PODSTAWY BAZ DANYCH

Zaprojektowanie i implementacja systemu bazodanowego dla firmy oferującej kursy i szkolenia

Autorzy: Maria Stalmach, Izabela Szpunar, Natalia Wrześniak

Cel projektu

Celem projektu było stworzenie systemu bazodanowego dla firmy zajmującej się organizacją szkoleń i kursów, które są prowadzone w modelu hybrydowym (stacjonarnie i online). System ma umożliwiać zarządzanie informacjami o webinarach, kursach oraz studiach, uwzględniając różnorodne formy uczestnictwa i płatności.

Zastosowane technologie

1. Vertabelo Database Modeler - zaprojektowanie schematu bazy danych, dodanie warunków integralnościowych, wygenerowanie kodów tworzących tabele
2. MS SQL Server - implementacja zaprojektowanej bazy danych.
3. dbForge Data Generator for SQL Server - wygenerowanie danych testowych

Opis systemu

W systemie zidentyfikowano 4 role, jakie może pełnić użytkownik, a wraz z tym określono funkcje, jakie może pełnić każdy z nich. Każdy z zarejestrowanych użytkowników (po nadaniu odpowiednich uprawnień i ich zweryfikowaniu) może pełnić każdą z ról osobno.

Student

- uczestniczyć w różnych formach kształcenia (webinary, kursy, studia, pojedyncze moduły w ramach kursów czy studiów),
- dokonywać zakupu dostępu do różnych płatnych form kształcenia (studia, kursy, płatne webinary, płatne moduły studyjne)

Wykładowca

- uczestniczyć w ogólnodostępny webinarach
- prowadzić zajęcia
- być koordynatorem kursu/studium

Tłumacz

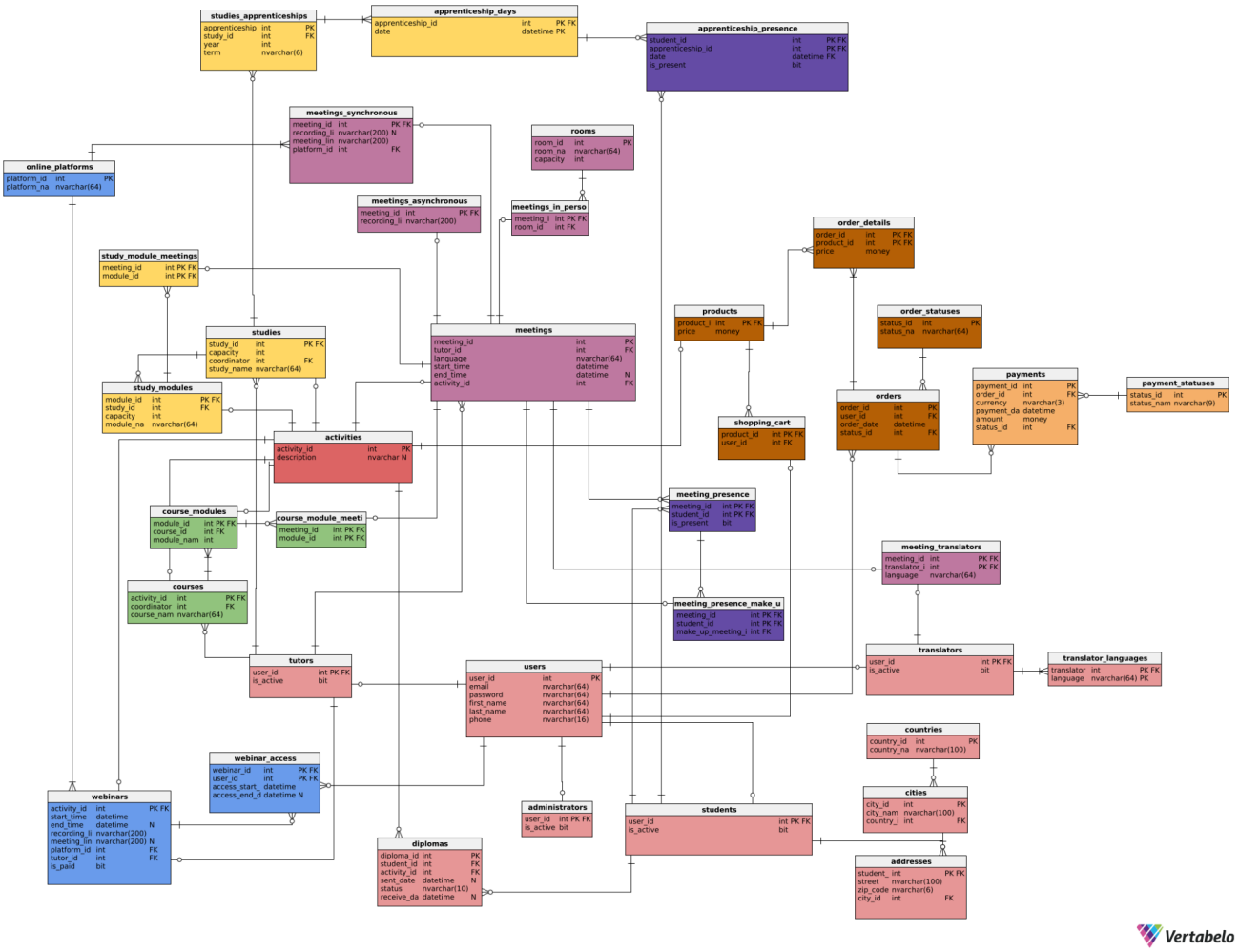
- uczestniczyć w ogólnodostępny webinarach
- tłumaczyć spotkania w ramach różnych form kształcenia

Administrator

- mieć dostęp do wszystkich tabel z możliwością ich modyfikacji

- skasować nagrania z webinariów czy spotkań online

Schemat bazy danych



Opisy tabel

Poniżej przedstawiono opisy poszczególnych tabel (nazwy pól, typ danych i znaczenie każdego pola, a także opis warunków integralności, jakie zostały zdefiniowane dla każdego pola. Uwzględniono również kody generujące tabele oraz powiązania kluczami obcymi pomiędzy nimi.

Tabele dotyczące użytkowników systemu

USERS

Zawiera informacje o wszystkich zarejestrowanych w systemie użytkownikach.

Pole	Typ danych	Znaczenie	Warunki integralnościowe

user_id	int	unikalny identyfikator użytkownika	not null, identitity (1, 1), klucz główny
email	nvarchar(64)	adres e-mail użytkownika	not null, unique, musi zawierać ‘@’
password	nvarchar(64)	hasło użytkownika do logowania	not null, musi zawierać min. 1 wielką literę, 1 małą oraz 1 cyfrę, musi mieć min. 8 znaków
first_name	nvarchar(64)	imię użytkownika	not null
last_name	nvarchar(64)	nazwisko użytkownika	not null
phone	nvarchar(64)	numer telefonu użytkownika	not null, unique

```
CREATE TABLE users (
   user_id int  NOT NULL IDENTITY(1, 1),
   email nvarchar(64)  NOT NULL,
   password nvarchar(64)  NOT NULL,
   first_name nvarchar(64)  NOT NULL,
   last_name nvarchar(64)  NOT NULL,
   phone nvarchar(16)  NOT NULL,
   CONSTRAINT unique_email UNIQUE (email),
   CONSTRAINT unique_phone UNIQUE (phone),
   CONSTRAINT email_format CHECK (email LIKE '%@%'),
   CONSTRAINT users_password_format CHECK (PATINDEX('%[A-Z]%', password)
> 0 AND PATINDEX('%[a-z]%', password) > 0 AND PATINDEX('%[0-9]%',
password) > 0),
   CONSTRAINT users_password_length CHECK (LEN(password) >= 8),
   CONSTRAINT users_pk PRIMARY KEY  (user_id)
);
```

ADMINISTRATORS

Tabela zawierająca listę administratorów systemu.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
user_id	int	identyfikator użytkownika, który jest administratorem	not null, klucz główny, klucz obcy wskazujący na pole user_id w tabeli users
is_active	bit	określa, czy użytkownik jest aktywny w systemie jako administrator:	not null, domyślnie przypisywana jest wartość 0

		1 - jest aktywny 0 - jest nieaktywny	

```
 CREATE TABLE administrators (
   user_id int  NOT NULL,
   is_active bit  NOT NULL DEFAULT 0,
   CONSTRAINT administrators_pk PRIMARY KEY  (user_id)
);
```

```
ALTER TABLE administrators ADD CONSTRAINT users_administrators
   FOREIGN KEY (user_id)
   REFERENCES users (user_id);
```

STUDENTS

Tabela zawierająca informacje o osobach uczących się (uczestnikach kursów, studiów, webinarów).

Pole	Typ danych	Znaczenie	Warunki integralnościowe
user_id	int	identyfikator użytkownika, który jest studentem	not null, klucz główny, klucz obcy wskazujący na pole user_id w tabeli users
is_active	bit	określa, czy użytkownik jest aktywny w systemie jako student: 1 - jest aktywny 0 - jest nieaktywny	not null, domyślnie przypisywana jest wartość 1

```
CREATE TABLE students (
   user_id int  NOT NULL,
   is_active bit  NOT NULL DEFAULT 1,
   CONSTRAINT students_pk PRIMARY KEY  (user_id)
);
```

```
ALTER TABLE students ADD CONSTRAINT students_users
   FOREIGN KEY (user_id)
   REFERENCES users (user_id);
```

ADDRESSES

Tabela zawierająca adresy studentów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
student_id	int	identyfikator studenta, dla którego zapisano adres.	not null, klucz główny, klucz obcy wskazujący na pole user_id w tabeli students
street	nvarchar(100)	nazwa ulicy	not null
zip_code	nvarchar(6)	kod pocztowy	not null
city_id	int	identyfikator miasta	not null, klucz obcy wskazujący na pole city_id w tabeli cities

```
CREATE TABLE addresses (
   student_id int  NOT NULL,
   street nvarchar(100)  NOT NULL,
   zip_code nvarchar(6)  NOT NULL,
   city_id int  NOT NULL,
   CONSTRAINT addresses_pk PRIMARY KEY  (student_id)
);
```

```
ALTER TABLE addresses ADD CONSTRAINT students_addresses
   FOREIGN KEY (student_id)
   REFERENCES students (user_id);
```

CITIES

Tabela zawierająca informacje o miastach.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
city_id	int	unikalny identyfikator miasta	not null, klucz główny, autoinkrementowany
city_name	nvarchar(100)	nazwa miasta	not null
country_id	int	identyfikator państwa	not null, klucz obcy wskazujący na pole country_id w tabeli countries

```
CREATE TABLE cities (
   city_id int  NOT NULL IDENTITY(1, 1),
   city_name nvarchar(100)  NOT NULL,
   country_id int  NOT NULL,
```

```
   CONSTRAINT cities_pk PRIMARY KEY  (city_id)
);
```

```
ALTER TABLE cities ADD CONSTRAINT cities_countries
   FOREIGN KEY (country_id)
   REFERENCES countries (country_id);
```

COUNTRIES

Tabela zawierająca informacje o krajach.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
country_id	int	unikalny identyfikator państwa	not null, klucz główny, autoinkremnetowany
country_name	nvarchar(100)	nazwa państwa	not null

```
CREATE TABLE countries (
   country_id int  NOT NULL IDENTITY(1, 1),
   country_name nvarchar(100)  NOT NULL,
   CONSTRAINT countries_pk PRIMARY KEY  (country_id)
);
```

DIPLOMAS

Tabela zawierająca informacje na temat wysyłek dyplomów ukończenia studiów/kursów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
diploma_id	int	unikalny identyfikator dyplomu	not null, klucz główny, autoinkrementowany
student_id	int	identyfikator studenta, który otrzymuje dyplom	not null, klucz obcy wskazujący na pole user_id w tabeli students
activity_id	int	identyfikator aktywności, po ukończeniu której wydano dyplom	not null, klucz obcy wskazujący na pole activity_id w tabeli activities
sent_date	datetime	data wysłania dyplomu	może być null, jeśli dyplom nie został jeszcze wysłany

status	nvarchar(10)	stan wysłania dyplomu	not null, domyślnie przypisana wartość to ‘not sent’, może przyjmować wartości: ‘not sent’, ‘sent’, ‘received’
receive_date	datetime	data odbioru dyplomu przez studenta	może być null, jeśli dyplom nie został jeszcze odebrany, musi być większe od sent_date

```
CREATE TABLE diplomas (
   diploma_id int  NOT NULL, IDENTITY(1, 1),
   student_id int  NOT NULL,
   activity_id int  NOT NULL,
   sent_date datetime  NULL,
   status nvarchar(10)  NOT NULL DEFAULT 'not sent',
   receive_date datetime  NULL,
   CONSTRAINT date CHECK (receive_date IS NULL OR receive_date >
sent_date),
   CONSTRAINT diplomas_ck_status CHECK (status IN ('not sent', 'sent',
'received')),
   CONSTRAINT diplomas_pk PRIMARY KEY  (diploma_id)
);
```

```
ALTER TABLE diplomas ADD CONSTRAINT activities_diploma
   FOREIGN KEY (activity_id)
   REFERENCES activities (activity_id);
```

```
ALTER TABLE diplomas ADD CONSTRAINT students_diploma
   FOREIGN KEY (student_id)
   REFERENCES students (user_id);
```

TRANSLATORS

Tabela zawierająca informacje o tłumaczach różnych form kształcenia.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
user_id	int	identyfikator użytkownika, który jest tłumaczem	not null, klucz główny, klucz obcy wskazujący na pole user_id w tabeli users
is_active	bit	określa, czy użytkownik jest	not null, domyślnie

		aktywny w systemie jako tłumacz: 1 - jest aktywny 0 - jest nieaktywny	przypisywana jest wartość 1

```
CREATE TABLE translators (
   user_id int  NOT NULL,
   is_active bit  NOT NULL DEFAULT 1,
   CONSTRAINT translators_pk PRIMARY KEY  (user_id)
);
```

```
ALTER TABLE translators ADD CONSTRAINT translators_users
   FOREIGN KEY (user_id)
   REFERENCES users (user_id);
```

TRANSLATOR_LANGUAGES

Tabela zawierająca informacje o tym jakimi językami posługują się tłumacze.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
translator_id	int	identyfikator użytkownika, który jest tłumaczem	not null, część klucza głównego, klucz obcy wskazujący na pole user_id w tabeli translators
language	nvarchar(64)	nazwa języka	not null, część klucza głównego

```
CREATE TABLE translator_languages (
   translator_id int  NOT NULL,
   language nvarchar(64)  NOT NULL,
   CONSTRAINT translator_languages_pk PRIMARY KEY  (translator_id)
);
```

```
ALTER TABLE translator_languages ADD CONSTRAINT
translator_languages_translators
   FOREIGN KEY (translator_id)
   REFERENCES translators (user_id);
```

TUTORS

Tabela zawierająca informacje o prowadzących różne formy kształcenia.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
user_id	int	identyfikator użytkownika, który jest tłumaczem	not null, klucz główny, klucz obcy wskazujący na pole user_id w tabeli users
is_active	bit	określa, czy użytkownik jest aktywny w systemie jako tłumacz: 1 - jest aktywny 0 - jest nieaktywny	not null, domyślnie przypisywana jest wartość 1

```
CREATE TABLE tutors (
   user_id int  NOT NULL,
   is_active bit  NOT NULL DEFAULT 1,
   CONSTRAINT tutors_pk PRIMARY KEY  (user_id)
);
```

Tabele dotyczące różnych form kształcenia

ACTIVITIES

Tabela gromadząca informacje o wszystkich formach kształcenia.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
activity_id	int	unikalny identyfikator aktywności - kursu, studium, webinaru.	not null, klucz główny, autoinkrementowany
description	nvarchar	opis aktywności	może być null

```
CREATE TABLE activities (
   activity_id int  NOT NULL IDENTITY(1, 1),
   description nvarchar(100)  NULL,
   CONSTRAINT activities_pk PRIMARY KEY  (activity_id)
);
```

COURSES

Tabela zawierająca informacje na temat kursów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
activity_id	int	unikalny identyfikator kursu	not null, klucz główny, klucz obcy wskazujący na pole activity_id w tabeli activities
coordinator_id	int	identyfikator koordynatora	not null, klucz obcy wskazujący na pole user_id w tabeli tutors
course_name	nvarchar(64)	nazwa kursu	not null, unique

```
CREATE TABLE courses (
   activity_id int  NOT NULL,
   coordinator_id int  NOT NULL,
   course_name nvarchar(64) NOT NULL,

   CONSTRAINT course_name_uq UNIQUE(course_name)
   CONSTRAINT courses_pk PRIMARY KEY  (activity_id)
);
```

```
ALTER TABLE courses ADD CONSTRAINT activities_courses
   FOREIGN KEY (activity_id)
   REFERENCES activities (activity_id);
```

```
ALTER TABLE courses ADD CONSTRAINT tutors_courses
   FOREIGN KEY (coordinator_id)
   REFERENCES tutors (user_id);
```

COURSE_MODULES

Tabela zawierająca informacje na temat modułów, składających się na kurs.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
module_id	int	unikalny identyfikator modułu	not null, klucz główny, klucz obcy wskazujący na pole activity_id w tabeli activities
course_id	int	identyfikator kursu, do którego należy moduł	not null, klucz obcy wskazujący na pole activity_id w tabeli courses

module_name	nvarchar(64)	nazwa modułu	not null

```
CREATE TABLE course_modules (
   module_id int  NOT NULL,
   course_id int  NOT NULL,
   module_name nvarchar(64) NOT NULL,
   CONSTRAINT course_modules_pk PRIMARY KEY  (module_id)
);
```

```
ALTER TABLE course_modules ADD CONSTRAINT course_modules_courses
   FOREIGN KEY (course_id)
   REFERENCES courses (activity_id);
```

```
ALTER TABLE course_modules ADD CONSTRAINT activities_course_modules
   FOREIGN KEY (module_id)
   REFERENCES activities (activity_id);
```

COURSE_MODULE_MEETINGS

Tabela zawierająca spotkania wchodzące w moduł kursu.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	unikalny identyfikator spotkania	not null, część klucza głównego, klucz obcy wskazujący na pole meeting_id w tabeli meetings
module_id	int	identyfikator modułu, do którego należy spotkanie	not null, część klucza głównego, klucz obcy wskazujący na pole module_id w tabeli course_modules

```
CREATE TABLE course_module_meetings (
   meeting_id int  NOT NULL,
   module_id int  NOT NULL,
   CONSTRAINT module_meetings_pk PRIMARY KEY  (meeting_id,module_id)
);
```

```
ALTER TABLE course_module_meetings ADD CONSTRAINT
course_modules_module_meetings
```

```
   FOREIGN KEY (module_id)
   REFERENCES course_modules (module_id);
```

```
ALTER TABLE course_module_meetings ADD CONSTRAINT
meetings_module_meetings
   FOREIGN KEY (meeting_id)
   REFERENCES meetings (meeting_id);
```

WEBINARS

Tabela zawierająca informacje na temat webinarów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
activity_id	int	unikalny identyfikator webinaru	not null, klucz główny, klucz obcy wskazujący na pole activity_id w tabeli activities
start_time	datetime	data rozpoczęcia webinaru	not null
end_time	datetime	data zakończenia webinaru	może być null, jeśli spotkanie jeszcze trwa, musi być późniejsza od start_time
recording_link	nvarchar (200)	link do nagrania webinaru	może być null, jeśli webinar jeszcze trwa / link jeszcze nie został wygenerowany, musi być zgodny z formatem URL, rozpoczynającym się od http
meeting_link	nvarchar (200)	link do spotkania	not null, musi być zgodny z formatem URL, rozpoczynającym się od http
platform_id	int	identyfikator platformy, na której odbywa się webinar	not null, klucz obcy wskazujący na pole platform_id w tabeli online_platforms
tutor_id	int	identyfikator wykładowcy prowadzącego webinar	not null, klucz obcy wskazujący na pole user_id w tabeli tutors
is_paid	bit	informacja o tym czy webinar jest płatny; 0 - bezpłatny 1 - płatny	not null

```
CREATE TABLE webinars (
   activity_id int  NOT NULL,
```

```
   start_time datetime  NOT NULL,
   end_time datetime  NULL,
   recording_link nvarchar(200)  NOT NULL,
   meeting_link nvarchar(200)  NULL,
   platform_id int  NOT NULL,
   tutor_id int  NOT NULL,
   is_paid bit  NOT NULL,
   CONSTRAINT start_before_end CHECK (end_time IS NULL OR end_time >
start_time),
   CONSTRAINT webinars_recording_link_format CHECK (recording_link LIKE
'http%'),
   CONSTRAINT webinars_meeting_link_format CHECK (meeting_link LIKE
'http%'),
   CONSTRAINT webinars_pk PRIMARY KEY  (activity_id)
);
```

```
ALTER TABLE webinars ADD CONSTRAINT activities_webinars
   FOREIGN KEY (activity_id)
   REFERENCES activities (activity_id);
```

```
ALTER TABLE webinars ADD CONSTRAINT online_platforms_webinars
   FOREIGN KEY (platform_id)
   REFERENCES online_platforms (platform_id);
```

```
ALTER TABLE webinars ADD CONSTRAINT tutors_webinars
   FOREIGN KEY (tutor_id)
   REFERENCES tutors (user_id);
```

WEBINAR_ACCESS

Tabela zawierająca informacje na temat dostępu użytkowników do webinarów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
webinar_id	int	identyfikator webinaru, do którego jest określony dostęp	not null, część klucza głównego, klucz obcy wskazujący na pole activity_id w tabeli webinars
user_id	int	identyfikator zarejestrowanego użytkownika, który ma dostęp do kursu	not null, część klucza głównego, klucz obcy wskazujący na pole user_id w tabeli users

access_start_date	datetime	data przydzielenia dostępu do webinaru	not null, musi być wcześniejsza niż access_end_date
access_end_date	datetime	data zakończenia dostępu do webinaru	not null,musi być późniejsza niż access_start_date

```
CREATE TABLE webinar_access (
   webinar_id int  NOT NULL,
   user_id int  NOT NULL,
   access_start_date datetime  NOT NULL,
   access_end_date datetime  NOT NULL,
   CONSTRAINT start_before_end CHECK (access_start_date <
access_end_date),
   CONSTRAINT webinar_access_pk PRIMARY KEY  (webinar_id,user_id)
);
```

```
ALTER TABLE webinar_access ADD CONSTRAINT users_webinar_access
   FOREIGN KEY (user_id)
   REFERENCES users (user_id);
```

```
ALTER TABLE webinar_access ADD CONSTRAINT webinar_access_webinars
   FOREIGN KEY (webinar_id)
   REFERENCES webinars (activity_id);
```

STUDIES

Tabela zawierająca informacje na temat studiów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
study_id	int	unikalny identyfikator studium	not null, klucz główny, klucz obcy wskazujący na pole activity_id w tabeli activities
capacity	int	limit miejsc	not null, musi być większa od 0
coordinator_id	int	identyfikator koordynatora	not null, klucz obcy wskazujący na pole user_id w tabeli tutors
study_name	nvarchar(64)	nazwa studium	not null, unique

```
CREATE TABLE studies (
```

```
   study_id int  NOT NULL,
   capacity int  NOT NULL,
   coordinator_id int  NOT NULL,
   study_name nvarchar(64) NOT NULL,
   CONSTRAINT name_uk UNIQUE(study_name)
   CONSTRAINT capacity CHECK (capacity>0),
   CONSTRAINT studies_pk PRIMARY KEY  (study_id)
);
```

```
ALTER TABLE studies ADD CONSTRAINT studies_activities
   FOREIGN KEY (study_id)
   REFERENCES activities (activity_id);
```

```
ALTER TABLE studies ADD CONSTRAINT studies_tutors
   FOREIGN KEY (coordinator_id)
   REFERENCES tutors (user_id);
```

STUDIES_APPRENTICESHIP

Tabela zawierająca informacje o praktykach w ramach konkretnego studium.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
apprenticeship_id	int	unikalny identyfikator praktyki	not null, klucz główny, auto inkrementowany
study_id	int	identyfikator studium	not null, klucz obcy wskazujący na pole study_id w tabeli studies
year	int	rok	not null, musi być większy niż 2000
term	nvarchar(6)	semestr	not null, musi przyjmować wartość ‘zimowy’ albo ‘letni’

```
CREATE TABLE studies_apprenticeships (
   apprenticeship_id int  NOT NULL IDENTITY(1, 1),
   study_id int  NOT NULL,
   year int  NOT NULL,
   term nvarchar(6)  NOT NULL,
   CONSTRAINT term_ck CHECK (term IN ('Zimowy', 'Letni')),
   CONSTRAINT year_ck CHECK (year > 2000),
   CONSTRAINT studies_apprenticeships_pk PRIMARY KEY
(apprenticeship_id)
```

```
);
```

```
ALTER TABLE studies_apprenticeships ADD CONSTRAINT
studies_apprenticeships_studies
   FOREIGN KEY (study_id)
   REFERENCES studies (study_id);
```

APPRENTICESHIP_DAYS

Tabela zawierająca informacje, kiedy odbyły się pojedyncze dni praktyk.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
apprenticeship_id	int	identyfikator praktyki	not null, część klucza głównego
date	datetime	data odbycia się dnia praktyki	not null, część klucza głównego

```
CREATE TABLE apprenticeship_days (
   apprenticeship_id int  NOT NULL,
   date datetime  NOT NULL,
   CONSTRAINT apprenticeship_days_pk PRIMARY KEY
(apprenticeship_id,date)
);
```

```
ALTER TABLE apprenticeship_days ADD CONSTRAINT
apprenticeship_days_studies_apprenticeships
   FOREIGN KEY (apprenticeship_id)
   REFERENCES studies_apprenticeships (apprenticeship_id);
```

APPRENTICESHIP_PRESENCE

Tabela zawierająca informacje o obecności studenta, na konkretnym dniu konkretnej praktyki.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
student_id	int	identyfikator studenta	not null, część klucza głównego, klucz obcy wskazujący na pole user_id w tabeli users
apprenticeship	int	identyfikator praktyki	not null, część klucza głównego,

_id			klucz obcy wskazujący na pole apprenticeship_id w tabeli apprenticeship_days
date	datetime	dzień odbywania się praktyki	not null, klucz obcy wskazujący na pole date w tabeli apprenticeship_days
is_present	bit	informacja o obecności; 1 - obecny 0 - nieobecny	not null, domyślnie ustawiony na 0

```
CREATE TABLE apprenticeship_presence (
   student_id int  NOT NULL,
   apprenticeship_id int  NOT NULL,
   date datetime  NOT NULL,
   is_present bit  NOT NULL DEFAULT 0,
   CONSTRAINT apprenticeship_presence_pk PRIMARY KEY
(student_id,apprenticeship_id)
);
```

```
ALTER TABLE apprenticeship_presence ADD CONSTRAINT
apprenticeship_days_apprenticeship_attendance
   FOREIGN KEY (apprenticeship_id,date)
   REFERENCES apprenticeship_days (apprenticeship_id,date);
```

```
ALTER TABLE apprenticeship_presence ADD CONSTRAINT
students_apprenticeship_attendance
   FOREIGN KEY (student_id)
   REFERENCES students (user_id);
```

STUDY_MODULES

Tabela zawierająca moduły wchodzące w studia.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
module_id	int	unikalny identyfikator modułu	not null, klucz główny, klucz obcy wskazujący na pole activity_id w tabeli activities
study_id	int	identyfikator studium, do którego należy moduł	not null, klucz obcy wskazujący na pole activity_id w tabeli studies
capacity	int	limit zapisu na dane spotkanie	not null, musi być większe od 0
module_name	nvarchar(64)	nazwa modułu	not null

```
CREATE TABLE study_modules (
   module_id int  NOT NULL,
   study_id int  NOT NULL,
   capacity int  NOT NULL,
   module_name nvarchar(64) NOT NULL
   CONSTRAINT capacity_ck CHECK (capacity > 0),
   CONSTRAINT study_modules_pk PRIMARY KEY  (module_id)
);
```

```
ALTER TABLE study_modules ADD CONSTRAINT studies_modules_activities
   FOREIGN KEY (module_id)
   REFERENCES activities (activity_id);
```

```
ALTER TABLE study_modules ADD CONSTRAINT studies_modules_studies
   FOREIGN KEY (study_id)
   REFERENCES studies (study_id);
```

STUDY_MODULE_MEETINGS

Tabela zawierająca spotkania wchodzące w moduł kursu.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	unikalny identyfikator spotkania	not null, część klucza głównego, klucz obcy wskazujący na pole meeting_id w tabeli meetings
module_id	int	identyfikator modułu, do którego należy spotkanie	not null, część klucza głównego, klucz obcy wskazujący na pole module_id w tabeli course_modules

```
CREATE TABLE study_module_meetings (
   meeting_id int  NOT NULL,
   module_id int  NOT NULL,
   CONSTRAINT study_module_meetings_pk PRIMARY KEY
(meeting_id,module_id)
);
```

```
ALTER TABLE study_module_meetings ADD CONSTRAINT
```

```
meetings_study_studies_modules
   FOREIGN KEY (module_id)
   REFERENCES study_modules (module_id);
```

```
ALTER TABLE study_module_meetings ADD CONSTRAINT meetings_study_meetings
   FOREIGN KEY (meeting_id)
   REFERENCES meetings (meeting_id);
```

MEETINGS

Tabela zawierająca informacje o poszczególnych spotkaniach w ramach spotkań studyjnych i modułów kursów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	unikalny identyfikator spotkania	not null, klucz główny, autoinkremnetowany
tutor_id	int	identyfikator wykładowcy prowadzącego spotkanie	not null, klucz obcy wskazujący na pole user_id w tabeli tutors
language	nvarchar (64)	język, w jakim jest prowadzone spotkanie	not null, domyślnie ustawiony na ‘polski’
start_time	datetime	Data rozpoczęcia spotkania.	not null
end_time	datetime	Data zakończenia spotkania.	może być null, jeśli spotkanie jeszcze trwa, musi być późniejsza niż start_time
activity_id	int	Identyfikator aktywności w ramach której odbywa się spotkanie	not null, klucz obcy wskazujący na pole activity_id w tabeli activities

```
CREATE TABLE meetings (
   meeting_id int  NOT NULL, IDENTITY(1, 1)
   tutor_id int  NOT NULL,
   language nvarchar(64)  NOT NULL DEFAULT 'polski',
   start_time datetime  NOT NULL,
   end_time datetime  NULL,
   activity_id INT NOT NULL
   CONSTRAINT end_time_ck CHECK (end_time IS NULL OR end_time >
start_time),
   CONSTRAINT meetings_pk PRIMARY KEY  (meeting_id)
```

```
);
```

```
ALTER TABLE meetings ADD CONSTRAINT meetings_tutors
   FOREIGN KEY (tutor_id)
   REFERENCES tutors (user_id);
```

```
ALTER TABLE meetings ADD CONSTRAINT meetings_activities
   FOREIGN KEY (activity_id)
   REFERENCES activities (activity_id);
```

MEETING_TRANSLATORS

Tabela przechowuje informacje o tłumaczonych spotkaniach.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	identyfikator spotkania, do którego przypisano tłumacza	not null, część klucza głównego, klucz obcy wskazujący na pole meeting_id w tabeli meetings
translator_id	int	identyfikator tłumacza, przypisanego do danego spotkania	not null, część klucza głównego, klucz obcy wskazujący na pole user_id w tabeli translators
language	nvarchar (64)	język tłumaczenia	not null

```
CREATE TABLE meeting_translators (
   meeting_id int  NOT NULL,
   translator_id int  NOT NULL,
   language nvarchar(64)  NOT NULL,
   CONSTRAINT meeting_translators_pk PRIMARY KEY
(meeting_id,translator_id)
);
```

```
ALTER TABLE meeting_translators ADD CONSTRAINT
meeting_translators_meetings
   FOREIGN KEY (meeting_id)
   REFERENCES meetings (meeting_id);
```

```
ALTER TABLE meeting_translators ADD CONSTRAINT
translators_meeting_translators
   FOREIGN KEY (translator_id)
```

```
   REFERENCES translators (user_id);
```

MEETING_PRESENCE

Tabela zawierająca informacje o obecności danego studenta na konkretnych spotkaniach.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	identyfikator spotkania	not null, część klucza głównego, klucz obcy, wskazujący na pole meeting_id w tabeli meetings
student_id	int	identyfikator obecnego studenta	not null, część klucza głównego, klucz obcy, wskazujący na pole user_id w tabeli students
is_present	bit	określenie, czy student był obecny na spotkaniu; 0 - nieobecny, 1 - obecny	not null, domyślna wartość to 0

```
CREATE TABLE meeting_presence (
   meeting_id int  NOT NULL,
   student_id int  NOT NULL,
   is_present bit  NOT NULL DEFAULT 0,
   CONSTRAINT meeting_presence_pk PRIMARY KEY  (meeting_id,student_id)
);
```

```
ALTER TABLE meeting_presence ADD CONSTRAINT meeting_presence_meetings
   FOREIGN KEY (meeting_id)
   REFERENCES meetings (meeting_id);
```

```
ALTER TABLE meeting_presence ADD CONSTRAINT meeting_presence_students
   FOREIGN KEY (student_id)
   REFERENCES students (user_id);
```

MEETINGS_PRESENCE_MAKE_UP

Tabela przechowująca informacje o obrabianych zajęciach.

Pole	Typ danych	Znaczenie	Warunki integralnościowe

meeting_id	int	identyfikator spotkania, którego obecność jest nadrabiana	not null, część klucza głównego, klucz obcy, wskazujący na pole meeting_id w tabeli meeting_presence
student_id	int	identyfikator studenta, który nadrabia nieobecność	not null, część klucza głównego, klucz obcy, wskazujący na pole user_id w tabeli meeting_presence
make_up_meeting_id	int	identyfikator spotkania, na którym odbyło się odrabianie	not null, klucz obcy wskazujący na pole meeting_id w tabeli meetings

```
CREATE TABLE meeting_presence_make_up (
   meeting_id int  NOT NULL,
   student_id int  NOT NULL,
   make_up_meeting_id int  NOT NULL,
   CONSTRAINT meeting_presence_make_up_pk PRIMARY KEY
(meeting_id,student_id)
);
```

```
ALTER TABLE meeting_presence_make_up ADD CONSTRAINT
meeting_presence_make_up_meeting_presence
   FOREIGN KEY (meeting_id,student_id)
   REFERENCES meeting_presence (meeting_id,student_id);
```

```
ALTER TABLE meetings ADD CONSTRAINT meetings_meeting_presence_make_up
  FOREIGN KEY (meeting_id)
  REFERENCES meeting_presence_make_up (make_up_meeting_id);
```

MEETINGS_IN_PERSON

Tabela zawierająca informacje o spotkaniach stacjonarnych.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	identyfikator spotkania stacjonarnego	not null, klucz główny, klucz obcy, wskazujący na pole meeting_id w tabeli meetings
room_id	int	identyfikator pomieszczenia, w którym odbywa się spotkanie	not null, klucz obcy wskazujący na pole room_id w tabeli rooms

```
CREATE TABLE meetings_in_person (
   meeting_id int  NOT NULL,
   room_id int  NOT NULL,
   CONSTRAINT meetings_in_person_pk PRIMARY KEY  (meeting_id)
);
```

```
ALTER TABLE meetings_in_person ADD CONSTRAINT
meetings_in_person_meetings
   FOREIGN KEY (meeting_id)
   REFERENCES meetings (meeting_id);
```

```
ALTER TABLE meetings_in_person ADD CONSTRAINT rooms_meetings_in_person
   FOREIGN KEY (room_id)
   REFERENCES rooms (room_id);
```

ROOMS

Tabela zawierająca informacje o pomieszczeniach, w których odbywają się stacjonarne spotkania.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
room_id	int	unikalny identyfikator pomieszczenia	not null, klucz główny, autoinkrementacja
room_name	int	nazwa pomieszczenia	not null, unique
capacity	int	limit miejsc	not null, musi być większy od 0

```
CREATE TABLE rooms (
   room_id int  NOT NULL IDENTITY(1, 1),
   room_name nvarchar(64)  NOT NULL,
   capacity int  NOT NULL,
   CONSTRAINT rooms_ak_1 UNIQUE (room_name),
   CONSTRAINT capacity CHECK (capacity>0),
   CONSTRAINT rooms_pk PRIMARY KEY  (room_id)
);
```

MEETINGS_ASYNCHRONOUS

Tabela zawierająca informacje o spotkaniach online, odbywających się asynchronicznie.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	identyfikator spotkania	not null, klucz główny, klucz obcy, wskazujący na pole meeting_id w tabeli meetings
recording_link	nvarchar(200)	link do nagranego spotkania	not null

```
CREATE TABLE meetings_asynchronous (
   meeting_id int  NOT NULL,
   recording_link nvarchar(200)  NOT NULL,
   CONSTRAINT meetings_asynchronous_pk PRIMARY KEY  (meeting_id)
);
```

```
ALTER TABLE meetings_asynchronous ADD CONSTRAINT
meetings_asynchronous_meetings
   FOREIGN KEY (meeting_id)
   REFERENCES meetings (meeting_id);
```

MEETINGS_SYNCHRONOUS

Tabela zawierająca informacje o spotkaniach online, odbywających się synchronicznie.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
meeting_id	int	identyfikator spotkania	not null, klucz główny, klucz obcy, wskazujący na pole meeting_id w tabeli meetings
recording_link	nvarchar(200)	link do nagranego spotkania	może być null, jeśli spotkanie jeszcze trwa / link jeszcze nie wygenerowany, musi zaczynać się od ‘http’
meeting_link	nvarchar(200)	link do spotkania	not null, musi zaczynać się od ‘http’
platform_id	int	identyfikator platformy, na której odbywa się spotkanie	not null, klucz obcy wskazujący na pole platfrom_id w tabeli online_platforms

```
CREATE TABLE meetings_synchronous (
   meeting_id int  NOT NULL,
   recording_link nvarchar(200)  NULL,
   meeting_link nvarchar(200)  NOT NULL,
   platform_id int  NOT NULL,
```

```
   CONSTRAINT meeting_link_ch CHECK (meeting_link LIKE 'http%'),
   CONSTRAINT recording_link_ch CHECK (recording_link LIKE 'http%'),
   CONSTRAINT meetings_synchronous_pk PRIMARY KEY  (meeting_id)
);
```

```
ALTER TABLE meetings_synchronous ADD CONSTRAINT
meetings_synchronous_online_platforms
   FOREIGN KEY (platform_id)
   REFERENCES online_platforms (platform_id);
```

```
ALTER TABLE meetings_synchronous ADD CONSTRAINT
meetings_synchronous_meetings
   FOREIGN KEY (meeting_id)
   REFERENCES meetings (meeting_id);
```

ONLINE_PLATFORMS

Tabela określająca platformy używane podczas zdalnych zajęć

Pole	Typ danych	Znaczenie	Warunki integralnościowe
platform_id	int	identyfikator platformy	not null, klucz główny, autoinkrementacja
platform_name	nvarchar(64)	nazwa platformy	not null

```
CREATE TABLE online_platforms (
   platform_id int  NOT NULL IDENTITY(1, 1),
   platform_name nvarchar(64)  NOT NULL,
   CONSTRAINT online_platforms_ak_1 UNIQUE (platform_name),
   CONSTRAINT online_platforms_pk PRIMARY KEY  (platform_id)
);
```

Tabele dotyczące zamówień i płatności

SHOPPING_CART

Tabela przechowująca informacje o produktach znajdujących się w koszykach zakupowych studentów.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
product_id	int	identyfikator produktu	not null, część klucza głównego, klucz obcy wskazujący pole

			product_id w tabeli products
user_id	int	identyfikator użytkownika, do którego należy koszyk	not null, część klucza głównego, klucz obcy wskazujący na pole user_id w users

```
CREATE TABLE shopping_cart (
   product_id int  NOT NULL,
   student_id int  NOT NULL,
   CONSTRAINT shopping_cart_pk PRIMARY KEY  (product_id,student_id)
);
```

```
ALTER TABLE shopping_cart ADD CONSTRAINT shopping_cart_users
   FOREIGN KEY (user_id)
   REFERENCES users (user_id);
```

```
ALTER TABLE shopping_cart ADD CONSTRAINT shopping_cart_products
   FOREIGN KEY (product_id)
   REFERENCES products (product_id);
```

PRODUCTS

Tabela zawierająca wszystkie możliwe do kupienia produkty (kursy, płatne webinary, studia, pojedyncze moduły w ramach studium itp.) oraz ich cenę.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
product_id	int	identyfikator produktu	not null, klucz główny, klucz obcy jako wskazanie na pole activity_id w tabeli activities
price	money	cena produktu	not null, musi być większe od 0

```
CREATE TABLE products (
   product_id int  NOT NULL,
   price money  NOT NULL CHECK (price>=0),
   CONSTRAINT products_pk PRIMARY KEY  (product_id)
);
```

```
ALTER TABLE products ADD CONSTRAINT products_activities
   FOREIGN KEY (product_id)
   REFERENCES activities (activity_id);
```

ORDERS

Tabela przechowująca zamówienia dokonane przez użytkownika.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
order_id	int	unikalny identyfikator zamówienia	not null, klucz główny, autoinkrementacja
student_id	int	identyfikator studenta, dokonującego zamówienie	not null, klucz obcy wskazujący pole user_id w tabeli users
order_date	datetime	data dokonania zamówienia	not null
status_id	int	identyfikator statusu zamówienia	not null, klucz obcy wskazujący pole status_id w tabeli order_statuses

```
CREATE TABLE orders (
   order_id int  NOT NULL IDENTITY(1, 1),
   user_id int  NOT NULL,
   order_date datetime  NOT NULL,
   status_id int  NOT NULL,
   CONSTRAINT order_id PRIMARY KEY  (order_id)
);
```

```
ALTER TABLE orders ADD CONSTRAINT orders_users
   FOREIGN KEY (user_id)
   REFERENCES users (user_id);
```

```
ALTER TABLE orders ADD CONSTRAINT order_statuses_orders
   FOREIGN KEY (status_id)
   REFERENCES order_statuses (status_id);
```

ORDER_DETAILS

Tabela przechowująca informacje o szczegółach konkretnego zamówienia związane z konkretnym produktem.

Pole	Typ danych	Znaczenie	Warunki integralnościowe

order_id	int	identyfikator zamówienia	not null, część klucza głównego, klucz obcy jako wskazanie na pole order_id w tabeli orders
product_id	int	produkt związany z zamówieniem	not null, część klucza głównego, klucz obcy jako wskazanie na pole product_id w tabeli products
price	money	cena produktu w momencie zakupu	not null, musi być większa od 0

```
CREATE TABLE order_details (
   order_id int  NOT NULL,
   product_id int  NOT NULL,
   price money  NOT NULL,
   CONSTRAINT price_ck CHECK (price > 0),
   CONSTRAINT order_details_pk PRIMARY KEY  (product_id,order_id)
);
```

```
ALTER TABLE order_details ADD CONSTRAINT products_order_information
   FOREIGN KEY (product_id)
   REFERENCES products (product_id);
```

```
ALTER TABLE order_details ADD CONSTRAINT order_information_orders
   FOREIGN KEY (order_id)
   REFERENCES orders (order_id);
```

ORDER_STATUSES

Tabela przechowująca informacje o możliwych statusach zamówień.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
status_id	int	unikalny identyfikator statusu	not null, klucz główny
status_name	nvarchar(64)	nazwa statusu	not null, przyjmuje jedną z wartości ‘Pending’, ‘Paid’, ‘Processing’, ‘Cancelled’, ‘Refunded’, ‘Received’’, unique

```
CREATE TABLE order_statuses (
   status_id int  NOT NULL,
   status_name nvarchar(64)  NOT NULL,
```

```
   CONSTRAINT order_statuses_pk PRIMARY KEY  (status_id)
   CONSTRAINT status_name_ck CHECK (status_name IN (N'Pending',
N'Paid', N'Processing', N'Cancelled', N'Refunded', N'Received')),
   CONSTRAINT name_uq UNIQUE(status_name)
);
```

PAYMENTS

Tabela zawierająca informacje o płatnościach.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
payment_id	int	unikalny identyfikator płatności	not null, klucz główny, autoinkrementacja
order_id	nvarchar(64)	identyfikator zamówienia, dla którego została dokonana płatność	not null, klucz obcy jako wskazanie na pole order_id w tabeli orders
currency	nvarchar(3)	rodzaj waluty	not null, domyślnie jest ustawiona na ‘PLN’, może przyjmować wartość ‘PLN’ lub ‘EUR’
payment_date	datetime	data dokonania płatności	not null
amount	money	zapłacona kwota	not null, musi być większa od 0
status_id	int	identyfikator statusu płatności	not null, klucz obcy jako wskazanie na polece status_id w tabeli payment_statuses

```
CREATE TABLE payments (
   payment_id int  NOT NULL IDENTITY(1, 1),
   order_id int  NOT NULL,
   currency nvarchar(3)  NOT NULL DEFAULT 'PLN',
   payment_date datetime  NOT NULL,
   amount money  NOT NULL,
   status_id int  NOT NULL,
   CONSTRAINT payments_ak_1 UNIQUE (order_id),
   CONSTRAINT amount_positive CHECK (amount > 0),
   CONSTRAINT currency_options CHECK (currency IN ('PLN', 'EUR')),
   CONSTRAINT payments_pk PRIMARY KEY  (payment_id)
);
```

```
ALTER TABLE payments ADD CONSTRAINT payments_orders
   FOREIGN KEY (order_id)
   REFERENCES orders (order_id);
```

```
ALTER TABLE payments ADD CONSTRAINT payments_statuses_payments
   FOREIGN KEY (status_id)
   REFERENCES payment_statuses (status_id);
```

PAYMENT_STATUSES

Tabela przechowująca informacje o możliwych statusach płatności.

Pole	Typ danych	Znaczenie	Warunki integralnościowe
status_id	int	unikalny identyfikator statusu	not null, klucz główny
status_name	nvarchar(9)	nazwa statusu	not null, przyjmuje jedną z wartości ‘pending’, ‘completed’, ‘failed’, jest unikalna

```
CREATE TABLE payment_statuses (
   status_id int  NOT NULL,
   status_name nvarchar(9)  NOT NULL,
   CONSTRAINT status_options CHECK (status_name IN ('pending',
'completed', 'failed')),
   CONSTRAINT status_uq UNIQUE(status_name)
   CONSTRAINT payment_statuses_pk PRIMARY KEY  (status_id)
);
```

Wygenerowane dane

Wszystkie tabele wypełniono danymi testowymi, używając do ich wygenerowania dbForge Data Generator for SQL Server. Gdzie tylko było to możliwe wygenerowano po 50 rekordów. Dane te posłużyły m.in. do testów procedur, widoków i triggerów.

Spis widoków

FINANCIAL REPORT

Działanie:

Wyświetla całkowity przychód dla każdego webinaru, kursu lub studium.

```
CREATE VIEW FinancialReport AS
```

```
SELECT
    a.activity_id,
    a.description AS activity_name,
    SUM(od.price) AS total_income,
    COUNT(DISTINCT o.order_id) AS total_orders
FROM
    activities a
INNER JOIN products p ON a.activity_id = p.product_id
INNER JOIN order_details od ON p.product_id = od.product_id
INNER JOIN orders o ON od.order_id = o.order_id
GROUP BY
    a.activity_id, a.description;
```

DEBTORS

Działanie:

Wyświetla listę użytkowników, którzy mają zaległe płatności.

```
CREATE VIEW Debtors AS
SELECT
    u.user_id,
    u.first_name,
    u.last_name,
    SUM(od.price) AS total_due
FROM
    orders o
INNER JOIN order_details od ON o.order_id = od.order_id
INNER JOIN products p ON od.product_id = p.product_id
INNER JOIN users u ON o.user_id = u.user_id
LEFT JOIN payments py ON o.order_id = py.order_id
WHERE
    py.status_id IS NULL - Brak płatności
    OR py.status_id <> (SELECT TOP 1 status_id FROM payment_statuses
WHERE status_name = 'completed')
GROUP BY
    u.user_id, u.first_name, u.last_name;
```

UPCOMING EVENTS ATTENDANCE

Działanie:

Wyświetla liczbę zapisanych osób na przyszłe wydarzenia oraz ich rodzaj.

```
CREATE VIEW UpcomingEventsAttendance AS
```

```sql
SELECT
    a.activity_id,
    a.description AS activity_name,
    CASE
        WHEN mip.meeting_id IS NOT NULL THEN 'stacjonarne' -- Jeżeli
spotkanie jest stacjonarne
        ELSE 'zdalne' -- Jeżeli zdalne
    END AS event_type,
    COUNT(DISTINCT mp.student_id) AS total_participants
FROM
    activities a
INNER JOIN meetings m ON a.activity_id = m.activity_id
LEFT JOIN meeting_presence mp ON m.meeting_id = mp.meeting_id -- Osoby
które powinny być na spotkaniu
LEFT JOIN meetings_in_person mip ON m.meeting_id = mip.meeting_id
WHERE
    m.start_time > GETDATE() -- Przyszłe zajęcia
GROUP BY
    a.activity_id, a.description,
    CASE
        WHEN mip.meeting_id IS NOT NULL THEN 'stacjonarne'
        ELSE 'zdalne'
    END;
```

EVENT ATTENDANCE

Działanie:

Wyświetla dane dotyczące obecności na zakończonych wydarzeniach.

```sql
CREATE VIEW EventAttendance AS
SELECT
    a.activity_id,
    a.description AS activity_name,
    COUNT(CASE WHEN mp.is_present = 1 THEN mp.student_id END) AS
total_present, -- Zlicz obecnych
    COUNT(m.meeting_id) - COUNT(CASE WHEN mp.is_present = 1 THEN
mp.student_id END) AS total_absent -- Zlicz nieobecnych
FROM
    activities a
INNER JOIN meetings m ON a.activity_id = m.activity_id -- Poprawione
połączenie
LEFT JOIN meeting_presence mp ON m.meeting_id = mp.meeting_id -- Dodano
uczestników
```

```
WHERE
    m.end_time < GETDATE() -- Tylko przeszłe wydarzenia
GROUP BY
    a.activity_id, a.description;
```

TIME COLLISIONS

Działanie:

Wyświetla listę osób zapisanych na co najmniej dwa kolidujące ze sobą przyszłe wydarzenia

```
CREATE VIEW TimeCollisions AS
SELECT DISTINCT
    mp1.student_id,
    u.first_name,
    u.last_name,
    m1.start_time AS event_1_start,
    m1.end_time AS event_1_end,
    m2.start_time AS event_2_start,
    m2.end_time AS event_2_end
FROM
    meeting_presence mp1
INNER JOIN meeting_presence mp2
    ON mp1.student_id = mp2.student_id
    AND mp1.meeting_id < mp2.meeting_id -- Unikaj duplikatów
INNER JOIN meetings m1 ON mp1.meeting_id = m1.meeting_id
INNER JOIN meetings m2 ON mp2.meeting_id = m2.meeting_id
INNER JOIN users u ON mp1.student_id = u.user_id
WHERE
    m1.start_time < m2.end_time -- Sprawdzenie, czy kolidują czasowo
    AND m1.end_time > m2.start_time
    AND m1.start_time > GETDATE(); -- Tylko przyszłe wydarzenia
```

ORDER DETAILS

Działanie:

Wyświetla szczegóły danego zamówienia

```
CREATE VIEW OrderDetails AS
SELECT
    o.order_id,
    o.user_id,
    o.order_date,
```

```
    od.product_id,
    od.price,
    os.status_name AS status
FROM orders o
INNER JOIN order_details od ON o.order_id = od.order_id
INNER JOIN order_statuses os ON o.status_id = os.status_id;
```

ORDERS BY STUDENT

Działanie:

Wyświetla informacje o zamówieniach danego studenta wraz ze szczegółami

```
CREATE VIEW OrdersByStudent AS
SELECT
    o.order_id,
    o.user_id,
    u.first_name,
    u.last_name,
    o.order_date,
    od.product_id,
    od.price,
    os.status_name AS status
FROM orders o
INNER JOIN users u ON o.user_id = u.user_id
INNER JOIN order_details od ON o.order_id = od.order_id
INNER JOIN order_statuses os ON o.status_id = os.status_id;
```

ATTENDANCE LIST

Działanie:

Wyświetla listę obecności studentów na spotkaniu

```
CREATE VIEW AttendanceList AS
SELECT
    mp.student_id,
    u.first_name,
    u.last_name,
    m.start_time AS meeting_date,
    mp.is_present
FROM
    meeting_presence mp
INNER JOIN
    students s ON mp.student_id = s.user_id
```

```
INNER JOIN
    users u ON s.user_id = u.user_id
INNER JOIN
    meetings m ON mp.meeting_id = m.meeting_id;
```

STUDY'S MODULES

Działanie:

Wyświetla moduły, z których składa się dany kurs

```
CREATE VIEW StudyModules AS
SELECT
    sm.module_id,
    sm.study_id,
    a.description AS activity_name
FROM
    study_modules sm
INNER JOIN
    activities a ON sm.module_id = a.activity_id;
```

Spis procedur

ADD COURSE

Procedura AddCourse służy do dodawania nowego kursu do bazy danych. Rozpoczyna się od sprawdzenia, czy kurs o podanej nazwie już istnieje. Jeśli kurs jest już obecny, transakcja jest wycofywana. Następnie dodawana jest nowa aktywność związana z kursem, a koordynator kursu jest weryfikowany, by upewnić się, że jest aktywnym tutorem.

Po pozytywnej weryfikacji, kurs jest dodawany do bazy, a jego cena zapisywana w osobnej tabeli. Jeśli zostały podane moduły kursu, są one przetwarzane i przypisywane do odpowiedniego kursu. Cały proces jest realizowany w ramach transakcji, która jest zatwierdzana lub wycofywana w przypadku błędów.

```
CREATE PROCEDURE [dbo].[AddCourse]
    @CourseName NVARCHAR(64),
    @Description NVARCHAR(100) = NULL,
    @CoordinatorEmail NVARCHAR(64),
      @Price money,
    @Modules NVARCHAR(MAX) = NULL -- Opcjonalna lista modułów rozdzielona
przecinami
AS
BEGIN
```

```
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @ActivityId INT;
        DECLARE @CoordinatorId INT;

        -- Sprawdzenie, czy kurs o podanej nazwie już istnieje
        IF EXISTS (SELECT 1 FROM courses WHERE course_name =
@CourseName)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Kurs o podanej nazwie już istnieje.';
            RETURN -1; -- Kod błędu: Kurs już istnieje
        END;

        -- Dodanie aktywności
        INSERT INTO activities (description)
        VALUES (@Description);

        SET @ActivityId = SCOPE_IDENTITY();

        -- Pobranie ID koordynatora
        SELECT @CoordinatorId = user_id
        FROM users
        WHERE email = @CoordinatorEmail;

        IF @CoordinatorId IS NULL
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Nie ma użytkownika o takim adresie email.';
            RETURN -1; -- Kod błędu: Brak użytkownika
        END;

        -- Sprawdzenie, czy użytkownik jest aktywnym tutorem
        IF NOT EXISTS (SELECT 1 FROM tutors WHERE user_id =
@CoordinatorId AND is_active = 1)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Użytkownik nie jest tutorem lub jest nieaktywny.';
            RETURN -2;
        END;

        -- Dodanie kursu
        INSERT INTO courses (activity_id, coordinator_id, course_name)
        VALUES (@ActivityId, @CoordinatorId, @CourseName);
```

```
            INSERT INTO products (product_id, price)
           VALUES (@ActivityId, @Price);

        -- Przetwarzanie modułów, jeśli podano
        IF @Modules IS NOT NULL AND LEN(@Modules) > 0
        BEGIN
            DECLARE @Module NVARCHAR(64);
            DECLARE @Pos INT = 1;
            DECLARE @Len INT;
            DECLARE @CommaPos INT;
                  DECLARE @ModuleId INT;

            SET @Len = LEN(@Modules);

            WHILE @Pos <= @Len
            BEGIN
                SET @CommaPos = CHARINDEX(',', @Modules, @Pos);

                IF @CommaPos = 0
                BEGIN
                    SET @Module = SUBSTRING(@Modules, @Pos, @Len - @Pos
+ 1);
                    SET @Pos = @Len + 1;
                END
                ELSE
                BEGIN
                    SET @Module = SUBSTRING(@Modules, @Pos, @CommaPos -
@Pos);
                    SET @Pos = @CommaPos + 1;
                END;

                SET @Module = LTRIM(RTRIM(@Module));

                -- Dodanie modułu do tabeli activities (jako moduł)
                INSERT INTO activities DEFAULT VALUES;


                SET @ModuleId = SCOPE_IDENTITY();

                -- Dodanie modułu do tabeli course_modules
                INSERT INTO course_modules (module_id, course_id,
module_name)
                VALUES (@ModuleId, @ActivityId, @Module);
            END;
        END;
```

```
        COMMIT TRANSACTION;

        RETURN 0;
    END TRY
    BEGIN CATCH
        PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        ROLLBACK TRANSACTION;
        RETURN -3;
    END CATCH;
END;
```

ADD COURSE MODULE

Procedura AddCourseModule jest odpowiedzialna za dodanie nowego modułu do kursu. Rozpoczyna się od sprawdzenia, czy kurs o podanej nazwie istnieje w tabeli courses. Jeśli kurs nie istnieje, procedura przerywa działanie, wycofując transakcję i zwracając odpowiedni kod błędu.

Następnie procedura sprawdza, czy moduł o podanej nazwie już istnieje w ramach kursu. Jeśli taki moduł istnieje, transakcja jest również wycofywana, a procedura kończy się z kodem błędu wskazującym na duplikat.

Po pozytywnych weryfikacjach, procedura dodaje nowy moduł do tabeli activities (z opisem), a następnie tworzy powiązanie modułu z kursem w tabeli course_modules, zapisując odpowiednie identyfikatory. Jeśli wszystkie operacje zakończą się sukcesem, transakcja jest zatwierdzana, a procedura zwraca kod sukcesu. W przypadku błędu transakcja jest wycofywana, a procedura zwraca kod błędu.

```
CREATE PROCEDURE [dbo].[AddCourseModule]
    @ModuleName NVARCHAR(64),
    @ModuleDescription NVARCHAR(100) = NULL,
    @CourseName NVARCHAR(64)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @CourseId INT;
        DECLARE @ModuleId INT;

        SET @CourseName = LTRIM(RTRIM(@CourseName));

        -- Sprawdzenie, czy kurs o podanej nazwie istnieje
        SELECT @CourseId = activity_id FROM courses WHERE course_name =
@CourseName;
```

```
        IF @CourseId IS NULL
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Kurs o takiej nazwie nie istnieje.';
            RETURN -1;
        END

        -- Sprawdzenie, czy moduł już istnieje w ramach tego kursu
        IF EXISTS (SELECT 1 FROM course_modules WHERE course_id =
@CourseId AND module_name = @ModuleName)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Moduł istnieje już w ramach tego kursu.';
            RETURN -2;
        END

        -- Dodanie modułu do tabeli activities (jako moduł)
        INSERT INTO activities (description)
        VALUES (@ModuleDescription);

        SET @ModuleId = SCOPE_IDENTITY();

        -- Dodanie modułu do tabeli course_modules
        INSERT INTO course_modules (course_id, module_id, module_name)
        VALUES (@CourseId, @ModuleId, @ModuleName);

        COMMIT TRANSACTION;
        RETURN 0;

    END TRY
    BEGIN CATCH
        PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        ROLLBACK TRANSACTION;
        RETURN -3;
    END CATCH
END;
```

ADD STUDY

Procedura AddStudy służy do dodawania nowego studium do bazy danych. Rozpoczyna działanie od sprawdzenia, czy studium o podanej nazwie już istnieje w tabeli studies.

Kolejnym krokiem jest dodanie aktywności związanej ze studium do tabeli activities. W tym celu zapisuje się opis aktywności, a identyfikator aktywności jest przypisany do zmiennej.

Procedura następnie sprawdza, czy podany koordynator (na podstawie adresu e-mail) istnieje w tabeli users. Dodatkowo, sprawdzane jest, czy użytkownik jest aktywnym tutorem.

Po pozytywnych weryfikacjach, procedura dodaje studium do tabeli studies., a jego cena zapisywana w tabeli products

Na koniec, jeśli wszystkie operacje zakończą się sukcesem, transakcja jest zatwierdzana. W przeciwnym przypadku, w razie błędu, transakcja jest wycofywana, a procedura zwraca kod błędu.

```
CREATE PROCEDURE [dbo].[AddStudy]
      @StudyName NVARCHAR(64),
    @Description NVARCHAR(100) = NULL,
    @CoordinatorEmail NVARCHAR(64),
      @Capacity int,
      @Price money
AS
BEGIN
      BEGIN TRY
            BEGIN TRANSACTION
            DECLARE @ActivityId INT;
        DECLARE @CoordinatorId INT;

        -- Sprawdzenie, czy kurs o podanej nazwie już istnieje
        IF EXISTS (SELECT 1 FROM studies WHERE study_name = @StudyName)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Studium o podanej nazwie już istnieje.';
            RETURN -1;
        END;

            -- Dodanie aktywności
        INSERT INTO activities (description)
        VALUES (@Description);

        SET @ActivityId = SCOPE_IDENTITY();

            -- Pobranie ID koordynatora
        SELECT @CoordinatorId = user_id
        FROM users
        WHERE email = @CoordinatorEmail;

        IF @CoordinatorId IS NULL
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Nie ma użytkownika o takim adresie email.';
            RETURN -2; -- Kod błędu: Brak użytkownika
```

```
        END;

        -- Sprawdzenie, czy użytkownik jest aktywnym tutorem
        IF NOT EXISTS (SELECT 1 FROM tutors WHERE user_id =
@CoordinatorId AND is_active = 1)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Użytkownik nie jest tutorem lub jest nieaktywny.';
            RETURN -3;
        END;

            -- Dodanie studium
        INSERT INTO studies (study_id, capacity, coordinator_id,
study_name)
        VALUES (@ActivityId, @Capacity, @CoordinatorId, @StudyName);

            -- Dodanie do tabeli products
            INSERT INTO products(product_id, price)
            VALUES (@ActivityId, @Price)

        COMMIT TRANSACTION;

        RETURN 0;

      END TRY

      BEGIN CATCH
            PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        ROLLBACK TRANSACTION;
        RETURN -4;
      END CATCH
END
```

ADD STUDY MODULE

Procedura AddStudyModule służy do dodawania nowego modułu do studium. Rozpoczyna się od sprawdzenia, czy studium o podanej nazwie istnieje w tabeli studies.

Następnie procedura sprawdza, czy moduł o podanej nazwie już istnieje w ramach tego studium.

Jeśli obie weryfikacje zakończą się pomyślnie, procedura dodaje nowy moduł do tabeli activities (z opisem), a następnie zapisuje go w tabeli study_modules, wiążąc go z odpowiednim studium. Dodatkowo, cena modułu jest zapisywana w tabeli products.

Na koniec, jeśli wszystkie operacje zakończą się sukcesem, transakcja jest zatwierdzana, a procedura zwraca kod sukcesu. W przypadku jakiegokolwiek błędu transakcja jest wycofywana i zwracany jest kod błędu.

```
CREATE PROCEDURE [dbo].[AddStudyModule]
      @ModuleName nvarchar(64),
      @ModuleDescription nvarchar(100) = NULL,
      @StudyName nvarchar(64),
      @Price money
AS
BEGIN
      BEGIN TRY
            BEGIN TRANSACTION
            DECLARE @StudyId INT;
            DECLARE @ModuleId INT;

            SET @StudyName = LTRIM(RTRIM(@StudyName));
            SELECT @StudyId = study_id FROM studies WHERE study_name =
@StudyName;

            IF @StudyId IS NULL
            BEGIN
                  ROLLBACK TRANSACTION
                  PRINT 'Studium o takiej nazwie nie istnieje.'
                  RETURN -1
            END

            IF EXISTS (SELECT 1 FROM study_modules WHERE study_id =
@StudyId AND module_name= @ModuleName)
            BEGIN
                  ROLLBACK TRANSACTION
                  PRINT 'Moduł istnieje już w ramach tego kursu.'
                  RETURN -2
            END

             INSERT INTO activities (description)
         VALUES (@ModuleDescription);

             SET @ModuleId = SCOPE_IDENTITY();

             INSERT INTO study_modules(study_id, module_id, module_name)
             VALUES (@StudyId, @ModuleId, @ModuleName)

             INSERT INTO products(product_id, price)
             VALUES(@ModuleId, @Price)
```

```
            COMMIT TRANSACTION
            RETURN 0
      END TRY
      BEGIN CATCH

      END CATCH
END
```

ADD WEBINAR

Procedura AddWebina służy do dodawania nowego webinaru do bazy danych. Rozpoczyna się od sprawdzenia, czy webinar o podanej nazwie już istnieje w tabeli webinars.

Następnie, procedura pobiera identyfikator wykładowcy na podstawie podanego adresu e-mail. Dodatkowo, sprawdzane jest, czy użytkownik jest aktywnym tutorem..

Kolejnym krokiem jest sprawdzenie, czy platforma, na której odbywa się dany webinar już istnieje w tabeli online_platforms. Jeśli platforma nie istnieje, zostaje dodana do bazy danych, a jej identyfikator jest pobierany.

Po tych weryfikacjach, procedura dodaje aktywność związana z webinaru do tabeli activities, a następnie wstawia informacje o webinarze do tabeli webinars, łącząc go z wykładowcą i platformą. Jeśli cena webinaru jest większa niż 0, procedura dodaje informacje o produkcie (webinarze) i jego cenie do tabeli products.

Na koniec, jeśli wszystkie operacje zakończą się sukcesem, transakcja jest zatwierdzana. W przypadku jakiegokolwiek błędu transakcja jest wycofywana, a procedura zwraca kod błędu.

```
CREATE PROCEDURE [dbo].[AddWebinar]
      @WebinarName nvarchar(64),
      @StartTime datetime,
      @EndTime datetime = NULL,
      @RecordingLink nvarchar(200)=NULL,
      @MeetingLink nvarchar(200),
      @PlatformName nvarchar(64),
      @TutorEmail nvarchar(200),
      @WebinarDescription nvarchar(200)=NULL,
      @Price money
AS
BEGIN
      BEGIN TRY
            BEGIN TRANSACTION
            DECLARE @WebinarId INT;
            DECLARE @TutorId INT;
            DECLARE @PlatformId INT;
            DECLARE @IsPaid BIT;
```

```
            IF EXISTS(SELECT 1 FROM webinars WHERE webinar_name=
@WebinarName)
            BEGIN
                  ROLLBACK TRANSACTION
                  RETURN -1
            END

             -- Pobranie ID wykładowcy
        SELECT @TutorId = user_id
        FROM users
        WHERE email = @TutorEmail;

        IF @TutorId IS NULL
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Nie ma użytkownika o takim adresie email.';
            RETURN -2; -- Kod błędu: Brak użytkownika
        END;

        -- Sprawdzenie, czy użytkownik jest aktywnym tutorem
        IF NOT EXISTS (SELECT 1 FROM tutors WHERE user_id = @TutorId AND
is_active = 1)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Użytkownik nie jest tutorem lub jest nieaktywny.';
            RETURN -3;
        END;

             -- Sprawdzenie, czy wpisana platforma już istnieje
      SELECT @PlatformId = platform_id
      FROM online_platforms
      WHERE platform_name = @PlatformName;

      -- Jeśli platforma nie istnieje, dodajemy ją
      IF @PlatformId IS NULL
      BEGIN
         INSERT INTO online_platforms(platform_name)
         VALUES (@PlatformName);

         SET @PlatformId = SCOPE_IDENTITY();  -- Pobranie ID platformy
      END

            -- Dodanie aktywności
        INSERT INTO activities (description)
        VALUES (@WebinarDescription);
```

```
            SET @WebinarId = SCOPE_IDENTITY()

            IF @Price > 0
            BEGIN
                  SET @IsPaid = 1
                  INSERT INTO products (product_id, price)
            VALUES (@WebinarId, @Price);
            END
            ELSE
            BEGIN
                  SET @IsPaid = 1
            END


             -- Dodanie webinaru
        INSERT INTO webinars (activity_id, start_time, end_time,
recording_link, meeting_link, tutor_id, is_paid, platform_id)
        VALUES (@WebinarId, @StartTime, @EndTime, @RecordingLink,
@MeetingLink, @TutorId, @IsPaid, @PlatformId);

            COMMIT TRANSACTION
            RETURN 0;


      END TRY

      BEGIN CATCH
            PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        ROLLBACK TRANSACTION;
        RETURN -4;
      END CATCH
END
```

ADD STUDENT

Procedura AddStudent służy do dodawania nowego studenta do systemu. Proces rozpoczyna się od sprawdzenia, czy użytkownik z podanym adresem e-mail oraz numerem telefonu już istnieje w systemie. W przeciwnym razie, jeśli użytkownik nie jest studentem, kontynuowana jest rejestracja nowego studenta.

Jeśli użytkownik nie istnieje w systemie, zostaje dodany do tabeli users. Następnie sprawdzane jest, czy kraj studenta istnieje w tabeli countries. Jeżeli nie, kraj jest dodawany do bazy, a jego identyfikator jest pobierany. Potem sprawdzane jest, czy miasto w danym

kraju istnieje w tabeli cities. Jeśli miasto nie istnieje, zostaje dodane do bazy danych, a identyfikator miasta jest pobierany.

Po dodaniu użytkownika, kraju i miasta, procedura dodaje nowego studenta do tabeli students oraz jego dane adresowe do tabeli addresses

Jeśli wszystkie operacje zakończą się sukcesem, transakcja jest zatwierdzana. Jeśli wystąpi jakikolwiek błąd, transakcja jest wycofywana, a procedura zwraca kod błędu.

```
CREATE PROCEDURE [dbo].[AddStudent]
   @Email nvarchar(64),
   @Password nvarchar(64),
   @FirstName nvarchar(64),
   @LastName nvarchar(64),
   @Phone nvarchar(16),
   @Street nvarchar(100),
   @ZipCode nvarchar(6),
   @CityName nvarchar(100),
   @CountryName nvarchar(100)
AS
BEGIN
   BEGIN TRY
      -- Rozpoczęcie transakcji
      BEGIN TRANSACTION;

      DECLARE @UserId INT;
      DECLARE @CityId INT;
      DECLARE @CountryId INT;

      -- Sprawdzenie, czy użytkownik już istnieje
      SELECT @UserId = user_id
      FROM users
      WHERE email = @Email AND phone = @Phone;

      IF @UserId IS NOT NULL
      BEGIN
         -- Sprawdzenie, czy użytkownik jest już studentem
         IF EXISTS (
            SELECT 1
            FROM students
            WHERE user_id = @UserId
         )
         BEGIN
            -- Wycofanie transakcji, jeśli użytkownik jest już studentem
            ROLLBACK TRANSACTION;
```

```
            RETURN -1; -- Kod błędu: użytkownik już istnieje jako student
         END
         -- Użytkownik istnieje, ale nie jest studentem; kontynuujemy
proces
      END
      ELSE
      BEGIN
         -- Dodanie nowego użytkownika do tabeli users
         INSERT INTO users (email, password, first_name, last_name,
phone)
         VALUES (@Email, @Password, @FirstName, @LastName, @Phone);

         SET @UserId = SCOPE_IDENTITY();  -- Pobranie UserId
      END

      -- Sprawdzenie, czy kraj już istnieje
      SELECT @CountryId = country_id
      FROM countries
      WHERE country_name = @CountryName;

      -- Jeśli kraj nie istnieje, dodajemy go
      IF @CountryId IS NULL
      BEGIN
         INSERT INTO countries (country_name)
         VALUES (@CountryName);

         SET @CountryId = SCOPE_IDENTITY();  -- Pobranie ID dodanego
kraju
      END

      -- Sprawdzenie, czy miasto już istnieje w danym kraju
      SELECT @CityId = city_id
      FROM cities
      WHERE city_name = @CityName AND country_id = @CountryId;

      -- Jeśli miasto nie istnieje, dodajemy je
      IF @CityId IS NULL
      BEGIN
         INSERT INTO cities (city_name, country_id)
         VALUES (@CityName, @CountryId);

         SET @CityId = SCOPE_IDENTITY();  -- Pobranie ID dodanego miasta
      END

      -- Dodanie użytkownika do tabeli students
      INSERT INTO students (user_id)
```

```
        VALUES (@UserId);

        -- Dodanie adresu studenta do tabeli addresses (bez IDENTITY)
        INSERT INTO addresses (student_id, street, zip_code, city_id)
        VALUES (@UserId, @Street, @ZipCode, @CityId);

        -- Zatwierdzenie transakcji po poprawnym zakończeniu operacji
        COMMIT TRANSACTION;
        RETURN 0;  -- Zwrócenie kodu sukcesu
    END TRY
    BEGIN CATCH
        -- Wycofanie transakcji w przypadku błędu
        PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        ROLLBACK TRANSACTION;
        RETURN -3;  -- Zwrócenie kodu błędu w przypadku wyjątku
    END CATCH
END;
```

ADD TUTOR

Procedura AddTutor służy do dodawania nowego tutora do systemu. Proces rozpoczyna się od sprawdzenia, czy użytkownik z podanym adresem e-mail oraz numerem telefonu już istnieje w systemie. Jeśli taki użytkownik istnieje, procedura sprawdza, czy jest już zarejestrowany jako tutor.

Jeśli użytkownik nie istnieje w systemie, zostaje dodany do tabeli users, następnie jest dodawany do tutors.

Po pomyślnym zakończeniu operacji, transakcja jest zatwierdzana. W przypadku wystąpienia błędu, transakcja jest wycofywana, a procedura zwraca odpowiedni komunikat o błędzie.

```
CREATE PROCEDURE [dbo].[AddTutor]
    @Email nvarchar(64),
    @Password nvarchar(64),
    @FirstName nvarchar(64),
    @LastName nvarchar(64),
    @Phone nvarchar(16)
AS
BEGIN
    BEGIN TRY
        -- Rozpoczęcie transakcji
        BEGIN TRANSACTION;

        DECLARE @UserId INT;
```

```
        SELECT @UserId = user_id
        FROM users
        WHERE email = @Email AND phone = @Phone;

        IF @UserId IS NOT NULL
        BEGIN
            IF EXISTS (SELECT 1 FROM tutors WHERE user_id=@UserId)
            BEGIN
                  ROLLBACK TRANSACTION
                  RETURN -1
            END
        END
        ELSE
        BEGIN
      -- Dodanie użytkownika do tabeli users
      INSERT INTO users (email, password, first_name, last_name, phone)
      VALUES (@Email, @Password, @FirstName, @LastName, @Phone);

      SET @UserId = SCOPE_IDENTITY();
        END

      -- Dodanie tutora do tabeli tutors
      INSERT INTO tutors (user_id)
      VALUES (@UserId);

      -- Zatwierdzenie transakcji po poprawnym zakończeniu operacji
      COMMIT TRANSACTION;
   END TRY
   BEGIN CATCH
      -- Wycofanie transakcji w przypadku błędu
      PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
      ROLLBACK TRANSACTION;
   END CATCH
END;
```

ADD TRANSLATOR

Procedura AddTranslator służy do dodawania nowego tłumacza do systemu. Proces rozpoczyna się od sprawdzenia, czy użytkownik o podanym adresie e-mail i numerze telefonu już istnieje w systemie. Jeśli taki użytkownik istnieje, procedura sprawdza, czy jest już zarejestrowany jako tłumacz.

Następnie sprawdzane jest, czy lista języków tłumacza nie jest pusta. Jeśli języki są dostępne, procedura przechodzi przez listę języków (oddzielonych przecinkami) i dla każdego języka sprawdza, czy jest on już przypisany do tłumacza w tabeli translator_languages. Jeśli język nie został jeszcze przypisany, jest dodawany do tej tabeli.

Na końcu, po pomyślnym zakończeniu operacji, transakcja jest zatwierdzana. W przypadku wystąpienia błędu, transakcja jest wycofywana, a odpowiedni kod błędu jest zwracany.

```
CREATE PROCEDURE [dbo].[AddTranslator]
   @Email nvarchar(64),
   @Password nvarchar(64),
   @FirstName nvarchar(64),
   @LastName nvarchar(64),
   @Phone nvarchar(16),
   @Languages nvarchar(MAX)  -- Lista języków oddzielonych przecinkami
AS
BEGIN
   BEGIN TRY
      -- Rozpoczęcie transakcji
      BEGIN TRANSACTION;
      PRINT 'Transakcja rozpoczęta.';

      DECLARE @UserId INT;

        SELECT @UserId = user_id
        FROM users
        WHERE email= @Email AND phone = @Phone

        IF @UserId IS NOT NULL
        BEGIN
            IF EXISTS (SELECT 1 FROM translators WHERE user_id= @UserId)
            BEGIN
                  ROLLBACK TRANSACTION
                  RETURN -1
            END
        END
        ELSE
        BEGIN
            INSERT INTO users (email, password, first_name, last_name,
phone)
      VALUES (@Email, @Password, @FirstName, @LastName, @Phone);
        SET @UserId = SCOPE_IDENTITY();  -- Pobranie ID użytkownika
        END
         IF LEN(@Languages) = 0
         BEGIN
            PRINT 'Brak języków do przypisania.';
```

```
            ROLLBACK TRANSACTION;
            RETURN -3;  -- Brak języków
        END

        -- Dodanie języków tłumacza do tabeli translator_languages
        DECLARE @Language nvarchar(64);
        DECLARE @Pos INT = 1;
        DECLARE @Len INT;
        DECLARE @CommaPos INT;

        SET @Len = LEN(@Languages);

        -- Pętla do dodania języków z listy oddzielonej przecinkami
        WHILE @Pos <= @Len
        BEGIN
            SET @CommaPos = CHARINDEX(',', @Languages, @Pos);

            IF @CommaPos = 0
            BEGIN
                SET @Language = SUBSTRING(@Languages, @Pos, @Len - @Pos +
1);
                SET @Pos = @Len + 1;
            END
            ELSE
            BEGIN
                SET @Language = SUBSTRING(@Languages, @Pos, @CommaPos -
@Pos);
                SET @Pos = @CommaPos + 1;
            END

            -- Usuwanie nadmiarowych spacji wokół języka
            SET @Language = LTRIM(RTRIM(@Language));

            -- Sprawdzenie, czy język już istnieje dla tłumacza
            IF NOT EXISTS (SELECT 1 FROM translator_languages WHERE
translator_id = @UserId AND language = @Language)
            BEGIN
                -- Dodanie języka do tabeli translator_languages, tylko
jeśli nie ma go jeszcze przypisanego
                INSERT INTO translator_languages (translator_id,
language)
                VALUES (@UserId, @Language);
            END
        END

    -- Zatwierdzenie transakcji po poprawnym zakończeniu operacji
```

```
      COMMIT TRANSACTION;
      PRINT 'Transakcja zakończona pomyślnie.';
      RETURN 0;  -- Sukces
   END TRY
   BEGIN CATCH
      -- Logowanie błędu w przypadku wyjątku
      PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
      -- Zatwierdzenie transakcji po błędzie
      IF @@TRANCOUNT > 0
         ROLLBACK TRANSACTION;
      RETURN -4;  -- Błąd w procesie
   END CATCH
END;
```

CHANGE STUDENT STATUS

Procedura ChangeStudentStatus służy do zmiany statusu aktywności studenta w systemie. Jeśli student istnieje, jego status aktywności w tabeli students jest aktualizowany na nową wartość podaną jako parametr.
Po zakończeniu operacji transakcja jest zatwierdzana. W przypadku błędu, transakcja jest wycofywana, a kod błędu jest zwracany.

```
CREATE PROCEDURE [dbo].[ChangeStudentStatus]
      -- Przykładowe (dla tutora/translatora/administratora mogłoby być to
samo)
      @StudentEmail nvarchar(64),
      @StatusOfActiveness BIT -- status na jaki ma zostać zmieniony
AS

BEGIN
      BEGIN TRY
            BEGIN TRANSACTION

            DECLARE @StudentId INT;

             -- Pobranie ID studenta
        SELECT @StudentId = user_id
        FROM users
        WHERE email = @StudentEmail;

        IF @StudentId IS NULL
        BEGIN
            ROLLBACK TRANSACTION;
```

```
            PRINT 'Nie ma użytkownika o takim adresie email.';
            RETURN -1; -- Kod błędu: Brak użytkownika
        END;
            UPDATE students
            SET is_active = @StatusOfActiveness
            WHERE user_id = @StudentId;

            COMMIT TRANSACTION

            RETURN 0

      END TRY

      BEGIN CATCH
            -- Wycofanie transakcji w przypadku błędu
      PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
      ROLLBACK TRANSACTION;
      RETURN -3;  -- Zwrócenie kodu błędu w przypadku wyjątku
      END CATCH
END
```

ADD ORDER

Procedura AddOrder służy do tworzenia nowego zamówienia w systemie. Sprawdzane jest, czy użytkownik istnieje w systemie. Jeśli użytkownik istnieje, tworzone jest nowe zamówienie w tabeli orders z domyślnym statusem 'Pending' (oczekujące).

Następnie, jeśli lista produktów nie jest pusta, procedura przetwarza każdy produkt z listy, sprawdzając jego istnienie w tabeli `products`. Dla każdego produktu, który istnieje, szczegóły zamówienia są dodawane do tabeli order_details (wraz z ceną produktu).

Po zakończeniu operacji, transakcja jest zatwierdzana. W przypadku błędu, transakcja jest wycofywana, a procedura zwraca odpowiedni kod błędu.

```
CREATE PROCEDURE [dbo].[AddOrder]
    @UserId INT,                -- Id użytkownika, który tworzy
zamówienie
    @OrderDate DATETIME,        -- Data zamówienia
    @Products NVARCHAR(MAX)     -- Lista Id produktów oddzielona
przecinkami
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
```

```
            DECLARE @OrderStatusId INT;

        -- Sprawdzenie, czy użytkownik istnieje
        IF NOT EXISTS (SELECT 1 FROM users WHERE user_id = @UserId)
        BEGIN
            ROLLBACK TRANSACTION;
            PRINT 'Użytkownik o takim ID nie istnieje.';
            RETURN -1;
        END;

        -- Tworzenie zamówienia
        DECLARE @OrderId INT;

            SELECT @OrderStatusId=status_id
            FROM order_statuses
            WHERE status_name = 'Pending'

        INSERT INTO orders (user_id, order_date, status_id)
        VALUES (@UserId, @OrderDate, @OrderStatusId);

        SET @OrderId = SCOPE_IDENTITY();

        -- Przetwarzanie produktów
        IF @Products IS NOT NULL AND LEN(@Products) > 0
        BEGIN
            DECLARE @ProductId INT;
            DECLARE @ProductPos INT = 1;
            DECLARE @CommaPos INT;
            DECLARE @Product NVARCHAR(MAX);
            DECLARE @Price MONEY;

            WHILE @ProductPos <= LEN(@Products)
            BEGIN
                SET @CommaPos = CHARINDEX(',', @Products, @ProductPos);

                IF @CommaPos = 0
                BEGIN
                    SET @Product = SUBSTRING(@Products, @ProductPos,
LEN(@Products) - @ProductPos + 1);
                    SET @ProductPos = LEN(@Products) + 1;
                END
                ELSE
                BEGIN
                    SET @Product = SUBSTRING(@Products, @ProductPos,
@CommaPos - @ProductPos);
                    SET @ProductPos = @CommaPos + 1;
```

```
                END;

                SET @ProductId = CAST(LTRIM(RTRIM(@Product)) AS INT);

                -- Sprawdzenie, czy produkt istnieje
                SELECT @Price = price FROM products WHERE product_id =
@ProductId;

                IF @Price IS NULL
                BEGIN
                    ROLLBACK TRANSACTION;
                    PRINT 'Produkt o ID ' + CAST(@ProductId AS NVARCHAR)
+ ' nie istnieje.';
                    RETURN -2;
                END;

                -- Dodanie szczegółów zamówienia
                INSERT INTO order_details (order_id, product_id, price)
                VALUES (@OrderId, @ProductId, @Price);
            END;
        END;

        COMMIT TRANSACTION;
        PRINT 'Zamówienie zostało pomyślnie utworzone.';
        RETURN 0;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        RETURN -3;
    END CATCH
END;
```

MARK ORDER AS PAID

Procedura MarkOrderAsPaid aktualizuje status zamówienia o podanym identyfikatorze na "Paid" . Sprawdza, czy zamówienie istnieje, a następnie zmienia jego status. Jeśli operacja przebiegnie pomyślnie, transakcja jest zatwierdzana. W przypadku błędu transakcja jest wycofywana, a procedura zwraca odpowiedni kod błędu.

```
CREATE PROCEDURE [dbo].[MarkOrderAsPaid]
    @OrderId INT
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
```

```
        DECLARE @OrderStatusId INT;

        IF NOT EXISTS (SELECT 1 FROM orders WHERE order_id = @OrderId)
        BEGIN
            ROLLBACK TRANSACTION;
            RETURN -1;
        END

        SELECT @OrderStatusId = status_id
        FROM order_statuses
        WHERE status_name = 'Paid';

        UPDATE orders
        SET status_id = @OrderStatusId
        WHERE order_id = @OrderId;

        COMMIT TRANSACTION;
        RETURN 0;

    END TRY
    BEGIN CATCH
        PRINT 'Wystąpił błąd: ' + ERROR_MESSAGE();
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        RETURN -2;
    END CATCH
END;
```

ADD COURSE MODULE ASYNCHRONOUS MEETING

Procedura dodaje asynchroniczne spotkanie do modułu kursu, zapisując je w tabelach meetings i meetings asynchronous. Sprawdza istnienie modułu i aktywnego tutora. Na końcu przypisuje link do nagrania

```
CREATE PROCEDURE [dbo].[AddCourseModuleAsynchronousMeeting]
    @ModuleId INT,
    @TutorId INT,
    @Language NVARCHAR(64) = 'polski',
    @StartTime DATETIME,
    @EndTIme DATETIME = NULL,
    @RecordingLink NVARCHAR(200)
AS
BEGIN
    BEGIN TRY
```

```
        PRINT 'Rozpoczynam transakcję...';
        BEGIN TRANSACTION;

        DECLARE @MeetingID INT;

        -- Sprawdzamy, czy moduł istnieje
        IF NOT EXISTS(SELECT 1 FROM course_modules WHERE module_id =
@ModuleId)
        BEGIN
            PRINT 'Moduł o id ' + CAST(@ModuleId AS NVARCHAR) + ' nie
istnieje.';
            ROLLBACK TRANSACTION;
            RETURN -1; -- Moduł nie istnieje
        END

        -- Sprawdzamy, czy tutor jest aktywny
        IF NOT EXISTS(SELECT 1 FROM tutors WHERE user_id = @TutorId AND
is_active = 1)
        BEGIN
            PRINT 'Tutor o id ' + CAST(@TutorId AS NVARCHAR) + ' nie
jest aktywny.';
            ROLLBACK TRANSACTION;
            RETURN -2; -- Tutor nieaktywny
        END

        PRINT 'Moduł i tutor znalezieni, przechodzę do wstawiania
danych.';

        -- Dodanie spotkania
        INSERT INTO meetings (tutor_id, language, start_time, end_time,
activity_id)
        VALUES (@TutorId, @Language, @StartTime,
                CASE WHEN @EndTIme IS NULL THEN NULL ELSE @EndTIme END,
                @ModuleId);

        SET @MeetingId = SCOPE_IDENTITY();

        -- Dodanie szczegółów spotkania asynchronicznego
        INSERT INTO meetings_asynchronous (meeting_id, recording_link)
        VALUES (@MeetingId, @RecordingLink);

        COMMIT TRANSACTION;
        PRINT 'Transakcja zakończona pomyślnie.';
        RETURN 0; -- Sukces

    END TRY
```

```
    BEGIN CATCH
        PRINT 'Błąd: ' + ERROR_MESSAGE();
        PRINT 'Numer błędu: ' + CAST(ERROR_NUMBER() AS NVARCHAR);
        PRINT 'Stan transakcji: ' + CAST(@@TRANCOUNT AS NVARCHAR);

        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        RETURN -3; -- Błąd w trakcie transakcji
    END CATCH
END;
```

Spis funkcji

GET TRANSLATOR LANGUAGES

Funkcja GetTranslatorLanguages zwraca listę języków przypisanych do tłumacza na podstawie jego translator_id.

```
CREATE FUNCTION [dbo].[GetTranslatorLanguages] (@TranslatorId INT)
RETURNS TABLE
AS
RETURN
(
    SELECT language
    FROM translator_languages
    WHERE translator_id = @TranslatorId
);
```

GET MEETING ATTENDANCE LIST

Funkcja GetTranslatorLanguages zwraca listę obecnych na spotkaniu studentów.

```
CREATE FUNCTION [dbo].[GetMeetingAttendanceList] (@MeetingId INT)
RETURNS TABLE
AS
RETURN
(
```

```
    SELECT student_id
    FROM meeting_presence
    WHERE meeting_id = @MeetingId and is_present=1
);
```

Spis triggerów

SET WEBINAR ACCESS END DATE

Po dodaniu rekordu do tabeli webinar_access ustawiana jest data zakończenia dostępu do webinaru na 30 dni po dacie rozpoczęcia.

```
CREATE TRIGGER SetWebinarAccessEndDate
ON webinar_access
AFTER INSERT
AS
BEGIN
    UPDATE webinar_access
    SET access_end_date = DATEADD(DAY, 30, access_start_date)
    WHERE webinar_id IN (SELECT webinar_id FROM inserted)
      AND user_id IN (SELECT user_id FROM inserted);
END;
```

Indeksy

Indeksy ram czasowych oraz cen

Ramy czasowe spotkania studyjnego:

```
CREATE INDEX Meeting_Time
ON meetings (start_time, end_time);
```

Ramy czasowe modułu w ramach studiów:

```
CREATE INDEX Meeting_Time
ON meetings (start_time, end_time);
```

Ramy czasowe webinaru:

```sql
CREATE INDEX Webinar_Time
ON webinars (start_time, end_time);
```

Cena za pojedynczy produkt:

```sql
CREATE INDEX Product_Price
ON products (price);
```

Cena za całe studia:

```sql
CREATE INDEX Studies_Price
ON studies (capacity);
```

Indeksy na kluczach obcych

Tabela `meeting_presence`:

```sql
CREATE INDEX MeetingPresence_MeetingID
ON meeting_presence (meeting_id);
```

```sql
CREATE INDEX MeetingPresence_StudentID
ON meeting_presence (student_id);
```

Tabela `study_modules`:

```sql
CREATE INDEX StudyModules_StudyID
ON study_modules (study_id);

CREATE INDEX StudyModules_ModuleID
ON study_modules (module_id);
```

Tabela `meetings`:

```sql
CREATE INDEX Meetings_TutorID
ON meetings (tutor_id);

CREATE INDEX Meetings_ActivityID
ON meetings (activity_id);
```

Tabela `order_details`:

```
CREATE INDEX OrderDetails_OrderID
ON order_details (order_id);

CREATE INDEX OrderDetails_ProductID
ON order_details (product_id);
```

Tabela `orders`:

```
CREATE INDEX Orders_UserID
ON orders (user_id);

CREATE INDEX Orders_StatusID
ON orders (status_id);
```

Tabela `webinar_access`:

```
CREATE INDEX WebinarAccess_WebinarID
ON webinar_access (webinar_id);

CREATE INDEX WebinarAccess_UserID
ON webinar_access (user_id);
```

Tabela `addresses`:

```
CREATE INDEX Addresses_StudentID
ON addresses (student_id);

CREATE INDEX Addresses_CityID
ON addresses (city_id);
```

Tabela `students`:

```
CREATE INDEX Students_UserID
ON students (user_id);
```

Tabela `diplomas`:

```
CREATE INDEX Diplomas_StudentID
ON diplomas (student_id);

CREATE INDEX Diplomas_ActivityID
```

```
ON diplomas (activity_id);
```

Tabela `meetings_synchronous`:

```
CREATE INDEX MeetingsSynchronous_PlatformID
ON meetings_synchronous (platform_id);
```