator NOT IN (

        SELECT e.EmployeeID FROM Employees AS e

            INNER JOIN EmployeeRoles AS er ON e.EmployeeID =
er.EmployeeID

            INNER JOIN Roles AS r ON er.RoleID = r.RoleID

```

```
        WHERE r.RoleName = 'Study Coordinator')

    BEGIN

        RAISERROR ('Given employee is not study coordinator',
16, 1)

        RETURN

    END

    --handling limit of students exceptions

    IF @LimitOfStudents <= 0

        BEGIN

            RAISERROR ('Invalid value: Limit of students must be
greater than 0', 16, 1)

            RETURN

        END

        --adding correct values to the table

        INSERT INTO Studies (StudyID, StudyName, FeePrice,
StudyCoordinatorID, LimitOfStudents)

            VALUES (@StudyID, @StudyName, @FeePrice, @StudyCoordinator,
@LimitOfStudents)

        PRINT 'Study added successfully'

    END
```

Zaliczenie praktyk studenta

Autor: Krzysztof Chmielewski

Procedura *PassStudentsInternship* służy do zaliczania praktyk konkretnemu studentowi.

```
CREATE PROCEDURE PassStudentsInternship
    @StudentID int,
    @InternshipID int
AS
BEGIN
    SET NOCOUNT ON

    --handling student ID exceptions
    IF @StudentID IS NULL
    BEGIN
        RAISERROR ('Student ID cannot be null', 16, 1)
        RETURN
    END

    IF @StudentID NOT IN (SELECT StudentID FROM Students)
    BEGIN
        RAISERROR ('There is no student with that ID', 16, 1)
        RETURN
    END

    --handling internship ID exceptions
    IF @InternshipID IS NULL
    BEGIN
        RAISERROR ('Internship ID cannot be null', 16, 1)
        RETURN
    END
```



```
END

IF @InternshipID NOT IN (SELECT InternshipID FROM Internships)

BEGIN

    RAISERROR ('There is no internship with that ID', 16,
1)

    RETURN

END

--checking if student-internship pair exists in internship
details

IF NOT EXISTS (SELECT 1

    FROM InternshipDetails

    WHERE StudentID = @StudentID AND InternshipID =
@InternshipID)

BEGIN

    RAISERROR ('There is no record of this student being
on this internship', 16, 1)

    RETURN

END

--passing students internship

UPDATE InternshipDetails

SET PASS = 1

WHERE StudentID = @StudentID AND InternshipID = @InternshipID

PRINT 'Updated record successfully'

END
```

Dodawanie rekordu praktyk

Autor: Krzysztof Chmielewski

Procedura *AddInternship* służy do dodawania rekordu praktyk do konkretnych studiów (dodaje oba cykle praktyk do danych studiów). Metoda służy jedynie do wpisywania rekordu praktyk w chwili dodania nowych studiów (każde studia powinny mieć dwa rekordy w tabeli Internships - jeden z cyklem 1 i drugie z cyklem 2).

```
CREATE PROCEDURE AddInternship
    @StudyID int
AS
BEGIN
    SET NOCOUNT ON

    --handling study id exceptions
    IF @StudyID IS NULL
    BEGIN
        RAISERROR ('Study ID cannot be null', 16, 1)

        RETURN
    END

    IF @StudyID NOT IN (SELECT StudentID FROM Students)
    BEGIN
        RAISERROR ('There is no study with that ID', 16, 1)

        RETURN
    END

    DECLARE @I int = 1;
    WHILE @I <= 2
    BEGIN
        DECLARE @InternshipID int = (SELECT MAX(InternshipID)
+ 1 FROM Internships)
```

```
        IF EXISTS (SELECT 1 FROM Internships WHERE StudyID =
@StudyID AND CycleNo = @I)
            BEGIN
                RAISERROR ('There already is internship with
that study id and cycle number', 16, 1)
                RETURN
            END

        INSERT INTO Internships (InternshipID, StudyID,
CycleNo)
            VALUES (@InternshipID, @StudyID, @I)

        SET @I = @I + 1
    END

    PRINT 'Internship added successfully'
END
```

Dodawanie studenta

Autor: Krzysztof Chmielewski

Procedura *AddStudent* służy do dodawania studenta na konkretne studia i semestr.

```
CREATE PROCEDURE AddStudent

    @UserID int,

    @StudyID int,

    @SemesterNo int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @StudentID int = (SELECT max(StudentID) + 1 FROM
Students)

    --handling user id exceptions
    IF @UserID IS NULL
        BEGIN
            RAISERROR ('User ID cannot be null', 16, 1)

            RETURN
        END

    IF @UserID NOT IN (SELECT UserID FROM Users)
        BEGIN
            RAISERROR ('There is no user with that ID', 16, 1)

            RETURN
        END

    --handling study id exceptions
    IF @StudyID IS NULL
```

```
BEGIN

    RAISERROR ('Study ID cannot be null', 16, 1)

    RETURN

END

IF @StudyID NOT IN (SELECT StudyID FROM Studies)

BEGIN

    RAISERROR ('There is no study with that ID', 16, 1)

    RETURN

END

--handling semester no exceptions

IF @SemesterNo IS NULL

BEGIN

    RAISERROR ('Semester number cannot be null', 16, 1)

    RETURN

END

IF @SemesterNo NOT IN (SELECT SemesterNo FROM Semesters)

BEGIN

    RAISERROR ('There is no semester with that number',
16, 1)

    RETURN

END

INSERT INTO Students (StudentID, UserID, StudyID, SemesterNo)
VALUES (@StudentID, @UserID, @StudyID, @SemesterNo)

PRINT 'Student added successfully'

END
```

Dodawanie krotki studia-semester

Autor: Krzysztof Chmielewski

Procedura *AddStudySemester* służy do dodawania rekordu do tabeli *StudySemesters*, jest wykorzystywana przez procedurę *FillStudySemesters*, która wywołuje tę procedurę dla każdego semestru studiów.

```
CREATE PROCEDURE AddStudySemester
    @StudyID int,
    @SemesterNo int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @StudySemesterID int = (SELECT MAX(StudySemesterID) +
1 FROM StudySemesters)

    --handling study id exceptions
    IF @StudyID IS NULL
        BEGIN
            RAISERROR ('Study ID cannot be null', 16, 1)

            RETURN
        END

    IF @StudyID NOT IN (SELECT StudyID FROM Studies)
        BEGIN
            RAISERROR ('There is no study with that ID', 16, 1)

            RETURN
        END

    --handling semester no exceptions
    IF @SemesterNo IS NULL
        BEGIN
```

```
        RAISERROR ('Semester number cannot be null', 16, 1)

        RETURN

    END

    IF @SemesterNo NOT IN (SELECT SemesterNo FROM Semesters)

    BEGIN

        RAISERROR ('There is no semester with that number',
16, 1)

        RETURN

    END

    IF EXISTS(SELECT 1 FROM StudySemesters WHERE StudyID =
@StudyID AND SemesterNo = @SemesterNo)

    BEGIN

        RAISERROR ('There is already study semester with that
values', 16, 1)

        RETURN

    END

    --adding correct values to the table

    INSERT INTO StudySemesters (StudySemesterID, StudyID,
SemesterNo)

    VALUES (@StudySemesterID, @StudyID, @SemesterNo)

    PRINT 'Study semester added successfully'

    END
```

Uzupełnianie semestrów dla studiów

Autor: Krzysztof Chmielewski

Procedura *FillStudySemesters* służy do dodawania rekordów do tabeli *StudySemesters*. Jeśli dodajemy nowe studia, to wywołując tę procedurę wpisujemy do tabeli *StudySemesters* wszystkie semestry na te studia.

```
CREATE PROCEDURE FillStudySemesters
    @StudyID int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @I int = 1

    WHILE @I <= 7
    BEGIN
        EXEC AddStudySemester @StudyID, @I

        SET @I = @I + 1
    END

    PRINT 'Study semester added successfully'

END
```


Dodanie nowego adresu

Autor: Szymon Migas

Procedura służąca do dodawania nowego adresu

```
CREATE PROCEDURE AddAddress
    @CityID int,
    @Street nvarchar(30),
    @ZipCode nvarchar(20)
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @AddressID int = (SELECT MAX(AddressID) + 1 FROM Addresses)

    IF @CityID NOT IN (SELECT CityID FROM Cities)
    BEGIN
        RAISERROR ('City does not exist', 16, 1)
        RETURN
    END

    IF @Street IS NULL
    BEGIN
        RAISERROR ('Street cannot be null', 16, 1)
        RETURN
    END

    IF @ZipCode IS NULL
    BEGIN
        RAISERROR ('Zip Code cannot be null', 16, 1)
        RETURN
    END

    IF @ZipCode IN (SELECT ZipCode FROM Addresses WHERE CityID =
@CityID)
    BEGIN
        RAISERROR ('Address already exists in this city', 16, 1)
        RETURN
    END
```

```

        END

        INSERT INTO Addresses (AddressID, CityID, Street, ZipCode)
        VALUES (@AddressID, @CityID, @Street, @ZipCode)

        PRINT 'Address added successfully'

    END

```

Dodawanie nowego miasta

Autor: Szymon Migas

Procedura służąca do wstawiania nowego miasta

```

CREATE PROCEDURE AddCity
    @CityName nvarchar(50),
    @CountryID int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @CityID int = (SELECT MAX(CityID) + 1 FROM Cities)

    IF @CityName IS NULL
        BEGIN
            RAISERROR ('City Name cannot be null', 16, 1)

            RETURN

        END

    IF @CountryID NOT IN (SELECT CountryID FROM Countries)
        BEGIN
            RAISERROR ('Country does not exist', 16, 1)

            RETURN

        END

    IF @CityName IN (SELECT CityName FROM Cities WHERE CountryID =
@CountryID)
        BEGIN
            RAISERROR ('City already exists in this country', 16, 1)

```

```

        RETURN

    END

    INSERT INTO Cities (CityID, CityName, CountryID)
    VALUES (@CityID, @CityName, @CountryID)

    PRINT 'City added successfully'

END

```

Dodawanie nowej sali

Autor: Szymon Migas

Procedura służąca do wstawiania nowej sali

```

CREATE PROCEDURE AddClassroom
    @AddressID int,
    @RoomNumber int,
    @Capacity int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @ClassroomID int = (SELECT MAX(ClassroomID) + 1 FROM
Classrooms)

    IF @AddressID NOT IN (SELECT AddressID FROM Addresses)
    BEGIN
        RAISERROR ('Address does not exist', 16, 1)

        RETURN

    END

    IF @RoomNumber IS NULL OR @RoomNumber < 0
    BEGIN
        RAISERROR ('Room Number cannot be null or less than 0', 16,
1)

        RETURN

    END

    IF @Capacity IS NULL OR @Capacity < 0
    BEGIN

```

```

        RAISERROR ('Capacity cannot be null or less than 0', 16, 1)
        RETURN
    END

    IF @RoomNumber IN (SELECT RoomNo FROM Classrooms WHERE AddressID =
@AddressID)
        BEGIN
            RAISERROR ('Classroom already exists at this address', 16,
1)

            RETURN
        END

    INSERT INTO Classrooms (ClassroomID, AddressID, RoomNo, Capacity)
    VALUES (@ClassroomID, @AddressID, @RoomNumber, @Capacity)

    PRINT 'Classroom added successfully'

END

```

Dodawanie odroczenia płatności

Autor: Szymon Migas

Procedura służąca do wystawiania przez dyrektora odroczenia płatności za daną usługę

```

CREATE PROCEDURE AddHeadteacherPostponement
    @UserID int,
    @ServiceTypeID int,
    @ServiceID int,
    @DueDate datetime
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @PostponementID int = (select max(PostponementID) + 1 from
dbo.HeadTeacherPaymentPostponements)

    IF @UserID NOT IN (SELECT UserID FROM dbo.Users)
        BEGIN

```

```

        RAISERROR ('User does not exist', 16, 1)

        RETURN

    END

    IF @ServiceTypeID NOT IN (SELECT ServiceTypeID FROM dbo.ServiceTypes)
    BEGIN
        RAISERROR ('Service Type does not exist', 16, 1)

        RETURN

    END

    IF @ServiceID NOT IN (SELECT ServiceID FROM dbo.AllServices where
ServiceType = @ServiceTypeID)
    BEGIN
        RAISERROR ('Service does not exist', 16, 1)

        RETURN

    END

    IF @DueDate IS NULL
    BEGIN
        RAISERROR ('Due Date cannot be null', 16, 1)

        RETURN

    END

    IF (SELECT COUNT(*)
        FROM OrderDetails OD
        INNER JOIN Orders O on OD.OrderID = O.OrderID
        WHERE UserID = @UserID AND ServiceTypeID = @ServiceTypeID AND
ServiceID = @ServiceID) = 0
    BEGIN
        RAISERROR ('User did not order this service', 16, 1)

        RETURN

    END

    INSERT INTO dbo.HeadTeacherPaymentPostponements (PostponementID, UserID,
ServiceTypeID, ServiceID, DueDate)

    VALUES (@PostponementID, @UserID, @ServiceTypeID, @ServiceID, @DueDate)

    PRINT 'HeadteacherPostponement added successfully'

END

```

Dodawanie nowego zamówienia

Autor: Szymon Migas

Procedura służąca do wstawiania informacji o zamówieniu złożonym przez użytkownika

```
CREATE PROCEDURE CreateOrder
    @UserID int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @Date datetime = GETDATE()
    DECLARE @OrderID int = (select max(OrderID) + 1 from Orders)

    IF @UserID NOT IN (SELECT UserID FROM dbo.Users)
    BEGIN
        RAISERROR ('User does not exist', 16, 1)
        RETURN
    END

    INSERT INTO dbo.Orders (OrderID, UserID, OrderDate)
    VALUES (@OrderID, @UserID, @Date)

    PRINT 'Order created successfully'
END
```

Dodawanie dodatkowych informacji o zamówieniu

Autor: Szymon Migas

Procedura służąca do wstawiania detali zamówienia danego użytkownika

```
CREATE PROCEDURE AddOrderDetails
    @OrderID int,
    @ServiceTypeID int,
    @ServiceID int
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @OrderDetailID int = (select max(OrderDetailID) + 1
from OrderDetails)

    IF @OrderID NOT IN (SELECT OrderID FROM dbo.Orders)
    BEGIN
        RAISERROR ('Order does not exist', 16, 1)

        RETURN
    END

    IF @ServiceTypeID NOT IN (SELECT ServiceTypeID FROM
dbo.ServiceTypes)
    BEGIN
        RAISERROR ('Service Type does not exist', 16, 1)

        RETURN
    END

    IF @ServiceID NOT IN (SELECT ServiceID FROM dbo.AllServices
where ServiceType = @ServiceTypeID)
    BEGIN
        RAISERROR ('Service does not exist', 16, 1)

        RETURN
    END

    IF @ServiceTypeID IN (SELECT ServiceTypeID from ServiceTypes
where ServiceTypeName = 'Course')
    BEGIN
```

```

        IF NOT EXISTS (SELECT 1 FROM FutureCourses where
@ServiceID = CourseID)
        BEGIN
            RAISERROR ('Course has already started', 16, 1)
        end
        DECLARE @limit int = (SELECT limit FROM CourseLimits
WHERE CourseID = @ServiceID)
        IF @limit IS NOT NULL AND @limit <= (SELECT TOP 1
Users FROM NumberOfUsersAuthorizedForCourseMeeting N
        INNER JOIN CoursesMeetings CM ON CM.MeetingID =
N.MeetingID
        INNER JOIN Courses C ON C.CourseID = CM.CourseID
        WHERE C.CourseID = @ServiceID)
        BEGIN
            RAISERROR ('No more available spots for course',
16, 1)
        end
    end
    IF @ServiceTypeID IN (SELECT ServiceTypeID from ServiceTypes
where ServiceTypeName = 'Studies Meetup')
    BEGIN
        IF NOT EXISTS (SELECT 1 FROM StudyMeetups where
StudyMeetupID = @ServiceID and StartDate > GETDATE())
        BEGIN
            RAISERROR ('Service in the past', 16, 1)
            RETURN
        end
    end
    IF @ServiceTypeID IN (SELECT ServiceTypeID from ServiceTypes
where ServiceTypeName = 'Studies')
    BEGIN
        DECLARE @StudyLimit int = (SELECT LimitOfStudents FROM
Studies WHERE @ServiceID = StudyID)
        DECLARE @registered int = (SELECT COUNT(*) FROM
Students WHERE StudyID = @ServiceID)

```



```

        IF @registered >= @Studylimit
        BEGIN
            RAISERROR ('No more available spots for studies',
16,1)
        end
        IF (SELECT MIN(StartDate) FROM StudyMeetups SM
            inner join
StudySemesters SS on SM.StudySemesterID = SS.StudySemesterID
            inner join Studies
S on SS.StudyID = S.StudyID
            where S.StudyID = @ServiceID) < GETDATE()
        BEGIN
            RAISERROR ('Study from past', 16,1)
        end

    end

    IF @ServiceTypeID IN (SELECT ServiceTypeID from ServiceTypes
where ServiceTypeName = 'Webinar')
    BEGIN
        IF NOT EXISTS(SELECT 1 FROM Webinars WHERE WebinarID =
@ServiceID and Date > GETDATE())
        BEGIN
            RAISERROR ('Service in the past', 16, 1)
            RETURN
        end
    end

    INSERT INTO dbo.OrderDetails (OrderDetailID, OrderID,
ServiceTypeID, ServiceID)
    VALUES (@OrderDetailID, @OrderID, @ServiceTypeID, @ServiceID)
    PRINT 'Order details added successfully'
END

```

Dodawanie nowego webinaru

Autor: Szymon Migas

Procedura służąca do wstawiania nowego webinaru

```
CREATE PROCEDURE AddWebinar
    @WebinarName varchar(80),
    @RecordingLink varchar(100),
    @Date datetime,
    @Price money,
    @WebinarPresenterID int,
    @TranslatorLanguageID int = null
AS
BEGIN
    DECLARE @WebinarID int = (select max(WebinarID) + 1 from Webinars)
    SET NOCOUNT ON
    IF @Date < GETDATE()
    BEGIN
        RAISERROR ('Date cannot be in the past', 16, 1)
        RETURN
    END
    IF @WebinarPresenterID NOT IN (SELECT UserID FROM dbo.Users)
    BEGIN
        RAISERROR ('Webinar Presenter does not exist', 16, 1)
        RETURN
    END
    IF @WebinarName IN (SELECT WebinarName FROM dbo.Webinars)
    BEGIN
        RAISERROR ('Webinar already exists', 16, 1)
        RETURN
    END
    IF @TranslatorLanguageID IS NOT NULL AND @TranslatorLanguageID NOT
    IN (SELECT TranslatorLanguageID FROM Translators)
    BEGIN
```

```

        RAISERROR ('Translator does not exist', 16, 1)
    RETURN

END

IF @Price < 0 OR @Price IS NULL OR @Price > 999999
BEGIN
    RAISERROR ('Price must be between 0 and 999999', 16, 1)
    RETURN
END

IF @RecordingLink IN (SELECT RecordingLink FROM dbo.Webinars)
BEGIN
    RAISERROR ('Recording Link already exists.', 16, 1)
    RETURN
END

IF NOT (@RecordingLink LIKE 'https://%' OR @RecordingLink LIKE
'http://%')
BEGIN
    RAISERROR ('Invalid link, must be either http://* or
https://*', 16, 1)
    RETURN
END

INSERT INTO dbo.Webinars (WebinarID, WebinarName, RecordingLink,
Date, Price, WebinarPresenterID, TranslatorLanguageID)
VALUES (@WebinarID,@WebinarName, @RecordingLink,
@Date,@Price,@WebinarPresenterID, @TranslatorLanguageID)

PRINT 'Webinar added successfully'

END

```

Dodawanie płatności

Autor: Szymon Migas

Procedura służąca do wstawiania nowej płatności, wpływającej za dane zamówienie

```
CREATE PROCEDURE CreatePayment
    @OrderDetailID int,
    @PaymentAmount money
AS
BEGIN
    SET NOCOUNT ON

    DECLARE @PaymentID int = (select max(PaymentID) + 1 from Payments)
    DECLARE @Date datetime = GETDATE()

    IF @OrderDetailID NOT IN (SELECT OrderDetailID FROM
dbo.OrderDetails)
        BEGIN
            RAISERROR ('Order Detail does not exist', 16, 1)
            RETURN
        END

    IF @PaymentAmount < 0 OR @PaymentAmount IS NULL OR @PaymentAmount >
999999
        BEGIN
            RAISERROR ('Payment Amount must be between 0 and 999999',
16, 1)
            RETURN
        END

    INSERT INTO dbo.Payments (PaymentID, OrderDetailID, PayDate, Amount)
    VALUES (@PaymentID, @OrderDetailID, @Date, @PaymentAmount)

    PRINT 'Payment created successfully'

END
```

Funkcje

Procentowa obecność danego użytkownika na spotkaniach w ramach danego kursu

Autor: Wiktor Sędzimir

Funkcja *GetCourseUserPresencePercentage* zwraca procentową wartość obecności danego użytkownika na spotkaniach w ramach danego kursu.

```
CREATE FUNCTION GetCourseUserPresencePercentage (@CourseID
INT, @UserID INT)
RETURNS REAL
AS
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM CourseMeetingPresence
        INNER JOIN CoursesMeetings
            ON CoursesMeetings.MeetingID =
CourseMeetingPresence.MeetingID
        WHERE CourseID = @CourseID AND UserID = @UserID
    )
    BEGIN
        RETURN 0.0;
    END;

    DECLARE @PresenceCount INT, @MeetingsCount INT;

    SELECT
```

```

        @PresenceCount = COUNT(*)

FROM CourseMeetingPresence

INNER JOIN CoursesMeetings

    ON CoursesMeetings.MeetingID =
CourseMeetingPresence.MeetingID

WHERE CourseID = @CourseID AND UserID = @UserID AND
Presence = 'Present';

SELECT

    @MeetingsCount = COUNT(*)

FROM CourseMeetingPresence

INNER JOIN CoursesMeetings

    ON CoursesMeetings.MeetingID =
CourseMeetingPresence.MeetingID

WHERE CourseID = @CourseID AND UserID = @UserID;

RETURN CAST(@PresenceCount AS REAL) * 100 /
@MeetingsCount;

END;

```

Procentowa obecność danego studenta na spotkaniach w ramach danego przedmiotu na studiach

Autor: Wiktor Sędzimir

Funkcja *GetSubjectStudentPresencePercentage* zwraca procentową wartość obecności danego studenta na spotkaniach w ramach danego przedmiotu na studiach.

```

CREATE FUNCTION

GetSubjectStudentPresencePercentage (@SubjectID INT,

@StudentID INT)

RETURNS REAL

```

```
AS
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM StudyMeetingPresence
        INNER JOIN StudiesMeetings
            ON StudiesMeetings.MeetingID =
StudyMeetingPresence.MeetingID
        WHERE SubjectID = @SubjectID AND StudentID =
@StudentID
    )
    BEGIN
        RETURN 0.0;
    END;

    DECLARE @PresenceCount INT, @MeetingsCount INT;

    SELECT
        @PresenceCount = COUNT(*)
    FROM StudyMeetingPresence
    INNER JOIN StudiesMeetings
        ON StudiesMeetings.MeetingID =
StudyMeetingPresence.MeetingID
    WHERE SubjectID = @SubjectID AND StudentID = @StudentID
    AND Presence = 'Present';
```

```

SELECT

    @MeetingsCount = COUNT(*)

FROM StudyMeetingPresence

INNER JOIN StudiesMeetings

    ON StudiesMeetings.MeetingID =
StudyMeetingPresence.MeetingID

WHERE SubjectID = @SubjectID AND StudentID = @StudentID

RETURN CAST(@PresenceCount AS REAL) * 100 /
@MeetingsCount;

END;

```

Przyszłe zajęcia dla danego użytkownika

Autor: Wiktor Sędzimir

Funkcja *GetUserSchedule* zwraca szczegóły na temat wszystkich przyszłych aktywności danego użytkownika.

```

CREATE FUNCTION GetUserSchedule(@UserID INT)

RETURNS TABLE

AS

RETURN (

    SELECT

        'Course Meeting' AS ServiceTypeName,

        CourseID AS ServiceID,

        CourseName AS ServiceName,

        CoursesMeetingsOrganization.MeetingID,

        Location,

```



```

        MeetingStartDate,
        MeetingEndDate

FROM CoursesMeetingsOrganization

INNER JOIN UsersAuthorizedForCourseMeeting
    ON CoursesMeetingsOrganization.MeetingID =
UsersAuthorizedForCourseMeeting.MeetingID

WHERE UserID = @UserID AND MeetingStartDate > GETDATE()

UNION

SELECT
    'Study Meeting' AS ServiceTypeName,
    StudyID AS ServiceID,
    SubjectName + ' (study: ' + StudyName + ')' AS
ServiceName,
    StudiesMeetingsOrganization.MeetingID,
    Location,
    MeetingStartDate,
    MeetingEndDate

FROM StudiesMeetingsOrganization

INNER JOIN UsersAuthorizedForStudyMeeting
    ON StudiesMeetingsOrganization.MeetingID =
UsersAuthorizedForStudyMeeting.MeetingID

WHERE UserID = @UserID AND MeetingStartDate > GETDATE()

UNION

SELECT
    'Webinar' AS ServiceTypeName,
    Webinars.WebinarID AS ServiceID,

```

```
WebinarName AS ServiceName,  
Webinars.WebinarID AS MeetingID,  
RecordingLink AS Location,  
Date AS MeetingStartDate,  
DATEADD(HOUR, 2, Date) AS MeetingEndDate  
FROM Webinars  
INNER JOIN UsersAuthorizedForWebinar  
ON Webinars.WebinarID =  
UsersAuthorizedForWebinar.WebinarID  
WHERE UserID = @UserID AND Date > GETDATE()  
);
```

Sprawdzenie stanu zaliczenia studenta

Autor: Krzysztof Chmielewski

Funkcja *DidStudentPass* przyjmuje jako argument ID studenta i zwraca informację w postaci BIT, o tym czy dany student zaliczył studia czy nie.

```
CREATE FUNCTION DidStudentPass (@StudentID int)
RETURNS BIT
AS
BEGIN
    DECLARE @AllStudentMeetings int = (SELECT COUNT(*) FROM
StudyMeetingPresence WHERE StudentID = @StudentID)

    DECLARE @MeetingWhereStudentWasPresent int = (SELECT COUNT(*) FROM
StudyMeetingPresence
                                                    WHERE StudentID =
@StudentID AND Presence = 'Present')

    DECLARE @NumberOfPassedInternships int = (SELECT COUNT(*) FROM
InternshipDetails
                                                    WHERE StudentID = @StudentID
AND Pass = 1)

    DECLARE @PassedThreshold decimal = (100 *
@MeetingWhereStudentWasPresent) / CAST(@AllStudentMeetings AS DECIMAL)

    DECLARE @StudentGrade decimal = (SELECT GradeValue FROM FinalExams
INNER JOIN dbo.Grades G ON G.GradeID =
FinalExams.GradeID
                                WHERE FinalExams.StudentID =
@StudentID)

    IF @PassedThreshold >= CAST(80 AS DECIMAL) AND
@NumberOfPassedInternships = 2 AND @StudentGrade >= 3.0
    BEGIN
        RETURN 1
    END
    RETURN 0
END
```

Uzyskiwanie harmonogramu zajęć danego ćwiczeniowca

Autor: Krzysztof Chmielewski

Funkcja *GetLecturerSchedule* przyjmuje jako argument ID ćwiczeniowca i zwraca tabelę z wszelkimi informacjami odnośnie zajęć danego ćwiczeniowca.

```
CREATE FUNCTION GetLecturerSchedule(@LecturerID int)
RETURNS TABLE
AS
RETURN (
    SELECT s.StudyName, sub.SubjectName, sm.StudyMeetupID,
    m.MeetingID,
        m.StartDate, m.EndDate, m.LimitOfMeetingParticipants,
    l.LanguageName,
        e.Firstname + ' ' + e.Lastname AS Translator
    FROM StudiesMeetings AS m
    INNER JOIN Subjects AS sub ON m.SubjectID = sub.SubjectID
    INNER JOIN StudyMeetups AS sm ON m.StudyMeetupID =
sm.StudyMeetupID
    INNER JOIN StudySemesters AS sem ON sem.StudySemesterID =
sm.StudySemesterID
    INNER JOIN Studies AS s ON sem.StudyID = s.StudyID
    INNER JOIN Translators AS t ON m.TranslatorLanguageID =
t.TranslatorLanguageID
    INNER JOIN Languages AS l ON t.LanguageID = l.LanguageID
    INNER JOIN Employees AS e ON t.EmployeeID = e.EmployeeID
    WHERE m.LecturerID = @LecturerID
)
```

Uzyskiwanie adresu studenta

Autor: Krzysztof Chmielewski

Funkcja *GetStudentsAddress* przyjmuje jako argument ID studenta i zwraca tabelę z informacjami o adresie danego studenta.

```
CREATE FUNCTION GetStudentsAddress (@StudentID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT a.* FROM Addresses AS a
        INNER JOIN Users AS u ON u.AddressID = a.AddressID
        INNER JOIN Students AS s ON u.UserID = s.UserID
        WHERE s.StudentID = @StudentID
);
```

Uzyskiwanie harmonogramu konkretnych studiów

Autor: Krzysztof Chmielewski

Funkcja *GetStudyTimetable* przyjmuje jako argument ID studiów i zwraca tabelę, którą jest harmonogram danych studiów, czyli daty rozpoczęcia i zakończenia wszystkich zjazdów na danym semestrze, oraz daty rozpoczęcia i zakończenia wszystkich spotkań w ramach tego zjazdu oraz informacja kto prowadzi dane spotkanie.

```
CREATE FUNCTION GetStudyTimetable (@StudyID int)
RETURNS TABLE
AS
RETURN (
    SELECT sem.StudyID, sem.SemesterNo, sm.StudyMeetupID, sm.StartDate
    AS 'MEETUP START DATE',
        sm.EndDate AS 'MEETUP END DATE', m.MeetingID, m.StartDate
    AS 'MEETING START DATE',
        m.EndDate AS 'MEETING END DATE', sub.SubjectName,
    e.Firstname + ' ' + e.Lastname AS LECTURER
    FROM StudySemesters AS sem
        INNER JOIN StudiesSchedule AS ss ON sem.StudySemesterID =
ss.StudySemesterID
        INNER JOIN Subjects AS sub ON ss.SubjectID = sub.SubjectID
        INNER JOIN StudyMeetups AS sm ON sem.StudySemesterID =
sm.StudySemesterID
        INNER JOIN StudiesMeetings AS m on sm.StudyMeetupID =
m.StudyMeetupID and sub.SubjectID = m.SubjectID
        INNER JOIN Employees AS e ON m.LecturerID = e.EmployeeID
    WHERE sem.StudyID = @StudyID
)
```

Uzyskiwanie rocznego przychodu

Autor: Szymon Migas

Funkcja *GetAnnualIncome* służy uzyskiwaniu dochodu generowanego przez uczelnię w danym roku

```
CREATE FUNCTION GetAnnualIncome
    (@Year int)
RETURNS decimal(10, 2)
AS
BEGIN
    RETURN CONVERT(decimal(10, 2), (SELECT sum(Amount)
                                     FROM Payments
                                     INNER JOIN dbo.OrderDetails OD on OD.OrderDetailID =
Payments.OrderDetailID
                                     WHERE YEAR(PayDate) = @Year))
END
```

Uzyskiwanie przychodu z kursów

Autor: Szymon Migas

Funkcja *GetCourseIncome* służy uzyskiwaniu przychodu generowanego przez dany kurs, może się to przydać w sytuacji sprawdzenia opłacalności prowadzenia takiego kursu

```
CREATE FUNCTION GetCourseIncome
    (@CourseID int)
RETURNS decimal(10, 2)
BEGIN
    DECLARE @CourseServiceTypeID int = (SELECT ServiceTypeID FROM
ServiceTypes WHERE ServiceTypeName = 'Course')

    IF NOT EXISTS(SELECT 1 FROM Courses WHERE CourseID = @CourseID)
    BEGIN
        RETURN 0.0
    END

    RETURN CONVERT(decimal(10, 2), (
        SELECT sum(Amount)
```

```

        FROM Payments
        INNER JOIN dbo.OrderDetails OD on OD.OrderDetailID =
Payments.OrderDetailID
        WHERE ServiceTypeID = @CourseServiceTypeID and ServiceID = @CourseID
    ))
end

```

Uzyskiwanie przychodu z webinarów

Autor: Szymon Migas

Funkcja *GetWebinarIncome* służy uzyskiwaniu przychodu generowanego przez dany webinar, może się to przydać w sytuacji sprawdzenia opłacalności prowadzenia takiego webinaru

```

CREATE FUNCTION GetWebinarIncome
    (@WebinarID int)
RETURNS decimal(10, 2)
AS
BEGIN
    DECLARE @WebinarServiceTypeID int = (SELECT ServiceTypeID FROM
ServiceTypes WHERE ServiceTypeName = 'Webinar')

    IF NOT EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID)
    BEGIN
        RETURN 0.0
    END

    RETURN CONVERT(decimal(10, 2),
        (SELECT sum(Amount)
        FROM Payments
        INNER JOIN dbo.OrderDetails OD on OD.OrderDetailID =
Payments.OrderDetailID
        WHERE ServiceTypeID = @WebinarServiceTypeID and ServiceID =
@WebinarID))
END

```

Uzyskiwanie przychodu ze studiów

Autor: Szymon Migas

Funkcja *GetStudyIncome* służy uzyskiwaniu przychodu generowanego przez dane studia (zjazdy + wpisowe)

```
CREATE FUNCTION GetStudyIncome
    (@StudyID int)
RETURNS decimal(10,2)
AS
BEGIN
    DECLARE @ServiceTypeID int = (SELECT ServiceTypeID FROM ServiceTypes
    WHERE ServiceTypeName = 'Studies')

    DECLARE @MeetupServiceID int = (SELECT ServiceTypeID FROM
    ServiceTypes WHERE ServiceTypeName = 'Studies Meetup')

    IF NOT EXISTS(SELECT 1 FROM Studies WHERE StudyID = @StudyID)
        BEGIN
            RETURN 0.0
        END

    RETURN CONVERT(decimal(10, 2), (SELECT sum(Amount)
        FROM Payments
            INNER JOIN dbo.OrderDetails OD on
    OD.OrderDetailID = Payments.OrderDetailID
        WHERE ServiceTypeID = @ServiceTypeID and ServiceID =
    @StudyID

        OR (ServiceTypeID = @MeetupServiceID
            AND
            (ServiceID IN
                (SELECT StudyMeetupID
                FROM StudyMeetups
                WHERE StudySemesterID IN
                    (SELECT StudySemesterID
                    FROM StudySemesters
                    WHERE StudyID = @StudyID))
            )))
END
```

Uzyskiwanie okresowego dochodu

Autor: Szymon Migas

Funkcja *GetRangeIncome* służy uzyskiwaniu przychodu uczelni z danego zakresu dat

```
CREATE FUNCTION GetRangeIncome
    (@StartDate date, @EndDate date)
RETURNS decimal(10, 2)
AS
BEGIN
    IF @StartDate > @EndDate
        BEGIN
            RETURN 0.0
        END
    RETURN CONVERT(decimal(10, 2), (SELECT sum(Amount)
                                     FROM Payments
                                     INNER JOIN dbo.OrderDetails OD on OD.OrderDetailID =
Payments.OrderDetailID
                                     WHERE PayDate BETWEEN @StartDate AND @EndDate))
END
```

Triggery

Automatyczne dodawanie studenta po zamówieniu studiów

Autor: Wiktor Sędzimir

Trigger *AddStudentOnStudyOrderPayment* automatycznie dodaje użytkownika, który zamówił studia oraz wpłacił całą zaliczkę jako studenta do tabeli *Students*.

```
CREATE TRIGGER AddStudentOnStudyOrderPayment
ON Payments
AFTER INSERT
AS
BEGIN
    DECLARE @NewPaymentID INT = (SELECT MAX(PaymentID) FROM
Payments);

    -- check if user paid for studies
    IF NOT EXISTS (
        SELECT 1
        FROM OrderDetails
        INNER JOIN ServiceTypes
            ON OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID
        INNER JOIN Payments
            ON OrderDetails.OrderDetailID =
Payments.OrderDetailID
        WHERE PaymentID = @NewPaymentID AND ServiceTypeName =
'Studies'
    )
```

```
BEGIN

    RETURN;

END

DECLARE @UserID INT;

DECLARE @StudyID INT;

DECLARE @OrderID INT;

SELECT

    @UserID = UserID,

    @StudyID = ServiceID,

    @OrderID = Orders.OrderID

FROM OrderDetails

INNER JOIN Orders

    ON OrderDetails.OrderID = Orders.OrderID

INNER JOIN Payments

    ON OrderDetails.OrderDetailID = Payments.OrderDetailID

WHERE PaymentID = @NewPaymentID;

DECLARE @TotalAmountPaid MONEY;

SELECT

    @TotalAmountPaid = SUM(Amount)

FROM Payments

INNER JOIN OrderDetails
```

```
        ON Payments.OrderDetailID = OrderDetails.OrderDetailID

    INNER JOIN ServiceTypes

        ON OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID

    INNER JOIN Orders

        ON OrderDetails.OrderID = Orders.OrderID

WHERE OrderDetails.OrderID = @OrderID

    AND

    ServiceTypeName = 'Studies'

    AND

    ServiceID = @StudyID

    AND

    UserID = @UserID;

    IF @TotalAmountPaid < (SELECT FeePrice FROM Studies WHERE
StudyID = @StudyID)

    BEGIN

        RETURN;

    END

EXEC AddStudent

    @UserID = @UserID,

    @StudyID = @StudyID,

    @SemesterNo = 1

END;
```


GO

Automatyczne dodawanie szczegółów zamówienia na zjazdy po dodaniu nowego zjazdu

Autor: Wiktor Sędzimir

Trigger *AddStudyMeetupOrderDetailOnNewMeetup* automatycznie dodaje szczegóły zamówienia, dla każdego użytkownika, który zamówił dane studia jest na studiach oraz semestrze, którego dotyczy nowy zjazd.

```
CREATE TRIGGER AddStudyMeetupOrderDetailOnNewMeetup
ON StudyMeetups
AFTER INSERT
AS
BEGIN
    -- GET ServiceID of 'Study Meetup' service
    DECLARE @MeetupServiceID INT = (
        SELECT
            ServiceTypeID
        FROM ServiceTypes
        WHERE ServiceTypeName = 'Studies Meetup'
    );

    DECLARE @NewMeetupID INT, @StudySemesterID INT;

    -- GET newest meetup information
    SELECT
        @NewMeetupID = StudyMeetupID,
        @StudySemesterID = StudySemesterID
    FROM StudyMeetups
```

```
WHERE StudyMeetupID = (
    SELECT MAX(sm1.StudyMeetupID) FROM StudyMeetups AS sm1
);

DECLARE @StudyID INT;

-- GET StudyID for given meetup
SELECT
    @StudyID = StudyID
FROM StudySemesters
WHERE StudySemesterID = @StudySemesterID;

DECLARE @OrderID INT;

-- FIND all orders for studies; consider only orders made
by students
-- WHO ARE currently enrolled in the studies and in the
semester related
-- TO the new meetup
DECLARE OrderCursor CURSOR FOR
SELECT
    Orders.OrderID
FROM Orders
INNER JOIN OrderDetails
    ON Orders.OrderID = OrderDetails.OrderID
```

```
INNER JOIN ServiceTypes
    ON OrderDetails.ServiceTypeID =
ServiceTypes.ServiceTypeID

INNER JOIN Students
    ON Students.UserID = Orders.UserID

INNER JOIN StudySemesters
    ON Students.SemesterNo = StudySemesters.SemesterNo AND
Students.StudyID = StudySemesters.StudyID

WHERE ServiceTypeName = 'Studies'

    AND ServiceID = @StudyID

    AND StudySemesterID = @StudySemesterID;

-- ADD new order details for given study meetup

OPEN OrderCursor;

FETCH NEXT FROM OrderCursor INTO @OrderID;

WHILE @@FETCH_STATUS = 0
BEGIN

    EXEC AddOrderDetails

        @OrderID = @OrderID,

        @ServiceTypeID = @MeetupServiceID,

        @ServiceID = @NewMeetupID;

    FETCH NEXT FROM OrderCursor INTO @OrderID;
```



```
END

CLOSE OrderCursor;

DEALLOCATE OrderCursor;

END;
```

Automatyczne dodawanie semestrów po dodaniu studiów

Autor: Krzysztof Chmielewski

Trigger *FillStudySemestersOnAddStudy* automatycznie dodaje wszystkie semestry dla tych studiów do tabeli *StudySemesters*.

```
CREATE TRIGGER FillStudySemestersOnAddStudy
ON Studies
AFTER INSERT AS
BEGIN
    DECLARE @NewStudyID int = (SELECT MAX(StudyID) FROM Studies)
    EXECUTE FillStudySemesters
        @StudyID = @NewStudyID;
END
```

Automatyczne dodawanie rekordów dla praktyk po dodaniu studiów

Autor: Krzysztof Chmielewski

Trigger *AddInternshipsOnAddStudy* automatycznie dodaje rekordy obu cykli praktyk do tabeli *Internships* dla tych studiów.

```
CREATE TRIGGER AddInternshipsOnAddStudy
ON Studies
AFTER INSERT AS
BEGIN
    DECLARE @NewStudyID int = (SELECT MAX(StudyID) FROM Studies)
    EXECUTE AddInternship
        @StudyID = @NewStudyID;
END
```

Automatyczne dodawanie zjazdów do detali zamówienia, podczas zamawiania studiów

Autor: Szymon Migas

Trigger *AddStudyMeetupOnStudyOrder* służy do dodawania do zamówienia wszystkich zjazdów, które znajdują się na zamówionych przez użytkownika studiach, aby mógł je spłacać.

```
CREATE TRIGGER AddStudyMeetupOnStudyOrder
    ON OrderDetails
    AFTER INSERT
    AS
BEGIN
    IF (SELECT ServiceTypeID FROM inserted) != (SELECT ServiceTypeID
from ServiceTypes where ServiceTypeName = 'Studies')
        BEGIN
            RETURN
        end
    DECLARE @MeetupID int
    DECLARE @OrderID int = (SELECT OrderID FROM inserted)
    DECLARE @ServiceID int = (SELECT ServiceID FROM inserted)
    DECLARE @MeetupTypeID int = (SELECT ServiceTypeID from
ServiceTypes where ServiceTypeName = 'Studies meetup')
    DECLARE MeetupCursor CURSOR FOR
        SELECT StudyMeetupID FROM StudyMeetups SM
                                INNER JOIN StudySemesters SS ON
SM.StudySemesterID = SS.StudySemesterID
                                INNER JOIN Studies S ON
SS.StudyID = S.StudyID
        where S.StudyID = @ServiceID

    -- add new order details for given study meetup
    OPEN MeetupCursor;

    FETCH NEXT FROM MeetupCursor INTO @MeetupID;
```

```
WHILE @@FETCH_STATUS = 0
    BEGIN
        EXEC AddOrderDetails
            @OrderID = @OrderID,
            @ServiceTypeID = @MeetupTypeID,
            @ServiceID = @MeetupID
        FETCH NEXT FROM MeetupCursor INTO @OrderID;
    END
CLOSE MeetupCursor
DEALLOCATE MeetupCursor
END;
```

Role

Stworzenie roli dla Studenta, Wykładowcy i Koordynatora studiów

Autor: Krzysztof Chmielewski

```
CREATE ROLE StudyCoordinator;

GRANT EXECUTE ON AddStudy TO StudyCoordinator;

GRANT EXECUTE ON AddFinalExam TO StudyCoordinator;

GRANT EXECUTE ON AddInternship TO StudyCoordinator;

GRANT EXECUTE ON AddInternshipDetail TO StudyCoordinator;

GRANT EXECUTE ON AddStudiesSchedule TO StudyCoordinator;

GRANT EXECUTE ON AddStudyMeetup TO StudyCoordinator;

GRANT EXECUTE ON AddSubject TO StudyCoordinator;

GRANT EXECUTE ON DidStudentPass TO StudyCoordinator;

GRANT EXECUTE ON PassStudentsInternship TO StudyCoordinator;

GRANT SELECT, INSERT, UPDATE, DELETE ON Studies TO StudyCoordinator;

GRANT SELECT, INSERT, UPDATE, DELETE ON Internships TO
StudyCoordinator;

GRANT SELECT, INSERT, UPDATE, DELETE ON InternshipDetails TO
StudyCoordinator;

CREATE ROLE Lecturer;

GRANT EXECUTE ON AddOnlineStudyMeeting TO Lecturer;

GRANT EXECUTE ON AddStationaryStudyMeeting TO Lecturer;

GRANT EXECUTE ON AddStudyAttendance TO Lecturer;

GRANT SELECT ON NumberOfUsersAuthorizedForStudyMeeting TO Lecturer;

GRANT SELECT ON StudyMeetingPresence TO Lecturer;

GRANT SELECT ON StudyAttendance TO Lecturer;

GRANT SELECT ON StudiesMeetingsOrganization TO Lecturer;
```

```
GRANT SELECT ON UsersAuthorizedForStudyMeeting TO Lecturer;
GRANT SELECT, INSERT, UPDATE, DELETE ON OnlineStudy TO Lecturer;
GRANT SELECT, INSERT, UPDATE, DELETE ON StationaryStudy TO Lecturer;
GRANT SELECT, INSERT, UPDATE, DELETE ON StudyAttendance TO Lecturer;

CREATE ROLE Student;

GRANT EXECUTE ON DidStudentPass TO Student;
GRANT EXECUTE ON GetSubjectStudentPresencePercentage TO Student;
GRANT EXECUTE ON GetStudyTimetable TO Student;
GRANT SELECT ON StudyMeetingPresence TO Student;
GRANT SELECT ON StudiesMeetingsOrganization TO Student;
GRANT SELECT ON FinalExams TO Student;
GRANT SELECT ON StudiesMeetings TO Student;
GRANT SELECT ON OnlineStudy TO Student;
GRANT SELECT ON StationaryStudy TO Student;
GRANT SELECT ON StudyMeetups TO Student;

ALTER ROLE Lecturer ADD MEMBER StudyCoordinator;
ALTER ROLE RegisteredUser ADD MEMBER Student;
```

Stworzenie roli koordynatora kursu

Autor: Wiktor Sędzimir

```
create role CourseCoordinator;

grant execute on AddCourse to CourseCoordinator;
grant execute on AddStationaryCourseMeeting to
CourseCoordinator;
```

```
grant execute on AddOnlineSyncCourseMeeting to
CourseCoordinator;

grant execute on AddOnlineAsyncCourseMeeting to
CourseCoordinator;

grant select, insert, update on Courses to CourseCoordinator;
grant select, insert, update on CoursesMeetings to
CourseCoordinator;

grant select, insert, update on StationaryCourse to
CourseCoordinator;

grant select, insert, update on OnlineSyncCourse to
CourseCoordinator;

grant select, insert, update on OnlineAsyncCourse to
CourseCoordinator;
```

Stworzenie roli prowadzącego kursu

Autor: Wiktor Sędzimir

```
create role CourseInstructor;

alter role CourseInstructor add member CourseCoordinator;

grant execute on AddCourseMeetingAttendance to
CourseInstructor;

grant execute on GetCourseUserPresencePercentage to
CourseInstructor;

grant select on CourseMeetingPresence to CourseInstructor;
grant select on CourseMeetingPresence to CourseInstructor;
grant select on CoursesMeetingsOrganization to
CourseInstructor;

grant select on CoursesTypes to CourseInstructor;
```

```
grant select on UsersAuthorizedForCourseMeeting to
CourseInstructor;

grant select on NumberOfUsersAuthorizedForCourseMeeting to
CourseInstructor;

grant select on CourseAttendace to CourseInstructor;

grant select, insert, update on CoursesAttendance to
CourseInstructor;
```

Stworzenie roli tłumacza

Autor: Wiktor Sędzimir

```
create role Translator;

grant select on CoursesMeetingsOrganization to Translator;
grant select on StudiesMeetingsOrganization to Translator;
grant select on Webinars to Translator;
grant select on FutureActivitiesReport to Translator;
```

Stworzenie roli użytkownika

Autor: Wiktor Sędzimir

```
create role RegisteredUser;

grant select on FutureActivitiesReport to RegisteredUser;
grant select on ServiceOffer to RegisteredUser;
grant select on GetUserSchedule to RegisteredUser;
grant execute on GetCourseUserPresencePercentage to
RegisteredUser;
```


Stworzenie roli pracownika administracyjnego

Autor: Szymon Migas

```
CREATE ROLE AdministrativeEmployee
GRANT SELECT ON BilocationReport TO AdministrativeEmployee
GRANT SELECT ON WebinarsIncome TO AdministrativeEmployee
GRANT SELECT ON StudyIncome TO AdministrativeEmployee
GRANT SELECT ON CoursesIncome TO AdministrativeEmployee
GRANT SELECT ON SummaryIncome TO AdministrativeEmployee
GRANT SELECT ON DiplomaInfo TO AdministrativeEmployee
GRANT SELECT ON CoursesDebtors TO AdministrativeEmployee
GRANT SELECT ON StudentDebtors TO AdministrativeEmployee
GRANT EXECUTE ON GetAnnualIncome TO AdministrativeEmployee
GRANT EXECUTE ON GetRangeIncome TO AdministrativeEmployee
GRANT EXECUTE ON GetStudyIncome TO AdministrativeEmployee
GRANT EXECUTE ON GetWebinarIncome TO AdministrativeEmployee
GRANT EXECUTE ON GetCourseIncome TO AdministrativeEmployee
```

Stworzenie roli dyrektora

Autor: Szymon Migas

```
CREATE ROLE HeadTeacher
GRANT SELECT ON AttendanceReport TO HeadTeacher
GRANT SELECT ON SummaryIncome TO HeadTeacher
GRANT SELECT ON CourseAttendance TO HeadTeacher
GRANT SELECT ON StudyAttendance TO HeadTeacher
GRANT SELECT ON WebinarAttendance TO HeadTeacher
GRANT EXECUTE ON AddHeadteacherPostponement TO HeadTeacher
```

Stworzenie roli administratora systemu

Autor: Szymon Migas

```
CREATE ROLE SystemAdministrator
GRANT ALL PRIVILEGES ON dbo TO SystemAdministrator
```

Stworzenie roli prezentera webinaru

Autor: Szymon Migas

```
CREATE ROLE WebinarPresenter  
  
GRANT SELECT, UPDATE ON Webinars TO WebinarPresenter  
  
GRANT SELECT, INSERT, DELETE ON WebinarsAttendance TO  
WebinarPresenter  
  
GRANT SELECT ON WebinarMeetingPresence TO WebinarPresenter  
  
GRANT SELECT ON WebinarAttendance TO WebinarPresenter  
  
GRANT EXECUTE ON AddWebinar TO WebinarPresenter
```

Stworzenie roli użytkownika niezalogowanego

Autor: Szymon Migas

```
GRANT EXECUTE ON AddUser TO guest
```

Indeksy

Utworzenie indeksów

Dla każdej tabeli dodano indeksy na kolumny będące kluczami obcymi, aby przyspieszyć wykonywanie operacji łączenia tych tabel.

```
-- COURSE INDEXES

CREATE INDEX CoursesIDX ON Courses (CourseCoordinatorID,
TranslatorLanguageID);

CREATE INDEX CoursesMeetingsIDX ON CoursesMeetings (CourseID,
CourseInstructorID);

CREATE INDEX StationaryCourseIDX ON StationaryCourse
(ClassroomID);

-- STUDIES INDEXES

CREATE INDEX StudiesIDX ON Studies (StudyCoordinatorID);

CREATE INDEX StudentsIDX ON Students (UserID, StudyID,
SemesterNo);

CREATE INDEX InternshipsIDX ON Internships (StudyID);

CREATE INDEX StudySemestersIDX ON StudySemesters (StudyID,
SemesterNo);

CREATE INDEX StationaryStudyIDX ON
StationaryStudy (ClassroomID);

CREATE INDEX StudiesMeetingsIDX ON
StudiesMeetings (StudyMeetupID, SubjectID, LecturerID,
TranslatorLanguageID);

CREATE INDEX StudyMeetupsIDX ON
StudyMeetups (StudySemesterID);
```

```
CREATE INDEX FinalExamsIDX ON FinalExams (GradeID);

-- PEOPLE INDEXES

CREATE INDEX EmployeesIDX ON Employees (AddressID);

CREATE INDEX TranslatorsIDX ON Translators (LanguageID,
EmployeeID);

CREATE INDEX UsersIDX ON Users (AddressID);

-- ORDERS INDEXES

CREATE INDEX OrdersIDX ON Orders (UserID);

CREATE INDEX OrderDetailsIDX ON OrderDetails (OrderID,
ServiceID, ServiceTypeID);

CREATE INDEX PaymentsIDX on Payments (OrderDetailID);

CREATE INDEX HeadTeacherPaymentPostponementsIDX on
HeadTeacherPaymentPostponements (ServiceTypeID, ServiceID,
UserID);

-- WEBINAR INDEXES

CREATE INDEX WebinarAttendanceIDX on WebinarsAttendance
(UserID);

CREATE INDEX WebinarsIDX on Webinars (TranslatorLanguageID,
WebinarPresenterID);

-- MISC INDEXES

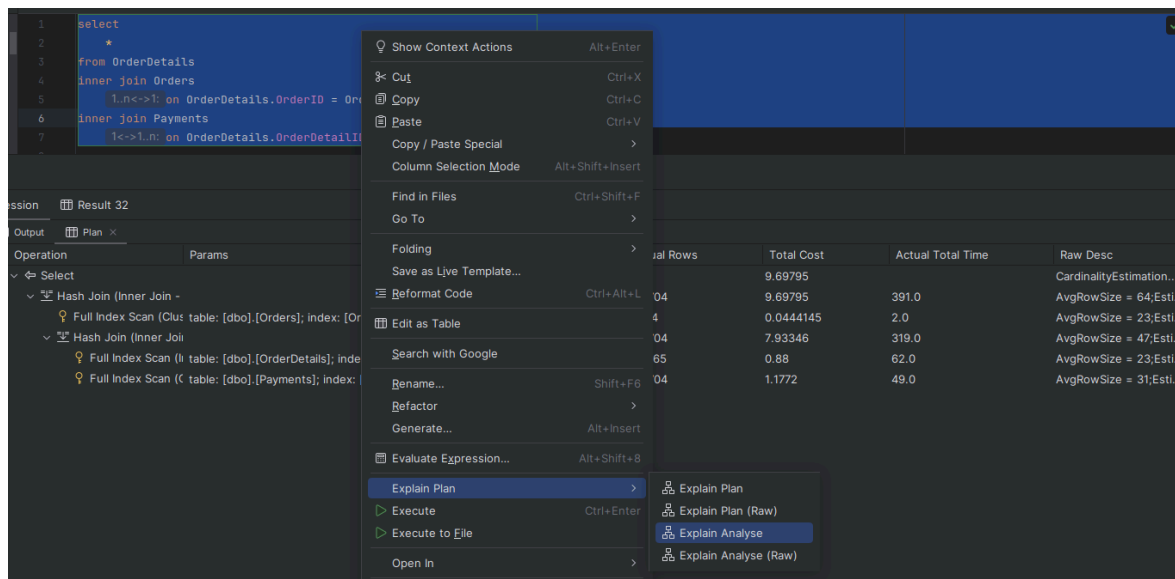
CREATE INDEX CitiesIDX on Cities (CountryID);

CREATE INDEX AddressesIDX on Addresses (CityID);
```

```
CREATE INDEX ClassroomsIDX on Classrooms (AddressID);
```

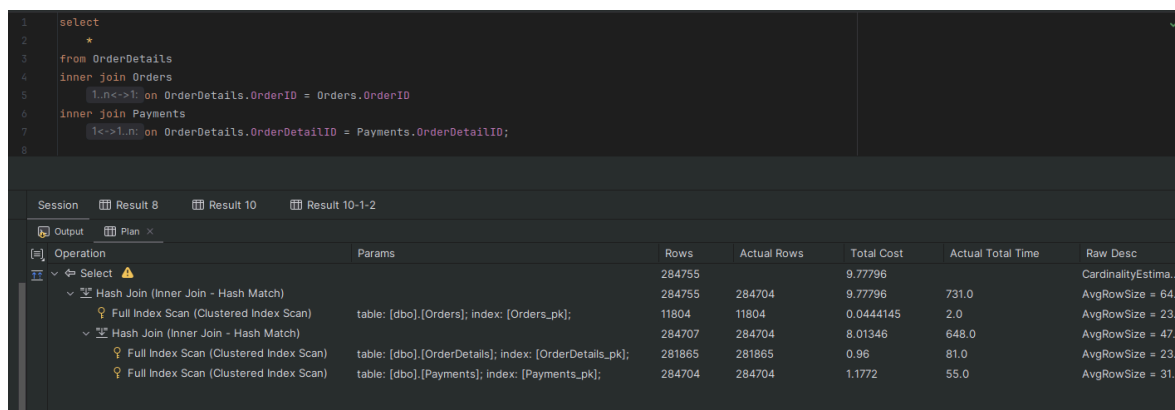
Porównanie czasów wykonywania zapytań

Do porównania czasów wykonywania zapytań na tabelach bez indeksów oraz z indeksami wykorzystano wbudowane narzędzie w programie *DataGrip*. Sposób jego uruchomienia przedstawiono na poniższym screenie.



Analiza wykonania zapytania łączącego table: *Orders*, *OrderDetails*, *Payments*:

Przed stworzeniem indeksów:



Po stworzeniu indeksów:

Session: u_sedzimir

Result 32

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Raw Desc
Select		284751		9.69795		CardinalityEstimation...
Hash Join (Inner Join)		284751	284704	9.69795	391.0	AvgRowSize = 64;Esti...
Full Index Scan (Clut table: [dbo].[Orders]; index: [Orders_pk];)		11804	11804	0.0444145	2.0	AvgRowSize = 23;Esti...
Hash Join (Inner Join)		284706	284704	7.93346	319.0	AvgRowSize = 47;Esti...
Full Index Scan (h table: [dbo].[OrderDetails]; index: [OrderDetailsIDX];)		281865	281865	0.88	62.0	AvgRowSize = 23;Esti...
Full Index Scan (table: [dbo].[Payments]; index: [Payments_pk];)		284704	284704	1.1772	49.0	AvgRowSize = 31;Esti...

Analiza wykonania widoku AttendanceReport:

Przed stworzeniem indeksów:

Session: u_sedzimir.dbo.AttendanceReport

Result 15

Result 10-1-2

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Raw Desc
Select		3		30.7912		Cardinality...
Union All (Concatenation)		3	3	30.7912	7757.0	AvgRowSiz...
Value (Compute Scalar)		1		4.31461		AvgRowSiz...
Value (Compute Scalar)		1		22.3771		AvgRowSiz...
Value (Compute Scalar)		1		22.3771		AvgRowSiz...
Aggregate (Aggregate - Stream Aggregate)		1	1	22.3771	7470.0	AvgRowSiz...
Hash Join (Left Outer Join - Hash M)		5931.67	26690	22.3735	4711.0	AvgRowSiz...
Value (Compute Scalar)		1		4.09956		AvgRowSiz...
Value (Compute Scalar)		1		4.09956		AvgRowSiz...
Aggregate (Aggregate - Stream Aggregate)		1	1	4.09956	145.0	AvgRowSiz...
Merge Join (Inner Join - Merge Joi)		197.212	417	4.09944	103.0	AvgRowSiz...
Merge Join (Inner Join - Merge .		272.834	417	0.0407144	1.0	AvgRowSiz...
Full Index Scan (Clustered Inc table: [dbo].[CoursesMeetings]; index: [CoursesMeetings_pk];)		425.281	422	0.006223...	0.0	AvgRowSiz...
Value (Compute Scalar)		417		0.0267778		AvgRowSiz...
Value (Compute Scalar)		469.849		4.05154		AvgRowSiz...
Aggregate (Aggregate - Stream		469.849	648	4.05154	101.0	AvgRowSiz...
Sort Unique (Distinct Sort - Si		517.677	13414	4.051	100.0	AvgRowSiz...
Union All (Concatenatic		517.687	14076	4.03235	94.0	AvgRowSiz...
Hash Join (Inner Joi		432.143	13227	3.98653	92.0	AvgRowSiz...
Hash Join (Inner Joi		85.5437	849	0.0457726	1.0	AvgRowSiz...

Po stworzeniu indeksów:

13
14 `select *`
15 `from AttendanceReport;`

Session: u_sedzimir.dbo.FutureActivitiesReport

Output Plan

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Raw Desc
Select		3		30.3601		Cardinality...
Union All (Concatenation)		3	3	30.3601	7540.0	AvgRowSiz...
Value (Compute Scalar)		1		4.23461		AvgRowSiz...
Value (Compute Scalar)		1		4.23461		AvgRowSiz...
Aggregate (Aggregate - Stream Aggregat)		1	1	4.23461	177.0	AvgRowSiz...
Merge Join (Inner Join - Merge Join)		470.302	554	4.23432	104.0	AvgRowSiz...
Nested Loops (Inner Join - Nested Loops)		1067.44	748	4.18624	103.0	AvgRowSiz...
Index Scan (Index Seek)	table: [dbo].[ServiceTypes]; index: [ServiceNameUnique];	1	1	0.0032831	0.0	AvgRowSiz...
Sort		276.53	748	4.1818	103.0	AvgRowSiz...
Value (Compute Scalar)		276.53		4.16694		AvgRowSiz...
Aggregate (Aggregate - Stream Aggregat)		276.53	748	4.16694	102.0	AvgRowSiz...
Hash Join (Inner Join - Hash Match)		8983.79	4013	4.10376	101.0	AvgRowSiz...
Hash Join (Inner Join - Hash Match)		11828.7	4013	1.39282	22.0	AvgRowSiz...
Full Index Scan	table: [dbo].[Webinars]; index: [Webinars_pk];	751	751	0.0181822	0.0	AvgRowSiz...
Full Index Scan	table: [dbo].[OrderDetails]; index: [OrderDetailsIDX];	70466.3	4013	0.88	20.0	AvgRowSiz...
Full Index Scan	table: [dbo].[Payments]; index: [Payments_pk];	284704	284704	1.1772	48.0	AvgRowSiz...
Merge Join (Inner Join - Merge Join)		416.446	554	0.0399857	1.0	AvgRowSiz...
Full Index Scan (Clustered Index)	table: [dbo].[Webinars]; index: [Webinars_pk];	565.176	565	0.0181822	0.0	AvgRowSiz...
Value (Compute Scalar)		554		0.0134481		AvgRowSiz...
Aggregate (Aggregate - Stream Aggregat)		554	554	0.0134481	1.0	AvgRowSiz...
Full Index Scan (Clustered Index)	table: [dbo].[WebinarsAttendance]; index: [WebinarsAttendance_pk];	2767	2767	0.0115109	0.0	AvgRowSiz...
Value (Compute Scalar)		1		22.2118		AvgRowSiz...

Analiza wykonania widoku *BilocationReport*:

Przed stworzeniem indeksów:

16 `select *`
17 `from BilocationReport;`
18

Session: u_sedzimir.dbo.BilocationReport

Output Plan

Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Raw Desc
Select		263582		45.5545		CardinalityE...
Transformation (Gather Streams - Parallelism)		263582	1451	45.5545	1956.0	AvgRowSiz...
Hash Join (Inner Join - Hash Match)		263582	1451	43.8696	1953.0	AvgRowSiz...
Transformation (Repartition Streams - Parallelism)		8501	8501	0.175333	9.0	AvgRowSiz...
Hash Join (Inner Join - Hash Match)		263551	1451	42.7823	1942.0	AvgRowSiz...
Union All (Concatenation)		287432	275832	15.2707	259.0	AvgRowSiz...
Transformation (Repartition Streams - Parallelism)		2911.55	980	2.35211	31.0	AvgRowSiz...
Transformation (Repartition Streams - Parallelism)		5624....	1249	2.47116	31.0	AvgRowSiz...
Transformation (Repartition Streams - Parallelism)		277909	273603	10.4331	192.0	AvgRowSiz...
Union All (Concatenation)		287432	275832	15.2707	357.0	AvgRowSiz...
Transformation (Repartition Streams - Parallelism)		2911.55	980	2.35211	25.0	AvgRowSiz...
Transformation (Repartition Streams - Parallelism)		5624....	1249	2.47116	28.0	AvgRowSiz...
Transformation (Repartition Streams - Parallelism)		277909	273603	10.4331	297.0	AvgRowSiz...

Po stworzeniu indeksów:

15	select *
16	from BilocationReport;

Session u_sedzimir.dbo.BilocationReport						
Operation	Params	Rows	Actual Rows	Total Cost	Actual Total Time	Raw Desc
Select		259740		44.5141		CardinalityEs...
Transformation (Gather Streams - Parallelism)		259740	1452	44.5141	1817.0	AvgRowSize ...
Hash Join (Inner Join - Hash Match)		259740	1452	42.8533	1817.0	AvgRowSize ...
Transformation (Repartition Streams - Parallelism)		8501	8501	0.175333	10.0	AvgRowSize ...
Full Index Scan (Clustered Index Scan)	table: [dbo].[Users]; index: [Users_pk];	8501	8501	0.124916	9.0	AvgRowSize ...
Hash Join (Inner Join - Hash Match)		259710	1452	41.7778	1805.0	AvgRowSize ...
Union All (Concatenation)		283560	274574	14.905	229.0	AvgRowSize ...
Transformation (Repartition Streams - Parallelism)		2907.79	974	2.25048	35.0	AvgRowSize ...
Sort Unique (Distinct Sort - Sort)		2907.79	974	2.21297	35.0	AvgRowSize ...
Value (Compute Scalar)		2908.14		2.18119		AvgRowSize ...
Transformation (Repartition Streams - Paralle		2908.14	974	2.18105	35.0	AvgRowSize ...
Hash Join (Inner Join - Hash Match)		2908.14	974	2.14393	34.0	AvgRowSize ...
Transformation (Bitmap Create - Bitmap		2908.04	974	2.01973	32.0	AvgRowSize ...
Transformation (Repartition Streams		2908.04	974	2.01973	32.0	AvgRowSize ...
Transformation (Repartition Streams - f		8501	899	0.0569347	1.0	AvgRowSize ...
Full Index Scan (Index Scan)	table: [dbo].[Users]; index: [UsersIDX];	8501	899	0.0182494	0.0	AvgRowSize ...
Transformation (Repartition Streams - Parallelism)		5601.23	1239	2.36811	31.0	AvgRowSize ...
Sort Unique (Distinct Sort - Sort)		5601.23	1239	2.3215	29.0	AvgRowSize ...
Value (Compute Scalar)		5617.07		2.24914		AvgRowSize ...
Transformation (Repartition Streams - Paralle		5617.07	1239	2.24886	28.0	AvgRowSize ...
Hash Join (Inner Join - Hash Match)		5617.07	1239	2.20498	28.0	AvgRowSize ...
Transformation (Bitmap Create - Bitmap		5616.88	1239	2.05276	25.0	AvgRowSize ...
Transformation (Repartition Streams		5616.88	1239	2.05276	25.0	AvgRowSize ...