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

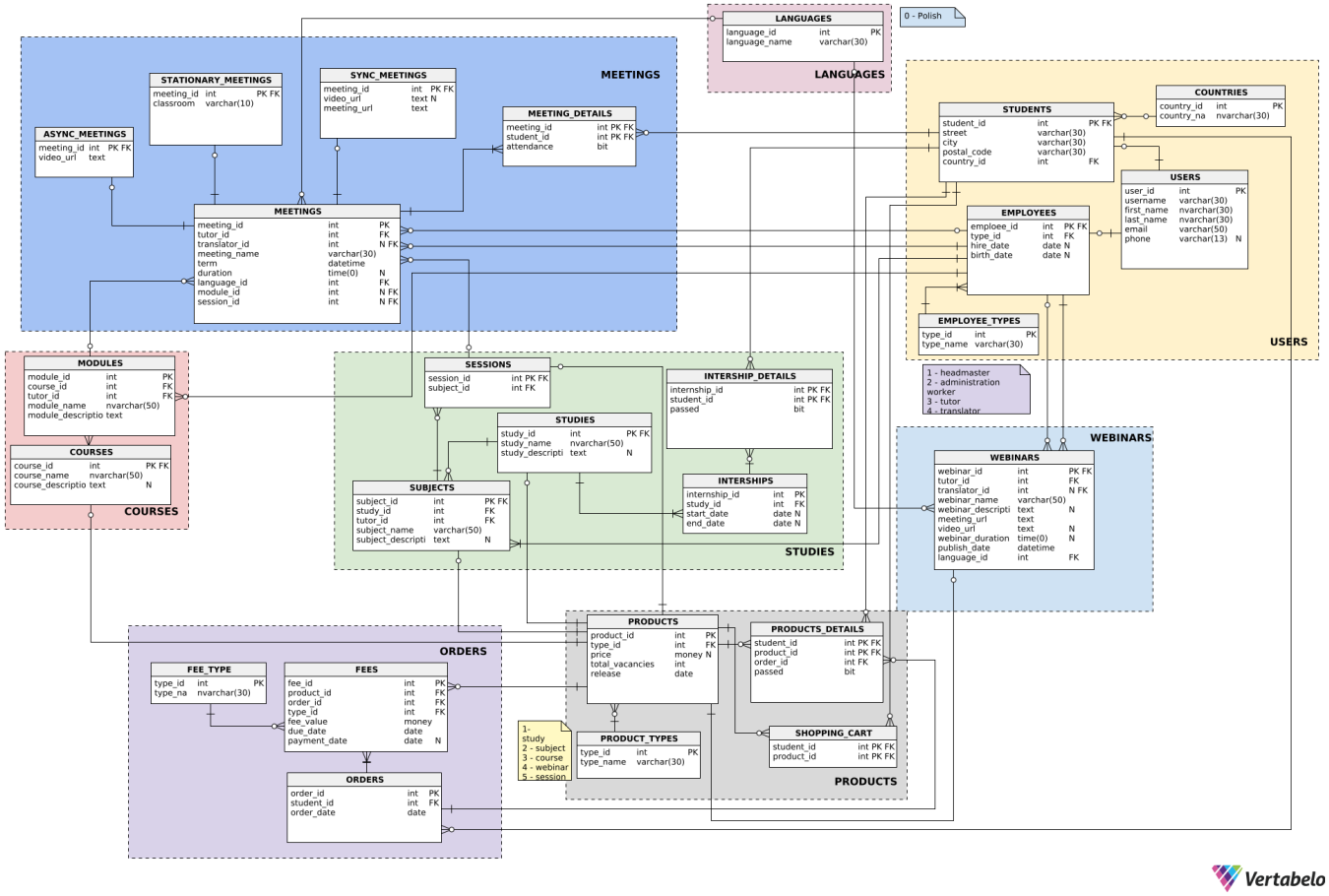
- tworzenie dyplomów potwierdzających ukończenie kursu / studium
- dodawanie / modyfikacja praktyk
- modyfikacja webinarów / kursów / studiów / przedmiotów
- dodawanie webinarów / kursów / studiów / przedmiotów
- dodawanie / usuwanie tłumacza do wybranych przedmiotów
- tworzenie sylabusu
- generowanie 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
country_id	int	Foreign Key
street	varchar(30)	
city	varchar(30)	

Column Name	Data Type	Properties
postal_code	varchar(30)	

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
employee_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)

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

```
-- Table: EMPLOYEES
CREATE TABLE EMPLOYEES (
```

```
    employee_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  (employee_id)
);
```

Tabela EMPLOYEES_TYPES

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
- type_name varchar(30) - nazwa pełnionej funkcji

Zdefiniowany typy:

- 1 - headmaster
- 2 - administration worker
- 3 - tutor
- 4 - translator

```
-- 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 - cena za produkt
    - warunek: price >= 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 PRODUCT_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
passed	bit	

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
- passed bit - indykator zaliczenia produktu edukacyjnego (1 - produkt zaliczony, 0 - produkt niezaliczony)

```
-- 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, identyfikator typu
- type_name varchar(30) - nazwa typu

Zdefiniowane typy:

- 1 - study
- 2 - subject
- 3 - course
- 4 - webinar
- 5 - session

```
-- 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 SHOPPING_CART

Column Name	Data Type	Properties
student_id	int	Primary Key Foreign Key
product_id	int	Primary Key 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: SHOPPING_CART
CREATE TABLE SHOPPING_CART (
    student_id int  NOT NULL,
    product_id int  NOT NULL,
    CONSTRAINT SHOPPING_CART_pk PRIMARY KEY  (student_id, product_id)
);
```

Kategoria Orders

Tabela ORDERS

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

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_TYPES

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

Zdefiniowane typy:

- 1 - session
- 2 - subject session
- 3 - study session
- 4 - study entry fee
- 5 - course
- 6 - course advance
- 7 - webinar

```
CREATE TABLE FEE_TYPES (
    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
language_id	int	Foreign Key
webinar_name	varchar(50)	

Column Name	Data Type	Properties
webinar_description	text	
meeting_url	text	
video_url	text	
webinar_duration	time(0)	
publish_date	datetime	

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_id int  NOT NULL DEFAULT 0,
   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	
advance_share	decimal(5,4)	

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
- advance_share - reprezentacja procentowej części ceny kursu, która jest uznawana za zaliczkę:
    - warunek: advance_share >= 0 and advance_share <= 1
    - default: 0.3000

```
-- Table: COURSES
CREATE TABLE COURSES (
    course_id int  NOT NULL,
    course_name nvarchar(50)  NOT NULL,
    course_description text  NULL,
    advance_share decimal(5,4) NOT NULL
      DEFAULT 0.3000
      CHECK (advance_share >= 0 and advance_share <= 1),
    CONSTRAINT COURSES_pk PRIMARY KEY  (course_id)
);
```

Tabela MODULES

Column Name	Data Type	Properties
module_id	int	Primary Key
course_id	int	Foreign Key
tutor_id	int	Foreign Key
module_name	nvarchar(50)	
module_description	text	

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 nvarchar(50) - nazwa modułu
- module_description text - opis modułu

```
-- Table: MODULES
CREATE TABLE MODULES (
    module_id int  NOT NULL IDENTITY,
    course_id int  NOT NULL,
    tutor_id int  NOT NULL,
    module_name nvarchar(50)  NOT NULL,
    module_description text  NOT NULL,
    CONSTRAINT MODULES_pk PRIMARY KEY  (module_id)
);
```

Kategoria STUDIES

Tabela STUDIES

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 - data rozpoczęcia praktyk
- end_date date - data zakończenia praktyk

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

Tabela INTERSHIP_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
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
meeting_id	int	Primary Key
tutor_id	int	Foreign Key
translator_id	int	Foreign Key
language_id	int	Foreign Key
module_id	int	Foreign Key
session_id	int	Foreign Key
meeting_name	varchar(30)	
term	datetime	
duration	time(0)	

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 1
- 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_id int  NOT NULL DEFAULT 0,
    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
video_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	employee_id	USERS	user_id
WEBINARS	tutor_id	EMPLOYEES	employee_id
FEES	order_id	ORDERS	order_id
FEES	product_id	PRODUCTS	product_id
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	employee_id
MEETINGS	translator_id	EMPLOYEES	employee_id
MODULES	course_id	COURSES	course_id
MODULES	tutor_id	EMPLOYEES	employee_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

Table Name	FK Column	Referenced Table	Referenced Column
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	employee_id
SUBJECTS	study_id	STUDIES	study_id
SYNC_MEETINGS	meeting_id	MEETINGS	meeting_id
STUDENTS	student_id	USERS	user_id
WEBINARS	translator_id	EMPLOYEES	employee_id
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.student_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,
        students.country_id AS country

    FROM students
    join users on users.user_id = students.student_id
    JOIN COUNTRIES on COUNTRIES.country_id = STUDENTS.country_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.employee_id as employee_id,
        users.first_name + ' ' + users.last_name AS name,
        emploee_type.type_name as rola

    FROM emploees
    join user on users.user_id = emploees.employee_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  AS tutor_name,
    Tu.last_name as tutor_last_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 AS tutor_name,
    Tu.last_name as tutor_last_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
    ),
    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
    ),
    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
        modules.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 = modules.module_id
    JOIN sync_course_MEETINGS_count AS syncmc
        ON syncmc.module_id = modules.module_id
    JOIN async_course_MEETINGS_count AS asyncmc
        ON asyncmc.module_id = modules.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.employee_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, 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, first_name,last_name as name
from ORDER_DETAILS
join USERS on USER.user_id=ORDER_DETAILS.student_id
```

Product_payment_information

Widok product_payment_information dla każdego produktu podaje jego typ, najpóźniejszy termin opłacenia całej aktywności, a także czy istnieje opcja wpłacenia zaliczki.

```
CREATE VIEW product_payment_information AS
SELECT
  p.product_id,
  pt.type_name,
  fee.due_date as due_date
  IIF(pt.type_name IN ('study', 'study',
  'course', 'session'), 1, 0) AS accepts_advance_payments
FROM
  products p
  join products_type pt on pt.product_id = p.product_id
  join fee on fee.product_id = p.product_id
```

Unpaid special permissions

Widok unpaid_special_permissions dla każdego klienta, któremu została odroczona płatność za zamówienie, pokazuje łączną kwotę jaką musi jeszcze dopłacić ze wszystkich zamówień.

Product_information

Widok product_information dla każdego produktu wylistowuje jego tytuł, opis, typ i cenę.

```
create view PRODUCT_OWNERS
select product_id,
  case
    WHEN pt.type_id = 1 then studies.study_name
    when pt.type_id = 2 then subjects.subject_name
    When pt.type_id = 3 then courses.course_name
    when pt.type_id = 4 then webinars.webinar_name
    when pt.type_id = 5 then null
  end as product_name,

  case
    WHEN pt.type_id = 1 then studies.study_description
    when pt.type_id = 2 then subjects.subject_description
    When pt.type_id = 3 then courses.course_description
    when pt.type_id = 4 then webinars.webinar_description
```

```
    when pt.type_id = 5 then null
  end as product_description

  price,
  pt.type_name,
from products p
join PRODUCT_TYPES pt on p.product_id = pt.product_id,
left join courses on courses.country_id = p.product_id,
left join subjects on subjects.subject_id = p.product_id,
left join studies on studies.study_id = p.product_id,
left join sessions on sessions.session_id = p.product_id,
left join webinars on webinars.webinar_id = p.product_id
```

Meetings

Attendance_list

Widok Attendance_list pokazuje id spotkań,jego nazwe, datę kiedy się spotkanie odbyło, rodzaj spotkania, jezyk spotkania, imie i nazwisko nauczyciela, imie i nazwisko tłumacza, imię i nazwisko studenta wraz z jego statusem obecności

```
create view ATTENDANCE_LIST as
SELECT
    m.meeting_id,
    m.meeting_name,
    m.term AS meeting_date,
    m.duration,
    CASE
        WHEN sm.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN sync.meeting_id IS NOT NULL THEN 'Synchronous'
        WHEN async.meeting_id IS NOT NULL THEN 'Asynchronous'
    END AS meeting_type,
    l.language_name,
    tutor.first_name + ' ' + tutor.last_name AS tutor_name,
    CASE
        WHEN trans.user_id IS NOT NULL
        THEN trans.first_name + ' ' + trans.last_name
        ELSE NULL
    END AS translator_name,
    u.first_name + ' ' + u.last_name AS student_name,
    u.email AS student_email,
    CASE
        WHEN md.attendance = 1 THEN 'Present'
        ELSE 'Absent'
    END AS attendance_status
FROM
    MEETINGS m
    INNER JOIN MEETING_DETAILS md ON m.meeting_id = md.meeting_id
    INNER JOIN USERS u ON md.student_id = u.user_id
    INNER JOIN USERS tutor ON m.tutor_id = tutor.user_id
    LEFT JOIN USERS trans ON m.translator_id = trans.user_id
```

```
    INNER JOIN LANGUAGES l ON m.language_id = l.language_id
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
WHERE
    m.term < GETDATE()
ORDER BY
    m.term DESC, m.meeting_id, u.last_name, u.first_name;
```

Future_meeting_attendee_count

Widok Future_meeting_attendee_count pokazuje id przyszłych spotkań, ich nazwę, termin, rodzaj oraz liczbę zapisanych na to spotkanie studentów

```
CREATE VIEW vw_future_meeting_attendee_count AS
SELECT
    m.meeting_id as meeting_id,
    m.meeting_name as meeting_name,
    m.term AS meeting_date,
    CASE
        WHEN sm.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN sync.meeting_id IS NOT NULL THEN 'Synchronous'
        WHEN async.meeting_id IS NOT NULL THEN 'Asynchronous'
    END AS meeting_type,
    COUNT(md.student_id) AS total_registered,
FROM
    MEETINGS m
    LEFT JOIN MEETING_DETAILS md ON m.meeting_id = md.meeting_id
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
WHERE
    m.term > GETDATE()
GROUP BY
    m.meeting_id,
    m.meeting_name,
    m.term,
    sm.meeting_id,
    sync.meeting_id,
    async.meeting_id
ORDER BY
    m.term;
```

Meeting_type

Widok Meeting_type wylistowuje id meetingu, jego nazwę, date, jego typ, czy należy do kursu, czy do studiów, nazwę studiów i nazwę kursów

```
CREATE VIEW vw_meeting_types AS
SELECT
    m.meeting_id,
    m.meeting_name,
    m.term,
    -- Determine meeting type
    CASE
        WHEN sm.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN sync.meeting_id IS NOT NULL THEN 'Synchronous'
        WHEN async.meeting_id IS NOT NULL THEN 'Asynchronous'
    END AS meeting_type,
    -- Check if meeting is part of a course
    CASE
        WHEN c.course_id IS NOT NULL THEN 'Yes'
        ELSE 'No'
    END AS is_part_of_course,
    c.course_name,
    -- Check if meeting is part of studies
    CASE
        WHEN s.study_id IS NOT NULL THEN 'Yes'
        ELSE 'No'
    END AS is_part_of_studies,
    sub.subject_name,
FROM
    MEETINGS m
    -- Join with meeting type tables
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
    -- Join with modules to get course information
    LEFT JOIN MODULES mod ON m.module_id = mod.module_id
    LEFT JOIN COURSES c ON mod.course_id = c.course_id
    -- Join with sessions to get study information
    LEFT JOIN SESSIONS ses ON m.session_id = ses.session_id
    LEFT JOIN SUBJECTS sub ON ses.subject_id = sub.subject_id
    LEFT JOIN STUDIES s ON sub.study_id = s.study_id
ORDER BY
    m.term, m.meeting_id;
```

Only_course_meeting

Widok Only_course_meeting pokazuje id spotkania, jego nazwę, date, czas trwania, nazwę kursu do którego należy, nazwę modułu, którego jest częścią, typ spotkania, imie i nazwisko nauczyciela oraz jezyk, w którym jest prowadzone spotkanie

```
CREATE VIEW Only_course_meeting AS
SELECT
    m.meeting_id,
    m.meeting_name,
    m.term AS meeting_date,
```

```
    m.duration,
    c.course_name,
    mod.module_name,
    CASE
        WHEN sm.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN sync.meeting_id IS NOT NULL THEN 'Synchronous'
        WHEN async.meeting_id IS NOT NULL THEN 'Asynchronous'
    END AS meeting_type,
    tutor.first_name + ' ' + tutor.last_name AS tutor_name,
    l.language_name
FROM
    MEETINGS m
    INNER JOIN MODULES mod ON m.module_id = mod.module_id
    INNER JOIN COURSES c ON mod.course_id = c.course_id
    INNER JOIN USERS tutor ON m.tutor_id = tutor.user_id
    INNER JOIN LANGUAGES l ON m.language_id = l.language_id
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
WHERE
    m.module_id IS NOT NULL
    AND m.session_id IS NULL
ORDER BY
    m.term;
```

Only_studies_meeting

Widok Only_studies_meeting pokazuje id spotkania, jego nazwę, date, czas trwania, nazwę studiów do którego należy, nazwę przedmiotu, którego jest częścią, numer zjazdu, do którego należy ,typ spotkania, imie i nazwisko nauczyciela oraz jezyk, w którym jest prowadzone spotkanie.

```
CREATE VIEW Only_studies_meeting AS
SELECT
    m.meeting_id,
    m.meeting_name,
    m.term AS meeting_date,
    m.duration,
    s.study_name,
    sub.subject_name,
    ses.session_id AS session_number,
    CASE
        WHEN sm.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN sync.meeting_id IS NOT NULL THEN 'Synchronous'
        WHEN async.meeting_id IS NOT NULL THEN 'Asynchronous'
    END AS meeting_type,
    tutor.first_name + ' ' + tutor.last_name AS tutor_name,
    l.language_name
FROM
    MEETINGS m
    INNER JOIN SESSIONS ses ON m.session_id = ses.session_id
    INNER JOIN SUBJECTS sub ON ses.subject_id = sub.subject_id
```

```
    INNER JOIN STUDIES s ON sub.study_id = s.study_id
    INNER JOIN USERS tutor ON m.tutor_id = tutor.user_id
    INNER JOIN LANGUAGES l ON m.language_id = l.language_id
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
WHERE
    m.session_id IS NOT NULL
    AND m.module_id IS NULL
ORDER BY
    m.term;
```

Room_schedule

Widok Room_schedule listuje id spotkania, jego nazwę, numer pokoju, termin startu i zakończenia spotkania, czas jego trwania, imie i nazwisko nauczyciela, nazwę kursu lub studiów z których dane spotkanie pochodzi, nazwę języka, w którym jest prowadzone spotkanie oraz określenie, czy spotkanie się już odbyło, czy dopiero odbędzie

```
CREATE VIEW Room_schedule AS
SELECT
    m.meeting_id,
    m.meeting_name,
    sm.classroom AS room,
    m.term AS start_time,
    DATEADD(MINUTE, DATEDIFF(MINUTE, '00:00:00', m.duration), m.term) AS end_time,
    m.duration,
    tutor.first_name + ' ' + tutor.last_name AS tutor_name,
    CASE
        WHEN mod.module_id IS NOT NULL THEN 'Course: ' + c.course_name
        WHEN ses.session_id IS NOT NULL THEN 'Study: ' + s.study_name
    END AS meeting_context,
    l.language_name,
    CASE
        WHEN m.term < GETDATE() THEN 'Past'
        ELSE 'Upcoming'
    END AS meeting_status
FROM
    MEETINGS m
    INNER JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    INNER JOIN USERS tutor ON m.tutor_id = tutor.user_id
    INNER JOIN LANGUAGES l ON m.language_id = l.language_id
    LEFT JOIN MODULES mod ON m.module_id = mod.module_id
    LEFT JOIN COURSES c ON mod.course_id = c.course_id
    LEFT JOIN SESSIONS ses ON m.session_id = ses.session_id
    LEFT JOIN SUBJECTS sub ON ses.subject_id = sub.subject_id
    LEFT JOIN STUDIES s ON sub.study_id = s.study_id
ORDER BY
    sm.classroom,
    m.term;
```

Studies

Study_information

Widok Study_information pokazuje id studiów, ich nazwe, opis, date rozpoczęcia pierwszych zajęć i date rozpoczęcia ostatnich zajęc, ilość przedmiotów, liczbę wszystkich zajęć, liczbę zajęć stacjonarnych, liczbe zajęć online oraz liczbę asynchronicznych zajęć online, liczbę praktyk podpiętych pod studia, limit miejsc na studia, opłatę za wpis oraz liczbę zapisanych studentów na dane studia.

```
CREATE VIEW Study_information AS
WITH study_timeframe AS (
    SELECT
        s.study_id,
        MIN(m.term) AS start_date,
        MAX(m.term) AS end_date
    FROM
        STUDIES s
        JOIN SUBJECTS sub ON s.study_id = sub.study_id
        JOIN SESSIONS ses ON sub.subject_id = ses.subject_id
        JOIN MEETINGS m ON ses.session_id = m.session_id
    GROUP BY
        s.study_id
),

SELECT
    s.study_id,
    s.study_name,
    s.study_description,
    tf.start_date AS study_start,
    tf.end_date AS study_end,
    -- Count of subjects
    COUNT(DISTINCT sub.subject_id) AS number_of_subjects,
    -- Count of all meetings
    COUNT(DISTINCT m.meeting_id) AS total_meetings,
    -- Count meetings by type
    SUM(CASE WHEN sm.meeting_id IS NOT NULL THEN 1 ELSE 0 END) AS stationary_meetings,
    SUM(CASE WHEN sync.meeting_id IS NOT NULL THEN 1 ELSE 0 END) AS online_meetings,
    SUM(CASE WHEN async.meeting_id IS NOT NULL THEN 1 ELSE 0 END) AS async_meetings,
    -- Count of internships
    (
        SELECT COUNT(*)
        FROM INTERSHIPS i
        WHERE i.study_id = s.study_id
    ) AS number_of_internships,
    -- Available languages
    m.language_id
    -- Product information
    p.total_vacancies AS place_limit,
    p.price AS entry_fee,
    -- Calculate occupancy
    (
        SELECT COUNT(DISTINCT pd.student_id)
```

```
        FROM PRODUCTS_DETAILS pd
        WHERE pd.product_id = s.study_id
    ) AS current_enrollment
FROM
    STUDIES s
    LEFT JOIN study_timeframe tf ON s.study_id = tf.study_id
    LEFT JOIN SUBJECTS sub ON s.study_id = sub.study_id
    LEFT JOIN SESSIONS ses ON sub.subject_id = ses.subject_id
    LEFT JOIN MEETINGS m ON ses.session_id = m.session_id
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
    LEFT JOIN PRODUCTS p ON s.study_id = p.product_id
GROUP BY
    s.study_id,
    s.study_name,
    s.study_description,
    tf.start_date,
    tf.end_date,
    sl.available_languages,
    p.total_vacancies,
    p.price;
```

Study_internship_information

Widok Study_internship_information wypisuje infomacje o praktykach. Wilistowuje id praktyk, nazwę studiów, z których te praktyki pochodzą, date ich rozpoczęcia i zakończenia, długość ich trwania, studentów zapisanych na te praktyki, ilość studentów, którzy zaliczyli praktyki, status praktyk.

```
CREATE VIEW study_internship_information AS
SELECT
    i.internship_id,
    s.study_id,
    s.study_name,
    i.start_date,
    i.end_date,
    DATEDIFF(day, i.start_date, i.end_date) AS duration_days,
    -- Count total students assigned
    COUNT(DISTINCT id.student_id) AS total_students,
    -- Count passed students
    SUM(CASE WHEN id.passed = 1 THEN 1 ELSE 0 END) AS students_passed,
    -- Status of internship
    CASE
        WHEN i.end_date < GETDATE() THEN 'Completed'
        WHEN i.start_date > GETDATE() THEN 'Upcoming'
        ELSE 'In Progress'
    END AS internship_status,
    -- Time until start or since end
FROM
    INTERSHIPS i
    INNER JOIN STUDIES s ON i.study_id = s.study_id
```

```
    LEFT JOIN INTERSHIP_DETAILS id ON i.internship_id = id.internship_id
GROUP BY
    i.internship_id,
    s.study_id,
    s.study_name,
    i.start_date,
    i.end_date
ORDER BY
    i.start_date DESC;
```

Study_meeting_information

Widok Study_meeting_information wylistowuje informacje na temat spotkań organizowanych w ramach studiów. wylisowuje id studiów, ich nazwe, id przedmiotu wraz z jego nazwą, id sesji, organizowanych dla danych przedmiotów, id spotkania, nazwę spotkania, datę spotkania oraz czas jego trwania, typ spotkania, link do spotkania albo numer sali stosownie do typu spotkania, status odbycia się spotkania oraz liczbe zapisanych studentów na to spotkanie

```
CREATE VIEW study_meeting_information AS
SELECT
    s.study_id,
    s.study_name,
    sub.subject_id,
    sub.subject_name,
    ses.session_id,
    m.meeting_id,
    m.meeting_name,
    m.term AS meeting_date,
    m.duration,
    -- Meeting type determination
    CASE
        WHEN sm.meeting_id IS NOT NULL THEN 'Stationary'
        WHEN sync.meeting_id IS NOT NULL THEN 'Synchronous'
        WHEN async.meeting_id IS NOT NULL THEN 'Asynchronous'
    END AS meeting_type,
    -- Location/URL information based on type
    CASE
        WHEN sm.meeting_id IS NOT NULL THEN sm.classroom
        WHEN sync.meeting_id IS NOT NULL THEN sync.meeting_url
        WHEN async.meeting_id IS NOT NULL THEN async.video_url
    END AS meeting_location,
    -- Meeting status
    CASE
        WHEN m.term < GETDATE() THEN 'Past'
        ELSE 'Upcoming'
    END AS meeting_status,
    -- Current enrollment count
    (
        SELECT COUNT(*)
        FROM MEETING_DETAILS md
        WHERE md.meeting_id = m.meeting_id
```

```
    ) AS current_enrollment
FROM
    STUDIES s
    INNER JOIN SUBJECTS sub ON s.study_id = sub.study_id
    INNER JOIN SESSIONS ses ON sub.subject_id = ses.subject_id
    INNER JOIN MEETINGS m ON ses.session_id = m.session_id
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
    LEFT JOIN PRODUCTS p ON ses.session_id = p.product_id
ORDER BY
    s.study_id,
    ses.session_id,  -- Added to ordering
    m.term;
```

Study_offers

Widok study_offers wylistowuje informacje o oferowanych aktualnie studiach. Podaje informacje o id studiów, ich nazwie, ich opisie, dacie rozpoczęcia pierwszych i ostatnich zajęć, ilości przedmiotów oraz spotkań z podziałem na typy spotkań, informacje o ilości wszystkich, zajętych oraz wolnych miejsc na studiach oraz opłacie za nie oraz statusie czsowym studiów.

```
CREATE VIEW study_offers AS
WITH study_timeframe AS (
    SELECT
        s.study_id,
        MIN(m.term) AS start_date,
        MAX(m.term) AS end_date
    FROM
        STUDIES s
        JOIN SUBJECTS sub ON s.study_id = sub.study_id
        JOIN SESSIONS ses ON sub.subject_id = ses.subject_id
        JOIN MEETINGS m ON ses.session_id = m.session_id
    GROUP BY
        s.study_id
),
study_languages AS (
    SELECT
        s.study_id,
        STRING_AGG(DISTINCT l.language_name, ', ') AS available_languages
    FROM
        STUDIES s
        JOIN SUBJECTS sub ON s.study_id = sub.study_id
        JOIN SESSIONS ses ON sub.subject_id = ses.subject_id
        JOIN MEETINGS m ON ses.session_id = m.session_id
        JOIN LANGUAGES l ON m.language_id = l.language_id
    GROUP BY
        s.study_id
),
current_enrollment AS (
    SELECT
```

```
        product_id,
        COUNT(DISTINCT student_id) AS enrolled_students
    FROM
        PRODUCTS_DETAILS
    GROUP BY
        product_id
)
SELECT
    s.study_id,
    s.study_name,
    s.study_description,
    tf.start_date,
    tf.end_date,
    -- Count of subjects
    COUNT(DISTINCT sub.subject_id) AS number_of_subjects,
    -- Count meetings by type
    COUNT(DISTINCT m.meeting_id) AS total_meetings,
    SUM(CASE WHEN sm.meeting_id IS NOT NULL THEN 1 ELSE 0 END) AS stationary_meetings,
    SUM(CASE WHEN sync.meeting_id IS NOT NULL THEN 1 ELSE 0 END) AS online_meetings,
    SUM(CASE WHEN async.meeting_id IS NOT NULL THEN 1 ELSE 0 END) AS async_meetings,
    -- Count of internships
    (
        SELECT COUNT(*)
        FROM INTERSHIPS i
        WHERE i.study_id = s.study_id
    ) AS number_of_internships,
    p.total_vacancies AS total_places,
    COALESCE(ce.enrolled_students, 0) AS current_enrollment,
    p.total_vacancies - COALESCE(ce.enrolled_students, 0) AS available_places,
    p.price AS entry_fee,
    -- Status information
    CASE
        WHEN tf.start_date > GETDATE() THEN 'Upcoming'
        ELSE 'In Progress'
    END AS study_status
FROM
    STUDIES s
    JOIN PRODUCTS p ON s.study_id = p.product_id
    LEFT JOIN study_timeframe tf ON s.study_id = tf.study_id
    LEFT JOIN study_languages sl ON s.study_id = sl.study_id
    LEFT JOIN current_enrollment ce ON s.study_id = ce.product_id
    LEFT JOIN SUBJECTS sub ON s.study_id = sub.study_id
    LEFT JOIN SESSIONS ses ON sub.subject_id = ses.subject_id
    LEFT JOIN MEETINGS m ON ses.session_id = m.session_id
    LEFT JOIN STATIONARY_MEETINGS sm ON m.meeting_id = sm.meeting_id
    LEFT JOIN SYNC_MEETINGS sync ON m.meeting_id = sync.meeting_id
    LEFT JOIN ASYNC_MEETINGS async ON m.meeting_id = async.meeting_id
WHERE
    p.total_vacancies - COALESCE(ce.enrolled_students, 0) > 0  -- Only studies with available places
    AND (tf.end_date IS NULL OR tf.end_date >= GETDATE())      -- Only current or future studies
GROUP BY
    s.study_id,
    s.study_name,
    s.study_description,
```

```
    tf.start_date,
    tf.end_date,
    sl.available_languages,
    p.total_vacancies,
    p.price,
    ce.enrolled_students
ORDER BY
    tf.start_date;
```

Study_passed

Widok Study_passed wylistowuje id studiów, ich nazwe, id studenta zapisanego na te studia, jego imie i nazwisko wraz z informacją, czy zdał dane studia i jego frekwencją w wykładach.

```
CREATE VIEW study_passes AS
SELECT
    s.study_id,
    s.study_name,
    u.user_id AS student_id,
    u.email,
    -- Study completion status directly from PRODUCTS_DETAILS
    CASE
        WHEN pd.passed = 1 THEN 'Passed'
        ELSE 'Not Passed'
    END AS study_status,
    -- Meeting attendance details
    COUNT(DISTINCT m.meeting_id) AS total_available_meetings,
    COUNT(DISTINCT CASE WHEN md.attendance = 1 THEN m.meeting_id END) AS meetings_attended,
    CASE
        WHEN COUNT(DISTINCT m.meeting_id) = 0 THEN 0
        ELSE CAST(COUNT(DISTINCT CASE WHEN md.attendance = 1 THEN m.meeting_id END) AS FLOAT) /
             COUNT(DISTINCT m.meeting_id) * 100
    END AS attendance_rate,
    -- Internship completion details
    id.passed as internships_passed
FROM
    STUDIES s
    INNER JOIN PRODUCTS_DETAILS pd ON s.study_id = pd.product_id
    INNER JOIN ORDERS o ON pd.order_id = o.order_id
    INNER JOIN USERS u ON pd.student_id = u.user_id
    -- Meeting attendance
    LEFT JOIN SUBJECTS sub ON s.study_id = sub.study_id
    LEFT JOIN SESSIONS ses ON sub.subject_id = ses.subject_id
    LEFT JOIN MEETINGS m ON ses.session_id = m.session_id
    LEFT JOIN MEETING_DETAILS md ON m.meeting_id = md.meeting_id AND md.student_id = pd.student_id
    -- Internship completion
    LEFT JOIN INTERSHIPS i ON s.study_id = i.study_id
    LEFT JOIN INTERSHIP_DETAILS id ON i.internship_id = id.internship_id AND id.student_id = pd.student_id
GROUP BY
    s.study_id,
    s.study_name,
```

```
    u.user_id,
    u.first_name,
    u.last_name,
    u.email,
    pd.order_id,
    o.order_date,
    pd.passed
ORDER BY
    s.study_name,
    u.last_name,
    u.first_name;
```

Study_session_schedule

Widok Study_session_schedule wylistowuje wszystkie zjazdy wraz z czasem rozpoczęcia pierwszych i ostatnich zajęć wramach tego zjazdu

```
CREATE VIEW study_session_schedule AS
  SELECT
  s.study_id,
  s.session_id,
  MIN(m.term) AS start_time,
  MAX(m.term) AS end_time
  FROM
    sessions s
    join meetings m on m.session_id =  s.session_id
  GROUP BY
    s.study_id,
    s.session_id
```

Study_syllabus

```
CREATE VIEW study_syllabus AS
SELECT
  s.study_id AS study_id,
  sub.subject_id AS subject_id,
  s.stady_name AS name,
  s.stady_description AS description,
  COUNT(m.meeting_id) AS meeting_count,
  MIN(m.start_time) AS start_time,
  MAX(m.end_time) AS end_time
  FROM
    stadies s
    JOIN subjects sub on sub.study_id = s.study_id
    join sessions on sub.subject_id = sessions.subject_id
    JOIN meetings m ON m.session_id = sessions.session_id
  GROUP BY
    s.study_id
```

Product_payment_information

Widok product_payment_information dla każdego produktu podaje jego typ, najpóźniejszy termin opłacenia całej aktywności, a także czy istnieje opcja wpłacenia zaliczki.

```
CREATE VIEW product_payment_information AS
SELECT
  p.product_id,
  pt.type_name,
  fee.due_date as due_date
  IIF(pt.type_name IN ('study', 'study',
  'course', 'session'), 1, 0) AS accepts_advance_payments
FROM
  products p
  join products_type pt on pt.product_id = p.product_id
  join fee on fee.product_id = p.product_id
```

Unpaid special permissions

Widok unpaid_special_permissions dla każdego klienta, któremu została odroczona płatność za zamówienie, pokazuje łączną kwotę jaką musi jeszcze dopłacić ze wszystkich zamówień.

Product_information

Widok product_information dla każdego produktu wylistowuje jego tytuł, opis, typ i cenę.

```
create view PRODUCT_OWNERS
select product_id,
  case
    WHEN pt.type_id = 1 then studies.study_name
    when pt.type_id = 2 then subjects.subject_name
    When pt.type_id = 3 then courses.course_name
    when pt.type_id = 4 then webinars.webinar_name
    when pt.type_id = 5 then null
  end as product_name,

  case
    WHEN pt.type_id = 1 then studies.study_description
    when pt.type_id = 2 then subjects.subject_description
    When pt.type_id = 3 then courses.course_description
    when pt.type_id = 4 then webinars.webinar_description
    when pt.type_id = 5 then null
  end as product_description

  price,
  pt.type_name,
from products p
join PRODUCT_TYPES pt on p.product_id = pt.product_id,
left join courses on courses.country_id = p.product_id,
```

```
left join subjects on subjects.subject_id = p.product_id,
left join studies on studies.study_id = p.product_id,
left join sessions on sessions.session_id = p.product_id,
left join webinars on webinars.webinar_id = p.product_id
```

Procedury

Sprawdzanie poprawności danych przed operacją

CheckWebinarExists

Procedura `CheckWebinarExists` sprawdza czy webinar o podanym ID istnieje w bazie. Jeśli nie - zgłasza błąd.

Argumenty:

- @webinar_id INT - Identyfikator webinaru do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckWebinarExists]
  @webinar_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM WEBINARS WHERE webinar_id = @webinar_id)
  BEGIN
    THROW 50001, 'Webinar nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

CheckOrderExists

Procedura `CheckOrderExists` weryfikuje czy zamówienie o podanym ID istnieje w bazie. W przypadku braku zamówienia zgłasza błąd.

Argumenty:

- @order_id INT - ID zamówienia do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckOrderExists]
  @order_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM ORDERS WHERE order_id = @order_id)
  BEGIN
    THROW 50001, 'Zamówienie nie istnieje.', 1;
    RETURN;
  END
```

```
END;
GO
```

CheckStudyExists

Procedura `CheckStudyExists` weryfikuje czy studia o podanym ID istnieją w bazie. W przypadku braku studiów zgłasza błąd.

Argumenty:

- @study_id INT - ID studiów do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckStudyExists]
  @study_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM STUDIES WHERE study_id = @study_id)
  BEGIN
    THROW 50001, 'Studia nie istnieją.', 1;
    RETURN;
  END
END;
GO
```

CheckProductExists

Procedura `CheckProductExists` weryfikuje czy produkt o podanym ID istnieje w bazie. W przypadku braku produktu zgłasza błąd.

Argumenty:

- @product_id INT - ID produktu do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckProductExists]
  @product_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM PRODUCTS WHERE product_id = @product_id)
  BEGIN
    THROW 50001, 'Produkt nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

CheckEmployeeExists

Procedura `CheckEmployeeExists` sprawdza czy pracownik o podanym ID istnieje w bazie. Jeśli nie - zgłasza błąd.

Argumenty:

- @employee_id INT - Identyfikator pracownika do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckEmployeeExists]
  @employee_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM EMPLOYEES WHERE employee_id = @employee_id)
  BEGIN
    THROW 50001, 'Pracownik nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

CheckLanguageExists

Procedura `CheckLanguageExists` weryfikuje czy język o podanym ID istnieje w bazie. W przypadku braku języka zgłasza błąd.

Argumenty:

- @language_id INT - ID języka do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckLanguageExists]
  @language_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM LANGUAGES WHERE language_id = @language_id)
  BEGIN
    THROW 50001, 'Język nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

CheckModuleExists

Procedura `CheckModuleExists` weryfikuje czy moduł o podanym ID istnieje w bazie. W przypadku braku modułu zgłasza błąd.

Argumenty:

- @module_id INT - ID modułu do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckModuleExists]
  @module_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM MODULES WHERE module_id = @module_id)
  BEGIN
    THROW 50001, 'Moduł nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

CheckMeetingExists

Procedura `CheckMeetingExists` sprawdza czy spotkanie o podanym ID istnieje w bazie. Jeśli nie - zgłasza błąd.

Argumenty:

- @meeting_id INT - Identyfikator spotkania do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckMeetingExists]
  @meeting_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM MEETINGS WHERE meeting_id = @meeting_id)
  BEGIN
    THROW 50001, 'Spotkanie nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

CheckSessionExists

Procedura `CheckSessionExists` sprawdza czy sesja o podanym ID istnieje w bazie. Jeśli nie - zgłasza błąd.

Argumenty:

- @session_id INT - Identyfikator sesji do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckSessionExists]
  @session_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM SESSIONS WHERE session_id = @session_id)
  BEGIN
    THROW 50001, 'Sesja nie istnieje.', 1;
```

```
    RETURN;
  END
END;
GO
```

CheckStudyExists

Procedura `CheckStudyExists` sprawdza czy studium o podanym ID istnieje w bazie. Jeśli nie - zgłasza błąd.

Argumenty:

- @study_id INT - Identyfikator studium do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckStudyExists]
  @study_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM STUDIES WHERE study_id = @study_id)
  BEGIN
    THROW 50001, 'Studia nie istnieją.', 1;
    RETURN;
  END
END;
GO
```

CheckCountryExists

Procedura `CheckCountryExists` sprawdza czy kraj o podanym ID istnieje w bazie. Jeśli nie - zgłasza błąd.

Argumenty:

- @country_id INT - Identyfikator kraju do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckCountryExists]
  @country_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM COUNTRIES WHERE country_id = @country_id)
  BEGIN
    THROW 50001, 'Państwo nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

CheckFeeExists

Procedura `CheckFeeExists` sprawdza czy należność o podanym ID istnieje w bazie. Jeśli nie - zgłasza błąd.

Argumenty:

- @fee_id INT - Identyfikator opłaty do sprawdzenia

```
CREATE PROCEDURE [dbo].[CheckFeeExists]
  @fee_id INT
AS
BEGIN
  IF NOT EXISTS (SELECT 1 FROM FEES WHERE fee_id = @fee_id)
  BEGIN
    THROW 50001, 'Należność nie istnieje.', 1;
    RETURN;
  END
END;
GO
```

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 [dbo].[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
```

```
        THROW 50000, 'Niepoprawny format adresu email.', 1;
        RETURN;
      END

      -- Validate phone number if provided
      IF @phone IS NOT NULL AND (LEN(@phone) != 9 OR ISNUMERIC(@phone) = 0)
      BEGIN
        THROW 50000, 'Niepoprawny format numeru telefonu.', 1;
        RETURN;
      END

      -- Check for existing email
      IF EXISTS (SELECT 1 FROM USERS WHERE email = @email)
      BEGIN
        THROW 50000, 'Email został już przypisany do innego użytkownika.', 1;
        RETURN;
      END

      -- Check for existing phone if provided
      IF @phone IS NOT NULL AND EXISTS (SELECT 1 FROM USERS WHERE phone = @phone)
      BEGIN
        THROW 50000, 'Numer telefonu został już przypisany do innego użytkownika.', 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
  GO
```

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 [dbo].[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_id INT = 0,
  @user_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate country exists
    EXEC [dbo].[CheckCountryExists] @country_id;

    -- 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;

    SET @user_id = @id;

    -- Insert student details
    INSERT INTO STUDENTS (
      student_id,
      street,
      city,
      postal_code,
      country_id
```

```
    ) VALUES (
      @user_id,
      @street,
      @city,
      @postal_code,
      @country_id
    );



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

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 [dbo].[CreateEmployee]
  @username VARCHAR(30),
  @first_name NVARCHAR(30),
  @last_name NVARCHAR(30),
  @email VARCHAR(50),
  @phone VARCHAR(9) = NULL,
  @employee_type_id INT,
  @birth_date DATE = NULL,
  @hire_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
      THROW 50000, 'Nieprawidłowy typ pracownika.', 1;
      RETURN;
    END

    -- 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;

    -- Set the returned value
    SET @employee_id = @id;

    -- Insert employee details
    INSERT INTO EMPLOYEES (
      employee_id,
      type_id,
      hire_date,
      birth_date
    ) VALUES (
      @employee_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
GO
```

LinkTranslatorToWebinar

Procedura `LinkTranslatorToWebinar` przypisuje tłumacza do istniejącego webinaru.

Argumenty:

- @webinar_id INT - Identyfikator webinaru, do którego ma zostać przypisany tłumacz
- @translator_id INT - Identyfikator tłumacza, który ma zostać przypisany do webinaru

```
CREATE PROCEDURE [dbo].[LinkTranslatorToWebinar]
  @webinar_id INT,
  @translator_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate webinar exists
    EXEC [dbo].[CheckWebinarExists] @webinar_id

    -- Validate translator exists
    EXEC [dbo].[CheckEmployeeExists] @translator_id

    -- Update the webinar to link the translator
    UPDATE WEBINARS
    SET translator_id = @translator_id
    WHERE webinar_id = @webinar_id;

    COMMIT TRANSACTION;
    PRINT 'Tłumacz został przypisany do webinaru pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

LinkTranslatorToMeeting

Procedura `LinkTranslatorToMeeting` przypisuje tłumacza do istniejącego spotkania.

Argumenty:

- @meeting_id INT - Identyfikator spotkania, do którego ma zostać przypisany tłumacz
- @translator_id INT - Identyfikator tłumacza, który ma zostać przypisany do spotkania

```
CREATE PROCEDURE [dbo].[LinkTranslatorToMeeting]
  @meeting_id INT,
```

```
  @translator_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate meeting exists
    EXEC [dbo].[CheckMeetingExists] @meeting_id

    -- Validate translator exists
    EXEC [dbo].[CheckEmployeeExists] @translator_id

    -- Update the meeting to link the translator
    UPDATE MEETINGS
    SET translator_id = @translator_id
    WHERE meeting_id = @meeting_id;

    COMMIT TRANSACTION;
    PRINT 'Tłumacz został przypisany do spotkania pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

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 [dbo].[CreateModule]
  @course_id INT,
  @tutor_id INT,
  @module_name NVARCHAR(50),
  @module_description TEXT,
  @module_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate that the course exists
    IF NOT EXISTS (SELECT 1 FROM COURSES WHERE course_id = @course_id)
    BEGIN
```

```
        THROW 50000, 'Kurs nie istnieje.', 1;
        RETURN;
      END

      -- Validate that the tutor exists
      EXEC [dbo].[CheckEmployeeExists] @tutor_id;

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

      -- Return the newly inserted module's ID
      SET @module_id = SCOPE_IDENTITY();

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

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 employee_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 employee_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 employee_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 [dbo].[CreateWebinar]
  @tutor_id INT,
  @translator_id INT = NULL,
  @webinar_name VARCHAR(50),
  @webinar_description TEXT = NULL,
  @video_url TEXT = NULL,
  @meeting_url TEXT = NULL,
  @webinar_duration TIME(0) = '01:30:00',
  @publish_date DATETIME = NULL,
  @language_id INT = 0,
  @product_price MONEY = 0,
  @vacancies INT = 30,
  @release DATE,
  @webinar_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate tutor exists
    EXEC [dbo].[CheckEmployeeExists] @tutor_id;

    -- Validate translator exists if provided
    IF @translator_id IS NOT NULL
    BEGIN
      EXEC [dbo].[CheckEmployeeExists] @translator_id;
    END

    -- Validate language exists
    EXEC [dbo].[CheckLanguageExists] @language_id;

    -- Insert product first (webinars are products)
    DECLARE @product_id INT;
    INSERT INTO PRODUCTS (
      type_id,
      price,
      total_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,
      meeting_url,
      webinar_duration,
      publish_date,
      language_id
    ) VALUES (
      @product_id,
      @tutor_id,
      @translator_id,
      @webinar_name,
      @webinar_description,
      @video_url,
      @meeting_url,
      @webinar_duration,
      COALESCE(@publish_date, GETDATE()),
      @language_id
    );

    -- 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
GO
```

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

```
CREATE PROCEDURE [dbo].[CreateStudy]
  @study_name NVARCHAR(50),
  @study_description TEXT = NULL,
  @product_price MONEY,
  @vacancies INT = 30,
  @release DATE,
  @study_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Insert product first (studies are products)
    INSERT INTO PRODUCTS (
      type_id,
      price,
      total_vacancies,
      release
    ) VALUES (
      1,
      @product_price,
      @vacancies,
      @release
    );

    -- Get the newly created product ID
    SET @study_id = SCOPE_IDENTITY();

    -- Insert study details
    INSERT INTO STUDIES (
      study_id,
      study_name,
      study_description
    ) VALUES (
      @study_id,
      @study_name,
      @study_description
    );

    COMMIT TRANSACTION;

    PRINT('Studium utworzono pomyślnie')
  END TRY
  BEGIN CATCH
```

```
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END
GO
```

CreateSubject

Procedura `CreateSubject` tworzy nowy przedmiot w ramach istniejących studiów. ID przedmiotu zwracane jest za pomocą @subject_id

Argumenty:

- @study_id - Identyfikator studiów
- @tutor_id - Identyfikator prowadzącego przedmiot
- @subject_name - Nazwa przedmiotu
- @subject_description - Opcjonalny opis przedmiotu
- @product_price - Cena przedmiotu
- @vacancies - Liczba dostępnych miejsc (domyślnie 30)
- @release - Data udostępnienia przedmiotu
- @subject_id - Zwracane ID przedmiotu

```
CREATE PROCEDURE [dbo].[CreateSubject]
  @study_id INT,
  @tutor_id INT,
  @subject_name NVARCHAR(50),
  @subject_description TEXT = NULL,
  @product_price MONEY = 0,
  @vacancies INT = 30,
  @release DATE,
  @subject_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate study exists
    EXEC [dbo].[CheckStudyExists] @study_id

    -- Validate tutor exists
    EXEC [dbo].[CheckEmployeeExists] @tutor_id

    -- Insert product first (subjects are products)
    INSERT INTO PRODUCTS (
      type_id,  -- Assuming type_id 2 is for subjects
      price,
      total_vacancies,
```

```
        release
      ) VALUES (
        2,
        @product_price,
        @vacancies,
        @release
      );

      -- Get the newly created product ID
      SET @subject_id = SCOPE_IDENTITY();

      -- Insert subject details
      INSERT INTO SUBJECTS (
        subject_id,
        study_id,
        tutor_id,
        subject_name,
        subject_description
      ) VALUES (
        @subject_id,
        @study_id,
        @tutor_id,
        @subject_name,
        @subject_description
      );

      COMMIT TRANSACTION;
      PRINT 'Przedmiot dodany pomyślnie.';
    END TRY
    BEGIN CATCH
      IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
      THROW;
    END CATCH
  END;
  GO
```

CreateSession

Procedura `CreateSession` tworzy nową sesję (zestaw spotkań) dla istniejącego przedmiotu. ID sesji zwracane jest za pomocą @session_id

Argumenty:

- @subject_id - Identyfikator przedmiotu
- @product_price - Cena sesji
- @vacancies - Liczba dostępnych miejsc (domyślnie 30)
- @release - Data udostępnienia sesji
- @session_id - Zwracane ID sesji

```
CREATE PROCEDURE [dbo].[CreateSession]
  @subject_id INT,
  @product_price MONEY = 0,
  @vacancies INT = 30,
  @release DATE,
  @session_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate subject exists
    IF NOT EXISTS (SELECT 1 FROM SUBJECTS WHERE subject_id = @subject_id)
    BEGIN
      THROW 50000, 'Przedmiot nie istnieje.', 1;
      RETURN;
    END

    -- Insert product first (sessions are products)
    INSERT INTO PRODUCTS (
      type_id,  -- Assuming type_id 5 is for sessions
      price,
      total_vacancies,
      release
    ) VALUES (
      5,
      @product_price,
      @vacancies,
      @release
    );

    -- Get the newly created product ID
    SET @session_id = SCOPE_IDENTITY();

    -- Insert session details
    INSERT INTO SESSIONS (
      session_id,
      subject_id
    ) VALUES (
      @session_id,
      @subject_id
    );

    COMMIT TRANSACTION;
    PRINT 'Sesja dodana pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
```

```
END;
GO
```

CreateSessionStationaryMeeting

Procedura `CreateSessionStationaryMeeting` tworzy nowe spotkanie stacjonarne w ramach sesji. ID spotkania zwracane jest za pomocą @meeting_id

Argumenty:

- @session_id - Identyfikator sesji
- @tutor_id - Identyfikator prowadzącego
- @translator_id - Opcjonalny identyfikator tłumacza
- @meeting_name - Nazwa spotkania
- @term - Termin spotkania
- @duration - Czas trwania (domyślnie 1h 30min)
- @language_id - Identyfikator języka (domyślnie 0)
- @classroom - Numer sali
- @meeting_id - Zwracane ID spotkania

```
CREATE PROCEDURE [dbo].[CreateSessionStationaryMeeting]
  @session_id INT,
  @tutor_id INT,
  @translator_id INT = NULL,
  @meeting_name VARCHAR(30),
  @term DATETIME,
  @duration TIME(0) = '01:30:00',
  @language_id INT = 0,
  @classroom VARCHAR(10),
  @meeting_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate session exists
    EXEC [dbo].[CheckSessionExists] @session_id

    -- Validate tutor exists
    EXEC [dbo].[CheckEmployeeExists] @tutor_id

    -- Insert meeting
    INSERT INTO MEETINGS (
      session_id,
      tutor_id,
      translator_id,
      meeting_name,
      term,
```

```
        duration,
        language_id
      ) VALUES (
        @session_id,
        @tutor_id,
        @translator_id,
        @meeting_name,
        @term,
        @duration,
        @language_id
      );

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

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

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

CreateSessionSyncMeeting

Procedura `CreateSessionSyncMeeting` tworzy nowe spotkanie synchroniczne w ramach sesji. ID spotkania zwracane jest za pomocą @meeting_id

Argumenty:

- @session_id - Identyfikator sesji
- @tutor_id - Identyfikator prowadzącego
- @translator_id - Opcjonalny identyfikator tłumacza
- @meeting_name - Nazwa spotkania
- @term - Termin spotkania
- @duration - Czas trwania (domyślnie 1h 30min)
- @language_id - Identyfikator języka (domyślnie 0)
- @meeting_url - Link do spotkania
- @video_url - Opcjonalny link do nagrania

- @meeting_id - Zwracane ID spotkania

```
CREATE PROCEDURE [dbo].[CreateSessionSyncMeeting]
  @session_id INT,
  @tutor_id INT,
  @translator_id INT = NULL,
  @meeting_name VARCHAR(30),
  @term DATETIME,
  @duration TIME(0) = '01:30:00',
  @language_id INT = 0,
  @meeting_url TEXT,
  @video_url TEXT = NULL,
  @meeting_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate session exists
    EXEC [dbo].[CheckSessionExists] @session_id

    -- Validate tutor exists
    EXEC [dbo].[CheckEmployeeExists] @tutor_id

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

    -- Get the newly created meeting ID
    SET @meeting_id = 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('Spotkanie synchroniczne utworzone pomyślnie.')
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

CreateSessionAsyncMeeting

Procedura `CreateSessionAsyncMeeting` tworzy nowe spotkanie asynchroniczne w ramach sesji. ID spotkania zwracane jest za pomocą @meeting_id

Argumenty:

- @session_id - Identyfikator sesji
- @tutor_id - Identyfikator prowadzącego
- @translator_id - Opcjonalny identyfikator tłumacza
- @meeting_name - Nazwa spotkania
- @term - Termin spotkania
- @duration - Czas trwania (domyślnie 1h 30min)
- @language_id - Identyfikator języka (domyślnie 0)
- @meeting_url - Link do materiałów
- @meeting_id - Zwracane ID spotkania

```
CREATE PROCEDURE [dbo].[CreateSessionAsyncMeeting]
  @session_id INT,
  @tutor_id INT,
  @translator_id INT = NULL,
  @meeting_name VARCHAR(30),
  @term DATETIME,
  @duration TIME(0) = '01:30:00',
  @language_id INT = 0,
  @video_url TEXT,
  @meeting_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;
```

```
    -- Validate session exists
    EXEC [dbo].[CheckSessionExists] @session_id

    -- Validate tutor exists
    EXEC [dbo].[CheckEmployeeExists] @tutor_id

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

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

    -- Insert async meeting details
    INSERT INTO ASYNC_MEETINGS (
      meeting_id,
      video_url
    ) VALUES (
      @meeting_id,
      @video_url
    );

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

CreateInternship

Procedura CreateInternship tworzy nową praktykę zawodową. ID praktyki zwracane jest za pomocą @internship_id

Argumenty:

- @product_price MONEY - Cena praktyki
- @vacancies INT - Liczba dostępnych miejsc (domyślnie 30)
- @release DATE - Data udostępnienia praktyki
- @name VARCHAR(30) - Nazwa praktyki
- @description TEXT - Opis praktyki
- @start_date DATE - Data rozpoczęcia praktyk
- @end_date DATE - Data zakończenia praktyk
- @internship_id INT OUTPUT - Zwracane ID praktyki

```
CREATE PROCEDURE [dbo].[CreateInternship]
  @study_id INT,
  @start_date DATE = NULL,
  @end_date DATE = NULL,
  @internship_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate study exists
    EXEC [dbo].[CheckStudyExists] @study_id

    -- Insert internship
    INSERT INTO INTERSHIPS (
      study_id,
      start_date,
      end_date
    ) VALUES (
      @study_id,
      @start_date,
      @end_date
    );

    -- Get the newly created internship ID
    SET @internship_id = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT 'Praktyka dodana pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
```

```
END;
GO
```

CreateInternshipDetails

Procedura CreateInternshipDetails dodaje szczegóły do istniejącej praktyki.

Argumenty:

- @internship_id INT - Identyfikator praktyki
- @tutor_id INT - Identyfikator opiekuna praktyk
- @student_id INT - Identyfikator studenta
- @company_id INT - Identyfikator firmy
- @completed BIT - Status ukończenia (domyślnie 0)
- @details_id INT OUTPUT - Zwracane ID szczegółów praktyki

```
CREATE PROCEDURE [dbo].[CreateInternshipDetails]
  @internship_id INT,
  @student_id INT,
  @passed BIT = 0,
  @internship_detail_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate internship exists
    IF NOT EXISTS (SELECT 1 FROM INTERSHIPS WHERE internship_id = @internship_id)
    BEGIN
      RAISERROR('Praktyka nie istnieje.', 16, 1);
      RETURN;
    END

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id

    -- Insert internship details
    INSERT INTO INTERSHIP_DETAILS (
      internship_id,
      student_id,
      passed
    ) VALUES (
      @internship_id,
      @student_id,
      @passed
    );

    -- Get the newly created internship detail ID
    SET @internship_detail_id = SCOPE_IDENTITY();
```

```
    COMMIT TRANSACTION;
    PRINT 'Szczegóły praktyki dodane pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

Orders

Typ productIdList

Typ tablicowy `productIdList` służący do przekazywania listy identyfikatorów produktów jako parametru do procedur składowanych.

Struktura:

- product_id INT - Identyfikator produktu

```
CREATE TYPE dbo.productIdList AS TABLE (
  product_id INT
);
```

CreateOrder

Procedura `CreateOrder` tworzy nowe zamówienie i generuje odpowiednie opłaty na podstawie typów zamawianych produktów.

Argumenty:

- @student_id INT - ID studenta składającego zamówienie
- @product_ids dbo.productIdList READONLY - Tabela z ID zamawianych produktów
- @order_id INT OUTPUT - Zwracane ID utworzonego zamówienia

Działanie:

1. Sprawdza czy student istnieje
2. Weryfikuje czy student nie ma już dostępu do któregoś z zamawianych produktów
3. Tworzy nowe zamówienie
4. Dla każdego produktu generuje odpowiednie opłaty zależnie od typu:
    - Studia: opłaty za sesje + opłata wpisowa
    - Przedmiot: opłaty za sesje
    - Kurs: opłata zaliczkowa + pozostała część
    - Webinar: opłata jednorazowa

- Sesja: opłata jednorazowa

```
CREATE PROCEDURE [dbo].[CreateOrder]
  @student_id INT,
  @product_ids dbo.productIdList READONLY,  -- productIdList type
  @order_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id

    -- Check if student already has access to any of the products
    IF EXISTS (
      SELECT 1
      FROM @product_ids pid
      JOIN PRODUCT_DETAILS pd ON pid.product_id = pd.product_id
      WHERE pd.student_id = @student_id
    )
    BEGIN
      THROW 50004, 'Student już ma dostęp do jednego lub więcej produktów z zamówienia.', 1;
    END

    -- Insert order
    INSERT INTO ORDERS (
      student_id,
      order_date
    ) VALUES (
      @student_id,
      GETDATE()
    );

    -- Get the newly created order ID
    SET @order_id = SCOPE_IDENTITY();

    -- Process each product in the list
    DECLARE @product_id INT;
    DECLARE @type_id INT;

    DECLARE product_cursor CURSOR FOR
    SELECT product_id FROM @product_ids;

    OPEN product_cursor;
    FETCH NEXT FROM product_cursor INTO @product_id;

    -- While there are rows to fetch from cursor and an error has not occured
    WHILE @@FETCH_STATUS = 0
    BEGIN
      -- Get product type
```

```
        SELECT @type_id = type_id FROM PRODUCTS WHERE product_id = @product_id;

        -- Process based on product type
        IF @type_id = 1
        -- study
        BEGIN
          EXEC [dbo].[createFeesForStudySession] @order_id, @product_id;
          EXEC [dbo].[createEntryFeeForStudy] @order_id, @product_id;
        END
        ELSE IF @type_id = 2
        -- subject
        BEGIN
          EXEC [dbo].[createFeesForSubject] @order_id, @product_id;
        END
        ELSE IF @type_id = 3
        -- course
        BEGIN
          EXEC [dbo].[createFeesForCourse] @order_id, @product_id;
        END
        ELSE IF @type_id = 4
        -- webinar
        BEGIN
          EXEC [dbo].[createFeeForWebinar] @order_id, @product_id;
        END
        ELSE IF @type_id = 5
        -- session
        BEGIN
          EXEC [dbo].[createFeeForSession] @order_id, @product_id;
        END

        -- Fetch next product
        FETCH NEXT FROM product_cursor INTO @product_id;
      END

      CLOSE product_cursor;
      DEALLOCATE product_cursor;

      COMMIT TRANSACTION;
      PRINT 'Zamówienie utworzone pomyślnie.';
    END TRY
    BEGIN CATCH
      IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;
      THROW;
    END CATCH
END;
GO
```

CreateFee

Procedura `CreateFee` tworzy pojedynczą opłatę w systemie.

Argumenty:

- @order_id INT - ID zamówienia
- @product_id INT - ID produktu, którego dotyczy opłata
- @type_id INT - Typ opłaty
- @due_date DATE - Termin płatności
- @fee_value MONEY - Kwota opłaty
- @fee_id INT OUTPUT - Zwracane ID utworzonej opłaty

Działanie:

1. Sprawdza czy zamówienie istnieje
2. Sprawdza czy produkt istnieje
3. Weryfikuje czy typ opłaty jest poprawny
4. Tworzy nową opłatę w systemie

```
CREATE PROCEDURE [dbo].[CreateFee]
  @order_id INT,
  @product_id INT,
  @type_id INT,
  @due_date DATE,
  @fee_value MONEY,
  @fee_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate order exists
    EXEC [dbo].[CheckOrderExists] @order_id

    -- Validate product exists
    EXEC [dbo].[CheckProductExists] @product_id

    -- Get student_id for checking
    DECLARE @student_id INT;
    SELECT @student_id = student_id FROM ORDERS WHERE order_id = @order_id;

    -- Check if fee for this product already exists
    IF EXISTS (
      SELECT 1
      FROM FEES
      JOIN ORDERS ON FEES.order_id = ORDERS.order_id
      WHERE ORDERS.student_id = @student_id
      AND FEES.product_id = @product_id
      AND FEES.payment_date IS NOT NULL
    )
    BEGIN
      PRINT 'Opłata za ten produkt już istnieje - pominięto.';
      RETURN;
```

```
    END

    -- Validate fee type exists
    IF NOT EXISTS (SELECT 1 FROM FEE_TYPES WHERE type_id = @type_id)
    BEGIN
      THROW 50000, 'Typ opłaty nie istnieje.', 1;
    END

    -- Insert fee
    INSERT INTO FEES (
      due_date,
      fee_value,
      type_id,
      order_id,
      product_id
    ) VALUES (
      @due_date,
      @fee_value,
      @type_id,
      @order_id,
      @product_id
    );

    -- Get the newly created fee ID
    SET @fee_id = SCOPE_IDENTITY();

    COMMIT TRANSACTION;
    PRINT 'Opłata dodana pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

createFeeForSession

Procedura `createFeeForSession` tworzy opłatę za pojedynczą sesję.

Argumenty:

- @order_id INT - ID zamówienia
- @session_id INT - ID sesji
- @fee_type INT - Typ opłaty (domyślnie 1)

```
CREATE PROCEDURE [dbo].[createFeeForSession]
  @order_id INT,
  @session_id INT,
  @fee_type INT = 1
```

```
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @first_meeting_date DATE;

    -- Get the date of the first meeting in the session
    SELECT TOP 1 @first_meeting_date = MIN(term)
    FROM MEETINGS
    WHERE session_id = @session_id
    GROUP BY term
    ORDER BY term;

    DECLARE @date datetime;
    SET @date = DATEADD(DAY, -1, @first_meeting_date);

    DECLARE @fee_value MONEY;
    SET @fee_value = (SELECT price FROM PRODUCTS WHERE product_id = @session_id);

    DECLARE @fee_id INT;

    EXEC [dbo].[CreateFee]
      @order_id = @order_id,
      @product_id = @session_id,
      @type_id = @fee_type,
      @due_date = @date,
      @fee_value =  @fee_value,
      @fee_id = @fee_id OUTPUT;

    COMMIT TRANSACTION;
    PRINT 'Opłata za sesję utworzona pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

createFeesForSubject

Procedura `createFeesForSubject` tworzy opłaty za wszystkie sesje w ramach przedmiotu.

Argumenty:

- @order_id INT - ID zamówienia
- @subject_id INT - ID przedmiotu
- @fee_type INT - Typ opłaty (domyślnie 1)

```
CREATE PROCEDURE [dbo].[createFeesForSubject]
  @order_id INT,
  @subject_id INT,
  @fee_type INT = 1
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @session_id INT;

    DECLARE session_cursor CURSOR FOR
      SELECT session_id
      FROM SESSIONS
      WHERE subject_id = @subject_id;

    OPEN session_cursor;
    FETCH NEXT FROM session_cursor INTO @session_id;

    WHILE @@FETCH_STATUS = 0
    BEGIN
      -- Create fee for each session in the subject
      EXEC [dbo].[createFeeForSession] @order_id, @session_id, @fee_type;

      FETCH NEXT FROM session_cursor INTO @session_id;
    END

    CLOSE session_cursor;
    DEALLOCATE session_cursor;

    COMMIT TRANSACTION;
    PRINT 'Opłaty za sesje przedmiotu utworzone pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

createFeesForStudySession

Procedura `createFeesForStudySession` tworzy opłaty za wszystkie sesje przedmiotów w ramach studiów.

Argumenty:

- @order_id INT - ID zamówienia
- @study_id INT - ID studiów

```
CREATE PROCEDURE [dbo].[createFeesForStudySession]
  @order_id INT,
  @study_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @subject_id INT;

    DECLARE subject_cursor CURSOR FOR
      SELECT subject_id
      FROM SUBJECTS
      WHERE study_id = @study_id;

    OPEN subject_cursor;
    FETCH NEXT FROM subject_cursor INTO @subject_id;

    WHILE @@FETCH_STATUS = 0
    BEGIN
      -- Create fees for each subject in the study
      EXEC [dbo].[createFeesForSubject] @order_id, @subject_id, 3;

      FETCH NEXT FROM subject_cursor INTO @subject_id;
    END

    CLOSE subject_cursor;
    DEALLOCATE subject_cursor;

    COMMIT TRANSACTION;
    PRINT 'Opłaty za sesje studiów utworzone pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

createEntryFeeForStudy

Procedura `createEntryFeeForStudy` tworzy opłatę wpisową za studia.

Argumenty:

- @order_id INT - ID zamówienia
- @study_id INT - ID studiów

```
CREATE PROCEDURE [dbo].[createEntryFeeForStudy]
  @order_id INT,
  @study_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @first_meeting_date DATE;

    -- Get the date of the first meeting in the study
    SELECT TOP 1 @first_meeting_date = MIN(term)
    FROM SESSIONS
    JOIN MEETINGS ON SESSIONS.session_id = MEETINGS.session_id
    WHERE SESSIONS.subject_id IN (SELECT subject_id FROM SUBJECTS WHERE study_id = @study_id)
    GROUP BY term
    ORDER BY term;

    DECLARE @date datetime;
    SET @date = DATEADD(DAY, -1, @first_meeting_date);

    DECLARE @fee_value MONEY;
    SET @fee_value = (SELECT price FROM PRODUCTS WHERE product_id = @study_id);

    -- Add entry fee
    DECLARE @fee_id INT;

    EXEC [dbo].[CreateFee]
      @order_id = @order_id,
      @product_id = @study_id,
      @type_id = 4,
      @due_date = @date,
      @fee_value =@fee_value,
      @fee_id = @fee_id OUTPUT;

    COMMIT TRANSACTION;
    PRINT 'Opłata wstępna za studia utworzona pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

createFeesForCourse

Procedura `createFeesForCourse` tworzy opłaty za kurs (zaliczkę i opłatę końcową).

Argumenty:

- @order_id INT - ID zamówienia
- @course_id INT - ID kursu

```
CREATE PROCEDURE [dbo].[createFeesForCourse]
  @order_id INT,
  @course_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @product_price MONEY;
    DECLARE @advance_share DECIMAL(5,4);
    DECLARE @first_meeting_date DATE;

    -- Get product price and advance share
    SELECT @product_price = price, @advance_share = advance_share
    FROM PRODUCTS
    JOIN COURSES ON PRODUCTS.product_id = COURSES.course_id
    WHERE PRODUCTS.product_id = @course_id;

    -- Get the date of the first meeting in the course
    SELECT TOP 1 @first_meeting_date = MIN(term)
    FROM MEETINGS
    JOIN MODULES ON MEETINGS.module_id = MODULES.module_id
    WHERE MODULES.course_id = @course_id
    GROUP BY term
    ORDER BY term;

    DECLARE @date_advance datetime;
    SET @date_advance = GETDATE();

    -- Add advance fee
    DECLARE @advance_value MONEY;
    SET @advance_value =  @product_price * @advance_share;

    DECLARE @fee_id INT;

    EXEC [dbo].[CreateFee]
      @order_id = @order_id,
      @product_id = @course_id,
      @type_id = 6,
      @due_date = @date_advance,
      @fee_value = @advance_value,
      @fee_id = @fee_id OUTPUT;

    -- Add remaining fee for course
```

```
    DECLARE @value_remaining MONEY;
    SET @value_remaining = @product_price * (1 - @advance_share)

    DECLARE @date_remaining datetime;
    SET @date_remaining = DATEADD(DAY, -3, @first_meeting_date);

    EXEC [dbo].[CreateFee]
      @order_id = @order_id,
      @product_id = @course_id,
      @type_id = 5,
      @due_date = @date_remaining,
      @fee_value = @value_remaining,
      @fee_id = @fee_id OUTPUT;

    COMMIT TRANSACTION;
    PRINT 'Opłaty za kurs utworzone pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

createFeeForWebinar

Procedura `createFeeForWebinar` tworzy opłatę za webinar.

Argumenty:

- @order_id INT - ID zamówienia
- @webinar_id INT - ID webinaru

```
CREATE PROCEDURE [dbo].[createFeeForWebinar]
  @order_id INT,
  @webinar_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    DECLARE @fee_id INT;
    DECLARE @product_price MONEY;

    -- Get product price
    SELECT @product_price = price
    FROM PRODUCTS
    WHERE product_id = @webinar_id;
```

```
    -- Add fee for webinar
    DECLARE @date datetime;
    SET @date = GETDATE();

    EXEC [dbo].[CreateFee]
      @order_id = @order_id,
      @product_id = @webinar_id,
      @type_id = 7,
      @due_date =  @date,
      @fee_value = @product_price,
      @fee_id = @fee_id OUTPUT;

    COMMIT TRANSACTION;
    PRINT 'Opłata za webinar utworzona pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

CreateOrderFromCart

Procedura `CreateOrderFromCart` tworzy zamówienie na podstawie koszyka studenta.

Argumenty:

- @student_id INT - ID studenta
- @order_id INT OUTPUT - Zwracane ID utworzonego zamówienia

```
CREATE PROCEDURE [dbo].[CreateOrderFromCart]
  @student_id INT,
  @order_id INT OUTPUT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id;

    -- Create a table variable to hold product IDs from the cart
    DECLARE @product_ids dbo.productIdList;

    -- Insert product IDs from the cart into the table variable
    INSERT INTO @product_ids (product_id)
```

```
    SELECT product_id FROM SHOPPING_CART WHERE student_id = @student_id;

    -- Create the order
    EXEC [dbo].[CreateOrder] @student_id, @product_ids, @order_id OUTPUT;

    -- Empty the cart
    DELETE FROM SHOPPING_CART WHERE student_id = @student_id;

    COMMIT TRANSACTION;
    PRINT 'Zamówienie utworzone z koszyka i koszyk opróżniony pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

addProductToCart

Procedura `addProductToCart` dodaje produkt do koszyka studenta.

Argumenty:

- @student_id INT - ID studenta, któremu dodajemy produkt do koszyka
- @product_id INT - ID produktu, który ma zostać dodany do koszyka

```
CREATE PROCEDURE [dbo].[addProductToCart]
  @student_id INT,
  @product_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id;

    -- Validate product exists
    EXEC [dbo].[CheckProductExists] @product_id;

    -- Add product to cart
    INSERT INTO SHOPPING_CART (student_id, product_id)
    VALUES (@student_id, @product_id);

    COMMIT TRANSACTION;
    PRINT 'Produkt dodany do koszyka pomyślnie.';
  END TRY
```

```
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

removeProductFromCart

Procedura `removeProductFromCart` usuwa produkt z koszyka studenta.

Argumenty:

- @student_id INT - ID studenta, któremu usuwamy produkt z koszyka
- @product_id INT - ID produktu, który ma zostać usunięty z koszyka

```
CREATE PROCEDURE [dbo].[removeProductFromCart]
  @student_id INT,
  @product_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id;

    -- Validate product exists in cart
    IF NOT EXISTS (SELECT 1 FROM SHOPPING_CART WHERE student_id = @student_id AND product_id = @product_id)
    BEGIN
      THROW 50005, 'Produkt nie znajduje się w koszyku.', 1;
      RETURN;
    END

    -- Remove product from cart
    DELETE FROM SHOPPING_CART
    WHERE student_id = @student_id AND product_id = @product_id;

    COMMIT TRANSACTION;
    PRINT 'Produkt usunięty z koszyka pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

UpdateFeePaymentDate

Procedura `UpdateFeePaymentDate` aktualizuje datę opłaty na bieżącą.

Argumenty:

- @fee_id INT - ID opłaty do zaktualizowania

```
CREATE PROCEDURE [dbo].[UpdateFeePaymentDate]
  @fee_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate fee exists
    EXEC [dbo].[CheckFeeExists] @fee_id;

    -- Update the fee with the current date as the payment date
    UPDATE FEES
    SET payment_date = GETDATE()
    WHERE fee_id = @fee_id;

    COMMIT TRANSACTION;
    PRINT 'Data płatności została zaktualizowana pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

Products

FillProductDetails

Procedura `FillProductDetails` dodaje szczegóły zakupionego produktu do tabeli PRODUCTS_DETAILS.

Argumenty:

- @student_id INT - ID studenta, który zakupił produkt
- @product_id INT - ID zakupionego produktu
- @order_id INT - ID zamówienia w ramach którego zakupiono produkt

```
CREATE PROCEDURE [dbo].[FillProductDetails]
  @student_id INT,
  @product_id INT,
  @order_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id

    -- Validate product exists
    EXEC [dbo].[CheckProductExists] @product_id

    -- Validate order exists
    EXEC [dbo].[CheckOrderExists] @order_id

    -- Insert into PRODUCTS_DETAILS
    INSERT INTO PRODUCTS_DETAILS (
      student_id,
      product_id,
      order_id
    ) VALUES (
      @student_id,
      @product_id,
      @order_id
    );

    COMMIT TRANSACTION;
    PRINT 'Tabela PRODUCTS_DETAILS została wypełniona pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

MarkProductAsPassed

Procedura `MarkProductAsPassed` oznacza produkt jako zaliczony przez danego studenta.

Argumenty:

- @product_id INT - ID produktu do oznaczenia
- @student_id INT - ID studenta, który zaliczył produkt

```
CREATE PROCEDURE [dbo].[MarkProductAsPassed]
  @product_id INT,
  @student_id INT
AS
BEGIN
  SET NOCOUNT ON;

  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id

    -- Validate product exists
    EXEC [dbo].[CheckProductExists] @product_id

    -- Update the product details to mark as passed
    UPDATE PRODUCTS_DETAILS
    SET passed = 1
    WHERE product_id = @product_id AND student_id = @student_id;

    COMMIT TRANSACTION;
    PRINT 'Produkt oznaczony jako zaliczony pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

MEETINGS

FillMeetingDetails

Procedura `FillMeetingDetails` rejestruje obecność studenta na spotkaniu poprzez dodanie wpisu do tabeli MEETING_DETAILS.

Argumenty:

- @meeting_id INT - ID spotkania
- @student_id INT - ID studenta uczestniczącego w spotkaniu

```
CREATE PROCEDURE [dbo].[FillMeetingDetails]
  @meeting_id INT,
  @student_id INT
AS
BEGIN
  SET NOCOUNT ON;
```

```
  BEGIN TRY
    BEGIN TRANSACTION;

    -- Validate meeting exists
    EXEC [dbo].[CheckMeetingExists] @meeting_id

    -- Validate student exists
    EXEC [dbo].[CheckStudentExists] @student_id

    -- Insert into MEETING_DETAILS
    INSERT INTO MEETING_DETAILS (
      meeting_id,
      student_id
    ) VALUES (
      @meeting_id,
      @student_id
    );

    COMMIT TRANSACTION;
    PRINT 'Tabela MEETING_DETAILS została wypełniona pomyślnie.';
  END TRY
  BEGIN CATCH
    IF @@TRANCOUNT > 0
      ROLLBACK TRANSACTION;
    THROW;
  END CATCH
END;
GO
```

Wyzwalacze (Triggers)

trg_AddMeetingDetails

Trigger `trg_AddMeetingDetails` automatycznie dodaje wpisy do tabeli MEETING_DETAILS gdy student zostaje przypisany do produktu w PRODUCTS_DETAILS.

Tabela wyzwalająca:

- PRODUCTS_DETAILS

Moment aktywacji:

- AFTER INSERT

```
CREATE TRIGGER trg_AddMeetingDetails
ON PRODUCTS_DETAILS
AFTER INSERT
AS
BEGIN
```

```
    SET NOCOUNT ON;

    DECLARE @student_id INT;
    DECLARE @product_id INT;
    DECLARE @type_id INT;

    -- Get the inserted student_id and product_id
    SELECT @student_id = inserted.student_id, @product_id = inserted.product_id
    FROM inserted;

    -- Get the type_id of the product
    SELECT @type_id = type_id FROM PRODUCTS WHERE product_id = @product_id;

    -- Return early if type_id does not match study, subject, course, or session
    IF @type_id NOT IN (1, 2, 3, 5)
    BEGIN
      RETURN;
    END

    -- Add meeting details based on product type
    IF @type_id = 1 -- study
    BEGIN
      INSERT INTO MEETING_DETAILS (meeting_id, student_id)
      SELECT meeting_id, @student_id
      FROM MEETINGS
      JOIN SESSIONS ON MEETINGS.session_id = SESSIONS.session_id
      JOIN SUBJECTS ON SESSIONS.subject_id = SUBJECTS.subject_id
      WHERE SUBJECTS.study_id = @product_id;
    END
    ELSE IF @type_id = 2 -- subject
    BEGIN
      INSERT INTO MEETING_DETAILS (meeting_id, student_id)
      SELECT meeting_id, @student_id
      FROM MEETINGS
      JOIN SESSIONS ON MEETINGS.session_id = SESSIONS.session_id
      WHERE SESSIONS.subject_id = @product_id;
    END
    ELSE IF @type_id = 3 -- course
    BEGIN
      INSERT INTO MEETING_DETAILS (meeting_id, student_id)
      SELECT meeting_id, @student_id
      FROM MEETINGS
      JOIN MODULES ON MEETINGS.module_id = MODULES.module_id
      WHERE MODULES.course_id = @product_id;
    END
    ELSE IF @type_id = 5 -- session
    BEGIN
      INSERT INTO MEETING_DETAILS (meeting_id, student_id)
      SELECT meeting_id, @student_id
      FROM MEETINGS
      WHERE session_id = @product_id;
    END
  END;
  GO
```

Funkcje

Kategoria zamówienia i produkty

Wyliczenie wartości koszyka

```
CREATE FUNCTION GetCartValue(@StudentId INT)
RETURNS MONEY
AS
BEGIN
    RETURN (
        SELECT SUM(p.price)
        FROM SHOPPING_CART sc
        JOIN PRODUCTS p ON sc.product_id = p.product_id
        WHERE sc.student_id = @StudentId
    );
END;
```

Wyliczenie wartości zamówienia

```
CREATE FUNCTION GetOrderValue(@OrderId INT)
RETURNS MONEY
AS
BEGIN
    RETURN (
        SELECT SUM(f.fee_value)
        FROM FEES f
        WHERE f.order_id = @OrderId
    );
END;
```

Sprawdzenie czy zjazd został zakupiony pojedynczo

```
CREATE FUNCTION isSingleProduct(@fee_id INT)
RETURNS BIT
AS
BEGIN
    DECLARE @result BIT;

    SELECT
        @result = CASE
                    WHEN ft.type_name = 'study session' or ft.type_name = 'subject session' THEN 0
                    ELSE 1
                END
```

```
    FROM FEES f
    JOIN FEE_TYPE ft ON f.type_id = ft.type_id
    WHERE f.fee_id = @fee_id;

    RETURN @result;
END;
```

Znalezienie studiów do których należy sesja

```
CREATE FUNCTION getParentId(@session_id INT)
RETURNS INT
AS
BEGIN
    DECLARE @study_id INT;

    SELECT
        @study_id = s.subject_id
    FROM
        SESSIONS ses
    JOIN SUBJECTS s ON ses.subject_id = s.subject_id
    WHERE
        ses.session_id = @session_id;

    RETURN @study_id;
END;
```

Wyliczenie wolnych miejsc dla danego produktu

```
CREATE FUNCTION GetVacanciesForProduct(@ProductId INT)
RETURNS INT
AS
BEGIN
    DECLARE @TotalVacancies INT, @EnrolledStudents INT;

    SELECT @TotalVacancies = p.total_vacancies
    FROM PRODUCTS p
    WHERE p.product_id = @ProductId;

    SELECT @EnrolledStudents = COUNT(*)
    FROM PRODUCTS_DETAILS pd
    WHERE pd.product_id = @ProductId;

    RETURN ISNULL(@TotalVacancies - @EnrolledStudents, 0);
END;
```

Sprawdzenie czy student posiada dany produkt

```
CREATE FUNCTION CheckStudentOwnsProduct(@student_id INT, @product_id INT)
RETURNS BIT
AS
BEGIN
    DECLARE @owns_product BIT;

    SELECT
        @owns_product = CASE
                            WHEN EXISTS (
                                SELECT 1
                                FROM PRODUCTS_DETAILS pd
                                WHERE pd.product_id = @product_id
                                  AND pd.student_id = @student_id
                            ) THEN 1
                            ELSE 0
                        END;

    RETURN @owns_product;
END;
```

Sprawdzenie czy student może dodać produkt do koszyka

```
CREATE FUNCTION CanAddToCart(@StudentId INT, @ProductId INT)
RETURNS BIT
AS
BEGIN
    DECLARE @OwnsProduct BIT;

    SELECT @OwnsProduct = CheckStudentOwnsProduct(@StudentId,@ProductId)

    RETURN @OwnsProduct
END;
```

Sprawdzenie czy student może kupić produkt

```
CREATE FUNCTION CanStudentBuyProduct(@StudentId INT, @ProductId INT)
RETURNS BIT
AS
BEGIN
    DECLARE @AvailableVacancies INT;

    SELECT @AvailableVacancies = GetVacanciesForProduct(@ProductId)

    RETURN (
        CASE
            WHEN @AvailableVacancies > 0 THEN 1
            ELSE 0
        END
```

```
    );
END;
```

Kursy, studia i webinary

Sprawdzenie czy student zdał praktyki

```
CREATE FUNCTION DoesStudentPassInternship(@StudentId INT)
RETURNS BIT
AS
BEGIN
    DECLARE @Result bit = 0
    IF EXISTS (SELECT 1 FROM INTERNSHIP_DETAILS WHERE @StudentId=student_id)
    BEGIN
        IF NOT EXISTS (
            SELECT 1
            FROM INTERNSHIP_DETAILS id
            JOIN INTERNSHIPS i ON i.internship_id=id.internship_id
            WHERE id.student_id=@StudentId AND i.end_date>GETDATE()
        )
        BEGIN
            IF NOT EXISTS (
                SELECT 1
                FROM INTERNSHIP_DETAILS id
                WHERE id.student_id=@StudentId and id.passed=0
            )
            SET @Result=1
        END
    END
    RETURN @Result;
END
```

Wyliczenie obecności studenta na danych studiach

```
CREATE FUNCTION GetAttendanceForStudy(@StudentId INT, @StudyId INT)
RETURNS DECIMAL(5, 2)
AS
BEGIN
    DECLARE @TotalMeetings INT, @AttendedMeetings INT;

    SELECT @TotalMeetings = COUNT(*)
    FROM MEETING_DETAILS md
    JOIN MEETINGS m ON md.meeting_id = m.meeting_id
    JOIN SESSIONS sess ON m.session_id = sess.session_id
    JOIN SUBJECTS subj ON sess.subject_id = subj.subject_id
    WHERE subj.study_id = @StudyId;

    SELECT @AttendedMeetings = COUNT(*)
    FROM MEETING_DETAILS md
```

```
    JOIN MEETINGS m ON md.meeting_id = m.meeting_id
    JOIN SESSIONS sess ON m.session_id = sess.session_id
    JOIN SUBJECTS subj ON sess.subject_id = subj.subject_id
    WHERE subj.study_id = @StudyId AND md.student_id = @StudentId AND md.attendance = 1;

    RETURN ISNULL((@AttendedMeetings * 100.0) / NULLIF(@TotalMeetings, 0), 0);
END;
```

Wyliczenie obecności studenta na danym kursie

```
CREATE FUNCTION GetAttendanceForCourse(@StudentId INT, @CourseId INT)
RETURNS DECIMAL(5, 2)
AS
BEGIN
    DECLARE @TotalMeetings INT, @AttendedMeetings INT;

    SELECT @TotalMeetings = COUNT(*)
    FROM MEETING_DETAILS md
    JOIN MEETINGS m ON md.meeting_id = m.meeting_id
    JOIN MODULES mod ON m.module_id = mod.module_id
    WHERE mod.course_id = @CourseId;

    SELECT @AttendedMeetings = COUNT(*)
    FROM MEETING_DETAILS md
    JOIN MEETINGS m ON md.meeting_id = m.meeting_id
    JOIN MODULES mod ON m.module_id = mod.module_id
    WHERE mod.course_id = @CourseId AND md.student_id = @StudentId AND md.attendance = 1;

    RETURN ISNULL((@AttendedMeetings * 100.0) / NULLIF(@TotalMeetings, 0), 0);
END;
```

Zwraca plan studiów w formie spotkań

```
CREATE FUNCTION GetStudySchedule(@StudyId INT)
RETURNS TABLE
AS
RETURN (
    SELECT m.meeting_id, m.meeting_name, m.term, m.duration, l.language_name
    FROM MEETING_DETAILS md
    JOIN MEETINGS m ON md.meeting_id = m.meeting_id
    JOIN LANGUAGES l ON m.language_id = l.language_id
    JOIN SESSIONS s ON m.session_id=s.session_id
    JOIN SUBJECTS sub on sub.subject_id=s.session_id
    JOIN STUDIES st on st.study_id=sub.study_id
    WHERE st.study_id=@StudyId
    ORDER BY m.term
);
```

Zwraca plan kursu w formie spotkań

```
CREATE FUNCTION GetCourseSchedule(@CourseId INT)
RETURNS TABLE
AS
RETURN (
    SELECT m.meeting_id, m.meeting_name, m.term, m.duration, l.language_name
    FROM MEETING_DETAILS md
    JOIN MEETINGS m ON md.meeting_id = m.meeting_id
    JOIN LANGUAGES l ON m.language_id = l.language_id
    JOIN MODULES mod ON mod.module_id=m.module_id
    JOIN COURSES c on c.course_id=m.course_id
    WHERE c.course_id=@CourseId
    ORDER BY m.term
);
```

Plan zajęć studenta

```
CREATE FUNCTION GetStudentTimetable(@StudentId INT)
RETURNS TABLE
AS
RETURN (
    SELECT m.meeting_id, m.meeting_name, m.term, m.duration, l.language_name
    FROM MEETING_DETAILS md
    JOIN MEETINGS m ON md.meeting_id = m.meeting_id
    JOIN LANGUAGES l ON m.language_id = l.language_id
    WHERE md.student_id = @StudentId
    ORDER BY m.term
);
```

Plan zajęć nauczyciela

```
CREATE FUNCTION GetTutorTimetable(@EmployeeId INT)
RETURNS TABLE
AS
RETURN (
    SELECT m.meeting_id, m.meeting_name, m.term, m.duration, l.language_name
    FROM MEETINGS m
    JOIN LANGUAGES l ON m.language_id = l.language_id
    WHERE m.tutor_id = @EmployeeId
    ORDER BY m.term
);
```

Plan zajęć tłumacza

```
CREATE FUNCTION GetTranslatorTimetable(@EmployeeId INT)
RETURNS TABLE
AS
RETURN (
    SELECT m.meeting_id, m.meeting_name, m.term, m.duration, l.language_name
    FROM MEETINGS m
    JOIN LANGUAGES l ON m.language_id = l.language_id
    WHERE m.translator_id = @EmployeeId
    ORDER BY m.term
);
```

Sprawdzenie czy spotkanie koliduje z resztą planu użytkownika

```
CREATE FUNCTION CheckMeetingConflict(@Id INT, @MeetingId INT)
RETURNS BIT
AS
BEGIN
    DECLARE @Conflict BIT = 0;

    DECLARE @MeetingTerm DATETIME, @MeetingDuration INT;
    SELECT @MeetingTerm = term, @MeetingDuration = duration
    FROM MEETINGS
    WHERE meeting_id = @MeetingId;

    IF @MeetingTerm IS NULL OR @MeetingDuration IS NULL
    BEGIN
        RETURN 0;
    END

    IF EXISTS (
        SELECT 1
        FROM MEETING_DETAILS md
        JOIN MEETINGS m ON md.meeting_id = m.meeting_id
        WHERE md.student_id = @Id
          AND m.term < DATEADD(MINUTE, @MeetingDuration, @MeetingTerm)
          AND DATEADD(MINUTE, m.duration, m.term) > @MeetingTerm
          AND m.meeting_id <> @MeetingId
        UNION ALL

        SELECT 1
        FROM MEETINGS m
        WHERE m.tutor_id = @Id
          AND m.term < DATEADD(MINUTE, @MeetingDuration, @MeetingTerm)
          AND DATEADD(MINUTE, m.duration, m.term) > @MeetingTerm
          AND m.meeting_id <> @MeetingId
        UNION ALL

        SELECT 1
        FROM MEETINGS m
        WHERE m.translator_id = @Id
```

```
            AND m.term < DATEADD(MINUTE, @MeetingDuration, @MeetingTerm)
            AND DATEADD(MINUTE, m.duration, m.term) > @MeetingTerm
            AND m.meeting_id <> @MeetingId
    )
    BEGIN
        SET @Conflict = 1;
    END

    RETURN @Conflict;
END;
```