# Projekt na Podstawy Baz Danych
# 2023/2024

Jakub Kotara, Ida Ciepiela, Igor Sikora

# Spis treści

# Użytkownicy systemu

Hierarchiczny układ użytkowników:

- Dyrektor szkoły
- Administrator

- Sekretariat
- Koordynator studiów
- Koordynator przedmiotu
- Prowadzący zajęcia
- Tłumacz
- Użytkownik zalogowany
- Użytkownik niezalogowany

# Opis funkcji systemu

- Sprawdzanie frekwencji na zajęciach zdalnych (System)
- Monitorowanie stanu płatności klientów (System)
- Dodawanie użytkowników którzy stworzyli konto (System)
- Wyliczenie wartości koszyka (System)
- Sprawdzanie czy jest wolne miejsce na kursie / studiach (System)
- Przekierowanie do systemu płatności (System)
- Odbieranie informacji zwrotnej o stanie płatności (System)
- Zapisanie na listę kursantów / studentów (System)
- Sprawdzanie czy płatność została dokonana (System)
- Przyznanie dostępu do kursu (System)
- Przyznanie dostępu do webinaru na 30 dni (System)
- Przyznanie dostępu do wykładu (System)

- Decyzja o odroczeniu płatności (Dyrektor)

- Usunięcie webinaru (Administrator)

- Generowanie raportów: (Sekretariat)
    - Finansowych
    - Pokazujących osoby, które jeszcze nie ziściły opłat
    - Lista osób z odroczonymi płatnościami
    - bilokacji zajęć
    - Lista osób zapisanych na przyszłe wydarzenia
    - Lista pracowników
- Skreślanie użytkownika z listy studentów (Sekretariat)
- Ustala ceny kursów / webinarów / studiów (Sekretariat)

- Dodawanie sylabusu studiów (Koordynator studiów)
- Potwierdzenia odbytych praktyk (Koordynator studiów)
- Raport dotyczący listy studentów (Koordynator studiów)

- Dodanie webinaru (Koordynator przedmiotu)
- Wystawianie zaliczeń (Koordynator przedmiotu)
- Przypisywanie prowadzących / tłumaczów do zajęć (Koordynator przedmiotu)
- Potwierdzenia odrobionych zajęć (Koordynator przedmiotu)

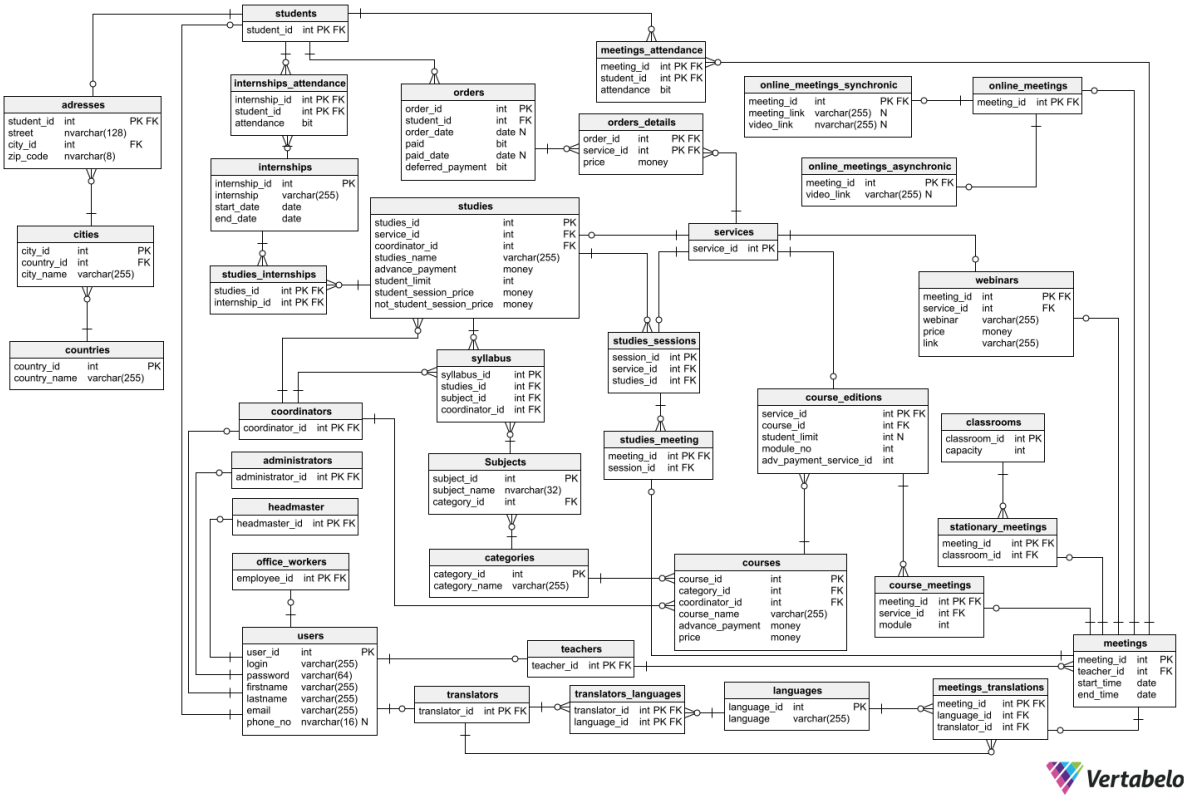
- Raport dotyczący osób zapisanych na przedmiot (Koordynator przedmiotu)
- Raport dotyczący prowadzących zajęcia i tłumaczów (Koordynator przedmiotu)
- Raport dotyczący frekwencji na zajęciach (Koordynator przedmiotu)
- Edytowanie harmonogramu zajęć (Koordynator przedmiotu)

- Edytowanie frekwencji studentów na stacjonarnych zajęciach (Prowadzący zajęcia)
- Wprowadzanie ocen z egzaminów końcowych (Prowadzący zajęcia)
- Raport dotyczący prowadzonych przez niego zajęć (Prowadzący zajęcia)

- Zapisanie na kursy / webinary / studia (Uż. zalogowany)
- Rezygnacja z kursów / studiów (Uż. zalogowany)
- Sprawdzenie swojego harmonogramu (Uż. zalogowany)
- Dokonanie opłaty za kursy / studia / webinaru (Uż. zalogowany)
- Dodawanie/usuwanie produktów do koszyka (Uż. zalogowany)
- Modyfikowanie danych osobowych i logowania (Uż. zalogowany)
- Otrzymanie raportu o zaległych płatnościach (Uż. zalogowany)
- Sprawdzenie swojego postępu w kursach / studiach (Uż. zalogowany)
- Sprawdzanie swojej frekwencji na zajęciach (Uż. zalogowany)
- Sprawdzanie swoich ocen z egzaminów (Uż. zalogowany)

- Przeglądanie oferty (Uż. niezalogowany)
- Rejestracja konta (Uż. niezalogowany)

# Schemat bazy danych




# Opis tabel

## administrators

wyszczególnia, którzy użytkownicy są administratorami

**klucz główny:** administrator_id

**klucz obcy:** administrator_id (z users:users_id)

**Indeksy:** administrator_id

administrator_id – identyfikator administratora

```
CREATE TABLE administrators (
    administrator_id int  NOT NULL,
    CONSTRAINT administrators_pk PRIMARY KEY  (administrator_id)
);
```

## addresses

zawiera informacje adresowe każdego studenta

**klucz główny:** student_id

**klucze obce:** street_id (ze streets), student_id (ze students)

**Indeksy:** student_it, street

**Warunki integracyjne:** zip_code jest numeryczny

student_id – unikalne ID studenta, któremu przypisuje się adres
street - nazwa ulicy wraz z numerem budynku/mieszkania
city_id - ID miasta
zip_code - kod pocztowy

```
CREATE TABLE adresses (
    student_id int  NOT NULL,
    street nvarchar(128)  NOT NULL,
    city_id int  NOT NULL,
    zip_code nvarchar(8)  NOT NULL CHECK (ISNUMERIC( [zip_code] ) = 1),
    CONSTRAINT adresses_pk PRIMARY KEY  (student_id)
);
```

## categories

zawiera możliwe tematyki zajęć

**klucz główny:** category_id

**Indeksy:** category_id, category_name

category_id – unikalne ID kategorii (tematyki zajęć)
category_name – nazwa kategorii

```
CREATE TABLE categories (
    category_id int  NOT NULL,
    category_name varchar(255)  NOT NULL,
    CONSTRAINT category_pk PRIMARY KEY  (category_id)
);
```

## cities

zawiera miasta, w których mieszkają studenci

**klucz główny:** city_id

**klucz obcy:** country_id (z countries)

**Indeksy:** city_id, city_name

city_id – unikalne ID miasta
country_id – ID państwa, w którym znajduje się miasto
city_name – nazwa miasta

```
CREATE TABLE cities (
    city_id int  NOT NULL,
    country_id int  NOT NULL,
    city_name varchar(255)  NOT NULL,
    CONSTRAINT cities_pk PRIMARY KEY  (city_id)
);
```

## classrooms

zawiera sale, w których odbywają się zajęcia

**klucz główny:** classroom_id

**Indeksy:** classroom_id

**warunki integralności**: minimalna pojemność sal > pojemność > 0

classroom_id – unikalne ID sali
capacity – pojemność sali (ile uczniów może w niej przebywać)

```
CREATE TABLE classrooms (
    classroom_id int  NOT NULL,
    capacity int  NOT NULL CHECK ([capacity] > 0),
    CONSTRAINT classrooms_pk PRIMARY KEY  (classroom_id)
);
```

## coordinators

wyszczególnia, którzy użytkownicy są koordynatorami zajęć

**klucz główny:** coordinator_id

**klucz obcy:** coordinators_id (z users:users_id)

**Indeksy:** coordinator_id

coordinator_id – unikalne ID koordynatora

```
CREATE TABLE coordinators (
    coordinator_id int  NOT NULL,
    CONSTRAINT coordinators_pk PRIMARY KEY  (coordinator_id)
);
```

## countries

zawiera państwa, w których mieszkają studenci

**klucz główny:** country_id

**Indeksy:** country_id, country_name

country_id – unikalne ID państwa
country_name – nazwa państwa

```
CREATE TABLE countries (
    country_id int  NOT NULL,
    country_name varchar(255)  NOT NULL,
    CONSTRAINT countries_pk PRIMARY KEY  (country_id)
);
```

## courses

zawiera ogólne informacje o kursach oferowanych przez szkołę

**klucz główny:** course_id

**klucze obce:** category_id (z categories), coordinator_id ( z coordinators)

**Indeksy:** course_id

**warunki integralności**: cena >= zaliczka >= 0

course_id – unikalne ID kursu
category_id – ID tematyki, jaką dotyczy kurs
coordinator_id – ID osoby odpowiedzialnej za kurs
advance_payment – wysokość zaliczki
price – cała cena za kurs (zawiera zaliczkę)

```
CREATE TABLE courses (
    course_id int  NOT NULL,
    category_id int  NOT NULL,
    coordinator_id int  NOT NULL,
    advance_payment money  NOT NULL CHECK ([advance_payment] >= 0),
    price money  NOT NULL CHECK ([price] >= 0),
    CONSTRAINT courses_pk PRIMARY KEY  (course_id)
);
```

## course_editions

zawiera informacje dotyczące konkretnej edycji kursu (te same kursy mogą odbywać się wielokrotnie)

**klucz główny:** service_id

**klucz obcy:** service_id (z services), course_id (z courses)

**Indeksy: Indeksy:** service_id, course_id

**warunki integralności:** student_limit > 0

service_id – unikalne ID edycji kursu
course_id – ID kursu, do którego ta edycja należy
start_date – data rozpoczęcia kursu
end_date – data zakończenia kursu
student_limit – maksymalna ilość studentów na kurs (NULL – brak limitu)
module_no - ilość modułów

```
CREATE TABLE course_editions (
    service_id int  NOT NULL,
    course_id int  NOT NULL,
    student_limit int  NULL CHECK ([student_limit] > 0),
    module_no int  NOT NULL,
    CONSTRAINT course_edition_pk PRIMARY KEY  (service_id)
);
```

## course_meetings

rozdziela kurs na poszczególne spotkania

**klucze główne:** meeting_id, service_id
**klucze obce:** meeting_id (z meetings), service_id (z course_editions)

**Indeksy:** meeting_id,  module

meeting_id – unikalne ID spotkania
service_id – unikalne ID edycji kursu, do którego należy spotkanie
module - numer modułu

```
CREATE TABLE course_meetings (
    meeting_id int  NOT NULL,
    service_id int  NOT NULL,
    module int  NOT NULL,
    CONSTRAINT course_meetings_pk PRIMARY KEY  (meeting_id,service_id)
);
```

## headmaster

zawiera ID dyrektora

**klucz główny:** headmaster_id

**klucz obcy:** headmaster_id (z users:users_id)

**Indeksy:** headmaster_id

headmaster_id – ID dyrektora

```
CREATE TABLE headmaster (
    headmaster_id int  NOT NULL,
    CONSTRAINT headmaster_pk PRIMARY KEY  (headmaster_id)
);
```

## internships

określa praktyki, które mogą być zaliczone przez studenta

**klucz główny:** internship_id

**Indeksy:** internship_id

internship_id - unikalne ID praktyk
internship - nazwa praktyk
start_date - czas rozpoczęcia praktyk
end_date - czas zakończenia praktyk

```
CREATE TABLE internships (
    internship_id int  NOT NULL,
    internship varchar(255)  NOT NULL,
    start_date date  NOT NULL,
    end_date date  NOT NULL,
    CONSTRAINT internship_pk PRIMARY KEY  (internship_id)
);
```

## internships_attendance

określa obecność studenta na praktykach

**klucz główny:** internship_id, student_id
**klucze obce:** interenship_id (z *internships*), student_id (z *students*)
**Indeksy:** internship_id, attendance
**warunki integralności:** czas rozpoczęcia praktyk po czasie teraźniejszym

internship_id - ID praktyk, którego dotyczy obecność
student_id - ID studenta, którego dotyczy obecność
attendance - informacja czy student był obecny na praktykach ( 1- tak, 0 - nie)

```
CREATE TABLE internships_attendance (
    internship_id int  NOT NULL,
    student_id int  NOT NULL,
    attendance bit  NOT NULL,
    CONSTRAINT internship_attendance_pk PRIMARY KEY  (internship_id,student_id)
);
```

## languages

określa języki, w których odbywają się spotkania lub które mogą tłumaczyć tłumacze

**klucz główny:** language_id

**Indeksy:** language_id

language_id - unikalne ID języka
language - nazwa języka

```
CREATE TABLE languages (
    language_id int  NOT NULL,
    language varchar(255)  NOT NULL,
    CONSTRAINT languages_pk PRIMARY KEY  (language_id)
);
```

## meetings_attendance

określa obecność studenta na spotkaniu

**klucz główny:** meeting_id, student_id
**klucze obce:** meeting_id (z *meetings*), student_id (z *students*)
**Indeksy:** meeting_id, attendance

meeting_id - ID spotkania, którego dotyczy obecność
student_id - ID studenta, którego dotyczy obecność
attendance - stwierdzenie czy student był obecny

```
CREATE TABLE meetings_attendance (
    meeting_id int  NOT NULL,
    student_id int  NOT NULL,
    attendance bit  NOT NULL,
    CONSTRAINT meeting_attendance_pk PRIMARY KEY  (meeting_id,student_id)
);
```

## meetings

określa pojedyncze spotkania w których można brać udział (webinary, zajęcia z kursu, zjazdy studyjne)

**klucz główny:** meeting_id
**klucze obce:** teacher_id (z *teachers*)
**Indeksy:** meeting_id, (start_time, end_time)
**warunki integralności:** czas rozpoczęcia spotkania przed czasem zakończenia spotkania

meeting_id - unikalne ID spotkania
teacher_id - id prowadzącego spotkanie
start_time - czas rozpoczęcia spotkania

end_time - czas zakończenia spotkania

```
CREATE TABLE meetings (
    meeting_id int  NOT NULL,
    teacher_id int  NOT NULL,
    start_time date  NOT NULL,
    end_time date  NOT NULL,
    CONSTRAINT meetings_pk PRIMARY KEY  (meeting_id)
);
```

## meetings_translations

określa który tłumacz tłumaczył które spotkanie jeśli nie było ono prowadzone w języku polskim

**klucz główny:** meeting_id
**klucze obce:** meeting_id (z *meetings*), language_id (z *languages*), translator_id (z *translators*)
**Indeksy:** meeting_id

meeting_id - ID spotkania, które było tłumaczone
language_id - ID języka, w którym odbyło się spotkanie
translator_id - ID tłumacza, który tłumaczył spotkanie

```
CREATE TABLE meetings_translations (
    meeting_id int  NOT NULL,
    language_id int  NOT NULL,
    translator_id int  NOT NULL,
    CONSTRAINT meetings_translations_pk PRIMARY KEY  (meeting_id)
);
```

## office_workers

określa pracowników biura

**klucz główny:** employee_id
**klucze obce:** employee_id (z users.user_id)
**Indeksy:** employee_id

employee_id - ID użytkownika, który jest pracownikiem biura

```
CREATE TABLE office_workers (
    employee_id int  NOT NULL,
    CONSTRAINT office_pk PRIMARY KEY  (employee_id)
);
```

## online_meetings

określa spotkania prowadzone zdalnie

**klucz główny:** meeting_id
**klucze obce:** meeting_id (z meetings)
**Indeksy:** meeting_id

meeting_id - ID spotkania, które jest prowadzone zdalnie

```
CREATE TABLE online_meetings (
    meeting_id int  NOT NULL,
    CONSTRAINT online_meetings_pk PRIMARY KEY  (meeting_id)
);
```

## online_meetings_asynchronic

określa spotkania prowadzone zdalnie i asynchronicznie

**klucz główny:** meeting_id
**klucze obce:** meeting_id (z *meeting*)
**Indeksy:** meeting_id, video_link
**Warunki integralności**: video_link unikalny

meeting_id - ID spotkania, które jest prowadzone zdalnie i asynchronicznie
video_link - link do spotkania

```
CREATE TABLE online_meetings_asynchronic (
    meeting_id int  NOT NULL,
    video_link varchar(255)  NULL,
    CONSTRAINT online_meetings_asynchronic_pk PRIMARY KEY  (meeting_id)
);
CREATE UNIQUE NONCLUSTERED INDEX online_meetings_asynchronic_idx_1 on online_meetings_asynchronic (video_link ASC)
    WHERE [video_link] IS NOT NULL
;
```

## online_meetings_synchronic

określa spotkania prowadzone zdalnie i synchronicznie

**klucz główny:** meeting_id
**klucze obce:** meeting_id (z *meeting*)
**Indeksy:** meeting_id
**Warunki integralności**: video_link,meeting_link unikalne

meeting_id - ID spotkania, które jest prowadzone zdalnie i synchronicznie
meeting_link - link do spotkania

```
CREATE TABLE online_meetings_synchronic (
    meeting_id int  NOT NULL,
    meeting_link varchar(255)  NULL,
    video_link nvarchar(255)  NULL,
    CONSTRAINT online_meetings_synchronic_pk PRIMARY KEY  (meeting_id)
);
CREATE UNIQUE NONCLUSTERED INDEX online_meetings_synchronic_idx_1 on online_meetings_synchronic (meeting_link ASC)
    WHERE [meeting_link] IS NOT NULL
;
```

## orders

określa zamówienia złożone przez użytkowników

**klucz główny:** order_id
**klucze obce:** student_id (z students)
**Indeksy:** order_id, paid, deferred_payment, paid_date
**warunki integralności:** czas złożenia zamówienia po czasie teraźniejszym
czas opłacenia zamówienia po czasie złożenia zamówienia

order_id - unikalne ID zamówienia
student_id - ID użytkownika, który składa zamówienie
order_date - czas złożenia zamówienia
paid - stwierdzenie czy zostało zapłacone
paid_date - czas opłacenia zamówienia
deferred_payment - stwierdzenie czy płatność została odroczona

```
CREATE TABLE orders (
    order_id int  NOT NULL,
    student_id int  NOT NULL,
    order_date date  NULL DEFAULT GETDATE() CHECK ([order_date] <= GETDATE()),
    paid bit  NOT NULL,
    paid_date date  NULL,
    deferred_payment bit  NOT NULL,
    CONSTRAINT orders_pk PRIMARY KEY  (order_id)
);
```

## order_details

określa szczegóły zamówienia

**klucz główny:** order_id, service_id
**klucze obce:** order_id (z *orders*), service_id (z *services*)
**Indeksy:** order_id
**warunki integralności:** cena usługi większa niż zero

order_id - numer zamówienia, którego to dotyczy
service_id - numer usługi, która została kupiona w danym zamówieniu
price - cena usługi

```
CREATE TABLE orders_details (
    order_id int  NOT NULL,
    service_id int  NOT NULL,
    price money  NOT NULL CHECK ([price] >= 0),
    CONSTRAINT orders_details_pk PRIMARY KEY  (order_id,service_id)
);
```

## services

określa usługi, które można kupić

**klucz główny**: service_id
**Indeksy:** service_id

service_id - unikalne ID usługi

```
CREATE TABLE services (
    service_id int  NOT NULL,
    CONSTRAINT services_pk PRIMARY KEY  (service_id)
);
```

## stationary_meetings

określa spotkania prowadzone stacjonarnie

**klucz główny**: meeting_id
**klucze obce**: meeting_id (z *meetings*), classroom_id (z *classrooms*)
**Indeksy:** meeting_id

meeting_id - unikalne ID spotkania które jest prowadzone stacjonarnie
classroom_id - numer sali w której odbywa się spotkanie

```
CREATE TABLE stationary_meetings (
    meeting_id int  NOT NULL,
    classroom_id int  NOT NULL,
    CONSTRAINT stationary_meetings_pk PRIMARY KEY  (meeting_id)
);
```

## students

określa studentów

**klucz główny**: student_id
**klucze obce**: student_id (z *users.user_id*)
**Indeksy:** student_id

student_id - ID użytkownika który jest studentem

```
CREATE TABLE students (
    student_id int  NOT NULL,
    CONSTRAINT students_pk PRIMARY KEY  (student_id)
);
```

## studies

zawiera kierunki studiów

**klucz główny**: studies_id
**klucze obce**: coordinator_id (z *coordinators*)
**Indeksy:** studies_id
**warunki integralności**:advance_payment > 0, student_limit > 0, student_session_price > 0, not_student_session_price > 0

studies_id - unikalne ID kierunku studiów
coordinator_id - unikalne ID koordynatora kierunku
advance_payment - kwota wpisowego
student_limit - liczba miejsc na roku kierunku
student_session_price - cena za pojedynczy zjazd dla studenta
not_student_session_price - cena za pojedynczy zjazd dla nie-studenta

```
CREATE TABLE studies (
    studies_id int  NOT NULL,
    coordinator_id int  NOT NULL,
    advance_payment money  NOT NULL CHECK ([advance_payment] >= 0),
    student_limit int  NOT NULL CHECK ([student_limit] >0),
    student_session_price money  NOT NULL ([student_session_price] >= 0),
    not_student_session_price money  NOT NULL ([not_student_session_price] >= 0),
    CONSTRAINT studies_pk PRIMARY KEY  (studies_id)
);
```

## studies_internships

określa które praktyki należą do których studiów

**klucz główny**: studies_id, internship_id
**klucze obce**: studies_id (z *studies*), internship_id (z *internships*)
**Indeksy:** studies_id

studies_id - ID studiów
internship_id - ID praktyk

```
CREATE TABLE studies_internships (
    studies_id int  NOT NULL,
    internship_id int  NOT NULL,
    CONSTRAINT studies_internships_pk PRIMARY KEY  (studies_id,internship_id)
);
```

## studies_meeting

rozdziela przedmioty studiów na pojedyncze spotkania

**klucz główny**: meeting_id
**klucze obce**: meeting_id (z *meetings*), service_id (z *services*), subject_id (z *syllabus*)
**Indeksy:** meeting_id

meeting_id - unikalne ID spotkania ze studiów
service_id - unikalne ID spotkania ze studiów jako usługi
subject_id - ID przedmiotu do którego to spotkanie należy

```
CREATE TABLE studies_meeting (
    meeting_id int  NOT NULL,
    service_id int  NOT NULL,
    subject_id int  NOT NULL,
    CONSTRAINT studies_meeting_pk PRIMARY KEY  (meeting_id)
);
```

## studies_sessions

rozdziela kierunki studiów na semestry

**klucz główny**: session_id
**klucze obce**: session_id (z *services.service_id*), studies_id (z *studies*)
**Indeksy:** session_id

session_id - ID zjazdu jako usługi
studies_id - ID kierunku studiów do którego należy zjazd kierunku

```
CREATE TABLE studies_sessions (
    session_id int  NOT NULL,
    studies_id int  NOT NULL,
    CONSTRAINT study_year_pk PRIMARY KEY  (session_id)
);
```

## subjects

Lista przedmiotów studyjnych
**Klucz główny:** subject_id
**Klucz obcy:** category_id (z categories)
**Indeksy:** subject_id

subject_id - ID przedmiotu
subject_name - nazwa przedmiotów
Category_id - ID kategorii, do jakiej należy przedmiot

```
CREATE TABLE subjects (
    subject_id int  NOT NULL,
    subject_name nvarchar(32)  NOT NULL,
    category_id int  NOT NULL,
    CONSTRAINT Subjects_pk PRIMARY KEY  (subject_id)
);
```

## syllabus

określa przedmioty na kierunku studiów

**klucz główny**: syllabus_id
**klucze obce**: session_id (z *studies_session*), coordinator_id (z *coordinators*), category_id (z *categories*)
**Indeksy:** syllabus_id

syllabus- ID przedmiotu na konkretnym kierunku
studies_id - ID kierunku studiów
coordinator_id - ID koordynatora przedmiotu
subject_id - ID przedmiotu

```
CREATE TABLE syllabus (
    subject_id int  NOT NULL,
    coordinator_id int  NOT NULL,
    category_id int  NOT NULL,
    session_id int  NOT NULL,
    CONSTRAINT syllabus_pk PRIMARY KEY  (subject_id)
);
```

## teachers

zawiera wykładowców

**klucz główny**: teacher_id
**klucze obce**: teacher_id (z *users.user_id*)
**Indeksy:** teacher_id

teacher_id - ID użytkownika który jest nauczycielem

```
CREATE TABLE teachers (
    teacher_id int  NOT NULL,
    CONSTRAINT teachers_pk PRIMARY KEY  (teacher_id)
);
```

## translators

zawiera tłumaczy

**klucz główny**: studies_id
**klucze obce**: translator_id (z *users.user_id*)
**Indeksy:** translator_id

translator_id - ID użytkownika który jest tłumaczem

```
CREATE TABLE translators (
    translator_id int  NOT NULL,
    CONSTRAINT translators_pk PRIMARY KEY  (translator_id)
);
```

## translators_languages

określa który tłumacz może tłumaczyć który język

**klucz główny**: translator_id, language_id
**klucze obce**: translator_id (z *translators*), language_id (z *languages*)
**Indeksy:** (translator_id, language_id)

translator_id - ID tłumacza
language_id - ID języka, który może tłumaczyć

```
CREATE TABLE translators_languages (
    translator_id int  NOT NULL,
    language_id int  NOT NULL,
    CONSTRAINT translators_languages_pk PRIMARY KEY  (translator_id,language_id)
);
```

## users

zawiera wszystkich użytkowników, ich dane logowania i podstawowe dane osobowe

**klucz główny**: user_id
**Indeksy:** user_id, email, phone_no, (firstname, lastname)
**warunki integralności:** email zawiera w sobie ‘@’ i jest unikalne, phone_no jest numeryczne lub jest zaczyna się od ‘+’ i jest numeryczne i jest unikalne

user_id - ID użytkownika
login - nazwa logowania użytkownika
password - zaszyfrowane hasło użytkownika
firstname - imię użytkownika
lastname - nazwisko użytkownika
email - adres e-mail użytkownika
phone_no - numer telefonu użytkownika

```
CREATE TABLE users (
    user_id int  NOT NULL,
    login varchar(255)  NOT NULL,
    password varchar(64)  NOT NULL,
    firstname varchar(255)  NOT NULL,
    lastname varchar(255)  NOT NULL,
    email varchar(255)  NOT NULL CHECK ([email] like '%@%'),
    phone_no nvarchar(16) NULL CHECK ([phone_no] IS NULL OR ISNUMERIC([phone_no]) = 1
                                      OR (LEFT([phone_no], 1) = '+'
                                      AND ISNUMERIC(SUBSTRING([phone_no], 2, LEN([phone_no]))) = 1)),
    CONSTRAINT phone UNIQUE NONCLUSTERED (phone_no),
    CONSTRAINT mail UNIQUE NONCLUSTERED (email),
    CONSTRAINT user_pk PRIMARY KEY  (user_id)
);
```

## webinars

zawiera webinary

**klucz główny**: meeting_id
**klucze obce**: meeting_id (z *meetings*), service_id (z *services*)
**Indeksy:** meeting_id, price, service_id

meeting_id - ID webinaru jako spotkania
service_id - ID webinaru jako usługi

webinar - nazwa webinaru
price - cena webinaru (0 znaczy że jest darmowy)
link - link do spotkania lub nagrania z webinaru

```
CREATE TABLE webinars (
    meeting_id int  NOT NULL,
    service_id int  NOT NULL,
    webinar varchar(255)  NOT NULL,
    price money  NOT NULL CHECK ([price] >= 0),
    link varchar(255)  NOT NULL,
    CONSTRAINT service_id UNIQUE (service_id),
    CONSTRAINT webinars_pk PRIMARY KEY  (meeting_id)
);
```

# REFERENCJE

```
-- foreign keys
-- Reference: FK_0 (table: studies)
ALTER TABLE studies ADD CONSTRAINT FK_0
    FOREIGN KEY (coordinator_id)
    REFERENCES coordinators (coordinator_id);

-- Reference: FK_1 (table: courses)
ALTER TABLE courses ADD CONSTRAINT FK_1
    FOREIGN KEY (coordinator_id)
    REFERENCES coordinators (coordinator_id);

-- Reference: FK_10 (table: teachers)
ALTER TABLE teachers ADD CONSTRAINT FK_10
    FOREIGN KEY (teacher_id)
    REFERENCES users (user_id);

-- Reference: FK_11 (table: online_meetings)
ALTER TABLE online_meetings ADD CONSTRAINT FK_11
    FOREIGN KEY (meeting_id)
    REFERENCES meetings (meeting_id);

-- Reference: FK_12 (table: stationary_meetings)
ALTER TABLE stationary_meetings ADD CONSTRAINT FK_12
    FOREIGN KEY (meeting_id)
    REFERENCES meetings (meeting_id);

-- Reference: FK_13 (table: stationary_meetings)
```

```sql
ALTER TABLE stationary_meetings ADD CONSTRAINT FK_13
    FOREIGN KEY (classroom_id)
    REFERENCES classrooms (classroom_id);

-- Reference: FK_14 (table: coordinators)
ALTER TABLE coordinators ADD CONSTRAINT FK_14
    FOREIGN KEY (coordinator_id)
    REFERENCES users (user_id);

-- Reference: FK_15 (table: students)
ALTER TABLE students ADD CONSTRAINT FK_15
    FOREIGN KEY (student_id)
    REFERENCES users (user_id);

-- Reference: FK_16 (table: orders)
ALTER TABLE orders ADD CONSTRAINT FK_16
    FOREIGN KEY (student_id)
    REFERENCES students (student_id);

-- Reference: FK_17 (table: orders_details)
ALTER TABLE orders_details ADD CONSTRAINT FK_17
    FOREIGN KEY (order_id)
    REFERENCES orders (order_id);

-- Reference: FK_18 (table: orders_details)
ALTER TABLE orders_details ADD CONSTRAINT FK_18
    FOREIGN KEY (service_id)
    REFERENCES services (service_id);

-- Reference: FK_19 (table: courses)
ALTER TABLE courses ADD CONSTRAINT FK_19
    FOREIGN KEY (category_id)
    REFERENCES categories (category_id);

-- Reference: FK_2 (table: meetings)
ALTER TABLE meetings ADD CONSTRAINT FK_2
    FOREIGN KEY (teacher_id)
    REFERENCES teachers (teacher_id);

-- Reference: FK_20 (table: meetings_translations)
ALTER TABLE meetings_translations ADD CONSTRAINT FK_20
    FOREIGN KEY (meeting_id)
    REFERENCES meetings (meeting_id);
```

```sql
-- Reference: FK_21 (table: meetings_translations)
ALTER TABLE meetings_translations ADD CONSTRAINT FK_21
    FOREIGN KEY (language_id)
    REFERENCES languages (language_id);

-- Reference: FK_22 (table: meetings_translations)
ALTER TABLE meetings_translations ADD CONSTRAINT FK_22
    FOREIGN KEY (translator_id)
    REFERENCES translators (translator_id);

-- Reference: FK_23 (table: online_meetings_asynchronic)
ALTER TABLE online_meetings_asynchronic ADD CONSTRAINT FK_23
    FOREIGN KEY (meeting_id)
    REFERENCES online_meetings (meeting_id);

-- Reference: FK_24 (table: online_meetings_synchronic)
ALTER TABLE online_meetings_synchronic ADD CONSTRAINT FK_24
    FOREIGN KEY (meeting_id)
    REFERENCES online_meetings (meeting_id);

-- Reference: FK_25 (table: studies_sessions)
ALTER TABLE studies_sessions ADD CONSTRAINT FK_25
    FOREIGN KEY (session_id)
    REFERENCES services (service_id);

-- Reference: FK_27 (table: studies_meeting)
ALTER TABLE studies_meeting ADD CONSTRAINT FK_27
    FOREIGN KEY (subject_id)
    REFERENCES syllabus (subject_id);

-- Reference: FK_28 (table: studies_meeting)
ALTER TABLE studies_meeting ADD CONSTRAINT FK_28
    FOREIGN KEY (meeting_id)
    REFERENCES meetings (meeting_id);

-- Reference: FK_29 (table: syllabus)
ALTER TABLE syllabus ADD CONSTRAINT FK_29
    FOREIGN KEY (coordinator_id)
    REFERENCES coordinators (coordinator_id);

-- Reference: FK_3 (table: meetings_attendance)
ALTER TABLE meetings_attendance ADD CONSTRAINT FK_3
```

```
    FOREIGN KEY (meeting_id)
    REFERENCES meetings (meeting_id);

-- Reference: FK_30 (table: syllabus)
ALTER TABLE syllabus ADD CONSTRAINT FK_30
    FOREIGN KEY (category_id)
    REFERENCES categories (category_id);

-- Reference: FK_31 (table: office_workers)
ALTER TABLE office_workers ADD CONSTRAINT FK_31
    FOREIGN KEY (employee_id)
    REFERENCES users (user_id);

-- Reference: FK_32 (table: headmaster)
ALTER TABLE headmaster ADD CONSTRAINT FK_32
    FOREIGN KEY (headmaster_id)
    REFERENCES users (user_id);

-- Reference: FK_33 (table: administrators)
ALTER TABLE administrators ADD CONSTRAINT FK_33
    FOREIGN KEY (administrator_id)
    REFERENCES users (user_id);

-- Reference: FK_34 (table: studies_sessions)
ALTER TABLE studies_sessions ADD CONSTRAINT FK_34
    FOREIGN KEY (studies_id)
    REFERENCES studies (studies_id);

-- Reference: FK_35 (table: course_editions)
ALTER TABLE course_editions ADD CONSTRAINT FK_35
    FOREIGN KEY (course_id)
    REFERENCES courses (course_id);

-- Reference: FK_36 (table: course_meetings)
ALTER TABLE course_meetings ADD CONSTRAINT FK_36
    FOREIGN KEY (service_id)
    REFERENCES course_editions (service_id);

-- Reference: FK_37 (table: course_editions)
ALTER TABLE course_editions ADD CONSTRAINT FK_37
    FOREIGN KEY (service_id)
    REFERENCES services (service_id);
```

```sql
-- Reference: FK_4 (table: meetings_attendance)
ALTER TABLE meetings_attendance ADD CONSTRAINT FK_4
    FOREIGN KEY (student_id)
    REFERENCES students (student_id);

-- Reference: FK_41 (table: cities)
ALTER TABLE cities ADD CONSTRAINT FK_41
    FOREIGN KEY (country_id)
    REFERENCES countries (country_id);

-- Reference: FK_44 (table: internships_attendance)
ALTER TABLE internships_attendance ADD CONSTRAINT FK_44
    FOREIGN KEY (internship_id)
    REFERENCES internships (internship_id);

-- Reference: FK_45 (table: webinars)
ALTER TABLE webinars ADD CONSTRAINT FK_45
    FOREIGN KEY (meeting_id)
    REFERENCES meetings (meeting_id);

-- Reference: FK_46 (table: webinars)
ALTER TABLE webinars ADD CONSTRAINT FK_46
    FOREIGN KEY (service_id)
    REFERENCES services (service_id);

-- Reference: FK_47 (table: studies_meeting)
ALTER TABLE studies_meeting ADD CONSTRAINT FK_47
    FOREIGN KEY (service_id)
    REFERENCES services (service_id);

-- Reference: FK_48 (table: internships_attendance)
ALTER TABLE internships_attendance ADD CONSTRAINT FK_48
    FOREIGN KEY (student_id)
    REFERENCES students (student_id);

-- Reference: FK_49 (table: studies_internships)
ALTER TABLE studies_internships ADD CONSTRAINT FK_49
    FOREIGN KEY (studies_id)
    REFERENCES studies (studies_id);

-- Reference: FK_5 (table: adresses)
ALTER TABLE adresses ADD CONSTRAINT FK_5
    FOREIGN KEY (student_id)
```

```
    REFERENCES students (student_id);

-- Reference: FK_50 (table: studies_internships)
ALTER TABLE studies_internships ADD CONSTRAINT FK_50
    FOREIGN KEY (internship_id)
    REFERENCES internships (internship_id);

-- Reference: FK_6 (table: translators)
ALTER TABLE translators ADD CONSTRAINT FK_6
    FOREIGN KEY (translator_id)
    REFERENCES users (user_id);

-- Reference: FK_7 (table: translators_languages)
ALTER TABLE translators_languages ADD CONSTRAINT FK_7
    FOREIGN KEY (translator_id)
    REFERENCES translators (translator_id);

-- Reference: FK_8 (table: course_meetings)
ALTER TABLE course_meetings ADD CONSTRAINT FK_8
    FOREIGN KEY (meeting_id)
    REFERENCES meetings (meeting_id);

-- Reference: FK_9 (table: translators_languages)
ALTER TABLE translators_languages ADD CONSTRAINT FK_9
    FOREIGN KEY (language_id)
    REFERENCES languages (language_id);

-- Reference: adresses_cities (table: adresses)
ALTER TABLE adresses ADD CONSTRAINT adresses_cities
    FOREIGN KEY (city_id)
    REFERENCES cities (city_id);
```

# WIDOKI

## Attendance_list (Ida C.)

Widok pokazujący obecność wszystkich studentów na poszczególnych spotkaniach

```
CREATE VIEW dbo.attendance_list AS
SELECT meetings_attendance.meeting_id, start_time AS date,
firstname + ' ' + lastname AS name, attendance
FROM meetings
        INNER JOIN meetings_attendance ON meetings.meeting_id =
meetings_attendance.meeting_id
        INNER JOIN students ON meetings_attendance.student_id =
students.student_id
        INNER JOIN users ON students.student_id = users.user_id
GO
```

## Attendance_raport (Ida C.)

Raport o ilości studentów na każdym z zajęć

```
CREATE VIEW dbo.attendance_report AS
SELECT meeting_id, SUM(IIF(attendance = 1, 1, 0)) AS attendance
FROM meetings_attendance
GROUP BY meeting_id
GO
```

## Bilocation_raport (JK)

Widok pokazujący raport bilokacji

```
CREATE VIEW dbo.bilocation_raport AS
WITH bilocs AS (SELECT m1.meeting_id AS meet1, m2.meeting_id AS
meet2
                FROM meetings AS m1
                        JOIN meetings AS m2 ON ((m1.meeting_id <
m2.meeting_id) AND
                                                ((m1.start_time <=
m2.start_time AND m2.start_time <= m1.end_time) OR
                                                 (m2.start_time <=
m1.start_time AND m1.start_time <= m2.end_time)))),
     studmeet AS (SELECT students.student_id, meetings.meeting_id
                  FROM students
                           INNER JOIN meetings_attendance ON
students.student_id = meetings_attendance.student_id
```

```
                            INNER JOIN meetings ON
meetings_attendance.meeting_id = meetings.meeting_id AND
                                                    start_time >
GETDATE())
SELECT s1.student_id, users.firstname + ' ' + users.lastname AS
student_name, meet1, meet2
FROM bilocs
        INNER JOIN studmeet AS s1 ON bilocs.meet1 = s1.meeting_id
        INNER JOIN studmeet AS s2 ON bilocs.meet2 = s2.meeting_id
        INNER JOIN users ON s1.student_id = users.user_id
WHERE s1.student_id = s2.student_id
GO
```

## Clients_statistics (IS)

Widok pokazujący raport z danymi studentów

```
CREATE VIEW dbo.client_statistics AS
SELECT u.firstname + ' ' + u.lastname                AS name,
      u.email,
      u.login,
      u.phone_no,
      countries.country_name,
      cities.city_name,
      adresses.zip_code,
      adresses.street,
      ((SELECT COUNT(orders.order_id)
        FROM orders
        WHERE u.user_id = orders.student_id))       AS
'order_count',
      ISNULL(((SELECT SUM(od.price)
               FROM orders o
                        INNER JOIN orders_details od ON
od.order_id = o.order_id
               WHERE u.user_id = o.student_id  AND o.paid = 1)), 0)
AS 'total_order_price'
FROM users u
        INNER JOIN students ON students.student_id = u.user_id
        INNER JOIN adresses ON adresses.student_id =
students.student_id
        INNER JOIN cities ON cities.city_id = adresses.city_id
```

```
        INNER JOIN countries ON countries.country_id =
cities.country_id
GO
```

## Courses_scheme(JK)

```
CREATE VIEW dbo.client_statistics AS
SELECT u.firstname + ' ' + u.lastname                AS name,
      u.email,
      u.login,
      u.phone_no,
      countries.country_name,
      cities.city_name,
      adresses.zip_code,
      adresses.street,
      ((SELECT COUNT(orders.order_id)
        FROM orders
        WHERE u.user_id = orders.student_id))       AS
'order_count',
      ISNULL(((SELECT SUM(od.price)
               FROM orders o
                        INNER JOIN orders_details od ON
od.order_id = o.order_id
               WHERE u.user_id = o.student_id)), 0) AS
'total_order_price'
FROM users u
        INNER JOIN students ON students.student_id = u.user_id
        INNER JOIN adresses ON adresses.student_id =
students.student_id
        INNER JOIN cities ON cities.city_id = adresses.city_id
        INNER JOIN countries ON countries.country_id =
cities.country_id
GO
```

## Debtors (IS)

Widok pokazujący raport dłużników, czyli osób które wpłaciły zaliczkę za kurs, ale nie opłaciły samego kursu który zaczyna się w ciągu trzech dni

```
CREATE VIEW debtors AS
SELECT u.firstname + ' ' + u.lastname AS 'Name'
FROM users u
        INNER JOIN orders o ON o.student_id = u.user_id AND
(o.paid = 1 OR o.deferred_payment = 1)
        INNER JOIN orders_details od ON od.order_id = o.order_id
```

```
        INNER JOIN course_editions ce ON
ce.advance_payment_service_id = od.service_id
        INNER JOIN course_meetings cm ON ce.service_id =
cm.service_id AND cm.module = 1
        INNER JOIN meetings m ON m.meeting_id = cm.meeting_id
WHERE ce.service_id NOT IN ((SELECT course_editions.service_id
                             FROM course_editions
                                      INNER JOIN orders_details
                                                 ON
orders_details.service_id = course_editions.service_id
                                      INNER JOIN orders ON
orders.order_id = orders_details.order_id
                             WHERE orders.student_id = u.user_id
AND (orders.paid = 1 OR orders.deferred_payment = 1)))
 AND DATEDIFF(DAY, GETDATE(), m.start_time) <= 3

GO
```

## Employees (JK)

```
CREATE VIEW dbo.employees AS
SELECT 'headmaster' AS rola, firstname + ' ' + lastname AS [imie i
nazwisko], email, phone_no AS telefon
FROM users
        INNER JOIN headmaster ON users.user_id =
headmaster.headmaster_id
UNION
SELECT 'administrator' AS rola, firstname + ' ' + lastname AS
[imie i nazwisko], email, phone_no AS telefon
FROM users
        INNER JOIN administrators ON users.user_id =
administrators.administrator_id
UNION
SELECT 'office worker' AS rola, firstname + ' ' + lastname AS
[imie i nazwisko], email, phone_no AS telefon
FROM users
        INNER JOIN office_workers ON users.user_id =
office_workers.employee_id
UNION
SELECT 'teacher' AS rola, firstname + ' ' + lastname AS [imie i
nazwisko], email, phone_no AS telefon
FROM users
        INNER JOIN teachers ON users.user_id = teachers.teacher_id
UNION
SELECT 'translator' AS rola, firstname + ' ' + lastname AS [imie i
nazwisko], email, phone_no AS telefon
FROM users
```

```
        INNER JOIN translators ON users.user_id =
translators.translator_id
UNION
SELECT 'coordinator' AS rola, firstname + ' ' + lastname AS [imie
i nazwisko], email, phone_no AS telefon
FROM users
        INNER JOIN coordinators ON users.user_id = coordinator_id
GO
```

## Enrolled_raport (Ida C.)

```
CREATE VIEW dbo.enrolled_raport AS
SELECT meetings_attendance.meeting_id,
      IIF(meetings_attendance.meeting_id IN (SELECT meeting_id
FROM stationary_meetings), 'stationary',
          'online') AS form,
      COUNT(*)      AS student_no
FROM meetings_attendance
        INNER JOIN meetings ON meetings_attendance.meeting_id =
meetings.meeting_id
WHERE start_time > GETDATE()
GROUP BY meetings_attendance.meeting_id
GO
```

## Financial_raport (Ida C.)

```
CREATE VIEW dbo.financial_raport AS
SELECT 'webinar' AS activity_type, webinar AS title,
SUM(orders_details.price) AS total_income
FROM webinars
        INNER JOIN services ON webinars.service_id =
services.service_id
        INNER JOIN orders_details ON services.service_id =
orders_details.service_id
        INNER JOIN orders ON orders_details.order_id =
orders.order_id
WHERE paid = 1
GROUP BY webinar
UNION
SELECT 'course', course_name, SUM(orders_details.price)
FROM courses
        INNER JOIN course_editions ON courses.course_id =
course_editions.course_id
```

```
        INNER JOIN services ON course_editions.service_id =
services.service_id
        INNER JOIN orders_details ON services.service_id =
orders_details.service_id
        INNER JOIN orders ON orders_details.order_id =
orders.order_id
WHERE paid = 1
GROUP BY course_name
UNION
SELECT 'studies', studies_name, SUM(orders_details.price)
FROM studies
        INNER JOIN studies_sessions ON studies.studies_id =
studies_sessions.studies_id
        INNER JOIN services ON studies_sessions.service_id =
services.service_id
        INNER JOIN orders_details ON services.service_id =
orders_details.service_id
        INNER JOIN orders ON orders_details.order_id =
orders.order_id
WHERE paid = 1
GROUP BY studies_name
GO
```

## Financial_raport_by_activity (Ida C.)

```
CREATE VIEW dbo.financial_raport_by_activity AS
SELECT activity_type, SUM(total_income) AS total_income
FROM financial_raport
GROUP BY activity_type
GO
```

## Free_webinars (JK)

Lista darmowych webinarów

```
CREATE VIEW dbo.free_webinars AS
SELECT webinars.webinar
FROM webinars
WHERE price = 0
GO
```

## Orders_raport(Ida C.)

```
CREATE VIEW dbo.orders_raport AS
SELECT orders.order_id, firstname + ' ' + lastname AS name,
service_id, price
```

```
FROM orders
        INNER JOIN orders_details ON orders.order_id =
orders_details.order_id
        INNER JOIN users ON orders.student_id = users.user_id
GO
```

## Orders_to_pay (IS)

Widok pokazujący listę zamówień które nie zostały jeszcze opłacone

```
CREATE VIEW orders_to_pay AS
SELECT u.user_id, orders.order_id, orders.order_date,
SUM(orders_details.price) AS 'order_price'
FROM users AS u
        INNER JOIN orders ON orders.student_id = u.user_id
        INNER JOIN orders_details ON orders_details.order_id =
orders.order_id
WHERE orders.paid = 0
 AND orders.deferred_payment = 0
GROUP BY u.user_id, orders.order_id, orders.order_date
GO
```

## Paid_webinar_access (IS)

Widok pokazujący studentów którzy mają dostęp na dany, płatny kurs

```
CREATE VIEW dbo.paid_webinar_access AS
SELECT w.webinar, u.firstname + ' ' + u.lastname AS name
FROM users u
        INNER JOIN orders o ON o.student_id = u.user_id
        INNER JOIN orders_details od ON od.order_id = o.order_id
        INNER JOIN webinars w ON w.service_id = od.service_id
WHERE (o.paid = 1 OR o.deferred_payment = 1)
 AND DATEDIFF(DAY, o.paid_date, GETDATE()) <= 30
GO
```

## Student_contact_info (JK)

Widok pokazujący dane adresowe studentów

```
CREATE VIEW dbo.student_contact_info AS
SELECT students.student_id,
      users.firstname + ' ' + users.lastname
AS student,
      adresses.street + ', ' + cities.city_name + ', ' +
country_name AS adres,
```

```
      users.phone_no
AS telefon,
      users.email
AS email
FROM students
        INNER JOIN users ON students.student_id = users.user_id
        INNER JOIN adresses ON students.student_id =
adresses.student_id
        INNER JOIN cities ON adresses.city_id = cities.city_id
        INNER JOIN countries ON cities.country_id =
countries.country_id
GO
```

## Studies_scheme (JK)

```
CREATE VIEW dbo.studies_scheme AS
SELECT studies_name,
      studies_sessions.session_id,
      subject_name,
      users.firstname + ' ' + users.lastname   AS teacher,
      meetings.start_time,
      ISNULL(language, 'polish')               AS language,
      users2.firstname + ' ' + users2.lastname AS translator,
      CAST(classroom_id AS varchar)            AS classroom
FROM studies
        INNER JOIN studies_sessions ON studies.studies_id =
studies_sessions.studies_id
        INNER JOIN studies_meeting ON studies_sessions.session_id
= studies_meeting.session_id
        INNER JOIN meetings ON studies_meeting.meeting_id =
meetings.meeting_id
        INNER JOIN syllabus ON studies.studies_id =
syllabus.studies_id
        INNER JOIN subjects ON syllabus.subject_id =
subjects.subject_id
        INNER JOIN teachers ON meetings.teacher_id =
teachers.teacher_id
        INNER JOIN users ON teachers.teacher_id = users.user_id
        LEFT OUTER JOIN meetings_translations ON
meetings.meeting_id = meetings_translations.meeting_id
        LEFT JOIN translators ON
meetings_translations.translator_id = translators.translator_id
        LEFT JOIN users AS users2 ON translators.translator_id =
users2.user_id
        LEFT JOIN languages ON meetings_translations.language_id =
languages.language_id
```

```
        INNER JOIN stationary_meetings ON meetings.meeting_id =
stationary_meetings.meeting_id
UNION
SELECT studies_name,
       studies_sessions.session_id,
       subject_name,
       users.firstname + ' ' + users.lastname   AS teacher,
       meetings.start_time,
       ISNULL(language, 'polish')               AS language,
       users2.firstname + ' ' + users2.lastname AS translator,
       online_meetings_synchronic.meeting_link  AS classroom
FROM studies
        INNER JOIN studies_sessions ON studies.studies_id =
studies_sessions.studies_id
        INNER JOIN studies_meeting ON studies_sessions.session_id
= studies_meeting.session_id
        INNER JOIN meetings ON studies_meeting.meeting_id =
meetings.meeting_id
        INNER JOIN syllabus ON studies.studies_id =
syllabus.studies_id
        INNER JOIN subjects ON syllabus.subject_id =
subjects.subject_id
        INNER JOIN teachers ON meetings.teacher_id =
teachers.teacher_id
        INNER JOIN users ON teachers.teacher_id = users.user_id
        LEFT OUTER JOIN meetings_translations ON
meetings.meeting_id = meetings_translations.meeting_id
        LEFT JOIN translators ON
meetings_translations.translator_id = translators.translator_id
        LEFT JOIN users AS users2 ON translators.translator_id =
users2.user_id
        LEFT JOIN languages ON meetings_translations.language_id =
languages.language_id
        INNER JOIN online_meetings ON meetings.meeting_id =
online_meetings.meeting_id
        INNER JOIN online_meetings_synchronic ON
online_meetings.meeting_id = online_meetings_synchronic.meeting_id
UNION
SELECT studies_name,
       studies_sessions.session_id,
       subject_name,
       users.firstname + ' ' + users.lastname   AS teacher,
       meetings.start_time,
       ISNULL(language, 'polish')               AS language,
       users2.firstname + ' ' + users2.lastname AS translator,
       online_meetings_asynchronic.meeting_link AS classroom
FROM studies
```

```
        INNER JOIN studies_sessions ON studies.studies_id =
studies_sessions.studies_id
        INNER JOIN studies_meeting ON studies_sessions.session_id
= studies_meeting.session_id
        INNER JOIN meetings ON studies_meeting.meeting_id =
meetings.meeting_id
        INNER JOIN syllabus ON studies.studies_id =
syllabus.studies_id
        INNER JOIN subjects ON syllabus.subject_id =
subjects.subject_id
        INNER JOIN teachers ON meetings.teacher_id =
teachers.teacher_id
        INNER JOIN users ON teachers.teacher_id = users.user_id
        LEFT OUTER JOIN meetings_translations ON
meetings.meeting_id = meetings_translations.meeting_id
        LEFT JOIN translators ON
meetings_translations.translator_id = translators.translator_id
        LEFT JOIN users AS users2 ON translators.translator_id =
users2.user_id
        LEFT JOIN languages ON meetings_translations.language_id =
languages.language_id
        INNER JOIN online_meetings ON meetings.meeting_id =
online_meetings.meeting_id
        INNER JOIN online_meetings_asynchronic
                    ON online_meetings.meeting_id =
online_meetings_asynchronic.meeting_id
GO
```

## Studies_info (JK)

Widok pokazujący informacje dotyczące poszczególnych studiów

```
CREATE VIEW dbo.studies_info AS
SELECT studies_name, users.firstname + ' ' + users.lastname AS
coordinator, advance_payment, student_limit
FROM studies
        INNER JOIN coordinators ON studies.coordinator_id =
coordinators.coordinator_id
        INNER JOIN users ON coordinators.coordinator_id =
users.user_id
GO
```

## Studies_syllabus (JK)

Lista przedmiotów prowadzonych na danych studiach wraz z koordynatorami przedmiotów

```
CREATE VIEW dbo.studies_syllabus AS
SELECT studies_name, users.firstname + ' ' + users.lastname AS
[koordynator przedmiotu], subject_name
FROM studies--,meetings.start_time from studies
        INNER JOIN studies_sessions ON studies.studies_id =
studies_sessions.studies_id
        INNER JOIN syllabus ON studies.studies_id =
syllabus.studies_id
        INNER JOIN subjects ON syllabus.subject_id =
subjects.subject_id
        INNER JOIN coordinators ON syllabus.coordinator_id =
coordinators.coordinator_id
        INNER JOIN users ON coordinators.coordinator_id =
users.user_id
GO
```

## Subjects_with_categories (JK)

```
CREATE VIEW dbo.subjects_with_categories AS
SELECT subject_name, category_name
FROM subjects
        INNER JOIN categories ON subjects.category_id =
categories.category_id
GO

CREATE VIEW dbo.total_price_by_orders AS
SELECT orders.order_id, lastname + ' ' + firstname AS name,
SUM(price) AS total_order_price
FROM orders
        INNER JOIN orders_details ON orders.order_id =
orders_details.order_id
        INNER JOIN users ON users.user_id = orders.student_id
GROUP BY orders.order_id, lastname + ' ' + firstname
GO
```

## total_price_by_students(Ida C.)

Widok pokazujący ile każdy student zapłacił łącznie za zamówienia

```
CREATE VIEW dbo.total_price_by_student AS
SELECT orders.student_id,
      firstname + ' ' + lastname AS student_name,
      orders.order_id,
      SUM(price)                 AS total_order_price
FROM orders
```

```
        INNER JOIN orders_details ON orders.order_id =
orders_details.order_id
        INNER JOIN students ON orders.student_id =
students.student_id
        INNER JOIN users ON students.student_id = users.user_id
GROUP BY orders.order_id, orders.student_id, firstname + ' ' +
lastname
GO
```

## Translation_language_raport (JK)

```
CREATE VIEW dbo.translation_language_raport AS
SELECT firstname + ' ' + lastname AS name, language
FROM translators
        INNER JOIN translators_languages ON
translators.translator_id = translators_languages.translator_id
        INNER JOIN languages ON translators_languages.language_id
= languages.language_id
        INNER JOIN users ON translators.translator_id =
users.user_id
GO
```

## Translators_schedue (JK)

```
CREATE VIEW dbo.translators_schedule AS
SELECT users.firstname + ' ' + users.lastname            AS
translator,
      language,
      meetings.meeting_id,
      meetings.start_time,
      CAST(stationary_meetings.classroom_id AS varchar) AS
[classroom/link]
FROM users
        INNER JOIN translators ON users.user_id =
translators.translator_id
        INNER JOIN meetings_translations ON
translators.translator_id = meetings_translations.translator_id
        INNER JOIN meetings ON meetings_translations.meeting_id =
meetings.meeting_id
        INNER JOIN languages ON meetings_translations.language_id
= languages.language_id
        INNER JOIN stationary_meetings ON meetings.meeting_id =
stationary_meetings.meeting_id
UNION
SELECT users.firstname + ' ' + users.lastname   AS translator,
      language,
      meetings.meeting_id,
```

```
      meetings.start_time,
      online_meetings_asynchronic.meeting_link AS classroom
FROM users
        INNER JOIN translators ON users.user_id =
translators.translator_id
        INNER JOIN meetings_translations ON
translators.translator_id = meetings_translations.translator_id
        INNER JOIN meetings ON meetings_translations.meeting_id =
meetings.meeting_id
        INNER JOIN languages ON meetings_translations.language_id
= languages.language_id
        INNER JOIN online_meetings ON meetings.meeting_id =
online_meetings.meeting_id
        INNER JOIN online_meetings_asynchronic
                   ON online_meetings.meeting_id =
online_meetings_asynchronic.meeting_id
UNION
SELECT users.firstname + ' ' + users.lastname  AS translator,
      language,
      meetings.meeting_id,
      meetings.start_time,
      online_meetings_synchronic.meeting_link AS classroom
FROM users
        INNER JOIN translators ON users.user_id =
translators.translator_id
        INNER JOIN meetings_translations ON
translators.translator_id = meetings_translations.translator_id
        INNER JOIN meetings ON meetings_translations.meeting_id =
meetings.meeting_id
        INNER JOIN languages ON meetings_translations.language_id
= languages.language_id
        INNER JOIN online_meetings ON meetings.meeting_id =
online_meetings.meeting_id
        INNER JOIN online_meetings_synchronic ON
online_meetings.meeting_id = online_meetings_synchronic.meeting_id
GO
```

## Webinar_scheme(JK)

```
CREATE VIEW dbo.webinar_scheme AS
SELECT webinar                                  AS webinar_name,
      start_time,
      users.firstname + ' ' + users.lastname    AS teacher,
      ISNULL(language, 'polish')                AS language,
      users2.firstname + ' ' + users2.lastname AS translator,
      link
FROM webinars
```

```
        INNER JOIN meetings ON webinars.meeting_id =
meetings.meeting_id
        INNER JOIN online_meetings ON meetings.meeting_id =
online_meetings.meeting_id
        INNER JOIN teachers ON meetings.teacher_id =
teachers.teacher_id
        INNER JOIN users ON teachers.teacher_id = users.user_id
        LEFT OUTER JOIN meetings_translations ON
meetings.meeting_id = meetings_translations.meeting_id
        LEFT JOIN translators ON
meetings_translations.translator_id = translators.translator_id
        LEFT JOIN users AS users2 ON translators.translator_id =
users2.user_id
        LEFT JOIN languages ON meetings_translations.language_id =
languages.language_id
GO
```

## total_price_by_orders(Ida C.)

Widok pokazujący łączną cenę każdego z zamówień

```
CREATE VIEW dbo.attendance_report AS
SELECT meeting_id, SUM(IIF(attendance = 1, 1, 0)) AS attendance
FROM meetings_attendance
GROUP BY meeting_id
GO
```

## Course_grades (IS)

Widok pokazujący frekwencję studenta na danym kursie, oraz stan zaliczenia

```
CREATE VIEW course_grades AS
SELECT u.user_id, u.firstname+' '+u.lastname AS 'name',
   ce.service_id AS 'course_edition', (COUNT(ma.attendance) * 100 / ce.module_no) AS
'attendance %',
   CASE WHEN ((COUNT(ma.attendance) * 1.0 / ce.module_no) >= 0.8) THEN 'True' ELSE
'False' END AS 'passed'
FROM users u
INNER JOIN orders o ON o.student_id = u.user_id AND (o.paid = 1 OR o.deferred_payment
= 1)
INNER JOIN orders_details od ON od.order_id = o.order_id
INNER JOIN course_editions ce ON ce.service_id = od.service_id
INNER JOIN course_meetings cm ON cm.service_id = ce.service_id
INNER JOIN meetings m ON m.meeting_id = cm.meeting_id
INNER JOIN meetings_attendance ma ON ma.meeting_id = m.meeting_id AND
ma.student_id = u.user_id
WHERE ma.attendance = 1
```

```
GROUP BY u.user_id, u.firstname, u.lastname, ce.module_no, ce.service_id
```

## Studies_grades (IS)

Widok pokazujący frekwencję studenta na danym studium, ocenę, stan zaliczenia praktyk i stan zaliczenia tego studia

```
CREATE VIEW studies_grades AS
SELECT
	user_id,
	name,
	attendance_percentage,
   CASE
	WHEN attendance_percentage >= 90 THEN 5
	WHEN attendance_percentage >= 70 THEN 4
	WHEN attendance_percentage >= 50 THEN 3
	WHEN attendance_percentage >= 0 THEN 2
	END AS 'grade',
	internship_passed,
	CASE WHEN internship_passed = 1 AND attendance_percentage >= 80 THEN
'True' ELSE 'False' END AS 'passed'
FROM (
	SELECT
	u.user_id,
	u.firstname + ' ' + u.lastname AS 'name',
	CAST (COUNT(CASE WHEN ma.attendance = 1 THEN m.meeting_id END) * 100.0
/ COUNT(m.meeting_id) AS int) AS 'attendance_percentage',
	ISNULL ((SELECT 1
		FROM internships_attendance ia
		INNER JOIN internships ON internships.internship_id = ia.internship_id
		INNER JOIN studies_internships ON studies_internships.internship_id =
internships.internship_id
		WHERE s.studies_id = studies_internships.studies_id AND ia.student_id =
u.user_id AND ia.attendance = 1
	), 0) AS 'internship_passed'
	FROM
	users u
	INNER JOIN orders o ON o.student_id = u.user_id AND o.paid = 1 OR
o.deferred_payment = 1
	INNER JOIN orders_details od ON od.order_id = o.order_id
	INNER JOIN studies s ON s.service_id = od.service_id
	INNER JOIN studies_sessions ss ON ss.studies_id = s.studies_id
	INNER JOIN studies_meeting sm ON sm.session_id = ss.session_id
	INNER JOIN meetings m ON m.meeting_id = sm.meeting_id
	INNER JOIN meetings_attendance ma ON ma.meeting_id = m.meeting_id AND
ma.student_id = u.user_id
	GROUP BY
	u.user_id, u.firstname, u.lastname, s.studies_id
) AS subquery;
```

# Procedury i funkcje

## Add_country (Ida C.)

```
CREATE PROCEDURE add_country(@country_name varchar(255))
AS
   SET NOCOUNT ON
   IF EXISTS(SELECT *
             FROM countries
             WHERE country_name = @country_name)
       BEGIN
           THROW 50000, 'There is already that country in the
database',1
       END
DECLARE @country_id int
SELECT @country_id = ISNULL(MAX(country_id), 0) + 1
FROM countries
INSERT INTO countries(country_id, country_name)
VALUES (@country_id, @country_name)
GO
```

### Add_course

```
CREATE PROCEDURE add_course(@category_id int, @coordinator_id int,
@course_name varchar(255),
                            @advance_payment money, @price
money)
   AS
       SET NOCOUNT ON
       IF NOT EXISTS(SELECT *
                     FROM categories
                     WHERE category_id = @category_id)
           BEGIN
               THROW 50000, 'This category does not exists in
database',1
           END
       IF NOT EXISTS(SELECT *
                     FROM coordinators
                     WHERE coordinator_id = @coordinator_id)
           BEGIN
               THROW 50000, 'This coordinator does not exists in
database',1
           END

   DECLARE @course_id int
   SELECT @course_id = ISNULL(MAX(course_id), 0) + 1
   FROM courses
```

```
   INSERT INTO courses(course_id, category_id, coordinator_id,
course_name, advance_payment, price)
   VALUES (@course_id, @category_id, @coordinator_id,
@course_name, @advance_payment, @price)
GO
```

## Add_course_edition (Ida C)

```
CREATE PROCEDURE add_course_edition(@course_id int, @student_limit
int, @module_no int)
   AS
       SET NOCOUNT ON
       IF NOT EXISTS(SELECT *
                     FROM courses
                     WHERE course_id = @course_id)
           BEGIN
               THROW 50000, 'This course does not exists in
database',1
           END
   DECLARE @service_id int
   SELECT @service_id = ISNULL(MAX(service_id), 0) + 1
   FROM services
   DECLARE @advance_payment_service_id int
   SELECT @advance_payment_service_id = ISNULL(MAX(service_id), 0)
+ 2
   FROM services
   INSERT INTO course_editions(service_id, course_id,
student_limit, module_no, advance_payment_service_id)
   VALUES (@service_id, @course_id, @student_limit, @module_no,
@advance_payment_service_id)
GO
```

## Add_language (JK)

```
CREATE PROCEDURE add_language(@language varchar(255))
AS
   SET NOCOUNT ON
   IF EXISTS(SELECT *
             FROM languages
             WHERE language = @language)
       BEGIN
           THROW 50000, 'There is already that language in the
database',1
       END
```

```
DECLARE @language_id int
SELECT @language_id = ISNULL(MAX(language_id), 0) + 1
FROM languages
INSERT INTO languages(language_id, language)
VALUES (@language_id, @language)
GO
```

## Add_language_to_translator (JK)

```
CREATE PROCEDURE add_language_to_translator(@translator_id int,
@language_id int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM translators
                 WHERE translator_id = @translator_id)
       BEGIN
           THROW 50000, 'This translator does not exists',1
       END
   IF NOT EXISTS(SELECT *
                 FROM languages
                 WHERE language_id = @language_id)
       BEGIN
           THROW 50000, 'This language does not exists',1
       END
INSERT INTO translators_languages(translator_id, language_id)
VALUES (@translator_id, @language_id)
GO
```

## Add_meeting (Ida C.)

```
CREATE PROCEDURE add_meeting(@teacher_id int, @start_time datetime,
@end_time datetime)
       AS
           SET NOCOUNT ON
           IF NOT EXISTS(SELECT *
                         FROM teachers
                         WHERE teacher_id = @teacher_id)
               BEGIN
                   THROW 50000, 'This teacher does not exists in
database',1
               END
           IF @start_time > @end_time
               BEGIN
                   THROW 50000, 'Wrong date',1
```

```
                END

        DECLARE @meeting_id int
        SELECT @meeting_id = ISNULL(MAX(meeting_id), 0) + 1
        FROM meetings
        INSERT INTO meetings(meeting_id, teacher_id, start_time,
end_time)
        VALUES (@meeting_id, @teacher_id, @start_time, @end_time)
GO
```

## Add_meeting_to_course (Ida C.)

```
CREATE PROCEDURE add_meeting_to_course(@meeting_id int, @service_id
int, @module int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM meetings
                 WHERE meeting_id = @meeting_id)
        BEGIN
            THROW 50000, 'This meeting does not exists in
database',1
        END
   IF NOT EXISTS(SELECT *
                 FROM services
                 WHERE service_id = @service_id)
        BEGIN
            THROW 50000, 'This service does not exists in
database',1
        END
   IF @module > (SELECT module_no
                 FROM course_editions
                 WHERE service_id = @service_id)
        BEGIN
            THROW 50000, 'This course does not have that many
modules',1
        END
   IF @module < 0
        BEGIN
            THROW 50000, 'That module does not exist',1
        END
INSERT INTO course_meetings(meeting_id, service_id, module)
VALUES (@meeting_id, @service_id, @module)
GO
```

## Add_meeting_to_session (Ida C.)

```
CREATE PROCEDURE add_meeting_to_session(@meeting_id int,
@session_id int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM meetings
                  WHERE meeting_id = @meeting_id)
       BEGIN
           THROW 50000, 'This meeting does not exists in
database',1
       END
   IF NOT EXISTS(SELECT *
                 FROM studies_sessions
                  WHERE session_id = @session_id)
       BEGIN
           THROW 50000, 'This session does not exists in
database',1
       END

INSERT INTO studies_meeting(meeting_id, session_id)
VALUES (@meeting_id, @session_id)
GO
```

## Add_online_meeting_asynchronic (Ida C.)

```
CREATE PROCEDURE add_online_meeting_asynchronic(@teacher_id int,
@start_time datetime, @end_time datetime,
                                          @meeting_link
varchar(255),@video_link varchar(255))
AS
   SET NOCOUNT ON

DECLARE @meeting_id int
SELECT @meeting_id = ISNULL(MAX(meeting_id), 0) + 1
FROM meetings
   EXEC add_meeting @teacher_id, @start_time, @end_time
INSERT INTO
online_meetings_asynchronic(meeting_id,meeting_link,video_link)
VALUES (@meeting_id, @meeting_link,@video_link)
INSERT INTO online_meetings(meeting_id)
VALUES (@meeting_id)
GO
```

## Add_meeting_synchronic (Ida C.)

```
CREATE PROCEDURE add_online_meeting_synchronic(@teacher_id int,
@start_time datetime, @end_time datetime,
                                          @meeting_link varchar(255))
AS
   SET NOCOUNT ON

DECLARE @meeting_id int
SELECT @meeting_id = ISNULL(MAX(meeting_id), 0) + 1
FROM meetings
   EXEC add_meeting @teacher_id, @start_time, @end_time
INSERT INTO online_meetings_synchronic(meeting_id,meeting_link)
VALUES (@meeting_id, @meeting_link)
INSERT INTO online_meetings(meeting_id)
VALUES (@meeting_id)
GO
```

## Add_session_to_studies (Ida C.)

```
CREATE PROCEDURE add_session_to_studies(@studies_id int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM studies
                 WHERE studies_id = @studies_id)
       BEGIN
           THROW 50000, 'This studies does not exists in
database',1
       END
DECLARE @session_id int
SELECT @session_id = ISNULL(MAX(session_id), 0) + 1
FROM studies_sessions
DECLARE @service_id int
SELECT @service_id = ISNULL(MAX(service_id), 0) + 1
FROM services
INSERT INTO studies_sessions(session_id, service_id, studies_id)
VALUES (@session_id, @service_id, @studies_id)
INSERT INTO services
VALUES (@service_id)
GO
```

## Add_stationary_meeting (Ida C.)

```
CREATE PROCEDURE add_stationary_meeting(@teacher_id int,
@start_time datetime, @end_time datetime,
```

```
                                                    @classroom_id
int)
            AS
                SET NOCOUNT ON
                IF NOT EXISTS(SELECT *
                              FROM classrooms
                               WHERE classroom_id = @classroom_id)
                    BEGIN
                        THROW 50000, 'This classroom does not
exists in database',1
                    END

            DECLARE @meeting_id int
            SELECT @meeting_id = ISNULL(MAX(meeting_id), 0) + 1
            FROM meetings
                EXEC add_meeting @teacher_id, @start_time, @end_time
            INSERT INTO stationary_meetings(meeting_id,
classroom_id)
            VALUES (@meeting_id, @classroom_id)
GO
```

## Add_studies (Ida C.)

```
CREATE PROCEDURE add_studies(@coordinator_id int, @studies_name
varchar(255), @advance_payment money,
                             @student_limit int,
@student_session_price money, @not_student_session_price money)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM coordinators
                  WHERE coordinator_id = @coordinator_id)
       BEGIN
           THROW 50000, 'This coordinator does not exists in
database',1
       END

DECLARE @studies_id int
SELECT @studies_id = ISNULL(MAX(studies_id), 0) + 1
FROM studies
DECLARE @service_id int
SELECT @service_id = ISNULL(MAX(service_id), 0) + 1
FROM services
INSERT INTO studies(studies_id, service_id, coordinator_id,
studies_name, advance_payment, student_limit,
                    student_session_price,
not_student_session_price)
```

```
VALUES (@studies_id, @service_id, @coordinator_id, @studies_name,
@advance_payment, @student_limit,
       @student_session_price, @not_student_session_price)
INSERT INTO services(service_id)
VALUES (@service_id)
go
```

## Add_to_order (Ida C.)

```
CREATE PROCEDURE add_to_order(@order_id int,@service_id int)
   AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM orders
                 WHERE order_id = @order_id)
       BEGIN
           THROW 50000, 'This order does not exists in database',1
       END
           IF NOT EXISTS(SELECT *
                 FROM services
                 WHERE service_id = @service_id)
       BEGIN
           THROW 50000, 'This service does not exists in
database',1
       END

       DECLARE @price MONEY
       DECLARE @student_id INT
       SELECT @student_id = student_id from orders where order_id
= @order_id
   SELECT @price = dbo.price_of_service(@service_id,@student_id)
   INSERT INTO orders_details(order_id, service_id, price)
   VALUES (@order_id,@service_id,@price)
GO
```

## Add_to_syllabus (Ida C.)

```
CREATE PROCEDURE add_to_syllabus(@studies_id int, @subject_id int,
@coordinator_id int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM studies
                 WHERE studies_id = @studies_id)
```

```
        BEGIN
            THROW 50000, 'This studies does not exists in
database',1
        END
    IF NOT EXISTS(SELECT *
                  FROM subjects
                  WHERE subject_id = @subject_id)
        BEGIN
            THROW 50000, 'This subject does not exists in
database',1
        END
    IF NOT EXISTS(SELECT *
                  FROM coordinators
                  WHERE coordinator_id = @coordinator_id)
        BEGIN
            THROW 50000, 'This coordinator does not exists in
database',1
        END

DECLARE @syllabus_id int
SELECT @syllabus_id = ISNULL(MAX(syllabus_id), 0) + 1
FROM syllabus
INSERT INTO syllabus(syllabus_id, studies_id, subject_id,
coordinator_id)
VALUES (@syllabus_id, @studies_id, @subject_id, @coordinator_id)
go
```

## Add_user ( Ida C.)

```
CREATE PROCEDURE add_user(@login varchar(255), @password
varchar(64),
                          @firstname varchar(255), @lastname
varchar(255), @email varchar(255), @phone_no varchar(16))
AS
    SET NOCOUNT ON
    IF EXISTS(SELECT *
              FROM users
              WHERE login = @login)
        BEGIN
            THROW 50000, 'This login already exists in database',1
        END
    IF EXISTS(SELECT *
              FROM users
              WHERE password = @password)
        BEGIN
            THROW 50000, 'This password already exists in
database',1
```

```
        END
    IF EXISTS(SELECT *
              FROM users
              WHERE email = @email)
        BEGIN
            THROW 50000, 'This email already exists in database',1
        END
    IF EXISTS(SELECT *
              FROM users
              WHERE phone_no = @phone_no)
        BEGIN
            THROW 50000, 'This phone number already exists in
database',1
        END
DECLARE @user_id INT
SELECT @user_id = ISNULL(MAX(user_id), 0) + 1
FROM users
INSERT INTO users(user_id, login, password, firstname, lastname,
email, phone_no)
VALUES (@user_id, @login, @password, @firstname, @lastname,
@email, @phone_no);
go
```

## Add_webinar (Ida C.)

```
CREATE PROCEDURE add_webinar(@webinar_name varchar(255), @price
money, @link varchar(255), @teacher_id int,
                             @start_time datetime, @end_time
datetime, @meeting_link varchar(255),
                             @video_link varchar(255))
AS
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT *
                  FROM teachers
                  WHERE teacher_id = @teacher_id)
        BEGIN
            THROW 50000, 'This teacher does not exists in
database',1
        END
    IF @start_time > @end_time
        BEGIN
            THROW 50000, 'Wrong date',1
        END

DECLARE @meeting_id int
SELECT @meeting_id = ISNULL(MAX(meeting_id), 0) + 1
FROM meetings
```

```
DECLARE @service_id int
SELECT @service_id = ISNULL(MAX(service_id), 0) + 1
FROM services
INSERT INTO meetings(meeting_id, teacher_id, start_time, end_time)
VALUES (@meeting_id, @teacher_id, @start_time, @end_time)
INSERT INTO services(service_id)
VALUES (@service_id)
INSERT INTO webinars(meeting_id, service_id, webinar, price, link)
VALUES (@meeting_id, @service_id, @webinar_name, @price, @link)
INSERT INTO online_meetings(meeting_id)
VALUES (@meeting_id)
INSERT INTO online_meetings_asynchronic(meeting_id, meeting_link,
video_link)
VALUES (@meeting_id, @meeting_link, @video_link)
go
```

## Change_order_status (JK)

```
CREATE PROCEDURE change_order_status(@order_id int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM orders
                 WHERE order_id = @order_id)
       BEGIN
           THROW 50000, 'This order does not exists in database',1
       END
DECLARE @paid bit = 1
DECLARE @paid_date datetime = GETDATE()
BEGIN
   UPDATE orders
   SET paid = @paid
   WHERE orders.order_id = @order_id
   UPDATE orders
   SET paid_date = @paid_date
   WHERE orders.order_id = @order_id
END
go
```

## Change_student_attendance (JK)

```
CREATE PROCEDURE change_student_attendance(@meeting_id int,
@student_id int, @attendance bit)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
```

```
                FROM meetings_attendance
                WHERE meeting_id = @meeting_id
                  AND student_id = @student_id)
        BEGIN
            THROW 50000, 'This attendance does not exist',1
        END
    IF NOT EXISTS(SELECT *
                FROM meetings
                WHERE meeting_id = @meeting_id)
        BEGIN
            THROW 50000, 'This meeting does not exists in
database',1
        END
    IF NOT EXISTS(SELECT *
                FROM students
                WHERE student_id = @student_id)
        BEGIN
            THROW 50000, 'This teacher does not exists in
database',1
        END
BEGIN
    UPDATE meetings_attendance
    SET attendance = @attendance
    WHERE meetings_attendance.student_id = @student_id
      AND meetings_attendance.meeting_id = @meeting_id
END
go
```

## Make_user_a_translator (IS)

```
CREATE PROCEDURE make_user_a_coordinator(@user_id int)
AS
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT *
                FROM users
                WHERE user_id = @user_id)
        BEGIN
            THROW 50000, 'This user does not exists',1
        END
INSERT INTO coordinators(coordinator_id)
VALUES (@user_id)
go
```

## Make_user_a_headmaster (IS)

```
CREATE PROCEDURE make_user_a_headmaster(@user_id int)
AS
```

```
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM users
                 WHERE user_id = @user_id)
       BEGIN
           THROW 50000, 'This user does not exists',1
       END
INSERT INTO headmaster(headmaster_id)
VALUES (@user_id)
go
```

## Make_user_a_student (IS)

```
CREATE PROCEDURE make_user_a_student(@user_id int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM users
                 WHERE user_id = @user_id)
       BEGIN
           THROW 50000, 'This user does not exists',1
       END
INSERT INTO students(student_id)
VALUES (@user_id)
go
```

## Make_user_a_teacher (IS)

```
CREATE PROCEDURE make_user_a_teacher(@user_id int)
AS
   SET NOCOUNT ON
   IF NOT EXISTS(SELECT *
                 FROM users
                 WHERE user_id = @user_id)
       BEGIN
           THROW 50000, 'This user does not exists',1
       END
INSERT INTO teachers(teacher_id)
VALUES (@user_id)
go
```

## Make_user_a_translator (IS)

```
CREATE PROCEDURE make_user_a_translator(@user_id int)
AS
```

```
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT *
                  FROM users
                  WHERE user_id = @user_id)
        BEGIN
            THROW 50000, 'This user does not exists',1
        END
INSERT INTO translators(translator_id)
VALUES (@user_id)
Go
```

## Make_user_an_administrator (IS)

```
CREATE PROCEDURE make_user_an_administrator(@user_id int)
AS
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT *
                  FROM users
                  WHERE user_id = @user_id)
        BEGIN
            THROW 50000, 'This user does not exists',1
        END
INSERT INTO administrators(administrator_id)
VALUES (@user_id)
go
```

## Place_order (Ida C.)

```
CREATE PROCEDURE place_order(@student_id int, @deferred_payment
bit)
AS
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT *
                  FROM students
                  WHERE student_id = @student_id)
        BEGIN
            THROW 50000, 'This student does not exists in
database',1
        END
DECLARE @order_id int
SELECT @order_id = ISNULL(MAX(order_id), 0) + 1
FROM orders
DECLARE @paid bit = 0
DECLARE @order_date datetime = GETDATE()
DECLARE @paid_date datetime = NULL
```

```
INSERT INTO orders(order_id, student_id, order_date, paid,
paid_date, deferred_payment)
VALUES (@order_id, @student_id, @order_date, @paid, @paid_date,
@deferred_payment)
go
```

## Price_of_service (Ida C.)

```
CREATE FUNCTION price_of_service(@service_id INT, @user_id INT)
   RETURNS MONEY
AS
BEGIN
   DECLARE @price MONEY;
   IF @service_id IN (SELECT service_id FROM webinars)
       BEGIN
           SELECT @price = price
           FROM webinars
           WHERE service_id = @service_id;
       END
   IF @service_id IN (SELECT service_id FROM course_editions)
       BEGIN
           SELECT @price = price
           FROM courses
           WHERE course_id = (SELECT course_id FROM
course_editions WHERE service_id = @service_id)
       END
   IF @service_id IN (SELECT advance_payment_service_id FROM
course_editions)
       BEGIN
           SELECT @price = advance_payment
           FROM courses
           WHERE course_id = (SELECT course_id FROM
course_editions WHERE service_id = @service_id)
       END
   IF @service_id IN (SELECT service_id FROM studies)
       BEGIN
           SELECT @price = advance_payment FROM studies WHERE
service_id = @service_id
       END
   IF @service_id IN (SELECT service_id FROM studies_sessions
WHERE service_id = @service_id)
       BEGIN
           IF @user_id IN (SELECT student_id FROM students)
               BEGIN
                   SELECT @price = student_session_price
                   FROM studies
                   WHERE studies_id IN
```

```
                        (SELECT studies_id FROM studies_sessions
WHERE studies_sessions.service_id = @service_id)
                END
            IF @user_id NOT IN (SELECT student_id FROM students)
                BEGIN
                    SELECT @price = not_student_session_price
                    FROM studies
                    WHERE studies_id IN
                        (SELECT studies_id FROM studies_sessions
WHERE studies_sessions.service_id = @service_id)
                END
        END
    RETURN @price;
END
GO
```

## Show_basket_content(IS)

```
CREATE PROCEDURE show_basket_content(@student_id int)
AS
    SET NOCOUNT ON
    IF NOT EXISTS(SELECT *
                  FROM students
                   WHERE student_id = @student_id)
        BEGIN
            THROW 50000, 'This student does not exists',1
        END

    SELECT service_id,price from orders
INNER JOIN orders_details on orders.order_id =
orders_details.order_id
WHERE student_id=@student_id and paid = 0
GO
```

## Show_course_schedule (Ida C.)

```
create procedure show_course_schedule @course_name nvarchar(64)
as
select * from courses_scheme where course_name like @course_name
Go
```

## Show_student_attendance (Ida C.)

```
CREATE PROCEDURE show_student_attendance @student_name varchar(255)
AS
SELECT meeting_id, date, attendance
FROM attendance_list
WHERE name = @student_name
Go
```

## Show_student_contact (Ida C.)

```
create procedure show_student_contact @student_name nvarchar(64)
as
select * from student_contact_info where
student_contact_info.student like @student_name
go
```

## Show_student_orders (Ida C.)

```
CREATE procedure show_student_orders @student_name varchar(255)
   as select order_id, total_order_price from
total_price_by_orders where name = @student_name
go
```

## Show_student_paid_webinar_access (Ida C.)

```
create procedure show_student_paid_webinar_access @student_name
varchar(255)
as select webinar from paid_webinar_access where name =
@student_name
go
```

## Show_studies_syllabus (JK)

```
create procedure show_studies_syllabus @studies_name varchar(255)
   as select * from studies_syllabus where studies_name like
@studies_name
go
```

## Show_study_schedule (JK)

```
CREATE procedure show_study_schedule @studies nvarchar(64)
as
```

```
select * from studies_scheme where studies_name like @studies
Go
```

## Show_translator_languages (JK)

```
create procedure show_translator_languages @translator
nvarchar(255)
   as select language from translation_language_raport where name
= @translator
Go
```

## Show_translator_schedule (JK)

```
CREATE procedure show_translator_schedule @translator varchar(64)
as
select * from translators_schedule where translator like
@translator
go
```

# Triggery

## Before_insert_order (Igor S.)

Zapewnia że data i czas opłacenia zamówienia następuje po złożeniu zamówienia

```
CREATE TRIGGER before_insert_order
ON orders
INSTEAD OF INSERT
AS
BEGIN
        IF EXISTS (SELECT 1 FROM inserted WHERE order_date > paid_date)
        BEGIN
        RAISERROR('order_date must be less than or equal to paid_date.', 16, 1);
        END;
        INSERT INTO orders SELECT * FROM inserted;
END;
```

## Before_insert_meeting (Igor S.)

Zapewnia że data i czas zakończenia spotkania następuje po rozpoczęciu spotkania

```
CREATE TRIGGER before_insert_meeting
ON meetings
```

```
INSTEAD OF INSERT
AS
BEGIN
   IF EXISTS (SELECT 1 FROM inserted WHERE start_time >= end_time)
        BEGIN
        RAISERROR('start_time must be less than end_time.', 16, 1);
        END;

INSERT INTO meetings SELECT * FROM inserted;
END;
```

## Before_insert_course_meeting (Igor S.)

Zapewnia że żadne dwa spotkania w kursie nie kolidują ze sobą czasowo

```
CREATE TRIGGER [dbo].[before_insert_course_meeting]
ON [dbo].[course_meetings]
INSTEAD OF INSERT
AS
BEGIN
   IF EXISTS (
        SELECT 1
        FROM INSERTED i
        INNER JOIN course_meetings cm ON cm.service_id = i.service_id
        INNER JOIN meetings m ON cm.meeting_id = m.meeting_id
        WHERE m.meeting_id <> i.meeting_id
                AND ((SELECT i_m.start_time FROM meetings i_m WHERE i_m.meeting_id
= i.meeting_id) BETWEEN m.start_time AND m.end_time
                OR (SELECT i_m.end_time FROM meetings i_m WHERE i_m.meeting_id =
i.meeting_id) BETWEEN m.start_time AND m.end_time
                OR m.start_time BETWEEN (SELECT i_m.start_time FROM meetings i_m
WHERE i_m.meeting_id = i.meeting_id) AND (SELECT i_m.end_time FROM meetings i_m
WHERE i_m.meeting_id = i.meeting_id))

        )
        BEGIN
        RAISERROR('Another meeting in the course collides timewise.', 16, 1);
        END;

   INSERT INTO course_meetings SELECT * FROM inserted;
END;
```

## Before_insert_studies_meeting (Igor S.)

Zapewnia że żadne dwa spotkania w studium nie kolidują ze sobą czasowo

```
CREATE TRIGGER [dbo].[before_insert_studies_meeting]
ON [dbo].[studies_meeting]
INSTEAD OF INSERT
AS
BEGIN
   IF EXISTS (
         SELECT 1
         FROM INSERTED i
          INNER JOIN studies_sessions ss0 ON i.session_id = ss0.session_id
          INNER JOIN studies s ON s.studies_id = ss0.studies_id
          INNER JOIN studies_sessions ss ON ss.studies_id = s.studies_id
         INNER JOIN studies_meeting sm ON sm.session_id = ss.session_id
         INNER JOIN meetings m ON sm.meeting_id = m.meeting_id
         WHERE m.meeting_id <> i.meeting_id
                  AND ((SELECT i_m.start_time FROM meetings i_m WHERE i_m.meeting_id
= i.meeting_id) BETWEEN m.start_time AND m.end_time
                  OR (SELECT i_m.end_time FROM meetings i_m WHERE i_m.meeting_id =
i.meeting_id) BETWEEN m.start_time AND m.end_time
                  OR m.start_time BETWEEN (SELECT i_m.start_time FROM meetings i_m
WHERE i_m.meeting_id = i.meeting_id) AND (SELECT i_m.end_time FROM meetings i_m
WHERE i_m.meeting_id = i.meeting_id))

         )
         BEGIN
         RAISERROR('Another meeting in the course collides timewise.', 16, 1);
         END;

   INSERT INTO studies_meeting SELECT * FROM inserted;
END;
```

# Role i uprawnienia

## Administrator (Igor S.)

```
USE u_jkotara
go

CREATE ROLE administrator AUTHORIZATION dbo
go

GRANT ALTER, EXECUTE, SELECT ON SCHEMA :: dbo TO administrator
go
```

## Koordynator (Igor S.)

```
USE u_jkotara
go

CREATE ROLE coordinator AUTHORIZATION dbo
go

GRANT INSERT, UPDATE ON dbo.internships_attendance TO coordinator
go

GRANT UPDATE ON dbo.meetings_translations TO coordinator
go

GRANT UPDATE ON dbo.syllabus TO coordinator
go

GRANT SELECT ON dbo.teachers TO coordinator
go

GRANT SELECT ON dbo.translators TO coordinator
go

GRANT INSERT ON dbo.webinars TO coordinator
go

GRANT SELECT ON dbo.enrolled_raport TO coordinator
go

GRANT INSERT ON dbo.studies_syllabus TO coordinator
go

GRANT EXECUTE ON dbo.change_student_attendance TO coordinator
go

GRANT EXECUTE ON dbo.show_student_attendance TO coordinator
go

GRANT EXECUTE ON dbo.show_student_contact TO coordinator
go
```

## Dyrektor (Igor S.)

```
USE u_jkotara
go
```

```
CREATE ROLE headmaster AUTHORIZATION dbo
go

GRANT ALTER, EXECUTE, SELECT ON SCHEMA :: dbo TO headmaster
go
```

## Sekretariat (Igor S.)

```
USE u_jkotara
go

CREATE ROLE office_worker AUTHORIZATION dbo
go

GRANT UPDATE ON dbo.course_editions TO office_worker
go

GRANT UPDATE (price) ON dbo.courses TO office_worker
go

GRANT SELECT ON dbo.orders_details TO office_worker
go

GRANT DELETE ON dbo.students TO office_worker
go

GRANT UPDATE ON dbo.studies TO office_worker
go

GRANT UPDATE (price) ON dbo.webinars TO office_worker
go

GRANT SELECT ON dbo.debtors TO office_worker
go

GRANT SELECT ON dbo.employees TO office_worker
go

GRANT SELECT ON dbo.financial_raport TO office_worker
go

GRANT SELECT ON dbo.financial_raport_by_activity TO office_worker
go

GRANT SELECT ON dbo.orders_raport TO office_worker
```

```
go

GRANT SELECT ON dbo.total_price_by_orders TO office_worker
go

GRANT SELECT ON dbo.total_price_by_student TO office_worker
go
```

## Student (Igor S.)

```
USE u_jkotara
go

CREATE ROLE student AUTHORIZATION dbo
go

GRANT SELECT ON dbo.meetings TO student
go

GRANT SELECT ON dbo.meetings_attendance TO student
go

GRANT INSERT ON dbo.orders TO student
go

GRANT INSERT ON dbo.orders_details TO student
go

GRANT UPDATE ON dbo.users TO student
go

GRANT SELECT ON dbo.course_grades TO student
go

GRANT SELECT ON dbo.debtors TO student
go

GRANT SELECT ON dbo.studies_grades TO student
go
```

## Nauczyciel (Igor S.)

```
USE u_jkotara
go

CREATE ROLE teacher AUTHORIZATION dbo
```

```
go

GRANT UPDATE ON dbo.meetings_attendance TO teacher
go

GRANT SELECT ON dbo.studies_meeting TO teacher
go

GRANT SELECT ON dbo.studies_sessions TO teacher
go
```

## Niezalogowany użytkownik

```
USE u_jkotara
go

CREATE ROLE unregistered AUTHORIZATION dbo
go

GRANT SELECT ON dbo.courses TO unregistered
go

GRANT SELECT ON dbo.studies TO unregistered
go

GRANT INSERT ON dbo.users TO unregistered
go

GRANT SELECT ON dbo.webinars TO unregistered
go
```