Podstawy baz danych

Dzień i godzina zajęć: Środa 15:00

Nr zespołu: 2

Autorzy: Dariusz Marecik, Filip Węgrzyn, Paweł Fornagiel

Link do repozytorium git: https://github.com/pFornagiel/bazy-danych-2025

Założenia dotyczące projektu:

- W zakres studiów wchodzą pojedyńcze przedmioty (studium), które mają przypisane spotkania

1. Wymagania i funkcje systemu

Opis Funkcjonalności Systemu

Funkcje Systemu

- Weryfikacja limitu zapisanych osób i blokowanie jego przekroczenia
- Blokowanie zapisu / dostępu do treści po upływie terminu ważności
- Blokowanie możliwości zapisania się na te same zajęcia wiele razy

Użytkownicy

- Studenci (użytkownicy zalogowani)
- Goście (użytkownicy niezalogowani)
- Prowadzący zajęcia
- Dyrektor Szkoły
- Administrator zasobów
- Dziekanat
- Tłumacz

Funkcje poszczególnych użytkowników

Studenci (użytkownicy zalogowani, rozszerzenie możliwości gości)

- możliwość zapisania się na kurs
- zapis na praktyki
- usunięcie konta
- dodanie i usunięcie adresu korespondencyjnego
- wyświetlenie wykazu zajęć w których brał udział / obecności
- wyświetlenie frekwencji / stopnia zaliczenia dla poszczególnych zajęć

- wyświetlenie dostępnych kursów / webinarów / studiów
- wyświetlanie linków dostępu do udostępnionych zasobów
- dodanie, usunięcie i przegląd elementów w koszyku
- stworzenie zamówienia
- opłacenie zamówienia

Goście (użytkownicy niezalogowani)

- dostęp do wybranych webinarów
- przegląd dostępnych webinarów / studiów / kursów
- założenie konta

Prowadzący zajęcia

- modyfikacja terminu zajęć
- modyfikacja udostępnionych zasobów
- sprawdzanie obecności dla każdych zajęć
- wyświetlenie wykazu prowadzonych zajęć

Administrator zasobów

- dodawanie / usuwanie webinarów, kursów i studiów
- dodawanie / usuwanie materiałów

Dyrektor

- dodawanie / usuwanie pracowników
- modyfikacja dostępu do kursu
- modyfikacja opłat za kurs
- modyfikacja czasu na dokonanie płatności dla danej osoby
- przegląd wszelkich danych dotyczących realizowanych zajęć

Dziekanat

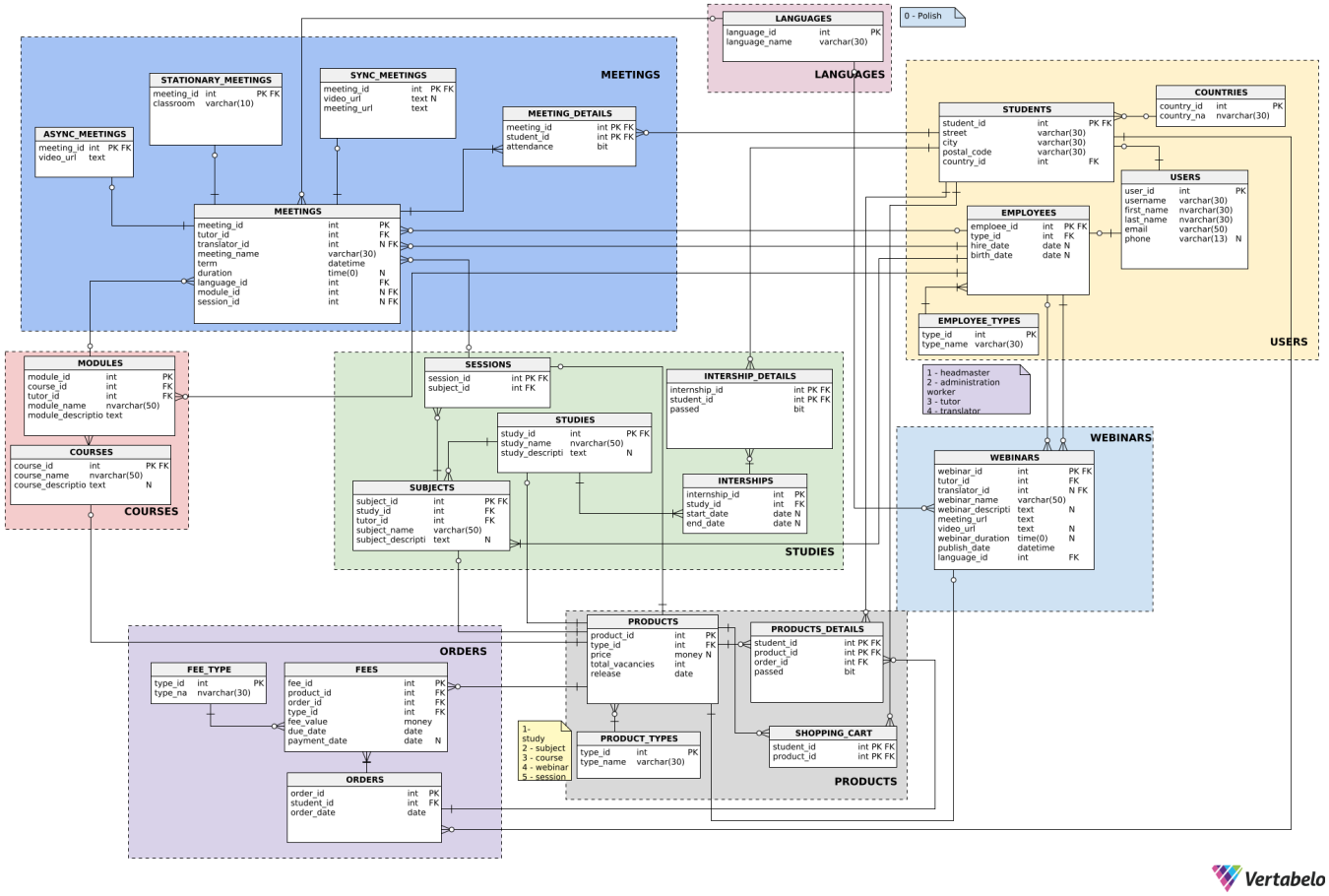
- tworzenie dyplomów potwierdzających ukończenie kursu / studium
- dodawanie / modyfikacja praktyk
- modyfikacja webinarów / kursów / studiów / przedmiotów
- dodawanie webinarów / kursów / studiów / przedmiotów
- dodawanie / usuwanie tłumacza do wybranych przedmiotów
- tworzenie sylabusu
- generowanie harmonogramu
- generowanie danych dotyczących realizowanych zajęć
- wyświetlenie zatrudnionych pracowników
- wyświetlenie studentów przypisanych do danego zasobu wraz z limitami zasobu
- wyświetlenie danych dotyczących wybranych form zajęć
- wykrywanie i wyświetlanie kolizji czasowych studentów
- Raportowanie:
    - Tworzenie raportu liczby zapisanych osób na przyszłe wydarzenia wraz z informacjami o wydarzeniach

- Tworzenie raportu dotyczącego frekwencji na zakończonych wydarzeniach
- Tworzenie raportu dotyczącego osób, które skorzystały z usług, ale nie uiściły opłat
- Tworzenie raportów finansowych
- Tworzenie list obecności dla poszczególnych form zajęć
- Tworzenie list kolizji czasowych wśród użytkowników

Tłumacz

- Dostęp do zasobów poszczególnych kursów / studiów i webinarów
- Dodawanie przetłumaczonych zasobów do kursów /studiów / webinarów

Schemat bazy danych



Opis tabel

Kategoria USERS

Tabela USERS

Column Name	Data Type	Properties
user_id	int	Primary Key
username	varchar(30)	

Column Name	Data Type	Properties
first_name	nvarchar(30)	
last_name	nvarchar(30)	
email	varchar(50)	
phone	varchar(9)	

Zawiera podstawowe informacje o każdym użytkowniku bazy.

- user_id int - klucz główny, identifikuje użytkownika
- username varchar(30) - nazwa użytkownika w bazie danych
- first_name nvarchar(30) - imię użytkownika
- last_name nvarchar(30) - nazwisko użytkownika
- email varchar(50) - email użytkownika
    - warunek: (mail LIKE '%_@%.%')
- phone varchar(9) nullable - numer telefonu użytkownika
    - warunek: LEN(Phone) = 15 AND ISNUMERIC(Phone) = 1

```
-- Table: USERS
CREATE TABLE USERS (
    user_id int  NOT NULL IDENTITY,
    username varchar(30)  NOT NULL,
    first_name nvarchar(30)  NOT NULL,
    last_name nvarchar(30)  NOT NULL,
    email varchar(50)  NOT NULL CHECK (mail LIKE '%_@%.%'),
    phone varchar(9)  NULL CHECK (LEN(Phone) = 9 AND ISNUMERIC(Phone) = 1),
    CONSTRAINT unique_email UNIQUE (email),
    CONSTRAINT unique_phone UNIQUE (phone),
    CONSTRAINT USERS_pk PRIMARY KEY  (user_id)
);
```

Tabela STUDENTS

Column Name	Data Type	Properties
student_id	int	Primary Key Foreign Key
street	varchar(30)	
city	varchar(30)	
postal_code	varchar(30)	

Column Name	Data Type	Properties
country_id	varchar(30)	Foreign Key

Zawiera infromacje specyficzne dla studenta

- *student_id* int - klucz główny, klucz obcy, identyfikuje studenta
- street varchar(30) - ulica, na której mieszka studenta
- city varchar(30) - miasto, w którym mieszka studenta
- postal_code varchar(30) - kod pocztowy studenta
- country_id int - klucz obcy, identyfikator pochodzenia studenta

```
-- Table: STUDENTS
CREATE TABLE STUDENTS (
   student_id int  NOT NULL,
   street varchar(30)  NOT NULL,
   city varchar(30)  NOT NULL,
   postal_code varchar(30)  NOT NULL,
   country varchar(30)  NOT NULL,
   CONSTRAINT STUDENTS_pk PRIMARY KEY  (student_id)
);
```

Tabela EMPLOYEES

Column Name	Data Type	Properties
emploee_id	int	Primary Key Foreign Key
type_id	int	Foreign Key
hire_date	date	
birth_date	date	

Zawiera szczególne informacje dla pracowników (dyrektora, pracownika dziekanatu, nauczyciela, tłumacza)

- *emploee_id* int - klucz główny, klucz obcy, identyfikator pracownika
- type_id int - sklucz obcy, typ pracownika (opisany poniżej)
- hire_date date nullable - data zatrudnienia
    - DEFAULT current_date
- birth_date date nullable - data urodzin pracownika
    - DEFAULT current_date

```
-- Table: EMPLOYEES
CREATE TABLE EMPLOYEES (
    emploee_id int  NOT NULL,
    type_id int  NOT NULL,
    hire_date date  NULL DEFAULT current_date,
    birth_date date  NULL DEFAULT current_date,
    CONSTRAINT EMPLOYEES_pk PRIMARY KEY  (emploee_id)
);
```

Tabela EMPLOYEES_TYPE

Column Name	Data Type	Properties
type_id	int	Primary Key
type_name	varchar(30)	

Zawiera opis typów pracowników

- *type_id* int - klucz główny, typ pracownika
  1 - headmaster
  2 - administration worker
  3 - tutor
  4 - translator
- type_name varchar(30) - nazwa pełnionej funkcji

```
-- Table: EMPLOYEE_TYPES
CREATE TABLE EMPLOYEE_TYPES (
    type_id int  NOT NULL IDENTITY,
    type_name varchar(30)  NOT NULL,
    CONSTRAINT EMPLOYEE_TYPES_pk PRIMARY KEY  (type_id)
);
```

Tabela COUNTRIES

Column Name	Data Type	Properties
country_id	int	Primary Key
country_name	nvarchar(30)	

Tabela słownikowa, przechowująca nazwy znanych państw, z których pocodzą studenci

- country_id int - klucz główny, identyfikator państwa
- country_name nvarchar(30) - nazwa państwa

```
-- Table: COUNTRIES
CREATE TABLE COUNTRIES (
```

```
    country_id int  NOT NULL,
    country_name nvarchar(30)  NOT NULL,
    CONSTRAINT COUNTRIES_pk PRIMARY KEY  (country_id)
);
```

Kategoria Products

Tabela Products

Column Name	Data Type	Properties
product_id	int	Primary Key Foreign Key
type_id	int	Foreign Key
price	money	
total_vacancies	int	
release	date	

Zawiera informacje o każdym produkcie w ofercie. Produkt jest rozumiany jako każda z form przeprowadzania zajęć.

- product*id* int - klucz główny, identyfikuje produkt
- type_id int - klucz obcy, numer kategorii produktu
- price money nullable - cena za produkt
    - warunek: prive >= 0
    - DEFAULT 1000
- total_vacancies int - ilość wolnych miejsc możliwych do zakupu na dane zajęcia
    - warunek: vacancies >= 0
- release date - data udostępnienia produktu do zakupu

```
-- Table: PRODUCTS
CREATE TABLE PRODUCTS (
    product_id int  NOT NULL,
    type_id int  NOT NULL,
    price money  NULL DEFAULT 1000 CHECK (price>=0),
    total_vacancies int  NOT NULL DEFAULT 30 CHECK (total_amount>0),
    release date  NOT NULL,
    CONSTRAINT product_id PRIMARY KEY  (product_id)
);
```

Tabela PRODUCTS_DETAILS

Column Name	Data Type	Properties
student_id	int	Primary Key Foreign Key
product_id	int	Primary Key Foreign Key
order_id	int	Foreign Key

Zawiera informacje o studentach zapisanych na dane zajęcia oraz o numerze zamówienia z jakiego został kupiony dostęp do zajęć

- student_id int - wchodzi w skład klucza głównego, klucz obcy, identyfikuje studenta
- product_id int - wchodzi w skład klucza głównego, klucz obcy, identifukuje produkt
- order_id int - klucz obcy, identifikuje zamówienie z jakiego został kupiony dostęp do zajęć

```
-- Table: PRODUCTS_DETAILS
CREATE TABLE PRODUCTS_DETAILS (
    student_id int  NOT NULL,
    product_id int  NOT NULL,
    order_id int  NOT NULL,
    CONSTRAINT PRODUCTS_DETAILS_pk PRIMARY KEY  (student_id,product_id)
);
```

Tabela PRODUCT_TYPES

Column Name	Data Type	Properties
type_id	int	Primary Key
type_name	varchar(30)	

Zawiera informacje o typach produktów

- type_id int - klucz główny, identyfikuje typ:
  1- study,
  2 - subject,
  3 - course,
  4 - webinar
- type_name varchar(30) - nazwa typu

```
-- Table: PRODUCT_TYPES
CREATE TABLE PRODUCT_TYPES (
    type_id int  NOT NULL IDENTITY,
    type_name varchar(30)  NOT NULL,
    CONSTRAINT PRODUCT_TYPES_pk PRIMARY KEY  (type_id)
);
```

Tabela CART

Column Name	Data Type	Properties
student_id	int	Primary Key Foreign Key
product_id	int	Foreign Key

Zawiera informacje o koszyku użytkownika

- student_id int - klucz główny, klucz obcy, identyfikator użytkownika
- product_id int - klucz główny, klucz obcy, identyfikator produktu

```
-- Table: CART
CREATE TABLE CART (
    student_id int  NOT NULL,
    product_id int  NOT NULL,
    CONSTRAINT CART_pk PRIMARY KEY  (student_id)
);
```

Kategoria Orders

Tabela ORDERS

Column Name	Data Type	Properties
order_id	int	Primary Key Foreign Key
student_id	int	Foreign Key
order_date	date	

Zawiera informacje na temat zamówienia pod danym identyfikatorem

- order_id int - klucz główny, identyfikator zamówienia
- student_id int - kluczo obcy, identyfikator studenta
- order_date datetime nullable - data złożenia zamówienia

```
-- Table: ORDERS
CREATE TABLE ORDERS (
    order_id int  NOT NULL IDENTITY,
    student_id int  NOT NULL,
    order_date date  NOT NULL DEFAULT actual_date,
    CONSTRAINT ORDERS_pk PRIMARY KEY  (order_id)
);
```

Tabela FEES

Column Name	Data Type	Properties
fee_id	int	Primary Key Foreign Key
due_date	date	
payment_date	date	
fee_value	money	
type_id	int	Foreign Key
order_id	int	Foreign Key
product_id	int	Foreign Key

Zawiera informacje o płatności za dany produkt dołączonej do danego zamówienia

- fee_id int - klucz główny, identyfikator płatności
- due_date date - data wymagania płatności, nieuregulowanie do podanego teminu skutkuje wpisem na liste dłużników
- payment_date date nullable - data dokonania płatności
- fee_value money - cena płatności
    - warunek: fee_value >= 0
- type_id int - klucz obcy, identyfikator typu płatności
- order_id int - klucz obcy, identifikator zamówienia
- product_id int, klucz obcy, identyfikator produktu

```
-- Table: PAYMENTS
CREATE TABLE FEES (
    fee_id int  NOT NULL IDENTITY,
    due_date date  NOT NULL DEFAULT actual_date,
    payment_date date  NULL,
    fee_value money  NOT NULL CHECK (payment_value>=0),
    type_id int  NOT NULL,
    order_id int  NOT NULL,
    product_id int  NOT NULL,
    CONSTRAINT FEES_pk PRIMARY KEY  (fee_id)
);
```

Tabela FEE_TYPE

Column Name	Data Type	Properties
type_id	int	Primary Key
type_name	nvarchar(30)	

Zawiera informacje o możliwych typach płatności

- type_id int - klucz główny, identyfikator typu płatności
- type_name nvarachar(30) - nazwa typu płatności

```
CREATE TABLE FEE_TYPE (
    type_id int  NOT NULL,
    type_name nvarchar(30)  NOT NULL,
    CONSTRAINT FEE_TYPE_pk PRIMARY KEY  (type_id)
);
```

Kategoria Webinars

Tabela Webinars

Column Name	Data Type	Properties
webinar_id	int	Primary Key Foreign Key
tutor_id	int	Foreign Key
translator_id	int	Foreign Key
webinar_name	varchar(50)	
webinar_description	text	
meeting_url	text	
video_url	text	
webinar_duration	time(0)	
publish_date	datetime	
language_id	int	Foreign Key

Zawiera informacje specyfinczne dla każdego produktu będącego webinarem

- webinar_id int - klucz główny, klucz obcy, identifikator webinaru
- tutor_id int - klucz obcy, identifikator nauczyciela
- translator_id int nullable - klucz obcy, identifikator tłumacza
- webinar_name varchar(50) - nazwa webinaru

- webinar_description text nullable - opis webinaru
- meeting_url text nullable - link do webinaru na żywo
- video_url text nullable - link do zapisu webinaru
- webinar_duration time(0) - czas trwania webinaru
    - warunek: DurationTime > '00:00:00'
    - DEFAULT 01:30:00
- publish_date datetime - data przeprowadzenia i udostępnienia materiałów video
- language_id int - klucz obcy, identyfikator języka, w jakim jest prowadzony Webinar
    - DEFAULT 0

```
-- Table: WEBINARS
CREATE TABLE WEBINARS (
   webinar_id int  NOT NULL,
   tutor_id int  NOT NULL,
   translator_id int  NULL,
   webinar_name varchar(50)  NOT NULL,
   webinar_description text  NULL,
   video_url text  NULL,
   webinar_duration time(0)  NULL DEFAULT 01:30:00 CHECK (DurationTime > '00:00:00'),
   publish_date datetime  NOT NULL,
   language varchar(50)  NOT NULL DEFAULT 'POLISH',
   CONSTRAINT WEBINARS_pk PRIMARY KEY  (webinar_id)
);
```

Kategoria COURSES

Tabela COURSES

Column Name	Data Type	Properties
course_id	int	Primary Key Foreign Key
course_name	nvarchar(50)	
course_description	text	

Zawiera informacje o produktach, które są kursami

- course_id int - klucz główny, klucz obcy, identifikator kursu
- course_name nvarchar(50) - nazwa kursu
- course_description text nullable - opis kursu

```
-- Table: COURSES
CREATE TABLE COURSES (
    course_id int  NOT NULL,
    course_name nvarchar(50)  NOT NULL,
    course_description text  NULL,
    CONSTRAINT COURSES_pk PRIMARY KEY  (course_id)
);
```

Tabela MODULES

Column Name	Data Type	Properties
module_id	int	Primary Key Foreign Key
course_id	int	Foreign Key
tutor_id	int	Foreign Key
module_name	int	
module_description	int	

Zawiera szczegółowe informacje dla każdego modułu kursu

- module_id int - klucz główny, identifikator modułu
- course_id int - klucz obcy, identifikator kursu, z którego pochodzi
- tutor_id int - klucz obcy, identifikator nauczyciela, który prowadzi dany moduł
- module_name - nazwa modułu
- module_description - opis modułu

```
-- Table: MODULES
CREATE TABLE MODULES (
    module_id int  NOT NULL IDENTITY,
    course_id int  NOT NULL,
    tutor_id int  NOT NULL,
    CONSTRAINT MODULES_pk PRIMARY KEY  (module_id)
);
```

Kategoria STUDIES

Tabela STUDIES

Column Name	Data Type	Properties

Column Name	Data Type	Properties
study_id	int	Primary Key Foreign Key
study_name	nvarchar(50)	
study_description	text	

Zawiera ogólne informacje o danych studiach

- study_id int - klucz główny, klucz obcy, identifikator studium
- study_name nvarchar(50) - nazwa studium
- study_description text nullable - opis studium

```
-- Table: STUDIES
CREATE TABLE STUDIES (
    study_id int  NOT NULL,
    study_name nvarchar(50)  NOT NULL,
    study_description text  NULL,
    CONSTRAINT STUDIES_pk PRIMARY KEY  (study_id)
);
```

Tabela SUBJECTS

Column Name	Data Type	Properties
subject_id	int	Primary Key Foreign Key
study_id	int	Foreign Key
tutor_id	int	Foreign Key
subject_name	varchar(50)	
subject_description	text	

Zawiera informacje szczegółowe inforamcje dotyczące przedmiotow

- subject_id int - klucz główny, klucz obcy, identifikator przedmiotu
- subject_name varchar(50) - nazwa przedmiotu
- subject_description text nullable - opis przedmiotu
- study_id int - klucz obcy, identifikator studiów, z których pochodzi przedmiot
- tutor_id int - klucz obcy, identifikator nauczyciela, który uczy dany przedmiot

```
-- Table: SUBJECTS
CREATE TABLE SUBJECTS (
   subject_id int  NOT NULL,
   study_id int  NOT NULL,
   tutor_id int  NOT NULL,
   subject_name varchar(50)  NOT NULL,
   subject_description text  NULL,
   CONSTRAINT SUBJECTS_pk PRIMARY KEY  (subject_id)
);
```

Tabela SESSIONS

Column Name	Data Type	Properties
session_id	int	Primary Key Foreign Key
subject_id	int	Foreign Key

Zawiera informacje o poszczególnych sesjach (grupach spotkań zjazdowych)

- sessions_id int - klucz główny, klucz obcy, identyfikator sesji
- subject_id int - klucz główny, klucz obcy, identifikator przedmiotu związanego z sesją

```
CREATE TABLE SESSIONS (
    session_id int  NOT NULL,
    subject_id int  NOT NULL,
    CONSTRAINT SESSIONS_pk PRIMARY KEY  (session_id)
);
```

Tabela INTERSHIPS

Column Name	Data Type	Properties
internship_id	int	Primary Key Foreign Key
study_id	int	Foreign Key
start_date	date	
end_date	date	

Zawiera informacje o praktykach prowadzonych na danych studiach

- internship_id - klucz główny, identifikator praktyk
- study_id int - klucz obcy, identifikator studiów
- start_date date nullable - data rozpoczęcia praktyk

- end_date date nullable - data zakończenia praktyk

```
-- Table: INTERSHIPS
CREATE TABLE INTERSHIPS (
    internship_id int  NOT NULL IDENTITY,
    study_id int  NOT NULL,
    start_date date  NULL,
    end_date date  NULL,
    CONSTRAINT INTERSHIPS_pk PRIMARY KEY  (internship_id)
);
```

Tabela INTERSHIPS_DETAILS

Column Name	Data Type	Properties
internship_id	int	Primary Key Foreign Key
student_id	int	Primary Key Foreign Key
passed	bit	

Zawiera szczegółowe informacje na temat danych praktyk

- internship_id int - klucz główny, klucz obcy, identifikator praktyk
- student_id int - klucz główny, klucz obcy, identifikator studenta biorącego udział w praktykach
- passed bit - zaliczenie danych praktyk,
  1 - student zaliczył praktyki (100% obecności),
  0 - student nie zaliczył praktyk (brak 100% obecności)

```
-- Table: INTERSHIP_DETAILS
-- Table: INTERSHIP_DETAILS
CREATE TABLE INTERSHIP_DETAILS (
    internship_id int  NOT NULL,
    student_id int  NOT NULL,
    passed bit  NOT NULL,
    CONSTRAINT INTERSHIP_DETAILS_pk PRIMARY KEY  (internship_id,student_id)
);
```

Kategoria MEETINGS

Tabela MEETINGS

Column Name	Data Type	Properties

Column Name	Data Type	Properties
meeting_id	int	Primary Key Foreign Key
tutor_id	int	Foreign Key
translator_id	int	Foreign Key
meeting_name	varchar(30)	
term	datetime	
duration	time(0)	
language_id	int	Foreign Key
module_id	int	Foreign Key
session_id	int	Foreign Key

Zawiera ogólne informacje na temat spotkania

- meeting_id int - klucz główny, identifikator spotkania
- tutor_id int - klucz obcy, identifikator nauczyciela prowadzącego spotkanie
- translator_id int nullable nullable - klucz obcy, identifikator tłumacza tłumaczącego spotkanie
- meeting_name varchar(30) - nazwa spotkania
- term datetime - data i godzina spotkania
- duration time(0) nullable - czas trwania spotkania
    - Warunek: duration > '00:00:00'
    - DEFAULT 01:30:00
- language_id int - klucz obcy, identyfikator języka w jakim przeprowadza się spotkanie
    - DEFAULT 0
- module_id int nullable - klucz obcy, identyfikator modułu kursu odpowiadającego spotkani
- sessions_id int nullable - klucz obcy, identyfikator sesji odpowiadającej spotkaniu

```
-- Table: MEETINGS
CREATE TABLE MEETINGS (
    meeting_id int  NOT NULL IDENTITY,
    tutor_id int  NOT NULL,
    translator_id int  NULL,
    meeting_name varchar(30)  NOT NULL,
    term datetime  NOT NULL,
    duration time(0)  NULL DEFAULT 01:30:00 CHECK (duration>'00:00:00'),
    language varchar(30)  NOT NULL DEFAULT 'POLISH',
```

```
    module_id int  NULL,
    session_id int  NULL,
    CONSTRAINT MEETINGS_pk PRIMARY KEY  (meeting_id)
);
```

Tabela MEETING_DETAILS

Column Name	Data Type	Properties
meeting_id	int	Primary Key Foreign Key
student_id	int	Primary Key Foreign Key
attendance	bit	

Zawiera szczegółowe informacje na temat osób biorących udział w spotkaniu

- meeting_id int - klucz główny, identyfikator spotkania
- student_id int - identyfiaktor studenta, zapisanego na spotkanie
- attendance bit - obecność,
  1 - student uczestniczył w spotkaniu,
  0 - student nie uczestniczył w spotkaniu

```
-- Table: MEETING_DETAILS
CREATE TABLE MEETING_DETAILS (
   meeting_id int  NOT NULL,
   student_id int  NOT NULL,
   attendance bit  NOT NULL,
   CONSTRAINT MEETING_DETAILS_pk PRIMARY KEY  (meeting_id,student_id)
);
```

Tabela ASYNC_MEETINGS

Column Name	Data Type	Properties
meeting_id	int	Primary Key Foreign Key
meeting_url	text	

Zawiera dane dotyczące spotkań internetowych, które nie są na żywo

- meeting_id int - klucz główny, klucz obcy, identyfikator spotkania
- meeting_url text - link do spotkania

```
-- Table: ASYNC_MEETINGS
CREATE TABLE ASYNC_MEETINGS (
    meeting_id int  NOT NULL,
    meeting_url text  NOT NULL,
    CONSTRAINT ASYNC_MEETINGS_pk PRIMARY KEY  (meeting_id)
);
```

Tabela SYNC_MEETINGS

Column Name	Data Type	Properties
meeting_id	int	Primary Key Foreign Key
video_url	text	
meeting_url	text	

Zawiera dane dotyczące spotkań internetowych, które są na żywo

- meeting_id int - klucz główny, klucz obcy, identyfikator spotkania
- video_url text nullable - link do zapisu video spotkania
- meeting_url text - link do spotkania

```
-- Table: SYNC_MEETINGS
CREATE TABLE SYNC_MEETINGS (
    meeting_id int  NOT NULL,
    video_url text  NULL,
    meeting_url text  NOT NULL,
    CONSTRAINT SYNC_MEETINGS_pk PRIMARY KEY  (meeting_id)
);
```

Tabela STATIONARY_MEETINGS

Column Name	Data Type	Properties
meeting_id	int	Primary Key Foreign Key
classroom	varchar(10)	

Zawiera dane dotyczące spotkań internetowych, które są stacjonarnie

- meeting_id int - klucz główny, klucz obcy, identyfikator spotkania
- classroom varchar(10) - numer pokoju, w którym przeprowadzane jest spotkanie

```
-- Table: STATIONARY_MEETINGS
CREATE TABLE STATIONARY_MEETINGS (
   meeting_id int  NOT NULL,
   classroom varchar(10)  NOT NULL,
   CONSTRAINT STATIONARY_MEETINGS_pk PRIMARY KEY  (meeting_id)
);
```

Kategoria LANGUAGES

Tabela LANGUAGES

Column Name	Data Type	Properties
language_id	int	Primary Key
language_name	varchar(30)	

Tabela słownikowa, zawierająca nazwy dostępnych język, w których są przeprowadzane formy kształcenia

- language_id int - klucz główny, identyfikator języka
- language_name varchar(30) - nazwa języka

```
-- Table: LANGUAGES
CREATE TABLE LANGUAGES (
    language_id int  NOT NULL,
    language_name varchar(30)  NOT NULL,
    CONSTRAINT LANGUAGES_pk PRIMARY KEY  (language_id)
);
```

Dokumentacja kluczy obcych

Table Name	FK Column	Referenced Table	Referenced Column
ASYNC_MEETINGS	meeting_id	MEETINGS	meeting_id
SHOPPING_CART	product_id	PRODUCTS	product_id
SHOPPING_CART	student_id	STUDENTS	student_id
COURSES	course_id	PRODUCTS	product_id
EMPLOYEES	type_id	EMPLOYEE_TYPES	type_id
EMPLOYEES	emploee_id	USERS	user_id
WEBINARS	tutor_id	EMPLOYEES	emploee_id
FEES	order_id	ORDERS	order_id
FEES	product_id	PRODUCTS	product_id

Table Name	FK Column	Referenced Table	Referenced Column
FEES	type_id	FEE_TYPE	type_id
INTERSHIPS	study_id	STUDIES	study_id
INTERSHIP_DETAILS	internship_id	INTERSHIPS	internship_id
INTERSHIP_DETAILS	student_id	STUDENTS	student_id
MEETINGS	language_id	LANGUAGES	language_id
WEBINARS	language_id	LANGUAGES	language_id
MEETINGS	module_id	MODULES	module_id
MEETINGS	session_id	SESSIONS	session_id
MEETING_DETAILS	meeting_id	MEETINGS	meeting_id
MEETING_DETAILS	student_id	STUDENTS	student_id
MEETINGS	tutor_id	EMPLOYEES	emploee_id
MEETINGS	translator_id	EMPLOYEES	emploee_id
MODULES	course_id	COURSES	course_id
MODULES	tutor_id	EMPLOYEES	emploee_id
PRODUCTS_DETAILS	order_id	ORDERS	order_id
PRODUCTS_DETAILS	product_id	PRODUCTS	product_id
PRODUCTS_DETAILS	student_id	STUDENTS	student_id
PRODUCTS	type_id	PRODUCT_TYPES	type_id
SUBJECTS	subject_id	PRODUCTS	product_id
SESSIONS	session_id	PRODUCTS	product_id
SESSIONS	subject_id	SUBJECTS	subject_id
STATIONARY_MEETINGS	meeting_id	MEETINGS	meeting_id
STUDENTS	country_id	COUNTRIES	country_id
ORDERS	student_id	STUDENTS	student_id
STUDIES	study_id	PRODUCTS	product_id
SUBJECTS	tutor_id	EMPLOYEES	emploee_id
SUBJECTS	study_id	STUDIES	study_id
SYNC_MEETINGS	meeting_id	MEETINGS	meeting_id
STUDENTS	student_id	USERS	user_id
WEBINARS	translator_id	EMPLOYEES	emploee_id

Table Name	FK Column	Referenced Table	Referenced Column
WEBINARS	webinar_id	PRODUCTS	product_id

Widoki

Users

Student_address

Widok student_address dla każdego studenta podaje jego imię i nazwisko i adres zamieszkania , czyli ulicę, kod pocztowy, miasto i państwo.

```
CREATE view student_address as
    SELECT
        students.user_id AS student_id,
        users.first_name + ' ' + users.last_name AS name,
        students.street AS street,
        students.postal_code AS zip_code,
        students.city AS city,
        student.country AS country

    FROM students
    join user on users.user_id = students.student_id
;
```

Emploee_list

Widok emploee_type_list wylisowuje wszystkich imiona i nazwiaska wszystkich pracowników oraz przypisane do nich role

```
CREATE view emploee_type_list as
    SELECT
        emploees.emploee_id as emploee_id,
        users.first_name + ' ' + users.last_name AS name,
        emploee_type.type_name as rola

    FROM emploees
    join user on users.user_id = emploees.emploee_id
    join emploee_type on emploee_type.type_id = emploees.type_id
;
```

User_information

Widok user_information dla każdego użytkownika podaje jego imię, nazwisko, adres e-mail, nr telefonu oraz czy jest studentem, czy pracownikiem

```
CREATE view user_information as
    SELECT
        users.user_id as user_id,
        users.first_name + ' ' + users.last_name AS name,
        users.email as email,
        users.phone as phone
        CASE
            WHEN users.id IN (SELECT user_id FROM students) AND
            users.id NOT IN (SELECT user_id FROM emploees) AND
            THEN 'student'

            WHEN users.id NOT IN (SELECT user_id FROM students) AND
            users.id IN (SELECT user_id FROM emploees) AND
            THEN 'employee'

    FROM emploees
;
```

Regular_customers

Widok regular_customers pokazuje stałych klientów, którzy są zdefiniowani jako osoby, które złożyły jakiekolwiek zamówienie w przeciągu ostatnich 2 lat

```
CREATE VIEW regular_customers AS
    SELECT
        student_id,
        COUNT(order_date) AS order_count
    FROM orders
    WHERE order_date >= DATEADD(year, -2, GETDATE())
    GROUP BY student_id
    HAVING COUNT(order_date) > 0
;
```

Webinars

Webinar_information

Widok webinar_information dla każdego webinaru podaje jego tytuł, opis, ID prowadzącego, ramy czasowe, ID tłumacza, link do spotkania, link do nagrania oraz jezyk w jakim jest prowadzony.

```
CREATE VIEW Webinar_information report AS
    SELECT
        webinar_id as webinar_id
        webinar_name as name
        webinar_description as description
        tutor_id as tutor
```

```
        publish_date as start_time
        webinar_duration as duration
        translator_id as translator
        meeting_url as meeting_url
        language as language
    FROM WEBINARS
;
```

alternatywnie

Widok webinar_information dla każdego webinaru podaje jego tytuł, opis, imie i nazwisko prowadzącego, ramy czasowe, imie i nazwisko tłumacza, link do spotkania, link do nagrania oraz jezyk w jakim jest prowadzony.

```
CREATE VIEW Webinar_information report AS
    SELECT
        webinar_id as webinar_id
        webinar_name as name
        webinar_description as description
        Tu.first_name + ' ' + Tu.last_name AS tutor_name,
        publish_date as start_time
        webinar_duration as duration
        Tr.first_name + ' ' + Tr.last_name AS translator_name,
        meeting_url as meeting_url
        language as language
    FROM WEBINARS
    left join users Tu
        on Tu.user_id = WEBINARS.tutor_id
    left join users Tr
        on Tr.user_id = WEBINARS.translator_id
;
```

Webinar_free_entry

Widok webinar_information wylistowuje webinary, które są darmowe. Dla każdego webinaru podaje jego tytuł, opis, imie i nazwisko prowadzącego, ramy czasowe, imie i nazwisko tłumacza, link do spotkania, link do nagrania oraz jezyk w jakim jest prowadzony.

```
CREATE VIEW Webinar_free_entry report AS
    SELECT
        webinar_id as webinar_id
        webinar_name as name
        webinar_description as description
        Tu.first_name + ' ' + Tu.last_name AS tutor_name,
        publish_date as start_time
        webinar_duration as duration
        Tr.first_name + ' ' + Tr.last_name AS translator_name,
        meeting_url as meeting_url
        language as language
```

```
    FROM WEBINARS
    left join users Tu
        on Tu.user_id = WEBINARS.tutor_id
    left join users Tr
        on Tr.user_id = WEBINARS.translator_id
    left join Products
        on webinar_id = product_id
    where products.price = 0
;
```

Webinar_available

Widok Webinar_available wylistowuje webinary, które odbędą się w przyszłości. Dla każdego webinaru podaje jego tytuł, opis, imie i nazwisko prowadzącego, ramy czasowe, imie i nazwisko tłumacza, link do spotkania, link do nagrania oraz jezyk w jakim jest prowadzony.

```
CREATE VIEW Webinar_available report AS
    SELECT
        webinar_id as webinar_id
        webinar_name as name
        webinar_description as description
        Tu.first_name + ' ' + Tu.last_name AS tutor_name,
        publish_date as start_time
        webinar_duration as duration
        Tr.first_name + ' ' + Tr.last_name AS translator_name,
        meeting_url as meeting_url
        language as language
    FROM WEBINARS
    join users Tu
        on Tu.user_id = WEBINARS.tutor_id
    join users Tr
        on Tr.user_id = WEBINARS.translator_id
    WHERE publish_date >= GETDATE()
;
```

Courses

Course_information

Widok Course_information dla każdego kursu podaje jego ID wraz z jego tytułem, opisem, ramami czasowymi, językiem w którym odbywają się spotkania, limitem miejsc i ceną.

```
CREATE VIEW Course_information report AS
    with course_start_end_date as (
        select
            course_id as course_id,
            min(term) as start_date,
```

```
            max(term) as end_date
        from meetings
        join modules on modules.module_id = meetings.module_id
        GROUP by module_id
        where module_id is not null
    )

    SELECT
        c.course_id as course_id
        c.course_name as name
        c.course_description as description
        start_end_date.start_date as start_date
        start_end_date.end_date as end_date
        c.meeting_url as meeting_url
        language as language
        products.price as price
        products.total_vacancies as amount_of_site
    FROM courses c
    join course_start_end_date
        on course_start_end_date.course_id = courses.course_id
    join products
        on products.product_id = course.course_id
;
```

Course_module_meeting_types

Widok course_module_meeting_types dla każdego modułu kursu podaje ile spotkań danego typu do niego należy.

```
CREATE VIEW course_module_meeting_types AS

    with STATIONARY_course_MEETINGS_count as (
        select
            module_id,
            count(*) as STATIONARY_MEETINGS_count
        from MEETINGS
        join STATIONARY_MEETINGS on STATIONARY_MEETINGS.meeting_id = MEETINGS.meeting_id
        GROUP by module_id
    ),
    with sync_course_MEETINGS_count as (
        select
            module_id,
            count(*) as sync_MEETINGS_count
        from MEETINGS
        join sync_MEETINGS on sync_MEETINGS.meeting_id = MEETINGS.meeting_id
        GROUP by module_id
    ),
    with async_course_MEETINGS_count as (
        select
            module_id,
            count(*) as async_MEETINGS_count
```

```
        from MEETINGS
        join async_MEETINGS on async_MEETINGS.meeting_id = MEETINGS.meeting_id
        GROUP by module_id
    )

    select
        module_id as module_id
        module_name as name
        STATIONARY_MEETINGS_count as STATIONARY_MEETINGS_count
        sync_MEETINGS_count as sync_MEETINGS_count
        async_MEETINGS_count as async_MEETINGS_count
    from
        MODULES
    JOIN STATIONARY_course_MEETINGS_count AS scmc
        ON scmc.module_id = m.module_id
    JOIN sync_course_MEETINGS_count AS syncmc
        ON syncmc.module_id = m.module_id
    JOIN async_course_MEETINGS_count AS asyncmc
        ON asyncmc.module_id = m.module_id;
```

Course_module_information

Widok course_module_information dla każdego modułu kursu podaje jego typ, limit miejsc oraz imię i nazwisko nauczyciela

```
CREATE VIEW Course_module_information report AS
    with STATIONARY_course_MEETINGS_count as (
        select
            module_id,
            count(*) as STATIONARY_MEETINGS_count
        from MEETINGS
        join STATIONARY_MEETINGS on STATIONARY_MEETINGS.meeting_id = MEETINGS.meeting_id
        GROUP by module_id
    ),
    with sync_course_MEETINGS_count as (
        select
            module_id,
            count(*) as sync_MEETINGS_count
        from MEETINGS
        join sync_MEETINGS on sync_MEETINGS.meeting_id = MEETINGS.meeting_id
        GROUP by module_id
    ),
    with async_course_MEETINGS_count as (
        select
            module_id,
            count(*) as async_MEETINGS_count
        from MEETINGS
        join async_MEETINGS on async_MEETINGS.meeting_id = MEETINGS.meeting_id
        GROUP by module_id
```

```
    )

    SELECT
    m.module_id,
    CASE
        WHEN scmc.STATIONARY_MEETINGS_count <> 0 AND
             syncmc.sync_MEETINGS_count = 0 AND
             asyncmc.async_MEETINGS_count = 0 THEN 'on_site'
        WHEN scmc.STATIONARY_MEETINGS_count = 0 AND
             syncmc.sync_MEETINGS_count <> 0 AND
             asyncmc.async_MEETINGS_count = 0 THEN 'online_synchronous'
        WHEN scmc.STATIONARY_MEETINGS_count = 0 AND
             syncmc.sync_MEETINGS_count = 0 AND
             asyncmc.async_MEETINGS_count <> 0 THEN 'online_asynchronous'
        WHEN scmc.STATIONARY_MEETINGS_count <> 0 OR
             syncmc.sync_MEETINGS_count <> 0 OR
             asyncmc.async_MEETINGS_count <> 0 THEN 'hybrid'
    END AS module_type,
    Tu.first_name + ' ' + Tu.last_name AS tutor_name,
    products.total_vacancies

FROM modules AS m
JOIN STATIONARY_course_MEETINGS_count AS scmc
    ON scmc.module_id = m.module_id
JOIN sync_course_MEETINGS_count AS syncmc
    ON syncmc.module_id = m.module_id
JOIN async_course_MEETINGS_count AS asyncmc
    ON asyncmc.module_id = m.module_id
join courses
    on courses.module_id = m.module_id
join products
    on courses.course_id = products.product_id
join emploees Tu
    on Tu.emploee_id = m.tutor_id
;
```

Course meeting information

Widok course_meeting_information dla każdego spotkania w ramach kursu podaje ID kursu do którego należy, ID modułu do którego należy, tytuł, opis spotkania, ramy czasowe oraz jego typ.

```
CREATE VIEW course_meeting_information report AS
    select
        modules.course_id AS course_id,
        modules.activity_id AS module_id,
        meetings.meeting_id AS meeting_id,
        meeting.name AS name,
        modules.description as description,
        meeting.term as start_time,
        meeting.duration as duration,
        CASE
```

```
            WHEN
                meetings.meeting_id IN (SELECT meeting_id FROM
                STATIONARY_MEETINGS) AND
                meetings.meeting_id not IN (SELECT meeting_id FROM
                async_meetings) AND
                meetings.meeting_id NOT IN (SELECT meeting_id FROM
                sync_meetings)
            THEN 'stationary'
            WHEN
                meetings.meeting_id not IN (SELECT meeting_id FROM
                STATIONARY_MEETINGS) AND
                meetings.meeting_id not IN (SELECT meeting_id FROM
                async_meetings) AND
                meetings.meeting_id IN (SELECT meeting_id FROM
                sync_meetings)
            THEN 'online_synchronous'
            WHEN
                meetings.meeting_id not IN (SELECT meeting_id FROM
                STATIONARY_MEETINGS) AND
                meetings.meeting_id IN (SELECT meeting_id FROM
                async_meetings) AND
                meetings.meeting_id NOT IN (SELECT meeting_id FROM
                sync_meetings)
            THEN 'online_asynchronous'
            END AS meeting_type,
    from modules
    join meeting on meeting.course_id = modules.module_id
```

Course passes

Widok course_passes dla każdego kursu podaje listę jego uczestników wraz z informacją o jego zaliczeniu.

```
CREATE VIEW course_passes AS
    SELECT
    courses.course_id as course_id,
    student_id as student_id,
    passed as passed
    from courses
    join products on products.product_id = courses.course_id
    join PRODUCTS_DETAILS on PRODUCTS_DETAILS.product_id = courses.course_id
```

Kategoria zamówienia i produkty

PRODUCT VACANCIES

Przedstawia ID produktu i wolne miejsca na dany produkt

```
create view PRODUCT_VACANCIES as
select product_id, total_vacancies-(select count(*) from FEES where FEES.product_id=PRODUCTS.product_id)
from PRODUCTS
```

USERS IN DEBT

Przedstawia użytkowników którzy nie opłacili danej usługi, ale z niej skorzystali co wykazane jest na liście obecności

```
create view USERS_IN_DEBT as
select student_id, concat_ws(' ',first_name,last_name) as name
from STUDENTS
join USERS on USERS.user_id=STUDENTS.student_id
where exists (select student_id
              from ORDERS
              join FEES on ORDERS.order_id=FEES.order_id
              join PRODUCTS on PRODUCTS.product_id = FEES.product_id
              join STUDIES on STUDIES.study_id=PRODUCTS.product_id
              join SUBJECTS on STUDIES.study_id=SUBJECTS.study_id
              join SESSIONS on SESSIONS.subject_id=SUBJECTS.subject_id
              join MEETINGS on MEETINGS.session_id=SESSIONS.session_id
              join MEETING_DETAILS on MEETINGS.meeting_id=MEETING_DETAILS.meeting_id
              where FEES.due_date>MEETINGS.term and FEES.payment_date=null
              union all
              select student_id
              from ORDERS
              join FEES on ORDERS.order_id=FEES.order_id
              join PRODUCTS on PRODUCTS.product_id = FEES.product_id
              join COURSES on COURSES.course_id=PRODUCTS.product_id
              join MODULES on COURSES.course_id=MODULES.course_id
              join MEETINGS on MEETINGS.module_id=MODULES.module_id
              join MEETING_DETAILS on MEETINGS.meeting_id=MEETING_DETAILS.meeting_id
              where FEES.due_date>MEETINGS.term and FEES.payment_date=null
              union all
              select student_id
              from ORDERS
              join FEES on ORDERS.order_id=FEES.order_id
              join PRODUCTS on PRODUCTS.product_id = FEES.product_id
              join MEETINGS on MEETINGS.session_id=PRODUCTS.product_id
              join MEETING_DETAILS on MEETINGS.meeting_id=MEETING_DETAILS.meeting_id
              where FEES.due_date>MEETINGS.term and FEES.payment_date=null
)
```

FINANCIAL REPORT

Przedstawia przychód dla każdego z produktów

```
create view FINANCIAL_REPORT as
select
```

```
  product_id,
  (select count(*)
    from FEES
    where FEES.product_id=PRODUCTS.product_id
  )*price as income, type_name
from PRODUCTS
join PRODUCT_TYPES on PRODUCTS.type_id=PRODUCT_TYPES.type_id
```

BILOCATION REPORT

Przedstawia studentów którzy mają kolizje wśród swoich zajęć

```
create view BILOCATION_REPORT as
with student_meetings as (
    select student_id, meeting_id,term,duration
    from MEETING_DETAILS
    join MEETINGS on MEETING_DETAILS.meeting_id=MEETINGS.meeting_id)
select student_id
from STUDENTS
join (select sm1.student_id,count(*) as collisions
      from student_meetings sm1
      join student_meetings sm2 on sm1.student_id=sm2.student_id and
      (
        datediff(hour,sm2.term-sm1.term)<sm1.duration or
        (datediff(hour,sm2.term-sm1.term)=sm1.duration and
        datediff(minute,sm2.term-sm1.term)<sm1.duration )
      ) or
      datediff(hour,sm1.term-sm2.term)<sm2.duration or
      (
        datediff(hour,sm1.term-sm2.term)=sm2.duration and
        datediff(minute,sm1.term-sm2.term)<sm2.duration
      )
      group by sm1.student_id) T on STUDENTS.student_id=T.student_id
where collisions>1
```

PRODUCT OWNERS

Przedstawia użytkowników i zakupione przez nich produkty

```
create view PRODUCT_OWNERS as
select student_id, concat_ws(' ',first_name,last_name) as name
from ORDER_DETAILS
join USERS on USER.user_id=ORDER_DETAILS.student_id
```

Procedury

Użytkownicy

CreateBasicUser

Procedura CreateBasicUser tworzy nowego użytkownika w systemie. ID użytkownika zwracane jest za pomocą @user_id.

Argumenty:

- @username - Nazwa użytkownika
- @first_name - Imię użytkownika
- @last_name - Nazwisko użytkownika
- @email - Adres email użytkownika
- @phone - Opcjonalny numer telefonu
- @user_id - Zwracany ID użytkownika

```
CREATE PROCEDURE CreateBasicUser
  @username VARCHAR(30),
  @first_name NVARCHAR(30),
  @last_name NVARCHAR(30),
  @email VARCHAR(50),
  @phone VARCHAR(9) = NULL,
  @user_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate email format
    IF @email NOT LIKE '%_@%.%'
    BEGIN
      RAISERROR('Niepoprawny format adresu email.', 16, 1);
      RETURN;
    END

    -- Validate phone number if provided
    IF @phone IS NOT NULL AND (LEN(@phone) != 9 OR ISNUMERIC(@phone) = 0)
    BEGIN
      RAISERROR('Niepoprawny format numeru telefonu.', 16, 1);
      RETURN;
    END

    -- Check for existing email
    IF EXISTS (SELECT 1 FROM USERS WHERE email = @email)
    BEGIN
      RAISERROR('Email został już przypisany do innego użytkownika.', 16, 1);
      RETURN;
    END

    -- Check for existing phone if provided
```

```
    IF @phone IS NOT NULL AND EXISTS (SELECT 1 FROM USERS WHERE phone = @phone)
    BEGIN
      RAISERROR('Numer telefonu został już przypisany do innego użytkownika.', 16, 1);
      RETURN;
    END

    -- Insert the new user
    INSERT INTO USERS (
      username,
      first_name,
      last_name,
      email,
      phone
    ) VALUES (
      @username,
      @first_name,
      @last_name,
      @email,
      @phone
    );

    -- Set the output parameter to the new user's ID
    SET @user_id = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT('Użytkownik utworzony pomyślnie.')
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END
```

CreateStudent

Procedura CreateStudent tworzy nowego studenta w systemie. ID studenta jest zwracane za pomocą @student_id

Argumenty:

@username - Nazwa użytkownika @first_name - Imię studenta @last_name - Nazwisko studenta @email - Adres email studenta @phone - Opcjonalny numer telefonu @street - Ulica zamieszkania @city - Miasto zamieszkania @postal_code - Kod pocztowy @country - Kraj zamieszkania @student_id - Wyjściowy identyfikator utworzonego studenta

```
CREATE PROCEDURE CreateStudent
  @username VARCHAR(30),
  @first_name NVARCHAR(30),
  @last_name NVARCHAR(30),
  @email VARCHAR(50),
```

```
  @phone VARCHAR(9) = NULL,
  @street VARCHAR(30),
  @city VARCHAR(30),
  @postal_code VARCHAR(30),
  @country VARCHAR(30),
  @user_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Create basic user first
    DECLARE @id INT;
    EXEC CreateBasicUser
      @username = @username,
      @first_name = @first_name,
      @last_name = @last_name,
      @email = @email,
      @phone = @phone,
      @user_id = @id OUTPUT;

    -- Insert student details
    INSERT INTO STUDENTS (
      student_id,
      street,
      city,
      postal_code,
      country
    ) VALUES (
      @user_id,
      @street,
      @city,
      @postal_code,
      @country
    );

    COMMIT TRANSACTION;
    PRINT("Student utworzony pomyślnie")
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END
```

CreateEmployee

Procedura CreateEmployee tworzy nowego pracownika w systemie. ID pracownika jest zwracane za pomocą @employee_id

Argumenty:

- @username - Nazwa użytkownika
- @first_name - Imię pracownika
- @last_name - Nazwisko pracownika
- @email - Adres email pracownika
- @phone - Opcjonalny numer telefonu
- @employee_type_id - Identyfikator typu pracownika
- @hire_date - Opcjonalna data zatrudnienia
- @birth_date - Opcjonalna data urodzenia
- @employee_id - Zwracany ID pracownika

```
CREATE PROCEDURE CreateEmployee
  @username VARCHAR(30),
  @first_name NVARCHAR(30),
  @last_name NVARCHAR(30),
  @email VARCHAR(50),
  @phone VARCHAR(9) = NULL,
  @employee_type_id INT,
  @hire_date DATE = NULL,
  @birth_date DATE = NULL,
  @employee_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate employee type exists
    IF NOT EXISTS (SELECT 1 FROM EMPLOYEE_TYPES WHERE type_id = @employee_type_id)
    BEGIN
      RAISERROR('Nieprawidłowy typ pracownika.', 16, 1);
      RETURN;
    END

    -- Create basic user first
    EXEC CreateBasicUser
      @username = @username,
      @first_name = @first_name,
      @last_name = @last_name,
      @email = @email,
      @phone = @phone,
      @employee_id = @user_id OUTPUT;

    -- Insert employee details
    INSERT INTO EMPLOYEES (
      emploee_id,
      type_id,
      hire_date,
      birth_date
```

```
    ) VALUES (
      @user_id,
      @employee_type_id,
      -- Check for NULL value
      COALESCE(@hire_date, GETDATE()),
      @birth_date
    );

    COMMIT TRANSACTION;
    PRINT("Pracownik utworzony pomyślnie.")
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END
```

Kursy

CreateCourse

Procedura `CreateCourse` tworzy nowy kurs na podstawie podanych danych oraz zwraca jego ID poprzez argument @course_id. Argumenty:

- @course_name - Nazwa kursu
- @course_description - Opis kursu
- @product_price MONEY - Cena kursu
- @vacancies - Ilość wolnych miejsc podczas zapisu na kurs
- @course_id - Zwracane ID kursu

```
CREATE PROCEDURE [dbo].[CreateCourse]
  @course_name NVARCHAR(50),
  @course_description TEXT = NULL,
  @product_price MONEY = 0,
  @vacancies INT,
  @release DATE,
  @course_id INT OUTPUT
AS
BEGIN
  BEGIN TRANSACTION;

  BEGIN TRY
    -- Add the product
    INSERT INTO PRODUCTS (type_id, price, total_vacancies, release)
    VALUES (3, @product_price, @vacancies, @release);
    -- Get created product ID, return it later
    SET @course_id = SCOPE_IDENTITY();

    -- Add the course
    INSERT INTO COURSES (course_id, course_name, course_description)
```

```
    VALUES (@course_id, @course_name, @course_description);

    COMMIT TRANSACTION;
    PRINT 'Kurs dodany pomyślnie.';
  END TRY
  BEGIN CATCH
    ROLLBACK TRANSACTION;
    THROW;
  END CATCH;
END;
```

CreateModule

Procedura `CreateModule` tworzy nowy moduł dla istniejącego kursu. Procedura sprawdza poprawność wprowadzanych danych i zwraca identyfikator nowo utworzonego modułu. ID modułu zwracane jest przez @module_id.

Argumenty:

- @course_id - Identyfikator istniejącego kursu, do którego zostanie dodany moduł
- @tutor_id - Identyfikator prowadzącego (nauczyciela) przypisanego do modułu
- @module_id - Zwracane ID modułu

```
CREATE PROCEDURE createModule
  @course_id INT,
  @tutor_id INT,
  @module_id INT OUTPUT
AS
BEGIN
  -- Validate that the course exists
  IF NOT EXISTS (SELECT 1 FROM COURSES WHERE course_id = @course_id)
  BEGIN
    RAISERROR('Kurs nie istnieje.', 16, 1);
    RETURN;
  END

  -- Validate that the tutor exists
  IF NOT EXISTS (SELECT 1 FROM EMPLOYEES WHERE emploee_id = @tutor_id)
  BEGIN
    RAISERROR('Tutor nie istnieje.', 16, 1);
    RETURN;
  END

  -- Insert the new module
  INSERT INTO MODULES (course_id, tutor_id)
  VALUES (@course_id, @tutor_id);

  -- Return the newly inserted module's ID
  SET @module_id = SCOPE_IDENTITY();
  PRINT("Moduł dodany pomyślnie.")
END
```

CreateModuleStationaryMeeting

Procedura CreateModuleStationaryMeeting tworzy nowe spotkanie stacjonarne dla modułu kursu. ID spotkania zwracane jest przez @meeting_id

Argumenty:

- @module_id - Identyfikator modułu kursu
- @tutor_id - Identyfikator prowadzącego
- @translator_id - Opcjonalny identyfikator tłumacza
- @meeting_name - Nazwa spotkania
- @term - Termin spotkania
- @duration - Czas trwania spotkania (domyślnie 1h 30min)
- @language - Język spotkania (domyślnie polski)
- @classroom - Numer sali, w której odbędzie się spotkanie
- @meeting_id - Zwracane ID spotkania

```
-- Stworzenie Stationary Meetingu dla modułu
CREATE PROCEDURE CreateModuleStationaryMeeting
  @module_id INT,
  @tutor_id INT,
  @translator_id INT = NULL,
  @meeting_name VARCHAR(30),
  @term DATETIME,
  @duration TIME(0) = '01:30:00',
  @language VARCHAR(30) = 'POLISH',
  @classroom VARCHAR(10),
  @meeting_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate module exists
    IF NOT EXISTS (SELECT 1 FROM MODULES WHERE module_id = @module_id)
    BEGIN
      RAISERROR('Moduł o podanym ID nie istnieje.', 16, 1);
      RETURN;
    END

    -- Validate tutor exists
    IF NOT EXISTS (SELECT 1 FROM EMPLOYEES WHERE emploee_id = @tutor_id)
    BEGIN
      RAISERROR('Tutor o podanym ID nie istnieje.', 16, 1);
      RETURN;
    END

    -- Insert meeting
```

```
    INSERT INTO MEETINGS (
      module_id,
      tutor_id,
      translator_id,
      meeting_name,
      term,
      duration,
      language
    ) VALUES (
      @module_id,
      @tutor_id,
      @translator_id,
      @meeting_name,
      @term,
      @duration,
      @language
    );

    -- Get the newly created meeting ID
    SET @meeting_id INT = SCOPE_IDENTITY();

    -- Insert stationary meeting details
    INSERT INTO STATIONARY_MEETINGS (
      meeting_id,
      classroom
    ) VALUES (
      @meeting_id,
      @classroom
    );

    COMMIT TRANSACTION;
    PRINT("Spotkanie dodane pomyślnie")
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END
```

CreateModuleSyncMeeting

Procedura CreateModuleSyncMeeting tworzy nowe spotkanie synchroniczne (na żywo) dla modułu kursu. ID spotkania zwracane jest za pomocą @meeting_id.

Argumenty:

- @module_id - Identyfikator modułu kursu
- @tutor_id - Identyfikator prowadzącego
- @translator_id - Opcjonalny identyfikator tłumacza
- @meeting_name - Nazwa spotkania
- @term - Termin spotkania

- @duration - Czas trwania spotkania (domyślnie 1h 30min)
- @language - Język spotkania (domyślnie polski)
- @meeting_url - Link do spotkania online
- @video_url - Opcjonalny link do nagrania wideo
- @meeting_id - Zwracane ID spotkania

```
CREATE PROCEDURE CreateModuleSyncMeeting
    @module_id INT,
    @tutor_id INT,
    @translator_id INT = NULL,
    @meeting_name VARCHAR(30),
    @term DATETIME,
    @duration TIME(0) = '01:30:00',
    @language VARCHAR(30) = 'POLISH',
    @meeting_url TEXT,
    @video_url TEXT = NULL,
    @meeting_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate module exists
    IF NOT EXISTS (SELECT 1 FROM MODULES WHERE module_id = @module_id)
    BEGIN
      RAISERROR('Module does not exist.', 16, 1);
      RETURN;
    END

    -- Validate tutor exists
    IF NOT EXISTS (SELECT 1 FROM EMPLOYEES WHERE emploee_id = @tutor_id)
    BEGIN
      RAISERROR('Tutor does not exist.', 16, 1);
      RETURN;
    END

    -- Insert meeting
    INSERT INTO MEETINGS (
      module_id,
      tutor_id,
      translator_id,
      meeting_name,
      term,
      duration,
      language
    ) VALUES (
      @module_id,
      @tutor_id,
      @translator_id,
      @meeting_name,
```

```
        @term,
        @duration,
        @language
      );

      -- Get the newly created meeting ID
      SET @meeting_id INT = SCOPE_IDENTITY();

      -- Insert sync meeting details
      INSERT INTO SYNC_MEETINGS (
        meeting_id,
        video_url,
        meeting_url
      ) VALUES (
        @meeting_id,
        @video_url,
        @meeting_url
      );

      COMMIT TRANSACTION;
      PRINT "Spoktanie synchroniczne utworzone pomyślnie."
    END TRY
    BEGIN CATCH
      IF @@TRANCOUNT > 0
          ROLLBACK TRANSACTION;
      THROW;
    END CATCH
  END
```

CreateModuleAsyncMeeting

Procedura CreateModuleAsyncMeeting tworzy nowe spotkanie asynchroniczne dla modułu kursu. ID spotkania zwracane jest za pomocą @meeting_id.

Argumenty:

- @module_id - Identyfikator modułu kursu
- @tutor_id - Identyfikator prowadzącego
- @translator_id - Opcjonalny identyfikator tłumacza
- @meeting_name - Nazwa spotkania
- @term - Termin spotkania
- @duration - Czas trwania spotkania (domyślnie 1h 30min)
- @language - Język spotkania (domyślnie polski)
- @meeting_url - Link do materiałów
- @meeting_id - Zwracane ID spotkania

```
CREATE PROCEDURE CreateModuleAsyncMeeting
  @module_id INT,
  @tutor_id INT,
  @translator_id INT = NULL,
```

```
  @meeting_name VARCHAR(30),
  @term DATETIME,
  @duration TIME(0) = '01:30:00',
  @language VARCHAR(30) = 'POLISH',
  @meeting_url TEXT,
  @meeting_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate module exists
    IF NOT EXISTS (SELECT 1 FROM MODULES WHERE module_id = @module_id)
    BEGIN
      RAISERROR('Module does not exist.', 16, 1);
      RETURN;
    END

    -- Validate tutor exists
    IF NOT EXISTS (SELECT 1 FROM EMPLOYEES WHERE emploee_id = @tutor_id)
    BEGIN
      RAISERROR('Tutor does not exist.', 16, 1);
      RETURN;
    END

    -- Insert meeting
    INSERT INTO MEETINGS (
      module_id,
      tutor_id,
      translator_id,
      meeting_name,
      term,
      duration,
      language
    ) VALUES (
      @module_id,
      @tutor_id,
      @translator_id,
      @meeting_name,
      @term,
      @duration,
      @language
    );

    -- Get the newly created meeting ID
    SET @meeting_id INT = SCOPE_IDENTITY();

    -- Insert async meeting details
    INSERT INTO ASYNC_MEETINGS (
      meeting_id,
      meeting_url
    ) VALUES (
```

```
      @meeting_id,
      @meeting_url
    );

    COMMIT TRANSACTION;
    PRINT("Spotkanie asynchroniczne utworzone pomyślnie.")
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END
```

Webinary

CreateWebinar

Procedura CreateWebinar tworzy nowy webinar w systemie. ID webinaru jest zwracane za pomocą @webinar_id

Argumenty:

- @tutor_id - Identyfikator prowadzącego webinar
- @translator_id - Opcjonalny identyfikator tłumacza
- @webinar_name - Nazwa webinaru
- @webinar_description - Opcjonalny opis webinaru
- @video_url - Opcjonalny link do nagrania wideo
- @webinar_duration - Czas trwania webinaru (domyślnie 1h 30min)
- @publish_date - Data publikacji webinaru (domyślnie aktualna data)
- @language - Język webinaru (domyślnie polski)
- @product_price - Cena webinaru (domyślnie 0)
- @vacancies - Liczba dostępnych miejsc (domyślnie 30)
- @webinar_id - Wyjściowy identyfikator utworzonego webinaru

```
CREATE PROCEDURE CreateWebinar
  @tutor_id INT,
  @translator_id INT = NULL,
  @webinar_name VARCHAR(50),
  @webinar_description TEXT = NULL,
  @video_url TEXT = NULL,
  @webinar_duration TIME(0) = '01:30:00',
  @publish_date DATETIME = NULL,
  @language VARCHAR(50) = 'POLISH',
  @product_price MONEY = 0,
  @vacancies INT = 30,
  @release INT,
  @webinar_id INT OUTPUT
AS
BEGIN
```

```
SET NOCOUNT ON;

BEGIN TRY
  BEGIN TRANSACTION;

  -- Validate tutor exists
  IF NOT EXISTS (SELECT 1 FROM EMPLOYEES WHERE emploee_id = @tutor_id)
  BEGIN
    RAISERROR('Tutor nie istnieje.', 16, 1);
    RETURN;
  END

  -- Validate translator exists if provided
  IF @translator_id IS NOT NULL AND
     NOT EXISTS (SELECT 1 FROM EMPLOYEES WHERE emploee_id = @translator_id)
  BEGIN
    RAISERROR('Tłumacz nie istnieje.', 16, 1);
    RETURN;
  END

  -- Insert product first (webinars are products)
  DECLARE @product_id INT;
  INSERT INTO PRODUCTS (
    type_id,  -- Assuming type_id 4 is for webinars
    price,
    vacancies,
    release
  ) VALUES (
    4,
    @product_price,
    @vacancies,
    @release
  );

  SET @product_id = SCOPE_IDENTITY();

  -- Insert webinar details
  INSERT INTO WEBINARS (
    webinar_id,
    tutor_id,
    translator_id,
    webinar_name,
    webinar_description,
    video_url,
    webinar_duration,
    publish_date,
    language
  ) VALUES (
    @product_id,
    @tutor_id,
    @translator_id,
    @webinar_name,
    @webinar_description,
    @video_url,
```

```
        @webinar_duration,
        COALESCE(@publish_date, GETDATE()),
        @language
      );

      -- Set the output parameter to the new webinar's ID
      SET @webinar_id = @product_id;

      COMMIT TRANSACTION;

      PRINT("Webinar utworzono pomyślnie.")
    END TRY
    BEGIN CATCH
      IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
      THROW;
    END CATCH
  END
```

Studia

CreateStudy

Procedura CreateStudy tworzy nowe studia w systemie. ID studium jest zwracane za pomocą @study_id

Argumenty:

- @study_name - Nazwa studiów
- @study_description - Opcjonalny opis studiów
- @product_price - Cena studiów (domyślnie 1000)
- @vacancies - Liczba dostępnych miejsc (domyślnie 30)
- @study_id - Wyjściowy identyfikator utworzonych studiów