Systemy Baz Danych 2024/2025 – mini projekt - documentation

Database engine: Microsoft SQL Server
Version: 2.4
Description:

Contents

1 Użytkownicy i funkcji systemu

1.1 Hierarchia użytkowników i konta w systemie

W naszej bazie danych, jeśli dana osoba jest wyżej w hierarchii uprawnień, to uprawnienia wszystkich użytkowników "pod nią" w hierarchii na nią przechodzą. Czyli na przykład administrator ma wszystkie uprawnienia prowadzącego przedmiotu i dodatkowe dla niego, ale prowadzący przedmiotu nie ma wszystkich uprawnień administratora. Hierarchia jest przedstawiona na rysunku 1 poniżej.

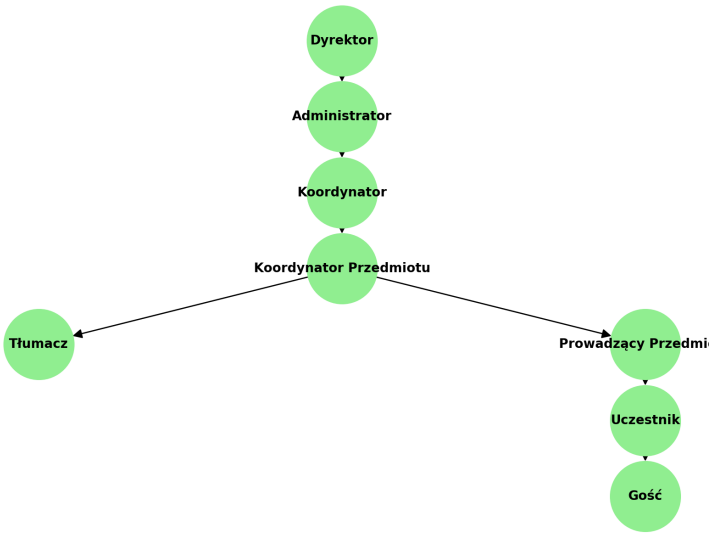


Figure 1: Drzewo hierarchii użytkowników

poniżej wypisani są wszyscy użytkownicy systemu:

- Dyrektor
- Administrator
- Koordynator
- Koordynator przedmiotu
- Prowadzący przedmiotu
- Tłumacz
- Uczestnik

1.2 Funkcje systemu

1.2.1 Zarządzanie kontami

- Dodawanie/Usuwanie konta [uprawnienia na poziomie Gościa, Tłumacza]
- Dodawanie/Usuwanie konta tłumacza/prowadzącego przedmiotu do kursu [uprawnienia na poziomie koordynatora przedmiotu]
- Dodanie/Usunięcie Administratora [Uprawnienia na poziomie Dyrektora]

1.2.2 Webinary

- Przeglądanie listy webinarów [uprawnienia na poziomie gościa/tłumacza]
- Zapis na bezpłatne webinary [uprawnienia na poziomie użytkownika]
- Możliwość oglądania webinarów [uprawnienia na poziomie użytkownika/tłumacza]
- Możliwość zakupu płatnych webinarów [uprawnienia na poziomie użytkownika]
- Dodawanie/usuwanie tłumaczeń do webinarów [uprawnienia na poziomie tłumacza/prowadzącego przedmiotu]
- Dodawanie zawartości do webinarów [uprawnienia na poziomie prowadzącego zajęcia]
- Modyfikacja webinaru [uprawnienia na poziomie koordynatora przedmiotu]
- Dodanie webinaru [uprawnienia na poziomie koordynatora]
- Możliwość przypisania/usunięcia prowadzących, tłumaczy i koordynatora przedmiotu [uprawnienia na poziomie koordynatora przedmiotu]
- Usuwanie webinarów [uprawnienia na poziomie Administratora]

1.2.3 Kursy

- Przeglądanie listy kursów [uprawnienia na poziomie Gościa]
- Zapis do kursu [uprawnienia na poziomie uczestnika]
- Po zapisie dostęp do szczegółowych danych kursu [uprawnienia na poziomie uczestnika]
- Możliwość dodawania/usuwania tłumaczenia kursu [uprawnienia na poziomie tłumacza]
- Dodanie treści do kursu [uprawnienia na poziomie prowadzącego przedmiotu]

- Dodawanie prowadzących przedmiotów i tłumaczy [uprawnienia na poziomie koordynatora przedmiotu]
- Modyfikacja kursu [uprawnienia na poziomie koordynatora przedmiotu]
- Dodawanie/usuwanie prowadzących przedmiotów, tłumaczy i koordynatorów przedmiotów [uprawnienia na poziomie koordynatora]
- Dodanie kursu [uprawnienia na poziomie koordynatora]
- Usunięcie kursu [uprawnienia na poziomie administratora]

1.2.4 Studia

- Przeglądanie sylabusu studiów [uprawnienia na poziomie gościa]
- Zapis na studia [uprawnienia na poziomie użytkownika]
- Zapis na pojedyńcze płatne spotkania [uprawnienia na poziomie użytkownika]
- Możliwość odrobienia zajęć [uprawnienia na poziomie uczestnika]
- Zaznaczenie obecności [uprawnienia na poziomie prowadzącego przedmiotu]
- Wskazanie daty odrobienia zajęć [uprawnienia na poziomie prowadzącego przedmiotu]
- Usprawiedliwienie nieobecności uczestnika [uprawnienia na poziomie prowadzącego przedmiotu]
- przypisanie tłumacza, prowadzącego przedmiotu, koordynatora przedmiotu [uprawnienia na poziomie koordynatora]
- Tworzenie spotkań [uprawnienia na poziomie koordynatora]
- Modyfikacja harmonogramu studiów [uprawnienia na poziomie koordynatora]
- Tworzenie sylabusu [uprawnienia na poziomie koordynatora przedmiotu]

1.2.5 Integracja z systemem płatności

- Płatność za webinary, kursy, studia [uprawnienia na poziomie uczestnika]
- Tworzenie zamówienia [uprawnienia na poziomie uczestnika]
- Płatność za webinary, kursy, studia [uprawnienia na poziomie uczestnika]
- Możliwość odroczenia płatności za kurs, webinar i studia [uprawnienia na poziomie dyrektora]

1.2.6 Generowanie raportów

- generowanie raportów dotyczących własnych zajęć takich jak swojej frekwencji, ocen itd.[uprawnienia na poziomie uczestnika]
- generowanie raportów dotyczących frekwencji i ocen wszystkich uczestników prowadzonego przedmiotu [uprawnienia na poziomie prowadzoncego przedmiotu]
- generowanie raportów dotyczących frekwencji i ocen koordynowanego webinaru, kursu lub studiów [koordynator przedmiotu]
- generowanie raportów dotyczących danych finansowych koordynowanych webinarów, kusów i studiów [uprawnienia na poziomie koordynatora]
- generowanie raportów odnośnie wszystkiego w bazie danych [uprawnienia na poziomie administratora]

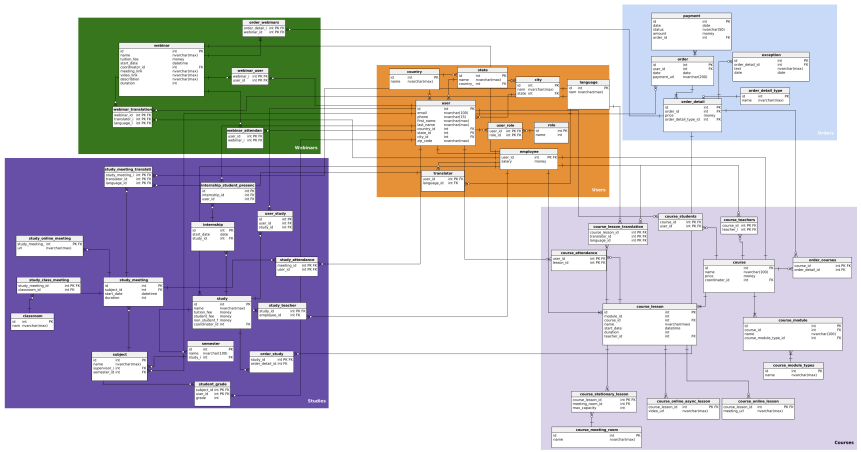


Figure 2: schemat bazy

2 Tables

2.1 Table study

2.1.1 Description

Przechowuje kierunki studiów.

2.1.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	
tuition_fee	money	NOT NULL, CHECK (tuition_fee >= 0)
student_limit	int	NOT NULL, CHECK (student_limit > 0)
coordinator_id	int	NOT NULL

2.1.3 SQL

```
CREATE TABLE study (
   id int NOT NULL,
   name nvarchar(max) NOT NULL,
   tuition_fee money NOT NULL CHECK (tuition_fee >= 0),
   student_limit int NOT NULL CHECK (student_limit > 0),
   coordinator_id int NOT NULL,
   CONSTRAINT study_pk PRIMARY KEY (id)
);
ALTER TABLE study ADD CONSTRAINT study_employee
    FOREIGN KEY (coordinator_id)
    REFERENCES employee (user_id);
```

2.2 Table internship

2.2.1 Description

Przechowuje praktyki zawodowe.

2.2.2 Columns

Column name	Type	Properties
id	int	PK
start_date	date	NOT NULL
study_id	int	NOT NULL

2.2.3 SQL

```
CREATE TABLE internship (
```

```
   id int NOT NULL,
   start_date date NOT NULL,
   study_id int NOT NULL,
   CONSTRAINT internship_pk PRIMARY KEY (id)
);
ALTER TABLE internship ADD CONSTRAINT internship_study
    FOREIGN KEY (study_id)
    REFERENCES study (id);
```

2.3 Table study_online_meeting

2.3.1 Description

Przechowuje spotkania online dla studiów.

2.3.2 Columns

Column name	Type	Properties
study_meeting_id	int	PK
url	nvarchar(max)	NOT NULL

2.3.3 SQL

```
CREATE TABLE study_online_meeting (
   study_meeting_id int NOT NULL,
   url nvarchar(max) NOT NULL,
   CONSTRAINT study_online_meeting_pk PRIMARY KEY (study_meeting_id)
);
ALTER TABLE study_online_meeting ADD CONSTRAINT study_online_meeting_study_meeting
    FOREIGN KEY (study_meeting_id)
    REFERENCES study_meeting (id);
```

2.4 Table subject

2.4.1 Description

Przechowuje przedmioty.

2.4.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL
supervisor_id	int	NOT NULL
semester_id	int	NOT NULL

2.4.3 SQL

```
CREATE TABLE subject (
   id int NOT NULL,
   name nvarchar(max) NOT NULL,
   supervisor_id int NOT NULL,
   semester_id int NOT NULL,
   CONSTRAINT subject_pk PRIMARY KEY (id)
);
ALTER TABLE subject ADD CONSTRAINT subject_semester
    FOREIGN KEY (semester_id)
    REFERENCES semester (id);

ALTER TABLE subject ADD CONSTRAINT subject_user
    FOREIGN KEY (supervisor_id)
    REFERENCES "user" (id);
```

2.5 Table study_class_meeting

2.5.1 Description

Przechowuje spotkania stacjonarne.

2.5.2 Columns

Column name	Type	Properties
study_meeting_id	int	PK
classroom_id	int	NOT NULL

2.5.3 SQL

```
CREATE TABLE study_class_meeting (
   study_meeting_id int NOT NULL,
   classroom_id int NOT NULL,
   CONSTRAINT study_class_meeting_pk PRIMARY KEY (study_meeting_id)
);
ALTER TABLE study_class_meeting ADD CONSTRAINT study_class_meeting_classrom
    FOREIGN KEY (classroom_id)
    REFERENCES classroom (id);

ALTER TABLE study_class_meeting ADD CONSTRAINT study_class_meeting_study_meeting
    FOREIGN KEY (study_meeting_id)
    REFERENCES study_meeting (id);
```

2.6 Table student_grade

2.6.1 Description

Przechowuje oceny studentów.

2.6.2 Columns

Column name	Type	Properties
subject_id	int	PK
user_id	int	PK
grade	int	NOT NULL, CHECK (grade <= 100), CHECK (grade >= 0)

2.6.3 SQL

```
CREATE TABLE student_grade (
   subject_id int NOT NULL,
   user_id int NOT NULL,
   grade int NOT NULL,
   CONSTRAINT check_grade_max CHECK (grade <= 100),
   CONSTRAINT check_grade_min CHECK (grade >= 0),
   CONSTRAINT student_grade_pk PRIMARY KEY (subject_id,user_id)
);
ALTER TABLE student_grade ADD CONSTRAINT student_grade_subject
    FOREIGN KEY (subject_id)
    REFERENCES subject (id);

ALTER TABLE student_grade ADD CONSTRAINT student_grade_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.7 Table order

2.7.1 Description

Przechowuje zamówienia.

2.7.2 Columns

Column name	Type	Properties
id	int	PK
user_id	int	NOT NULL
date	date	NOT NULL, DEFAULT GETDATE()
payment_url	nvarchar(200)	NOT NULL

2.7.3 SQL

```
CREATE TABLE "order" (
   id int NOT NULL,
   user_id int NOT NULL,
   date date NOT NULL DEFAULT GETDATE(),
   payment_url nvarchar(200) NOT NULL,
   CONSTRAINT order_pk PRIMARY KEY (id)
);
ALTER TABLE "order" ADD CONSTRAINT order_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.8 Table order_detail

2.8.1 Description

Przechowuje szczegóły zamówień.

2.8.2 Columns

Column name	Type	Properties
id	int	PK
order_id	int	NOT NULL
price	money	NOT NULL, CHECK (price >= 0)
order_detail_type_id	int	NOT NULL

2.8.3 SQL

```
CREATE TABLE order_detail (
   id int NOT NULL,
   order_id int NOT NULL,
```

```
   price money NOT NULL,
   order_detail_type_id int NOT NULL,
   CONSTRAINT price_check CHECK (price >= 0),
   CONSTRAINT order_detail_pk PRIMARY KEY (id)
);
ALTER TABLE order_detail ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (order_id)
    REFERENCES "order" (id);

ALTER TABLE order_detail ADD CONSTRAINT order_detail_order_detail_type
    FOREIGN KEY (order_detail_type_id)
    REFERENCES order_detail_type (id);
```

2.9 Table payment

2.9.1 Description

Przechowuje płatności.

2.9.2 Columns

Column name	Type	Properties
id	int	PK
order_id	int	NOT NULL
date	date	NOT NULL
amount	money	NOT NULL, CHECK (amount >= 0)
status	nvarchar(50)	NOT NULL

2.9.3 SQL

```
CREATE TABLE payment (
   id int NOT NULL,
   date date NOT NULL,
   status nvarchar(50) NOT NULL,
   amount money NOT NULL CHECK (amount >= 0),
   order_id int NOT NULL,
   CONSTRAINT amount_check CHECK (amount >= 0),
   CONSTRAINT payment_pk PRIMARY KEY (id)
);
ALTER TABLE payment ADD CONSTRAINT Payments_Orders
    FOREIGN KEY (order_id)
    REFERENCES "order" (id);
```

2.10 Table course

2.10.1 Description

Przechowuje kursy.

2.10.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(100)	NOT NULL
price	money	NOT NULL, CHECK (price >= 0)
coordinator_id	int	NOT NULL

2.10.3 SQL

```
CREATE TABLE course (
   id int NOT NULL,
   name nvarchar(100) NOT NULL,
   price money NOT NULL,
   coordinator_id int NOT NULL,
   CONSTRAINT price_check CHECK (price >= 0),
   CONSTRAINT course_pk PRIMARY KEY (id)
);
ALTER TABLE course ADD CONSTRAINT Courses_employee
    FOREIGN KEY (coordinator_id)
    REFERENCES employee (user_id);
```

2.11 Table course_module

2.11.1 Description

Przechowuje moduły kursów.

2.11.2 Columns

Column name	Type	Properties
id	int	PK
course_id	int	NOT NULL
name	nvarchar(100)	NOT NULL
course_module_type_id	int	NOT NULL

2.11.3 SQL

```
CREATE TABLE course_module (
   id int NOT NULL,
   course_id int NOT NULL,
   name nvarchar(100) NOT NULL,
   course_module_type_id int NOT NULL,
   CONSTRAINT course_module_pk PRIMARY KEY (id)
);
ALTER TABLE course_module ADD CONSTRAINT CourseModules_CourseModuleTypes
    FOREIGN KEY (course_module_type_id)
    REFERENCES course_module_types (id);

ALTER TABLE course_module ADD CONSTRAINT CourseModules_Courses
    FOREIGN KEY (course_id)
    REFERENCES course (id);
```

2.12 Table course_lesson

2.12.1 Description

Przechowuje lekcje kursu.

2.12.2 Columns

Column name	Type	Properties
id	int	PK
module_id	int	NOT NULL
course_id	int	NOT NULL
name	nvarchar(max)	NOT NULL
start_date	datetime	NOT NULL
duration	int	NOT NULL, CHECK (duration >= 15 AND duration <= 240)
teacher_id	int	NOT NULL

2.12.3 SQL

```
CREATE TABLE course_lesson (
   id int NOT NULL,
   module_id int NOT NULL,
   course_id int NOT NULL,
   name nvarchar(max) NOT NULL,
```

```
   start_date datetime NOT NULL,
   duration int NOT NULL,
   teacher_id int NOT NULL,
   CONSTRAINT duration_check CHECK (duration >= 15 AND duration <= 240),
   CONSTRAINT course_lesson_pk PRIMARY KEY (id)
);
ALTER TABLE course_lesson ADD CONSTRAINT CourseLessons_Courses
    FOREIGN KEY (course_id)
    REFERENCES course (id);

ALTER TABLE course_lesson ADD CONSTRAINT course_lesson_employee_2
    FOREIGN KEY (teacher_id)
    REFERENCES employee (user_id);
```

2.13 Table course_module_types

2.13.1 Description

Przechowuje typy modułów kursu.

2.13.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL

2.13.3 SQL

```
CREATE TABLE course_module_types (
   id int NOT NULL,
   name nvarchar(max) NOT NULL,
   CONSTRAINT course_module_types_pk PRIMARY KEY (id)
);
```

2.14 Table order_study

2.14.1 Description

Przechowuje zamówienia dotyczące studiów.

2.14.2 Columns

Column name	Type	Properties
study_id	int	PK
order_detail_id	int	NOT NULL

2.14.3 SQL

```
CREATE TABLE order_study (
   study_id int NOT NULL,
   order_detail_id int NOT NULL,
   CONSTRAINT order_study_pk PRIMARY KEY (study_id)
);
ALTER TABLE order_study ADD CONSTRAINT OrderDetails_OrderStudies
    FOREIGN KEY (order_detail_id)
    REFERENCES order_detail (id);

ALTER TABLE order_study ADD CONSTRAINT OrderStudies_study
    FOREIGN KEY (study_id)
    REFERENCES study (id);
```

2.15 Table order_courses

2.15.1 Description

Przechowuje zamówienia dotyczące kursów.

2.15.2 Columns

Column name	Type	Properties
course_id	int	PK
order_detail_id	int	NOT NULL

2.15.3 SQL

```
CREATE TABLE order_courses (
   course_id int NOT NULL,
   order_detail_id int NOT NULL,
   CONSTRAINT order_courses_pk PRIMARY KEY (course_id)
);
ALTER TABLE order_courses ADD CONSTRAINT Courses_OrderCourses
    FOREIGN KEY (course_id)
    REFERENCES course (id);
```

```
ALTER TABLE order_courses ADD CONSTRAINT OrderCourses_OrderDetails
    FOREIGN KEY (order_detail_id)
    REFERENCES order_detail (id);
```

2.16 Table order_webinars

2.16.1 Description

Przechowuje zamówienia dotyczące webinarów.

2.16.2 Columns

Column name	Type	Properties
order_detail_id	int	PK
webinar_id	int	PK

2.16.3 SQL

```
CREATE TABLE order_webinars (
   order_detail_id int NOT NULL,
   webinar_id int NOT NULL,
   CONSTRAINT order_webinars_pk PRIMARY KEY (order_detail_id, webinar_id)
);
ALTER TABLE order_webinars ADD CONSTRAINT OrderDetails_OrderWebinars
    FOREIGN KEY (order_detail_id)
    REFERENCES order_detail (id);

ALTER TABLE order_webinars ADD CONSTRAINT order_webinars_webinar
    FOREIGN KEY (webinar_id)
    REFERENCES webinar (id);
```

2.17 Table user

2.17.1 Description

Przechowuje informacje o użytkownikach.

2.17.2 Columns

Column name	Type	Properties
id	int	PK
email	nvarchar(100)	CHECK (email LIKE '%_@__%.__%')
phone	nvarchar(15)	CHECK (phone LIKE '+[0-9]%' OR phone LIKE '[0-9]%')
first_name	nvarchar(max)	NOT NULL
last_name	nvarchar(max)	NOT NULL
country_id	int	NOT NULL
state_id	int	NOT NULL
city_id	int	NOT NULL
zip_code	nvarchar(max)	NOT NULL

2.17.3 SQL

```
CREATE TABLE "user" (
    id int  NOT NULL,
    email nvarchar(100)  NOT NULL CHECK (email LIKE '%_@__%.__%'),
    phone nvarchar(15)  NOT NULL CHECK (phone LIKE '+[0-9]%' OR phone LIKE '[0-9]%'),
    first_name nvarchar(max)  NOT NULL,
    last_name nvarchar(max)  NOT NULL,
    country_id int  NOT NULL,
    state_id int  NOT NULL,
    city_id int  NOT NULL,
    zip_code nvarchar(max)  NOT NULL,
    CONSTRAINT user_pk PRIMARY KEY  (id)
);
ALTER TABLE "user" ADD CONSTRAINT user_city
    FOREIGN KEY (city_id)
    REFERENCES city (id);

ALTER TABLE "user" ADD CONSTRAINT user_country
    FOREIGN KEY (country_id)
    REFERENCES country (id);

ALTER TABLE "user" ADD CONSTRAINT user_state
    FOREIGN KEY (state_id)
    REFERENCES state (id);
```

2.18 Table employee

2.18.1 Description

Przechowuje informacje o pracownikach.

2.18.2 Columns

Column name	Type	Properties
user_id	int	PK
salary	money	NOT NULL, CHECK (salary > 0)

2.18.3 SQL

```
CREATE TABLE employee (
    user_id int  NOT NULL,
    salary money  NOT NULL CHECK (salary > 0),
    CONSTRAINT salary_check CHECK (salary > 0),
    CONSTRAINT employee_pk PRIMARY KEY  (user_id)
);
ALTER TABLE employee ADD CONSTRAINT employee_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.19 Table language

2.19.1 Description

Przechowuje języki.

2.19.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL

2.19.3 SQL

```
CREATE TABLE language (
    id int  NOT NULL,
    name nvarchar(max)  NOT NULL,
    CONSTRAINT language_pk PRIMARY KEY  (id)
);
```

2.20 Table study_meeting_translation

2.20.1 Description

Przechowuje informacje o tłumaczeniach spotkań.

2.20.2 Columns

Column name	Type	Properties
study_meeting_id	int	PK
language_id	int	PK
translator_id	int	NOT NULL
name	nvarchar(max)	NOT NULL
describtion	nvarchar(max)	NOT NULL

2.20.3 SQL

```
CREATE TABLE study_meeting_translation (
    study_meeting_id int  NOT NULL,
    translator_id int  NOT NULL,
    language_id int  NOT NULL,
    CONSTRAINT study_meeting_translation_pk PRIMARY KEY
    (study_meeting_id,language_id,translator_id)
);
ALTER TABLE study_meeting_translation
ADD CONSTRAINT study_meeting_translation_language
    FOREIGN KEY (language_id)
    REFERENCES language (id);

ALTER TABLE study_meeting_translation
ADD CONSTRAINT study_meeting_translation_study_meeting
    FOREIGN KEY (study_meeting_id)
    REFERENCES study_meeting (id);

ALTER TABLE study_meeting_translation
ADD CONSTRAINT study_meeting_translation_translator
    FOREIGN KEY (translator_id)
    REFERENCES translator (user_id);
```

2.21 Table translator

2.21.1 Description

Przechowuje tłumaczy.

2.21.2 Columns

Column name	Type	Properties
user_id	int	PK
language_id	int	NOT NULL

2.21.3 SQL

```
CREATE TABLE translator (
    user_id int  NOT NULL,
    language_id int  NOT NULL,
    CONSTRAINT translator_pk PRIMARY KEY  (user_id)
);
ALTER TABLE translator ADD CONSTRAINT translator_employee
    FOREIGN KEY (user_id)
    REFERENCES employee (user_id);

ALTER TABLE translator ADD CONSTRAINT translator_language
    FOREIGN KEY (language_id)
    REFERENCES language (id);
```

2.22 Table study_teacher

2.22.1 Description

Przechowuje relacje koordynatorów z kierunkami studiów.

2.22.2 Columns

Column name	Type	Properties
study_id	int	PK
employee_id	int	PK

2.22.3 SQL

```
CREATE TABLE study_teacher (
    study_id int  NOT NULL,
    employee_id int  NOT NULL,
    CONSTRAINT study_teacher_pk PRIMARY KEY  (study_id, employee_id)
);
ALTER TABLE study_teacher ADD CONSTRAINT study_teacher_employee
    FOREIGN KEY (employee_id)
    REFERENCES employee (user_id);

ALTER TABLE study_teacher ADD CONSTRAINT study_teacher_study
```

```
    FOREIGN KEY (study_id)
    REFERENCES study (id);
```

2.23 Table course_teachers

2.23.1 Description

Przechowuje relacje prowadzących z kursami.

2.23.2 Columns

Column name	Type	Properties
course_id	int	PK
teacher_id	int	PK

2.23.3 SQL

```
CREATE TABLE course_teachers (
    course_id int  NOT NULL,
    teacher_id int  NOT NULL,
    CONSTRAINT course_teachers_pk PRIMARY KEY  (course_id,teacher_id)
);
ALTER TABLE course_teachers ADD CONSTRAINT CourseTeachers_Courses
    FOREIGN KEY (course_id)
    REFERENCES course (id);

ALTER TABLE course_teachers ADD CONSTRAINT CourseTeachers_employee
    FOREIGN KEY (teacher_id)
    REFERENCES employee (user_id);
```

2.24 Table user_study

2.24.1 Description

Przechowuje relacje studentów z ich kierunkami studiów.

2.24.2 Columns

Column name	Type	Properties
id	int	PK
user_id	int	NOT NULL
study_id	int	NOT NULL

2.24.3 SQL

```
CREATE TABLE user_study (
    id int  NOT NULL,
    user_id int  NOT NULL,
    study_id int  NOT NULL,
    CONSTRAINT user_study_pk PRIMARY KEY  (id)
);
ALTER TABLE user_study ADD CONSTRAINT study_students_study
    FOREIGN KEY (study_id)
    REFERENCES study (id);

ALTER TABLE user_study ADD CONSTRAINT study_users_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.25 Table webinar_user

2.25.1 Description

Przechowuje użytkowników webinaru.

2.25.2 Columns

Column name	Type	Properties
webinar_id	int	PK
user_id	int	PK

2.25.3 SQL

```
CREATE TABLE webinar_user (
    webinar_id int  NOT NULL,
    user_id int  NOT NULL,
    CONSTRAINT webinar_user_pk PRIMARY KEY  (webinar_id,user_id)
);
ALTER TABLE webinar_user ADD CONSTRAINT webinar_students_Webinar
    FOREIGN KEY (webinar_id)
    REFERENCES webinar (id);

ALTER TABLE webinar_user ADD CONSTRAINT webinar_students_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.26 Table course_students

2.26.1 Description

Przechowuje uczestników kursu.

2.26.2 Columns

Column name	Type	Properties
course_id	int	PK
user_id	int	PK

2.26.3 SQL

```
CREATE TABLE course_students (
    course_id int  NOT NULL,
    user_id int  NOT NULL,
    CONSTRAINT course_students_pk PRIMARY KEY  (course_id,user_id)
);
ALTER TABLE course_students ADD CONSTRAINT CourseStudents_Courses
    FOREIGN KEY (course_id)
    REFERENCES course (id);

ALTER TABLE course_students ADD CONSTRAINT course_students_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.27 Table course_lesson_translation

2.27.1 Description

Przechowuje informacje o tłumaczeniach lekcji kursu.

2.27.2 Columns

Column name	Type	Properties
course_lesson_id	int	PK
language_id	int	PK
translator_id	int	NOT NULL
name	nvarchar(max)	NOT NULL
describtion	nvarchar(max)	NOT NULL

2.27.3 SQL

```
CREATE TABLE course_lesson_translation (
```

```
    course_lesson_id int  NOT NULL,
    translator_id int  NOT NULL,
    language_id int  NOT NULL,
    CONSTRAINT course_lesson_translation_pk
    PRIMARY KEY  (translator_id,course_lesson_id,language_id)
);
ALTER TABLE course_lesson_translation
ADD CONSTRAINT course_lesson_translation_course_lesson
    FOREIGN KEY (course_lesson_id)
    REFERENCES course_lesson (id);

ALTER TABLE course_lesson_translation ADD CONSTRAINT course_lesson_translation_employee
    FOREIGN KEY (translator_id)
    REFERENCES employee (user_id);

ALTER TABLE course_lesson_translation ADD CONSTRAINT course_lesson_translation_language
    FOREIGN KEY (language_id)
    REFERENCES language (id);
```

2.28 Table webinar_translation

2.28.1 Description

Przechowuje tłumaczenia webinaru.

2.28.2 Columns

Column name	Type	Properties
webinar_id	int	PK
language_id	int	PK
translator_id	int	NOT NULL
name	nvarchar(max)	NOT NULL
describtion	nvarchar(max)	NOT NULL

2.28.3 SQL

```
CREATE TABLE webinar_translation (
    webinar_id int  NOT NULL,
    translator_id int  NOT NULL,
    language_id int  NOT NULL,
    CONSTRAINT webinar_translation_pk
    PRIMARY KEY  (webinar_id,translator_id,language_id)
);
ALTER TABLE webinar_translation ADD CONSTRAINT language_webinar_translate
    FOREIGN KEY (language_id)
```

```
    REFERENCES language (id);

ALTER TABLE webinar_translation ADD CONSTRAINT webinar_translate_employee
    FOREIGN KEY (translator_id)
    REFERENCES employee (user_id);

ALTER TABLE webinar_translation ADD CONSTRAINT webinar_translate_webinar
    FOREIGN KEY (webinar_id)
    REFERENCES webinar (id);
```

2.29 Table classroom

2.29.1 Description

Przechowuje sale lekcyjne dla studiów.

2.29.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL

2.29.3 SQL

```
CREATE TABLE classroom (
    id int  NOT NULL,
    name nvarchar(max)  NOT NULL,
    CONSTRAINT classroom_pk PRIMARY KEY  (id)
);
```

2.30 Table semester

2.30.1 Description

Stores semesters.

2.30.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(100)	NOT NULL
study_id	int	NOT NULL

2.30.3 SQL

```
CREATE TABLE semester (
    id int NOT NULL,
    name nvarchar(100) NOT NULL,
    study_id int NOT NULL,
    CONSTRAINT semester_pk PRIMARY KEY (id)
);
ALTER TABLE semester ADD CONSTRAINT semester_study
    FOREIGN KEY (study_id)
    REFERENCES study (id);
```

2.31 Table country

2.31.1 Description

Stores user countries.

2.31.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL

2.31.3 SQL

```
CREATE TABLE country (
    id int NOT NULL,
    name nvarchar(max) NOT NULL,
    CONSTRAINT country_pk PRIMARY KEY (id)
);
```

2.32 Table state

2.32.1 Description

Stores states or provinces.

2.32.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL
country_id	int	NOT NULL

2.32.3 SQL

```
CREATE TABLE state (
    id int NOT NULL,
    name nvarchar(max) NOT NULL,
    country_id int NOT NULL,
    CONSTRAINT state_pk PRIMARY KEY (id)
);
ALTER TABLE state ADD CONSTRAINT state_country
    FOREIGN KEY (country_id)
    REFERENCES country (id);
```

2.33 Table course_stationary_lesson

2.33.1 Description

Stores stationary lessons for courses.

2.33.2 Columns

Column name	Type	Properties
course_lesson_id	int	PK
meeting_room_id	int	NOT NULL
max_capacity	int	NOT NULL, CHECK (max_capacity $>$ 0 and max_capacity $<$ 100)

2.33.3 SQL

```
CREATE TABLE course_stationary_lesson (
   course_lesson_id int NOT NULL,
   meeting_room_id int NOT NULL,
   max_capacity int NOT NULL CHECK (max_capacity > 0 and max_capacity < 100),
   CONSTRAINT course_stationary_lesson_pk PRIMARY KEY (course_lesson_id)
);
ALTER TABLE course_stationary_lesson
ADD CONSTRAINT course_meeting_room_course_stationary_lesson
    FOREIGN KEY (meeting_room_id)
    REFERENCES course_meeting_room (id);

ALTER TABLE course_stationary_lesson
ADD CONSTRAINT course_stationary_lesson_course_lesson
    FOREIGN KEY (course_lesson_id)
    REFERENCES course_lesson (id);
```

2.34 Table course_online_lesson

2.34.1 Description

Stores online lessons for courses.

2.34.2 Columns

Column name	Type	Properties
course_lesson_id	int	PK
meeting_url	nvarchar(max)	NOT NULL

2.34.3 SQL

```
CREATE TABLE course_online_lesson (
   course_lesson_id int NOT NULL,
   meeting_url nvarchar(max) NOT NULL,
   CONSTRAINT course_online_lesson_pk PRIMARY KEY (course_lesson_id)
);
ALTER TABLE course_online_lesson
ADD CONSTRAINT course_online_lesson_course_lesson
    FOREIGN KEY (course_lesson_id)
    REFERENCES course_lesson (id);
```

2.35 Table course_online_async_lesson

2.35.1 Description

Stores asynchronous online lessons for courses.

2.35.2 Columns

Column name	Type	Properties
course_lesson_id	int	PK
video_url	nvarchar(max)	NOT NULL

2.35.3 SQL

```
CREATE TABLE course_online_async_lesson (
   course_lesson_id int NOT NULL,
   video_url nvarchar(max) NOT NULL,
   CONSTRAINT course_online_async_lesson_pk
   PRIMARY KEY (course_lesson_id)
);
ALTER TABLE course_online_async_lesson
ADD CONSTRAINT course_online_async_lesson_course_lesson
```

```
    FOREIGN KEY (course_lesson_id)
    REFERENCES course_lesson (id);
```

2.36 Table course_meeting_room

2.36.1 Description

Stores meeting rooms for courses.

2.36.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL

2.36.3 SQL

```
CREATE TABLE course_meeting_room (
   id int NOT NULL,
   name nvarchar(max) NOT NULL,
   CONSTRAINT course_meeting_room_pk PRIMARY KEY (id)
);
```

2.37 Table order_detail_type

2.37.1 Description

Stores order types.

2.37.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL

2.37.3 SQL

```
CREATE TABLE order_detail_type (
   id int NOT NULL,
   name nvarchar(max) NOT NULL,
   CONSTRAINT order_detail_type_pk PRIMARY KEY (id)
);
```

2.38 Table exception

2.38.1 Description

Tabela wyjątków

2.38.2 Columns

Column name	Type	Properties
id	int	PK
order_detail_id	int	NOT NULL
text	nvarchar(max)	NOT NULL
date	date	NOT NULL

2.38.3 SQL

```
CREATE TABLE exception (
    id int NOT NULL,
    order_detail_id int NOT NULL,
    text nvarchar(max) NOT NULL,
    date date NOT NULL,
    CONSTRAINT exception_pk PRIMARY KEY (id)
);
ALTER TABLE exception ADD CONSTRAINT exception_order_detail
    FOREIGN KEY (order_detail_id)
    REFERENCES order_detail (id);
```

2.39 Table webinar_attendance

2.39.1 Description

Tabela przechowująca obecność na webinarach

2.39.2 Columns

Column name	Type	Properties
user_id	int	PK
webinar_id	int	PK

2.39.3 SQL

```
CREATE TABLE webinar_attendance (
    user_id int NOT NULL,
    webinar_id int NOT NULL,
    CONSTRAINT webinar_attendance_pk PRIMARY KEY (webinar_id, user_id)
```

```
);
ALTER TABLE webinar_attendance ADD CONSTRAINT webinar_attendance_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);

ALTER TABLE webinar_attendance ADD CONSTRAINT webinar_attendance_webinar
    FOREIGN KEY (webinar_id)
    REFERENCES webinar (id);
```

2.40 Table course_attendance

2.40.1 Description

Tabela przechowująca obecność na kursach

2.40.2 Columns

Column name	Type	Properties
user_id	int	PK
lesson_id	int	PK

2.40.3 SQL

```
CREATE TABLE course_attendance (
    user_id int NOT NULL,
    lesson_id int NOT NULL,
    CONSTRAINT course_attendance_pk PRIMARY KEY (user_id, lesson_id)
);
ALTER TABLE course_attendance ADD CONSTRAINT course_activity_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);

ALTER TABLE course_attendance ADD CONSTRAINT course_lesson_course_activity
    FOREIGN KEY (lesson_id)
    REFERENCES course_lesson (id);
```

2.41 Table study_attendance

2.41.1 Description

Tabela przechowująca obecność na zajęciach studiów

2.41.2 Columns

Column name	Type	Properties
meeting_id	int	PK
user_id	int	PK

2.41.3 SQL

```
CREATE TABLE study_attendance (
    meeting_id int NOT NULL,
    user_id int NOT NULL,
    CONSTRAINT study_attendance_pk PRIMARY KEY (meeting_id, user_id)
);
ALTER TABLE study_attendance ADD CONSTRAINT study_attendance_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);

ALTER TABLE study_attendance ADD CONSTRAINT study_meeting_study_attendance
    FOREIGN KEY (meeting_id)
    REFERENCES study_meeting (id);
```

2.42 Table internship_student_presence

2.42.1 Description

Stores internship presence.

2.42.2 Columns

Column name	Type	Properties
id	int	PK
internship_id	int	NOT NULL
user_id	int	NOT NULL

2.42.3 SQL

```
CREATE TABLE internship_student_presence (
    id int NOT NULL,
    internship_id int NOT NULL,
    user_id int NOT NULL,
    CONSTRAINT internship_student_presence_pk PRIMARY KEY (id)
);
ALTER TABLE internship_student_presence ADD CONSTRAINT student_internship_internship
    FOREIGN KEY (internship_id)
    REFERENCES internship (id);
```

```
ALTER TABLE internship_student_presence ADD CONSTRAINT student_internship_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.43 Table webinar

2.43.1 Description

Tabela zawierająca wszystkie webinaty

2.43.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL
tuition_fee	money	NOT NULL, DEFAULT 0, CHECK (tuition_fee >= 0)
start_date	datetime	NOT NULL
coordinator_id	int	NOT NULL
meeting_link	nvarchar(max)	NOT NULL
video_link	nvarchar(max)	NOT NULL
description	nvarchar(max)	NOT NULL
duration	int	NOT NULL, CHECK (duration > 0)

2.43.3 SQL

```
CREATE TABLE webinar (
    id int NOT NULL,
    name nvarchar(max) NOT NULL,
    tuition_fee money NOT NULL DEFAULT 0 CHECK (tuition_fee >= 0),
    start_date datetime NOT NULL,
    coordinator_id int NOT NULL,
    meeting_link nvarchar(max) NOT NULL,
    video_link nvarchar(max) NOT NULL,
    description nvarchar(max) NOT NULL,
    duration int NOT NULL CHECK (duration > 0),
    CONSTRAINT duration_check CHECK (duration >= 15 AND duration <= 240),
    CONSTRAINT activity_id PRIMARY KEY (id)
);
```

```
ALTER TABLE webinar ADD CONSTRAINT Webinar_employee
    FOREIGN KEY (coordinator_id)
    REFERENCES employee (user_id);
```

2.44 Table role

2.44.1 Description

Tabela ról

2.44.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL

2.44.3 SQL

```
CREATE TABLE role (
    id int NOT NULL,
    name int NOT NULL,
    CONSTRAINT role_pk PRIMARY KEY (id)
);
```

2.45 Table user_role

2.45.1 Description

Tabela ról użytkowników

2.45.2 Columns

Column name	Type	Properties
user_id	int	PK
role_id	int	PK

2.45.3 SQL

```
CREATE TABLE user_role (
    user_id int NOT NULL,
    role_id int NOT NULL,
    CONSTRAINT user_role_pk PRIMARY KEY (user_id, role_id)
);
ALTER TABLE user_role ADD CONSTRAINT user_role_role
    FOREIGN KEY (role_id)
```

```
    REFERENCES role (id);

ALTER TABLE user_role ADD CONSTRAINT user_role_user
    FOREIGN KEY (user_id)
    REFERENCES "user" (id);
```

2.46 Table city

2.46.1 Description

Miasta użytkowników

2.46.2 Columns

Column name	Type	Properties
id	int	PK
name	nvarchar(max)	NOT NULL
state_id	int	NOT NULL

2.46.3 SQL

```
CREATE TABLE city (
    id int NOT NULL,
    name nvarchar(max) NOT NULL,
    state_id int NOT NULL,
    CONSTRAINT city_pk PRIMARY KEY (id)
);
ALTER TABLE city ADD CONSTRAINT Table_86_state
    FOREIGN KEY (state_id)
    REFERENCES state (id);
```

2.47 Table study_meeting

2.47.1 Description

tabela spotkań na studiach

2.47.2 Columns

Column name	Type	Properties
id	int	PK
subject_id	int	NOT NULL
start_date	datetime	NOT NULL
duration	int	NOT NULL
non_student_fee	money	NOT NULL CHECK(non_student_fee >= 0)
student_fee	money	NOT NULL CHECK(student_fee >=0)

2.47.3 SQL

```
CREATE TABLE study_meeting (
   id int  NOT NULL,
   subject_id int  NOT NULL,
   start_date datetime  NOT NULL,
   duration int  NOT NULL,
   non_student_fee money NOT NULL CHECK(non_student_fee >= 0),
   student_fee money NOT NULL CHECK(student_fee >=0)
   CONSTRAINT duration_check CHECK (duration >= 15 AND duration <= 240),
   CONSTRAINT study_meeting_pk PRIMARY KEY  (id)
);
ALTER TABLE study_meeting ADD CONSTRAINT study_meeting_subject
    FOREIGN KEY (subject_id)
    REFERENCES subject (id);
```

3 Views

3.1 View Study Programme

3.1.1 Description

Program studiów..

3.1.2 Columns

Column name	Type	Properties
study_name	nvarchar(max)	
semester_name	nvarchar(100)	
subject_name	nvarchar(max)	

3.1.3 SQL

```
SELECT
    study.name AS study_name,
    semester.name AS semester_name,
    subject.name AS subject_name
FROM
    study
INNER JOIN
    semester ON study.id = semester.study_id
INNER JOIN
    subject ON semester.id = subject.semester_id;
```

Radosław Rogalski

3.2 View study_class_meeting_schedule

3.2.1 Description

Wykaz spotkań stacjonarnych dla studiów.

3.2.2 Columns

Column name	Type	Properties
subject_name	nvarchar(max)	
date	datetime	
classroom	nvarchar(max)	

Robert Raniszewski

3.3 View study_online_meeting_schedule

3.3.1 Description

Wykaz spotkań online dla studiów.

3.3.2 Columns

Column name	Type	Properties
subject_name	nvarchar(max)	
date	datetime	
url	nvarchar(max)	
duration	int	

3.3.3 SQL

```
SELECT
    subject.name AS subject_name,
    study_meeting.start_date AS date,
    study_online_meeting.url,
    study_online_meeting.duration
FROM
    study_meeting
INNER JOIN
    study_online_meeting
    ON study_meeting.id = study_online_meeting.study_meeting_id
INNER JOIN
    subject
    ON subject.id = study_meeting.subject_id;
```

Radosław Rogalski

3.4 View student_internship_status

3.4.1 Description

Obecność studentów na praktykach.

3.4.2 Columns

Column name	Type	Properties
first_name	nvarchar(max)	
last_name	nvarchar(max)	
did_attend	int	

3.4.3 SQL

```
SELECT
    user.first_name,
    user.last_name,
    CASE
```

```
        WHEN COUNT(student_internship.id) = 14 THEN 1
        ELSE 0
    END AS did_attend
FROM
    user
LEFT JOIN
    student_internship ON user.id = student_internship.student_id
GROUP BY
    user.id, user.first_name, user.last_name;
```

Radosław Rogalski

3.5 View student_attendance

3.5.1 Description

Obecność studentów.

3.5.2 Columns

Column name	Type	Properties
first_name	nvarchar(max)	
last_name	nvarchar(max)	
did_attend	int	

3.5.3 SQL

```
SELECT
    user.first_name,
    user.last_name,
    COUNT(study_attendance.meeting_id) AS attendence_num
FROM
    user
LEFT JOIN
    study_attendance ON user.id = study_attendance.user_id
GROUP BY
    user.id, user.first_name, user.last_name;
```

Radosław Rogalski

3.6 View WebinarRevenue

3.6.1 Description

Zestawienie przychodów dla każdego webinaru

3.6.2 Columns

Column name	Type	Properties
WebinarName	nvarchar(max)	
TotalRevenue	money	

3.6.3 SQL

```
SELECT
    w.name AS WebinarName,
    SUM(od.price) AS TotalRevenue
FROM
    webinar w
INNER JOIN
    order_webinars ow ON w.id = ow.webinar_id
INNER JOIN
    order_detail od ON ow.order_detail_id = od.id
GROUP BY
    w.name;
```

2.5.1. Columns

Column name	Type	Properties
WebinarName	nvarchar(max)	
TotalRevenue	money	

Robert Raniszewski

3.7 View CourseRevenue

3.7.1 Description

Zestawienie przychodów dla każdego kursu

3.7.2 SQL

```
SELECT
    c.name AS CourseName,
    SUM(od.price) AS TotalRevenue
FROM
    course c
JOIN
    order_courses oc ON c.id = oc.course_id
JOIN
```

```
    order_detail od ON oc.order_detail_id = od.id
GROUP BY
    c.name;
```

2.6.1. Columns

Column name	Type	Properties
CourseName	nvarchar(100)	
TotalRevenue	money	

Robert Raniszewski

3.8 View StudyRevenue

3.8.1 Description

Zestawienie przychodów dla każdego studia

3.8.2 SQL

```
SELECT
    s.name AS StudyName,
    SUM(od.price) AS TotalRevenue
FROM
    study s
JOIN
    order_study os ON s.id = os.study_id
JOIN
    order_detail od ON os.order_detail_id = od.id
GROUP BY
    s.name;
```

2.7.1. Columns

Column name	Type	Properties
StudyName	nvarchar(max)	
TotalRevenue	money	

Robert Raniszewski

3.9 View FinancialReports

3.9.1 Description

Zestawienie przychodów

3.9.2 SQL

```
SELECT
    'Webinar' AS Category,
    WebinarName AS Name,
    TotalRevenue
FROM
    WebinarRevenue

UNION ALL

SELECT
    'Course' AS Category,
    CourseName AS Name,
    TotalRevenue
FROM
    CourseRevenue

UNION ALL

SELECT
    'Study' AS Category,
    StudyName AS Name,
    TotalRevenue
FROM
    StudyRevenue;
```

2.8.1. Columns

Column name	Type	Properties
Category	nvarchar(max)	
Name	nvarchar(max)	
TotalRevenue	money	

Robert Raniszewski

3.10 View unpaid_orders

3.10.1 Description:

lista dłużników

3.10.2 SQL

```
SELECT
    u.first_name AS FirstName,
    u.last_name AS LastName,
    o.date AS OrderDate,
    ISNULL(SUM(od.price),0) AS TotalAmount,
    ISNULL(SUM(p.amount), 0) AS PaidAmount,
    (o.amount - ISNULL(SUM(p.amount), 0)) AS OutstandingAmount
FROM
    [user] u
INNER JOIN
    [order] o ON u.id = o.user_id
LEFT JOIN
    payment p ON o.id = p.order_id
INNER JOIN
 order_detail od on o.id = od.order_id
GROUP BY
    u.first_name, u.last_name, o.date, o.amount
HAVING
    (o.amount - ISNULL(SUM(p.amount), 0)) > 0;
```

2.9.1. Columns

Column name	Type	Properties
FirstName	nvarchar(max)	
LastName	nvarchar(max)	
OrderDate	date	
TotalAmount	money	
PaidAmount	money	
OutstandingAmount	money	

Robert Raniszewski

3.11 View FutureCourseModulesEnrollment

3.11.1 Description

Liczba zapisanych osób na przyszłe moduły kursów

3.11.2 SQL

```
SELECT
    c.name AS CourseName,
```

```
    cm.name AS ModuleName,
    cm.start_date AS StartDate,
    COUNT(cs.user_id) AS EnrolledStudents
FROM
    course c
JOIN
    course_module cm ON c.id = cm.course_id
LEFT JOIN
    course_students cs ON cm.id = cs.course_id
WHERE
    cm.start_date > GETDATE()
GROUP BY
    c.name, cm.name, cm.start_date
ORDER BY
    cm.start_date;
```

2.10.1. Columns

Column name	Type	Properties
CourseName	nvarchar(max)	
ModuleName	nvarchar(max)	
StartDate	date	
EnrolledStudents	int	

Robert Raniszewski 0

3.12 View CourseModuleAttendance

3.12.1 Description

Frekwencja na zakończonych modułach kursów

3.12.2 SQL

```
SELECT
    cm.name AS ModuleName,
    c.name AS CourseName,
    cm.start_date AS StartDate,
    cm.duration AS Duration,
    COUNT(DISTINCT ca.user_id) AS AttendedStudents,
    COUNT(DISTINCT cs.user_id) AS EnrolledStudents,
    CASE
        WHEN COUNT(DISTINCT cs.user_id) = 0 THEN 0
        ELSE CAST(COUNT(DISTINCT ca.user_id) AS FLOAT) / COUNT(DISTINCT cs.user_id) * 100
```

```
    END AS AttendanceRate
FROM
    course_module cm
JOIN
    course c ON cm.course_id = c.id
LEFT JOIN
    course_students cs ON cm.id = cs.course_id
LEFT JOIN
    course_attendance ca ON cm.id = ca.lesson_id
WHERE
    cm.start_date + cm.duration <= GETDATE() -- Moduły zakończone
GROUP BY
    cm.name, c.name, cm.start_date, cm.duration
ORDER BY
    cm.start_date;
```

2.11.1. Columns

Column name	Type	Properties
ModuleName	nvarchar(max)	
CourseName	nvarchar(max)	
StartDate	date	
Duration	int	
AttendedStudents	int	
EnrolledStudents	int	
AttendanceRate	float(2)	

Robert Raniszewski

3.13 View CourseModulesOverview

3.13.1 Description

Spis wszystkich modułów kursów z informacjami o kursie oraz ramach czasowych

3.13.2 SQL:

```
SELECT
    c.id AS CourseID,
    c.name AS CourseName,
    cm.id AS ModuleID,
    cm.name AS ModuleName,
    MIN(cl.start_date) AS StartDate,
    DATEADD(MINUTE, SUM(cl.duration), MIN(cl.start_date)) AS EndDate,
```

```
    SUM(cl.duration) AS TotalDurationMinutes,
    ROUND(SUM(cl.duration) / 60.0, 2) AS TotalDurationHours,
    ROUND(SUM(cl.duration) / 1440.0, 2) AS TotalDurationDays -- 1440 minut = 1 dzień
FROM
    course c
JOIN
    course_module cm ON c.id = cm.course_id
JOIN
    course_lesson cl ON cm.id = cl.module_id
GROUP BY
    c.id, c.name, cm.id, cm.name
```

2.12.1. Columns

Column name	Type	Properties
CourseID	int	
CourseName	nvarchar(100)	
ModuleID	int	
ModuleName	nvarchar(100)	
StartDate	datetime	
EndDate	datetime	
TotalDurationMinutes	int	
TotalDurationHours	int	
TotalDurationDays	int	

Robert Raniszewski

3.14 View upcoming_webinars

3.14.1 SQL

```
create view upcoming_webinars as
select webinar.id, webinar.name,
webinar.start_date as [enroled_students],
count(*) as future_attendance,
'zdalne' as miejsce from webinar
join webinar_user on webinar_user.webinar_id = webinar.id
where [start_date] > getdate()
group by webinar.id, webinar.name, webinar.start_date
```

3.14.2 Columns

Column name	Type	Properties
id	int	
name	nvarchar(max)	
start_date	datetime	
future_attendance	int	
user_id	int	

Piotr Sękulski

3.15 View upcoming_activities

3.15.1 Description

Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia (z informacją, czy wydarzenie jest stacjonarnie, czy zdalnie).

3.15.2 SQL

```
create view upcoming_activities as
select webinar.id, webinar.name,
webinar.start_date as [enroled_students],
count(*) as future_attendance, 'zdalne' as miejsce from webinar
join webinar_user on webinar_user.webinar_id = webinar.id
where [start_date] > getdate()
group by webinar.id, webinar.name, webinar.start_date
union
select sm.id, sub.name, sm.start_date, count(*), 'stacjonarne'
from study_meeting as sm
join study_class_meeting as scm on scm.study_meeting_id = sm.id
join subject as sub on sub.id = sm.subject_id
join semester as sem on sem.id = sub.semester_id
join study as s on s.id = sem.study_id
join user_study as us on us.study_id = s.id
where getdate() < sm.start_date
group by sm.id, sub.name, sm.start_date
union
select sm.id, sub.name, sm.start_date, count(*), 'zdalne'
from study_meeting as sm
join study_online_meeting as som on som.study_meeting_id = sm.id
join subject as sub on sub.id = sm.subject_id
join semester as sem on sem.id = sub.semester_id
join study as s on s.id = sem.study_id
```

```
join user_study as us on us.study_id = s.id
where getdate() < sm.start_date
group by sm.id, sub.name, sm.start_date
union
select cl.id, cl.name, cl.start_date, count(*),
'stacjonarne' from course_lesson as cl
join course_stationary_lesson as csl on csl.course_lesson_id = cl.id
join course as c on c.id = cl.course_id
join course_students as cs on cs.course_id = c.id
where getdate() < cl.start_date
group by cl.id, cl.name, cl.start_date
union
select cl.id, cl.name, cl.start_date, count(*),
'zdalne' from course_lesson as cl
join course_online_lesson as col on col.course_lesson_id = cl.id
join course as c on c.id = cl.course_id
join course_students as cs on cs.course_id = c.id
where getdate() < cl.start_date
group by cl.id, cl.name, cl.start_date
union
select cl.id, cl.name, cl.start_date, count(*),
'zdalne' from course_lesson as cl
join course_online_async_lesson as coal on coal.course_lesson_id = cl.id
join course as c on c.id = cl.course_id
join course_students as cs on cs.course_id = c.id
group by cl.id, cl.name, cl.start_date
```

3.15.3 Columns

Column name	Type	Properties
id	int	
name	nvarchar(max)	
start_date	datetime	
end_date	datetime	
future_attendance	int	
miejsce	nvarchar(max)	

Piotr Sękulski

3.16 View bilocation_raport

3.16.1 Describtion

Raport bilokacji - : lista osób, które są zapisane na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo.

3.16.2 SQL

```
CREATE VIEW bilocation_report as
WITH AllFutureEvents AS (
    -- Webinars (zdalne)
    SELECT
        w.id,
        wu.user_id,
        w.name,
        w.start_date,
        DATEADD(minute, w.duration, w.start_date) AS end_date,
        'zdalne' AS miejsce
    FROM dbo.webinar w
    INNER JOIN dbo.webinar_user wu ON w.id = wu.webinar_id
    WHERE w.start_date > GETDATE()

    UNION ALL

    -- Study meetings (stacjonarne)
    SELECT
        sm.id,
        us.user_id,
        sub.name,
        sm.start_date,
        DATEADD(minute, sm.duration, sm.start_date) AS end_date,
        'stacjonarne' AS miejsce
    FROM dbo.study_meeting sm
    INNER JOIN dbo.study_class_meeting scm ON sm.id = scm.study_meeting_id
    INNER JOIN dbo.subject sub ON sm.subject_id = sub.id
    INNER JOIN dbo.semester sem ON sub.semester_id = sem.id
    INNER JOIN dbo.study s ON sem.study_id = s.id
    INNER JOIN dbo.user_study us ON s.id = us.study_id
    WHERE sm.start_date > GETDATE()

    UNION ALL

    -- Study meetings (zdalne)
    SELECT
        sm.id,
        us.user_id,
```

```
    sub.name,
    sm.start_date,
    DATEADD(minute, sm.duration, sm.start_date) AS end_date,
    'zdalne' AS miejsce
FROM dbo.study_meeting sm
INNER JOIN dbo.study_online_meeting som ON sm.id = som.study_meeting_id
INNER JOIN dbo.subject sub ON sm.subject_id = sub.id
INNER JOIN dbo.semester sem ON sub.semester_id = sem.id
INNER JOIN dbo.study s ON sem.study_id = s.id
INNER JOIN dbo.user_study us ON s.id = us.study_id
WHERE sm.start_date > GETDATE()

UNION ALL

-- Course lessons (stacjonarne)
SELECT
    cl.id,
    cs.user_id,
    cl.name,
    cl.start_date,
    DATEADD(minute, cl.duration, cl.start_date) AS end_date,
    'stacjonarne' AS miejsce
FROM dbo.course_lesson cl
INNER JOIN dbo.course_stationary_lesson csl ON cl.id = csl.course_lesson_id
INNER JOIN dbo.course_module cm ON cl.module_id = cm.id
INNER JOIN dbo.course co ON cm.course_id = co.id
INNER JOIN dbo.course_students cs ON co.id = cs.course_id
WHERE cl.start_date > GETDATE()

UNION ALL

-- Course lessons (zdalne)
SELECT
    cl.id,
    cs.user_id,
    cl.name,
    cl.start_date,
    DATEADD(minute, cl.duration, cl.start_date) AS end_date,
    'zdalne' AS miejsce
FROM dbo.course_lesson cl
INNER JOIN dbo.course_online_lesson col ON cl.id = col.course_lesson_id
INNER JOIN dbo.course_module cm ON cl.module_id = cm.id
INNER JOIN dbo.course co ON cm.course_id = co.id
INNER JOIN dbo.course_students cs ON co.id = cs.course_id
WHERE cl.start_date > GETDATE()
```

```
    UNION ALL

    -- Course lessons (zdalne, asynchroniczne)
    SELECT
        cl.id,
        cs.user_id,
        cl.name,
        cl.start_date,
        DATEADD(minute, cl.duration, cl.start_date) AS end_date,
        'zdalne' AS miejsce
    FROM dbo.course_lesson cl
    INNER JOIN dbo.course_online_async_lesson coal ON cl.id = coal.course_lesson_id
    INNER JOIN dbo.course_module cm ON cl.module_id = cm.id
    INNER JOIN dbo.course co ON cm.course_id = co.id
    INNER JOIN dbo.course_students cs ON co.id = cs.course_id
    WHERE cl.start_date > GETDATE()
)

SELECT DISTINCT
    u.id AS user_id,
    u.first_name,
    u.last_name,
    e1.name AS Event1_Name,
    e1.start_date AS Event1_Start,
    e1.end_date AS Event1_End,
    e2.name AS Event2_Name,
    e2.start_date AS Event2_Start,
    e2.end_date AS Event2_End
FROM dbo.[user] u
INNER JOIN AllFutureEvents e1 ON u.id = e1.user_id
INNER JOIN AllFutureEvents e2
    ON e1.user_id = e2.user_id
    AND e1.id < e2.id
    AND e1.start_date < e2.end_date
    AND e1.end_date > e2.start_date
```

3.16.3 Columns

user_id	int	PRIMARY KEY
first_name	nvarchar(max)	NOT NULL
last_name	nvarchar(max)	NOT NULL
event1_name	nvarchar(max)	
event1_start	datetime	
event1_end	datetime	
event2_name	nvarchar(max)	
event2_start	datetime	
event2_end	datetime	

Piotr Sękulski

3.17 View completed_studies

3.17.1 Description

Widok zawiera studentów, którzy ukończyli studia

3.17.2 SQL

```
CREATE VIEW completed_studies AS
SELECT
    u.id AS UserID,
    u.first_name AS FirstName,
    u.last_name AS LastName,
    s.name AS StudyName
FROM
    "user" u
JOIN
    user_study us ON u.id = us.user_id
JOIN
    study s ON us.study_id = s.id
LEFT JOIN
    study_attendance sa ON sa.user_id = u.id
LEFT JOIN
    study_meeting sm ON sm.id = sa.meeting_id
LEFT JOIN
    internship i ON i.study_id = s.id
LEFT JOIN
    internship_student_presence isp
    ON isp.internship_id = i.id AND isp.user_id = u.id
WHERE
```

```
(
    -- Obecność na zajęciach zwykłych >= 80%
    (
        SELECT
            COUNT(DISTINCT sa.meeting_id) * 1.0 / COUNT(DISTINCT sm.id)
        FROM
            study_meeting sm
        LEFT JOIN
            study_attendance sa
            ON sa.meeting_id = sm.id AND sa.user_id = u.id
        WHERE
            sm.id IS NOT NULL
    ) >= 0.8
)
AND
(
    -- Obecność na internship = 100%
    (
        SELECT
            COUNT(DISTINCT isp.id)
        FROM
            internship i
        JOIN
            internship_student_presence isp
            ON isp.internship_id = i.id AND isp.user_id = u.id
        WHERE
            i.study_id = s.id
    ) =
    (
        SELECT
            COUNT(DISTINCT i.id)
        FROM
            internship i
        WHERE
            i.study_id = s.id
    )
);
```

3.17.3 Columns

Column name	Type	Properties
UserID	int	
FirstName	nvarchar(max)	
LastName	nvarchar(max)	
StudyName	nvarchar(max)	

3.18 user_role view

3.18.1 Description

view of every user role

3.18.2 SQL

```
    create view user_roles as
    select u.first_name, u.last_name, u.id, r.name from [user] as u
    join user_role as ur
    on ur.user_id = u.id
    join role as r
    on r.id = ur.role_id
```

Piotr Sękulski

3.19 unpaid_attendance view

3.19.1 Description

view of people who went on activities without paying

3.19.2 SQL

```
    create view unpaid_attendance as
WITH UnpaidOrders AS (
    SELECT o.id, u.first_name, u.last_name
    FROM [order] o
    JOIN payment p ON p.order_id = o.id
    JOIN order_detail od ON o.id = od.order_id
    JOIN [user] u ON u.id = o.user_id
    GROUP BY o.id, u.first_name, u.last_name
    HAVING ROUND(SUM(p.amount), 2) <= ROUND(SUM(od.price), 2)
)
SELECT DISTINCT uo.FirstName, uo.LastName, od.id AS order_detail_id, en.text AS constent_tex
FROM            dbo.unpaid_orders uo JOIN
```

```
                        order_detail od ON od.order_id = uo.oID JOIN
                        order_webinars ow ON ow.order_detail_id = od.id JOIN
                        webinar_attendance wa ON wa.webinar_id = ow.webinar_id LEFT JOIN
                        dbo.exception en ON en.order_detail_id = od.id
UNION
SELECT DISTINCT uo.FirstName, uo.LastName, od.id AS order_detail_id, en.text AS constent_tex
FROM            dbo.unpaid_orders uo JOIN
                        order_detail od ON od.order_id = uo.oID JOIN
                        order_courses oc ON oc.order_detail_id = od.id JOIN
                        course_module cm ON cm.id = oc.course_id JOIN
                        course_lesson cl ON cl.module_id = cm.id JOIN
                        course_attendance ca ON ca.lesson_id = cl.id LEFT JOIN
                        dbo.exception en ON en.order_detail_id = od.id
UNION
SELECT DISTINCT uo.FirstName, uo.LastName, od.id AS order_detail_id, en.text AS constent_tex
FROM            dbo.unpaid_orders uo JOIN
                        order_detail od ON od.order_id = uo.oID JOIN
                        order_study os ON os.order_detail_id = od.id JOIN
                        semester sem ON sem.study_id = os.study_id JOIN
                        subject sub ON sub.semester_id = sem.id JOIN
                        study_meeting sm ON sm.subject_id = sub.id JOIN
                        study_attendance sa ON sa.meeting_id = sm.id LEFT JOIN
                        dbo.exception en ON en.order_detail_id = od.id;
```

Piotr Sękulski Radosław Rogalski

3.20 vw_course_attendance

3.20.1 Description

Course attendance summary.

3.20.2 SQL

```
CREATE VIEW vw_course_attendance AS
SELECT
    u.id AS user_id,
    u.first_name,
    u.last_name,
    c.id AS course_id,
    c.name AS course_name,
    cm.id AS module_id,
    cm.name AS module_name,
    cl.id AS lesson_id,
    cl.name AS lesson_name,
    cl.start_date,
    CASE
```

```
        WHEN coa.course_lesson_id IS NOT NULL THEN 'online-async'
        WHEN col.course_lesson_id IS NOT NULL THEN 'online'
        WHEN csl.course_lesson_id IS NOT NULL THEN 'stationary'
        ELSE 'unknown'
    END AS course_type
FROM
    course_attendance ca
JOIN
    course c ON ca.lesson_id = c.id
JOIN
    course_module cm ON c.id = cm.course_id
JOIN
    course_lesson cl ON cm.id = cl.module_id
JOIN
    users u ON ca.user_id = u.id
LEFT JOIN
    course_online_async_lesson coa ON cl.id = coa.course_lesson_id
LEFT JOIN
    course_online_lesson col ON cl.id = col.course_lesson_id
LEFT JOIN
    course_stationary_lesson csl ON cl.id = csl.course_lesson_id;
```

Radosław Rogalski

3.21 vw_study_attendance

3.21.1 Description

Study attendance summary.

3.21.2 SQL

```
CREATE VIEW vw_study_attendance AS
SELECT
    u.id AS user_id,
    u.first_name,
    u.last_name,
    s.id AS study_id,
    s.name AS study_name,
    sem.id AS semester_id,
    sem.name AS semester_name,
    sub.id AS subject_id,
    sub.name AS subject_name,
    sm.id AS meeting_id, -- Added meeting_id
    sm.start_date,
    CASE
```

```
        WHEN som.study_meeting_id IS NOT NULL THEN 'online'
        WHEN scm.study_meeting_id IS NOT NULL THEN 'stationary'
        ELSE 'stationary'
    END AS meeting_type
FROM
    dbo.study_attendance AS sa
INNER JOIN
    dbo.study_meeting AS sm ON sa.meeting_id = sm.id
INNER JOIN
    dbo.subject AS sub ON sm.subject_id = sub.id
INNER JOIN
    dbo.semester AS sem ON sub.semester_id = sem.id
INNER JOIN
    dbo.study AS s ON sem.study_id = s.id
LEFT OUTER JOIN
    dbo.study_online_meeting AS som ON sm.id = som.study_meeting_id
LEFT OUTER JOIN
    dbo.study_class_meeting AS scm ON sm.id = scm.study_meeting_id
INNER JOIN
    dbo.[user] AS u ON sa.user_id = u.id;
```

Radosław Rogalski

3.22 vw_webinar_attendance

3.22.1 Description

Webinar attendance summary.

3.22.2 SQL

```
CREATE VIEW vw_webinar_attendance AS
SELECT
    u.id AS user_id,
    u.first_name,
    u.last_name,
    w.id AS webinar_id,
    w.start_date,
    w.name AS webinar_name
FROM
    webinar_attendance wa
JOIN
    webinar w ON wa.webinar_id = w.id
JOIN
```

```
    users u ON wa.user_id = u.id;
```

Radosław Rogalski

3.23 vw_user_activity

3.23.1 Description

Full user activity including studies, webinars and courses.

3.23.2 SQL

```
CREATE VIEW vw_user_activity AS
SELECT
    user_id,
    first_name,
    last_name,
    date,
    webinar_name AS activity_name
FROM
    vw_webinar_attendance

UNION ALL

SELECT
    user_id,
    first_name,
    last_name,
    start_date AS date,
    lesson_name AS activity_name
FROM
    vw_course_attendance

UNION ALL

SELECT
    user_id,
    first_name,
    last_name,
    start_date AS date,
    subject_name AS activity_name
FROM
    vw_student_attendance;
```

Radosław Rogalski

4 Procedures

4.1 Procedure AddOrderDetail

4.1.1 Description

Procedura dodaje nową pozycję zamówienia

4.1.2 SQL:

```
ALTER PROCEDURE [dbo].[AddOrderDetail]
    @OrderID INT,
    @DetailType NVARCHAR(MAX), -- Typ zamówienia: 'Webinar', 'Course', 'Study'
    @DetailID INT, -- ID webinaru, kursu lub studiów
    @StudentID INT,
    @OrderDetailTypeID INT
AS
BEGIN
    -- Start of the transaction
    BEGIN TRANSACTION;
 -- Sprawdzenie, czy student istnieje
    IF NOT EXISTS (SELECT 1 FROM [user] WHERE id = @StudentID)
    BEGIN
        RAISERROR('Student o podanym ID nie istnieje.', 16, 1);
  ROLLBACK TRANSACTION;
        RETURN;
    END


    -- Dodanie zamówienia

    -- Sprawdzenie, czy zamówienie o podanym ID już istnieje
        IF NOT EXISTS (SELECT 1 FROM [order] WHERE id = @OrderID)
        BEGIN
            -- Dodanie zamówienia
            INSERT INTO [order] (id, user_id, date, payment_url)
            VALUES (@OrderID, @StudentID, GETDATE(),
            CONCAT('https://payment.example.com/order/', @OrderID));
        END
        ELSE
        BEGIN
            PRINT 'Zamówienie o podanym ID już istnieje. Pomijam dodanie zamówienia.';
        END
```

```
  -- Obsługa webinarów
  IF @DetailType = 'Webinar'
  BEGIN
      -- Sprawdzenie, czy student jest już zapisany na ten webinar
      IF EXISTS
      (SELECT 1 FROM webinar_user
      WHERE webinar_id = @DetailID AND user_id = @StudentID)
      BEGIN
          RAISERROR('Student jest już zapisany na ten webinar.', 16, 1);
          ROLLBACK TRANSACTION;
          RETURN;
      END
DECLARE @OrderDetailID INT;
DECLARE @OrderWebinarID INT;
SELECT @OrderWebinarID = ISNULL(MAX(id), 0) + 1 FROM dbo.order_webinars;
-- Generowanie nowego ID
SELECT @OrderDetailID = ISNULL(MAX(id), 0) + 1 FROM order_detail;

-- Dodanie szczegółów zamówienia
      INSERT INTO order_detail
      (id, order_id, price, order_detail_type_id)
      VALUES (@OrderDetailID,@OrderID,
      (SELECT tuition_fee FROM webinar WHERE id = @DetailID), @OrderDetailTypeID);
INSERT INTO order_webinars (id, webinar_id, order_detail_id)
VALUES (@OrderWebinarID, @DetailID, @OrderDetailID);



  END

  -- Obsługa kursów
  ELSE IF @DetailType = 'Course'
  BEGIN
      -- Sprawdzenie, czy student jest już zapisany na ten kurs
      IF EXISTS
      (SELECT 1 FROM course_students
      WHERE course_id = @DetailID AND user_id = @StudentID)
      BEGIN
          RAISERROR('Student jest już zapisany na ten kurs.', 16, 1);
          ROLLBACK TRANSACTION;
          RETURN;
      END
      -- Generowanie nowego ID
      SELECT @OrderDetailID = ISNULL(MAX(id), 0) + 1 FROM order_detail;

IF @OrderDetailTypeID = 1
```

```
BEGIN
        -- Dodanie szczegółów zamówienia z wygenerowanym ID
        INSERT INTO order_detail
           (id, order_id, price, order_detail_type_id)
        VALUES
           (@OrderDetailID, @OrderID,
           (SELECT price FROM course WHERE id = @DetailID), 1);
END
ELSE IF @OrderDetailTypeID = 2
BEGIN
 -- Dodanie szczegółów zamówienia z wygenerowanym ID
        INSERT INTO order_detail
           (id, order_id, price, order_detail_type_id)
        VALUES (@OrderDetailID, @OrderID,
           (SELECT price FROM course WHERE id = @DetailID) * 0.3, 2);
END
ELSE IF @OrderDetailTypeID = 3
BEGIN
 -- Dodanie szczegółów zamówienia z wygenerowanym ID
        INSERT INTO order_detail
           (id, order_id, price, order_detail_type_id)
        VALUES (@OrderDetailID, @OrderID,
           (SELECT price FROM course WHERE id = @DetailID) * 0.7, 3);
END
ELSE
BEGIN
       RAISERROR('Nieprawidłowy typ szczegółów zamówienia.', 16, 1);
       ROLLBACK TRANSACTION;
       RETURN;
   END


DECLARE @OrderCourseID INT;
   SELECT @OrderCourseID = ISNULL(MAX(id), 0) + 1 FROM dbo.order_courses;
   INSERT INTO order_courses (id, course_id, order_detail_id)
   VALUES (@OrderCourseID,@DetailID, @OrderDetailID);
END
   -- Obsługa studiów
   ELSE IF @DetailType = 'Study'
   BEGIN
       -- Sprawdzenie, czy student jest już zapisany na te studia
       IF EXISTS
       (SELECT 1 FROM user_study
       WHERE study_id = @DetailID AND user_id = @StudentID)
       BEGIN
           RAISERROR('Student jest już zapisany na te studia.', 16, 1);
```

```
            ROLLBACK TRANSACTION;
            RETURN;
        END

 DECLARE @OrderStudyID INT;

        -- Generowanie nowego ID
        SELECT @OrderDetailID = ISNULL(MAX(id), 0) + 1 FROM order_detail;

IF @OrderDetailTypeID = 1
BEGIN
         -- Dodanie szczegółów zamówienia
         INSERT INTO order_detail
            (id, order_id, price, order_detail_type_id)
         VALUES (@OrderDetailID, @OrderID,
            (SELECT tuition_fee FROM study WHERE id = @DetailID), 3);
END
ELSE IF @OrderDetailTypeID = 2
BEGIN
 -- Dodanie szczegółów zamówienia
         INSERT INTO order_detail
            (id, order_id, price, order_detail_type_id)
         VALUES (@OrderDetailID, @OrderID,
            (SELECT tuition_fee FROM study WHERE id = @DetailID) * 0.3, 3);
END
ELSE IF @OrderDetailTypeID = 3
BEGIN
 -- Dodanie szczegółów zamówienia
         INSERT INTO order_detail
            (id, order_id, price, order_detail_type_id)
         VALUES (@OrderDetailID, @OrderID,
            (SELECT tuition_fee FROM study WHERE id = @DetailID) * 0.7, 3);
END
ELSE
BEGIN
        RAISERROR('Nieprawidłowy typ szczegółów zamówienia.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
 SELECT @OrderStudyID = ISNULL(MAX(id), 0) + 1 FROM dbo.order_study;
 INSERT INTO order_study (id, study_id, order_detail_id)
        VALUES (@OrderStudyID,@DetailID, @OrderDetailID);
   END
   ELSE
   BEGIN
        RAISERROR('Nieprawidłowy typ szczegółów zamówienia.', 16, 1);
```

```
        ROLLBACK TRANSACTION;
        RETURN;
    END

    -- If all operations are successful, commit the transaction
    COMMIT TRANSACTION;
END;
```

Robert Raniszewski

4.2 Procedure add user

4.2.1 Describtion

Procedue which adds user to database

4.3 SQL

```
    CREATE PROCEDURE dbo.add_user
    @p_email VARCHAR(100),
    @p_phone VARCHAR(15),
    @p_first_name NVARCHAR(MAX),
    @p_last_name NVARCHAR(MAX),
    @p_country_id INT,
    @p_state_id INT,
    @p_city_id INT,
    @p_zip_code NVARCHAR(MAX),
    @p_role_id INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @v_user_id INT;
    DECLARE @ErrorMessage NVARCHAR(4000);

    -- Input validation
    IF @p_email IS NULL OR @p_first_name IS NULL OR @p_last_name IS NULL
    BEGIN
        THROW 50001, 'Email, first name, and last name are required', 1;
        RETURN;
    END

    -- Verify foreign key references exist
    IF NOT EXISTS (SELECT 1 FROM country WHERE id = @p_country_id)
    BEGIN
        THROW 50002, 'Invalid country_id', 1;
```

```
    RETURN;
END

IF NOT EXISTS (SELECT 1 FROM state WHERE id = @p_state_id AND country_id = @p_country_id
BEGIN
    THROW 50003, 'Invalid state_id or state does not belong to specified country', 1;
    RETURN;
END

IF NOT EXISTS (SELECT 1 FROM city WHERE id = @p_city_id AND state_id = @p_state_id)
BEGIN
    THROW 50004, 'Invalid city_id or city does not belong to specified state', 1;
    RETURN;
END

IF NOT EXISTS (SELECT 1 FROM role WHERE id = @p_role_id)
BEGIN
    THROW 50005, 'Invalid role_id', 1;
    RETURN;
END

BEGIN TRY
    BEGIN TRANSACTION;

    -- Insert new user
    INSERT INTO [user] (
        email,
        phone,
        first_name,
        last_name,
        country_id,
        state_id,
        city_id,
        zip_code
    )
    VALUES (
        @p_email,
        @p_phone,
        @p_first_name,
        @p_last_name,
        @p_country_id,
        @p_state_id,
        @p_city_id,
        @p_zip_code
    );
```

```
        SET @v_user_id = SCOPE_IDENTITY();

        -- Assign role to user
        INSERT INTO user_role (user_id, role_id)
        VALUES (@v_user_id, @p_role_id);

        COMMIT TRANSACTION;

        -- Return the new user ID
        SELECT @v_user_id AS new_user_id;

    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        SET @ErrorMessage =
            ERROR_MESSAGE() + ' Line ' +
            CAST(ERROR_LINE() AS NVARCHAR(5)) +
            ' Error ' +
            CAST(ERROR_NUMBER() AS NVARCHAR(10));

        THROW 50000, @ErrorMessage, 1;
    END CATCH;
END;
```

Piotr Sękulski

4.4 Procedure Add City

4.4.1 Description

Procedure which adds Cities to database

4.4.2 SQL

```
    CREATE PROCEDURE dbo.add_city
    @p_name NVARCHAR(MAX),
    @p_state_id INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @v_city_id INT;

    IF @p_name IS NULL OR LTRIM(RTRIM(@p_name)) = ''
    BEGIN
```

```
        THROW 50001, 'City name is required', 1;
        RETURN;
    END

    -- Validate state exists
    IF NOT EXISTS (SELECT 1 FROM state WHERE id = @p_state_id)
    BEGIN
        THROW 50002, 'Invalid state_id', 1;
        RETURN;
    END

    -- Check if city already exists in this state
    IF EXISTS (SELECT 1 FROM city WHERE name = @p_name AND state_id = @p_state_id)
    BEGIN
        THROW 50003, 'City with this name already exists in the specified state', 1;
        RETURN;
    END

    BEGIN TRY
        INSERT INTO city (name, state_id)
        VALUES (@p_name, @p_state_id);

        SET @v_city_id = SCOPE_IDENTITY();

        SELECT @v_city_id AS new_city_id;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
```

Piotr Sękuslki

4.5 Procedure Add State

4.5.1 Describtion

Procedure which adds state to database

4.5.2 SQL

```
 CREATE PROCEDURE dbo.add_state
    @p_name NVARCHAR(MAX),
    @p_country_id INT
AS
BEGIN
    SET NOCOUNT ON;
```

```
    DECLARE @v_state_id INT;

    IF @p_name IS NULL OR LTRIM(RTRIM(@p_name)) = ''
    BEGIN
        THROW 50001, 'State name is required', 1;
        RETURN;
    END

    -- Validate country exists
    IF NOT EXISTS (SELECT 1 FROM country WHERE id = @p_country_id)
    BEGIN
        THROW 50002, 'Invalid country_id', 1;
        RETURN;
    END

    -- Check if state already exists in this country
    IF EXISTS (SELECT 1 FROM state WHERE name = @p_name AND country_id = @p_country_id)
    BEGIN
        THROW 50003, 'State with this name already exists in the specified country', 1;
        RETURN;
    END

    BEGIN TRY
        INSERT INTO state (name, country_id)
        VALUES (@p_name, @p_country_id);

        SET @v_state_id = SCOPE_IDENTITY();

        SELECT @v_state_id AS new_state_id;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
GO
```

Piotr Sękulski

4.6 Procedure Add Country

4.6.1 Description

Procedure which adds countris to database

4.6.2 SQL

```
    CREATE PROCEDURE dbo.add_country
    @p_name NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @v_country_id INT;

    IF @p_name IS NULL OR LTRIM(RTRIM(@p_name)) = ''
    BEGIN
        THROW 50001, 'Country name is required', 1;
        RETURN;
    END

    -- Check if country already exists
    IF EXISTS (SELECT 1 FROM country WHERE name = @p_name)
    BEGIN
        THROW 50002, 'Country with this name already exists', 1;
        RETURN;
    END

    BEGIN TRY
        INSERT INTO country (name)
        VALUES (@p_name);

        SET @v_country_id = SCOPE_IDENTITY();

        SELECT @v_country_id AS new_country_id;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
GO
```

Piotr Sękulski

4.7 Procedure Add Employee

4.7.1 Description

Procedure which adds employees to database

4.7.2 SQL

```
    CREATE PROCEDURE dbo.add_employee
```

```
    @p_email VARCHAR(100),
    @p_phone VARCHAR(15),
    @p_first_name NVARCHAR(MAX),
    @p_last_name NVARCHAR(MAX),
    @p_country_id INT,
    @p_state_id INT,
    @p_city_id INT,
    @p_zip_code NVARCHAR(MAX),
    @p_role_id INT,
    @p_salary MONEY
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @v_user_id INT;
    DECLARE @ErrorMessage NVARCHAR(4000);

    -- Input validation
    IF @p_email IS NULL OR @p_first_name IS NULL OR @p_last_name IS NULL
    BEGIN
        THROW 50001, 'Email, first name, and last name are required', 1;
        RETURN;
    END

    IF @p_salary <= 0
    BEGIN
        THROW 50002, 'Salary must be greater than zero', 1;
        RETURN;
    END

    -- Verify foreign key references exist
    IF NOT EXISTS (SELECT 1 FROM country WHERE id = @p_country_id)
    BEGIN
        THROW 50003, 'Invalid country_id', 1;
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM state WHERE id = @p_state_id AND country_id = @p_country_id
    BEGIN
        THROW 50004, 'Invalid state_id or state does not belong to specified country', 1;
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM city WHERE id = @p_city_id AND state_id = @p_state_id)
    BEGIN
        THROW 50005, 'Invalid city_id or city does not belong to specified state', 1;
```

```
    RETURN;
END

IF NOT EXISTS (SELECT 1 FROM role WHERE id = @p_role_id)
BEGIN
    THROW 50006, 'Invalid role_id', 1;
    RETURN;
END

-- Check if email already exists
IF EXISTS (SELECT 1 FROM [user] WHERE email = @p_email)
BEGIN
    THROW 50007, 'Email address already exists', 1;
    RETURN;
END

BEGIN TRY
    BEGIN TRANSACTION;

    -- Insert new user
    INSERT INTO [user] (
        email,
        phone,
        first_name,
        last_name,
        country_id,
        state_id,
        city_id,
        zip_code
    )
    VALUES (
        @p_email,
        @p_phone,
        @p_first_name,
        @p_last_name,
        @p_country_id,
        @p_state_id,
        @p_city_id,
        @p_zip_code
    );

    SET @v_user_id = SCOPE_IDENTITY();

    -- Assign role to user
    INSERT INTO user_role (user_id, role_id)
    VALUES (@v_user_id, @p_role_id);
```

```
        -- Create employee record
        INSERT INTO employee (user_id, salary)
        VALUES (@v_user_id, @p_salary);

        COMMIT TRANSACTION;

        -- Return the new user/employee ID
        SELECT @v_user_id AS new_employee_id;

    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        SET @ErrorMessage =
            ERROR_MESSAGE() + ' Line ' +
            CAST(ERROR_LINE() AS NVARCHAR(5)) +
            ' Error ' +
            CAST(ERROR_NUMBER() AS NVARCHAR(10));

        THROW 50000, @ErrorMessage, 1;
    END CATCH;
END;
```

Piotr Sękulski

4.8 Procedure Add Language

4.8.1 Description

Procedure which adds language to database

4.8.2 SQL

```
    CREATE PROCEDURE dbo.add_language
    @p_name NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @v_language_id INT;

    IF @p_name IS NULL OR LTRIM(RTRIM(@p_name)) = ''
    BEGIN
        THROW 50001, 'Language name is required', 1;
        RETURN;
```

```
    END

    -- Check if language already exists
    IF EXISTS (SELECT 1 FROM language WHERE name = @p_name)
    BEGIN
        THROW 50002, 'Language with this name already exists', 1;
        RETURN;
    END

    BEGIN TRY
        INSERT INTO language (name)
        VALUES (@p_name);

        SET @v_language_id = SCOPE_IDENTITY();

        SELECT @v_language_id AS new_language_id;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
GO
```

Piotr Sękulski

4.9 Procedure Add Translator

4.9.1 Description

Procedure which adds Translator to database

4.9.2 SQL

```
    CREATE PROCEDURE dbo.add_translator
    @p_user_id INT,
    @p_language_id INT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @translator_role_id INT;

    -- Validate user exists
    IF NOT EXISTS (SELECT 1 FROM [user] WHERE id = @p_user_id)
    BEGIN
        THROW 50001, 'Invalid user_id', 1;
        RETURN;
    END
```

```
    -- Validate language exists
    IF NOT EXISTS (SELECT 1 FROM language WHERE id = @p_language_id)
    BEGIN
        THROW 50002, 'Invalid language_id', 1;
        RETURN;
    END

    -- Check if translator already exists for this language
    IF EXISTS (SELECT 1 FROM translator
               WHERE user_id = @p_user_id
               AND language_id = @p_language_id)
    BEGIN
        THROW 50003, 'This user is already registered as a translator for this language', 1;
        RETURN;
    END

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Add translator
        INSERT INTO translator (user_id, language_id)
        VALUES (@p_user_id, @p_language_id);

        -- Make sure user has translator role
        SELECT @translator_role_id = id FROM role WHERE name = 'Translator';

        IF @translator_role_id IS NOT NULL
            AND NOT EXISTS (SELECT 1 FROM user_role
                            WHERE user_id = @p_user_id
                            AND role_id = @translator_role_id)
        BEGIN
            INSERT INTO user_role (user_id, role_id)
            VALUES (@p_user_id, @translator_role_id);
        END

        COMMIT TRANSACTION;

        SELECT @p_user_id AS translator_user_id, @p_language_id AS language_id;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
```

```
GO
```

Piotr Sękulski

4.10 Procedure Add Multiple Language

4.10.1 Description

Procedure which adds multiple language as id separated by coma to translator id

4.10.2 SQL

```
    CREATE PROCEDURE dbo.add_translator
    @p_user_id INT,
    @p_language_id INT
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @translator_role_id INT;

    -- Validate user exists
    IF NOT EXISTS (SELECT 1 FROM [user] WHERE id = @p_user_id)
    BEGIN
        THROW 50001, 'Invalid user_id', 1;
        RETURN;
    END

    -- Validate language exists
    IF NOT EXISTS (SELECT 1 FROM language WHERE id = @p_language_id)
    BEGIN
        THROW 50002, 'Invalid language_id', 1;
        RETURN;
    END

    -- Check if translator already exists for this language
    IF EXISTS (SELECT 1 FROM translator
               WHERE user_id = @p_user_id
               AND language_id = @p_language_id)
    BEGIN
        THROW 50003, 'This user is already registered as a translator for this language', 1;
        RETURN;
    END

    BEGIN TRY
        BEGIN TRANSACTION;
```

```
        -- Add translator
        INSERT INTO translator (user_id, language_id)
        VALUES (@p_user_id, @p_language_id);

        -- Make sure user has translator role
        SELECT @translator_role_id = id FROM role WHERE name = 'Translator';

        IF @translator_role_id IS NOT NULL
            AND NOT EXISTS (SELECT 1 FROM user_role
                            WHERE user_id = @p_user_id
                            AND role_id = @translator_role_id)
        BEGIN
            INSERT INTO user_role (user_id, role_id)
            VALUES (@p_user_id, @translator_role_id);
        END

        COMMIT TRANSACTION;

        SELECT @p_user_id AS translator_user_id, @p_language_id AS language_id;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

Piotr Sękulski

4.11 Procedure Add Webinar

4.11.1 Describtion

Procedure which adds Webiar into database

4.11.2 SQL

```
    -- Add Webinar Procedure
CREATE PROCEDURE dbo.add_webinar
    @p_name NVARCHAR(MAX),
    @p_tuition_fee MONEY,
    @p_start_date DATETIME,
    @p_coordinator_id INT,
    @p_meeting_link NVARCHAR(MAX),
    @p_video_link NVARCHAR(MAX),
    @p_description NVARCHAR(MAX),
```

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

    DECLARE @v_webinar_id INT;

    -- Basic validation
    IF @p_name IS NULL OR LTRIM(RTRIM(@p_name)) = ''
    BEGIN
        THROW 50001, 'Webinar name is required', 1;
        RETURN;
    END

    IF @p_start_date IS NULL OR @p_start_date < GETDATE()
    BEGIN
        THROW 50002, 'Start date must be in the future', 1;
        RETURN;
    END

    IF @p_duration <= 0
    BEGIN
        THROW 50003, 'Duration must be greater than zero', 1;
        RETURN;
    END

    -- Validate coordinator exists
    IF NOT EXISTS (SELECT 1 FROM [user] WHERE id = @p_coordinator_id)
    BEGIN
        THROW 50004, 'Invalid coordinator_id', 1;
        RETURN;
    END

    BEGIN TRY
        INSERT INTO webinar (
            name,
            tuition_fee,
            start_date,
            coordinator_id,
            meeting_link,
            video_link,
            describtion,
            duration
        )
        VALUES (
            @p_name,
```

```
            @p_tuition_fee,
            @p_start_date,
            @p_coordinator_id,
            @p_meeting_link,
            @p_video_link,
            @p_description,
            @p_duration
        );

        SET @v_webinar_id = SCOPE_IDENTITY();

        SELECT @v_webinar_id AS new_webinar_id;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
GO
```

Piotr Sękulski

4.12 Procedure Add Webinar Translation

4.12.1 Description

Procedure Which adds translator to webinar

4.12.2 SQL

```
    -- Add Webinar Procedure
CREATE PROCEDURE dbo.add_webinar
    @p_name NVARCHAR(MAX),
    @p_tuition_fee MONEY,
    @p_start_date DATETIME,
    @p_coordinator_id INT,
    @p_meeting_link NVARCHAR(MAX),
    @p_video_link NVARCHAR(MAX),
    @p_description NVARCHAR(MAX),
    @p_duration INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @v_webinar_id INT;

    -- Basic validation
    IF @p_name IS NULL OR LTRIM(RTRIM(@p_name)) = ''
```

```
BEGIN
    THROW 50001, 'Webinar name is required', 1;
    RETURN;
END

IF @p_start_date IS NULL OR @p_start_date < GETDATE()
BEGIN
    THROW 50002, 'Start date must be in the future', 1;
    RETURN;
END

IF @p_duration <= 0
BEGIN
    THROW 50003, 'Duration must be greater than zero', 1;
    RETURN;
END

-- Validate coordinator exists
IF NOT EXISTS (SELECT 1 FROM [user] WHERE id = @p_coordinator_id)
BEGIN
    THROW 50004, 'Invalid coordinator_id', 1;
    RETURN;
END

BEGIN TRY
    INSERT INTO webinar (
        name,
        tuition_fee,
        start_date,
        coordinator_id,
        meeting_link,
        video_link,
        describtion,
        duration
    )
    VALUES (
        @p_name,
        @p_tuition_fee,
        @p_start_date,
        @p_coordinator_id,
        @p_meeting_link,
        @p_video_link,
        @p_description,
        @p_duration
    );
```

```
        SET @v_webinar_id = SCOPE_IDENTITY();

        SELECT @v_webinar_id AS new_webinar_id;
    END TRY
    BEGIN CATCH
        THROW;
    END CATCH;
END;
GO
```

Piotr Sękulski

4.13 Add Webinar Attendance Procedure

4.13.1 Description

Procedure which adds user attendance to database

4.13.2 SQL

```
    CREATE PROCEDURE dbo.add_webinar_attendance
    @p_user_id INT,
    @p_webinar_id INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Validate user exists
    IF NOT EXISTS (SELECT 1 FROM [user] WHERE id = @p_user_id)
    BEGIN
        THROW 50001, 'Invalid user_id', 1;
        RETURN;
    END

    -- Validate webinar exists
    IF NOT EXISTS (SELECT 1 FROM webinar WHERE id = @p_webinar_id)
    BEGIN
        THROW 50002, 'Invalid webinar_id', 1;
        RETURN;
    END

    -- Check if attendance already exists
    IF EXISTS (
        SELECT 1
        FROM webinar_attendance
        WHERE user_id = @p_user_id
        AND webinar_id = @p_webinar_id
```

```
    )
    BEGIN
        THROW 50003, 'User is already registered for this webinar', 1;
        RETURN;
    END

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Add attendance record
        INSERT INTO webinar_attendance (
            user_id,
            webinar_id
        )
        VALUES (
            @p_user_id,
            @p_webinar_id
        );

        COMMIT TRANSACTION;

        -- Return attendance details
        SELECT
            wa.user_id,
            wa.webinar_id,
            w.name as webinar_name,
            w.start_date,
            w.duration
        FROM webinar_attendance wa
        JOIN webinar w ON w.id = wa.webinar_id
        WHERE wa.user_id = @p_user_id
        AND wa.webinar_id = @p_webinar_id;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;
        THROW;
    END CATCH;
END;
GO
```

Piotr Sękulski

4.14 Add study procedure

4.14.1 Description

Adds new study.

4.14.2 SQL

```
CREATE PROCEDURE AddStudy
    @Name NVARCHAR(MAX),
    @TuitionFee MONEY,
    @CoordinatorId INT,
    @StudentLimit INT
AS
BEGIN
    BEGIN TRY
        INSERT INTO study (name, tuition_fee, coordinator_id, student_limit)
        VALUES (@Name, @TuitionFee, @CoordinatorId, @StudentLimit);
        PRINT 'Study record added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'An error occurred while adding a study.';
        THROW;
    END CATCH;
END;
```

Radosław Rogalski

4.15 Add semester procedure

4.15.1 Description

Adds new semester.

4.15.2 SQL

```
CREATE PROCEDURE AddSemester
    @Name NVARCHAR(100),
    @StudyId INT
AS
BEGIN
    BEGIN TRY
        INSERT INTO semester (name, study_id)
        VALUES (@Name, @StudyId);

        PRINT 'Semester record added successfully.';
    END TRY
    BEGIN CATCH
```

```
        PRINT 'An error occurred while adding a semester.';

        THROW;
    END CATCH;
END;
```

Radosław Rogalski

4.16 Add subject procedure

4.16.1 Description

Adds new subject.

4.16.2 SQL

```
CREATE PROCEDURE AddSubject
    @Name NVARCHAR(MAX),
    @SupervisorId INT,
    @SemesterId INT
AS
BEGIN
    BEGIN TRY
        INSERT INTO subject (name, supervisor_id, semester_id)
        VALUES (@Name, @SupervisorId, @SemesterId);

        PRINT 'Subject record added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'An error occurred while adding a subject.';

        THROW;
    END CATCH;
END;
```

Radosław Rogalski

4.17 Add study online meeting procedure

4.17.1 Description

Adds new online meeting for study.

4.17.2 SQL

```
CREATE PROCEDURE AddStudyOnlineMeeting
    @SubjectId INT,
    @StartDate DATETIME,
```

```
    @Duration INT,
    @NonStudentFee MONEY,
    @StudentFee MONEY,
    @Url NVARCHAR(MAX)
AS
BEGIN
    BEGIN TRY
        INSERT INTO study_meeting (subject_id, start_date, duration, non_student_fee, studen
        VALUES (@SubjectId, @StartDate, @Duration, @NonStudentFee, @StudentFee);

        DECLARE @StudyMeetingId INT = SCOPE_IDENTITY();

        INSERT INTO study_online_meeting (study_meeting_id, url)
        VALUES (@StudyMeetingId, @Url);

        PRINT 'Study meeting and online meeting record added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'An error occurred while adding a study meeting or study online meeting.';

        THROW;
    END CATCH;
END;
```

Radosław Rogalski

4.18 Add study class meeting procedure

4.18.1 Description

Adds new class meeting for study.

4.18.2 SQL

```
CREATE PROCEDURE AddStudyClassMeeting
    @SubjectId INT,
    @StartDate DATETIME,
    @Duration INT,
    @NonStudentFee MONEY,
    @StudentFee MONEY,
    @ClassroomId INT
AS
BEGIN
    BEGIN TRY
        INSERT INTO study_meeting (subject_id, start_date, duration, non_student_fee, studen
        VALUES (@SubjectId, @StartDate, @Duration, @NonStudentFee, @StudentFee);
```

```
        DECLARE @StudyMeetingId INT = SCOPE_IDENTITY();

        INSERT INTO study_class_meeting (study_meeting_id, classroom_id)
        VALUES (@StudyMeetingId, @ClassroomId);

        PRINT 'Study meeting and study class meeting record added successfully.';
    END TRY
    BEGIN CATCH
        PRINT 'An error occurred while adding a study meeting or study class meeting.';

        THROW;
    END CATCH;
END;
```

Radosław Rogalski

4.19 Add student attendance

4.19.1 Description

Adds attendance for a student for a specified activity.

4.19.2 SQL

```
CREATE PROCEDURE AddStudentAttendance
    @UserID INT,
    @ActivityType NVARCHAR(50), -- 'Webinar', 'Course', 'Study'
    @ActivityID INT            -- ID of the webinar, course lesson, or study meeting
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        IF @ActivityType = 'Webinar'
        BEGIN
            IF EXISTS (SELECT 1 FROM webinar_attendance WHERE user_id = @UserID AND webinar_
            BEGIN
                RAISERROR('The student is already marked as attended for this webinar.', 16,
                ROLLBACK TRANSACTION;
                RETURN;
            END

            INSERT INTO webinar_attendance (user_id, webinar_id)
            VALUES (@UserID, @ActivityID);
        END
        ELSE IF @ActivityType = 'Course'
```

```
        BEGIN
            IF EXISTS (SELECT 1 FROM course_attendance WHERE user_id = @UserID AND lesson_id
            BEGIN
                RAISERROR('The student is already marked as attended for this course lesson.
                ROLLBACK TRANSACTION;
                RETURN;
            END

            INSERT INTO course_attendance (user_id, lesson_id)
            VALUES (@UserID, @ActivityID);
        END
        ELSE IF @ActivityType = 'Study'
        BEGIN
            IF EXISTS (SELECT 1 FROM study_attendance WHERE user_id = @UserID AND meeting_id
            BEGIN
                RAISERROR('The student is already marked as attended for this study meeting.
                ROLLBACK TRANSACTION;
                RETURN;
            END

            INSERT INTO study_attendance (user_id, meeting_id)
            VALUES (@UserID, @ActivityID);
        END
        ELSE
        BEGIN
            RAISERROR('Invalid activity type specified. Use ''Webinar'', ''Course'', or ''St
            ROLLBACK TRANSACTION;
            RETURN;
        END

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        -- Handle errors and rollback transaction
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        -- Re-throw the error
        THROW;
    END CATCH
END;
```

Radosław Rogalski

4.20 Get study meetings

4.20.1 Description

Gets study meetings for specified study id.

4.20.2 SQL

```
CREATE PROCEDURE GetStudyMeetings
    @StudyId INT
AS
BEGIN
    -- Retrieve meetings for the specified study
    SELECT
        sm.id AS meeting_id,
        sm.start_date,
        sm.duration,
        sub.id AS subject_id,
        sub.name AS subject_name,
        sem.id AS semester_id,
        sem.name AS semester_name,
        s.id AS study_id,
        s.name AS study_name,
        CASE
            WHEN som.study_meeting_id IS NOT NULL THEN 'online'
            WHEN scm.study_meeting_id IS NOT NULL THEN 'stationary'
            ELSE 'unknown'
        END AS meeting_type
    FROM
        study_meeting sm
    JOIN
        subject sub ON sm.subject_id = sub.id
    JOIN
        semester sem ON sub.semester_id = sem.id
    JOIN
        study s ON sem.study_id = s.id
    LEFT JOIN
        study_online_meeting som ON sm.id = som.study_meeting_id
    LEFT JOIN
        study_class_meeting scm ON sm.id = scm.study_meeting_id
    WHERE
        s.id = @StudyId
    ORDER BY
        sm.start_date;
END;
```

Radosław Rogalski

4.21 Get student attendance by meeting id

4.21.1 Description

Gets attendance by meeting id.

4.21.2 SQL

```
CREATE PROCEDURE GetStudyAttendanceByMeetingId
    @MeetingId INT
AS
BEGIN
    -- Filter the vw_study_attendance view based on the provided meetingId
    SELECT
        user_id,
        first_name,
        last_name,
        study_id,
        study_name,
        semester_id,
        semester_name,
        subject_id,
        subject_name,
        start_date,
        meeting_type,
        meeting_id
    FROM
        vw_study_attendance
    WHERE
        meeting_id = @MeetingId;
END;
```

Radosław Rogalski

4.22 Get subject attendance report

4.22.1 Description

Gets attendance report for subject.

4.22.2 SQL

```
CREATE PROCEDURE GetSubjectAttendanceReport
    @SubjectId INT
AS
```

```
BEGIN
    SELECT
        u.first_name,
        u.last_name,
        COUNT(sa.meeting_id) AS attendance_count
    FROM
        study_attendance sa
    JOIN
        study_meeting sm ON sa.meeting_id = sm.id
    JOIN
        [user] u ON sa.user_id = u.id
    WHERE
        sm.subject_id = @SubjectId
    GROUP BY
        u.first_name,
        u.last_name
    ORDER BY
        attendance_count DESC;
END;
```

Radosław Rogalski

4.23 GetStudentsByStudyId

4.23.1 Description

Gets students by study id.

4.23.2 SQL

```
CREATE PROCEDURE GetStudentsByStudyId
    @StudyId INT
AS
BEGIN
    SELECT
        u.id AS user_id,
        u.first_name,
        u.last_name,
        us.study_id
    FROM
        user_study us
    INNER JOIN
        [user] u ON us.user_id = u.id
    INNER JOIN
        study s ON us.study_id = s.id
    WHERE
        us.study_id = @StudyId;
```

```
END;
```

Radosław Rogalski

4.24 ProcessPayment

4.24.1 Description

Pays for the order

4.24.2 SQL

```
ALTER PROCEDURE [dbo].[ProcessPayment]
    @OrderID INT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        -- Start a transaction
        BEGIN TRANSACTION;

        -- Declare variables
        DECLARE @PaymentID INT;
        DECLARE @StudentID INT;
        DECLARE @PaidAmount DECIMAL(18, 2);

        -- Get the next Payment ID
        SELECT @PaymentID = ISNULL(MAX(id), 0) + 1 FROM dbo.payment;

        -- Get the Student ID from the Order
        SELECT @StudentID = user_id
        FROM [order]
        WHERE id = @OrderID;

        -- Calculate the total Paid Amount for the Order
        SELECT @PaidAmount = SUM(od.price)
        FROM order_detail od
        INNER JOIN [order] o ON od.order_id = o.id
        WHERE o.id = @OrderID;

        -- Check if @PaidAmount is NULL (e.g., no matching records)
        IF @PaidAmount IS NULL
        BEGIN
            THROW 50000, 'Order not found or has no associated order details.', 1;
        END
```

```
        -- Insert payment record
        INSERT INTO payment (id, order_id, date, amount, status)
        VALUES (@PaymentID, @OrderID, GETDATE(), @PaidAmount, 'Paid');

        -- Commit the transaction
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        -- Rollback the transaction in case of error
        ROLLBACK TRANSACTION;

        -- Re-throw the error
        THROW;
    END CATCH
END
```

Robert Raniszewski

4.25 CreateConsent

4.25.1 Description

Creates the director consent.

4.25.2 SQL

```
CREATE PROCEDURE CreateConsent
    @order_detail_id INT,
    @text NVARCHAR(MAX),
    @date DATETIME
AS
BEGIN
    -- Insert a new record into the exception table
    INSERT INTO exception (id, order_detail_id, text, date)
    VALUES (
        (SELECT ISNULL(MAX(id), 0) + 1 FROM exception),
        @order_detail_id,
        @text,
        @date
    );
END;
```

Radosław Rogalski

5 Functions

5.1 Function GetStudentSchedule

5.1.1 Description

Funkcja zwraca harmonogram wszystkich zajęć dla usera

5.1.2 SQL:

```
CREATE FUNCTION GetStudentSchedule (@UserID INT)
RETURNS TABLE
AS
RETURN
(
    -- Zajęcia stacjonarne ze studiów
    SELECT
        sm.start_date AS MeetingDate,
        sm.duration AS Duration,
        sub.name AS SubjectName,
        'Stacjonarne' AS MeetingType,
        c.name AS LocationOrLink
    FROM
        study_attendance sa
    JOIN
        study_meeting sm ON sa.meeting_id = sm.id
    JOIN
        subject sub ON sm.subject_id = sub.id
    LEFT JOIN
        study_class_meeting scm ON scm.study_meeting_id = sm.id
    LEFT JOIN
        classroom c ON scm.classroom_id = c.id
    WHERE
        sa.user_id = @UserID

    UNION ALL

    -- Zajęcia online ze studiów
    SELECT
        sm.start_date AS MeetingDate,
        sm.duration AS Duration,
        sub.name AS SubjectName,
        'Online' AS MeetingType,
        som.url AS LocationOrLink
    FROM
        study_attendance sa
```

```
JOIN
    study_meeting sm ON sa.meeting_id = sm.id
JOIN
    subject sub ON sm.subject_id = sub.id
LEFT JOIN
    study_online_meeting som ON som.study_meeting_id = sm.id
WHERE
    sa.user_id = @UserID

UNION ALL

-- Zajęcia stacjonarne z kursów
SELECT
    cl.start_date AS MeetingDate,
    cl.duration AS Duration,
    c.name AS SubjectName,
    'Stacjonarne' AS MeetingType,
    mr.name AS LocationOrLink
FROM
    course_students cs
JOIN
    course c ON cs.course_id = c.id
JOIN
    course_lesson cl ON c.id = cl.course_id
LEFT JOIN
    course_stationary_lesson csl ON cl.id = csl.course_lesson_id
LEFT JOIN
    course_meeting_room mr ON csl.meeting_room_id = mr.id
WHERE
    cs.user_id = @UserID
    AND csl.course_lesson_id IS NOT NULL

UNION ALL

-- Zajęcia online z kursów
SELECT
    cl.start_date AS MeetingDate,
    cl.duration AS Duration,
    c.name AS SubjectName,
    'Online' AS MeetingType,
    col.meeting_url AS LocationOrLink
FROM
    course_students cs
JOIN
    course c ON cs.course_id = c.id
JOIN
```

```
        course_lesson cl ON c.id = cl.course_id
    LEFT JOIN
        course_online_lesson col ON cl.id = col.course_lesson_id
    WHERE
        cs.user_id = @UserID
        AND col.course_lesson_id IS NOT NULL

    UNION ALL

    -- Zajęcia z webinarów
    SELECT
        w.start_date AS MeetingDate,
        w.duration AS Duration,
        w.name AS SubjectName,
        'Online' AS MeetingType,
        w.meeting_link AS LocationOrLink
    FROM
        webinar_user wu
    JOIN
        webinar w ON wu.webinar_id = w.id
    WHERE
        wu.user_id = @UserID
);
```

Robert Raniszewski

5.2 Function get_user_role

5.2.1 Description

Get user role by id input

5.2.2 SQL

```
    CREATE FUNCTION get_user_role(@user_id INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        u.first_name,
        u.last_name,
        r.name AS role_name
    FROM [user] AS u
    JOIN user_role AS ur
        ON ur.user_id = u.id
    JOIN role AS r
```

```
        ON r.id = ur.role_id
    WHERE u.id = @user_id
);
```

Piotr Sękulski

5.3 Function CalculateFreeSpots

5.3.1 Description

Calculates the number of available spots in the desired study program.

5.3.2 SQL

```
CREATE FUNCTION CalculateFreeSpots (@StudyId INT)
RETURNS INT
AS
BEGIN
    DECLARE @FreeSpots INT;

    SELECT @FreeSpots = s.student_limit - COUNT(us.user_id)
    FROM study s
    LEFT JOIN user_study us ON s.id = us.study_id
    WHERE s.id = @StudyId
    GROUP BY s.student_limit;

    RETURN @FreeSpots;
END;
```

Radosław Rogalski

6 Triggers

6.1 trg_AfterPaymentInsert

6.1.1 Description

Trigger automatycznie dodaje uzytkownika na kurs, studia lub webinar po opłaceniu zamówienia

6.1.2 SQL

```
ALTER TRIGGER [dbo].[trg_AfterPaymentInsert]
ON [dbo].[payment]
AFTER INSERT
AS
BEGIN
```

```
SET NOCOUNT ON;

-- Deklaracje zmiennych
DECLARE @OrderID INT;
DECLARE @UserID INT;
DECLARE @DetailType NVARCHAR(MAX);
DECLARE @DetailID INT;

-- Pobranie danych z nowo dodanego rekordu
SELECT @OrderID = order_id
FROM inserted;

-- Pobranie user_id powiązanego z zamówieniem
SELECT @UserID = user_id
FROM [order]
WHERE id = @OrderID;

-- Iteracja po szczegółach zamówienia
DECLARE DetailCursor CURSOR FOR
SELECT
    COALESCE(oc.course_id, os.study_id, ow.webinar_id) AS DetailID,
    CASE
        WHEN oc.course_id IS NOT NULL THEN 'Course'
        WHEN os.study_id IS NOT NULL THEN 'Study'
        WHEN ow.webinar_id IS NOT NULL THEN 'Webinar'
        ELSE NULL
    END AS DetailType
FROM order_detail od
LEFT JOIN order_courses oc ON od.id = oc.order_detail_id
LEFT JOIN order_study os ON od.id = os.order_detail_id
LEFT JOIN order_webinars ow ON od.id = ow.order_detail_id
WHERE od.order_id = @OrderID;

OPEN DetailCursor;
FETCH NEXT FROM DetailCursor INTO @DetailID, @DetailType;

WHILE @@FETCH_STATUS = 0
BEGIN
    -- Logika zapisywania studenta w zależności od typu szczegółu
    IF @DetailType = 'Webinar'
    BEGIN
        -- Dodanie rekordu do tabeli webinar_user
        IF NOT EXISTS (SELECT 1 FROM webinar_user
        WHERE webinar_id = @DetailID AND user_id = @UserID)
        BEGIN
            INSERT INTO webinar_user (webinar_id, user_id)
```

```
                VALUES (@DetailID, @UserID);
            END
        END
        ELSE IF @DetailType = 'Course'
        BEGIN
            -- Dodanie rekordu do tabeli course_students
            IF NOT EXISTS (SELECT 1 FROM course_students
            WHERE course_id = @DetailID AND user_id = @UserID)
            BEGIN
                INSERT INTO course_students (course_id, user_id)
                VALUES (@DetailID, @UserID);
            END
        END
        ELSE IF @DetailType = 'Study'
        BEGIN
            -- Dodanie rekordu do tabeli user_study
            IF NOT EXISTS (SELECT 1 FROM user_study
            WHERE study_id = @DetailID AND user_id = @UserID)
            BEGIN
    DECLARE @UserStudyID INT;
    SELECT @UserStudyID = ISNULL(MAX(id), 0) + 1 FROM dbo.user_study;
                INSERT INTO user_study (id,study_id, user_id)
                VALUES (@UserStudyID ,@DetailID, @UserID);
            END
        END

        FETCH NEXT FROM DetailCursor INTO @DetailID, @DetailType;
    END

    CLOSE DetailCursor;
    DEALLOCATE DetailCursor;
END;
```

Robert Raniszewski

6.2 trg_translation_language_check

6.2.1 Description

Trigger sprawdza czy tłumacz zna język spotkania.

6.2.2 SQL

```
CREATE TRIGGER trg_translation_language_check
ON course_lesson_translation
AFTER INSERT, UPDATE
AS
```

```
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        WHERE NOT EXISTS (
            SELECT 1
            FROM translator t
            WHERE t.user_id = i.translator_id
              AND t.language_id = i.language_id
        )
    )
    BEGIN
        RAISERROR ('Translator does not know the specified language.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Radosław Rogalski

6.3 trg_coordinator_employee_check

6.3.1 Description

Trigger sprawdza czy koordynator jest pracownikiem.

6.3.2 SQL

```
CREATE TRIGGER trg_coordinator_employee_check
ON course
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted
        WHERE coordinator_id NOT IN (SELECT user_id FROM employee)
    )
    BEGIN
        RAISERROR ('Coordinator must be an employee.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Piotr Sękulski

6.4 trg_study_student_limit

6.4.1 Description

Sprawdza czy limit studentów został przekroczony.

6.4.2 SQL

```
CREATE TRIGGER trg_study_student_limit
ON user_study
AFTER INSERT
AS
BEGIN
    DECLARE @StudyID INT = (SELECT study_id FROM inserted);
    DECLARE @CurrentStudents INT = (SELECT COUNT(*) FROM user_study WHERE study_id = @Studyl
    DECLARE @StudentLimit INT = (SELECT student_limit FROM study WHERE id = @StudyID);

    IF @CurrentStudents > @StudentLimit
    BEGIN
        RAISERROR ('Study student limit exceeded.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Radosław Rogalski

7 Roles

7.1 Guest

```
grant select on webinar to guest;
grant select on course to guest;
grant select on study to guest;
grant execute on add_user to guest;
```

7.2 Student

```
grant select on webinar to student;
grant select on course to student;
grant select on study to student;
grant select on GetStudentSchedule to student;
grant execute on AddOrderDetail to student;
grant execute on ProcessPayment to student;
```

7.3 Translator_role

```
grant select, insert, delete,
```

```
update on study_meeting_translation to translator_role;
grant select, insert, delete,
update on webinar_translation to translator_role;
grant select, insert, delete,
update on course_lesson_translation to translator_role;
```

7.4 Manager (Dyrektor)

```
GRANT SELECT ON dbo.bilocation_report TO manager;
GRANT SELECT ON dbo.completed_studies TO manager;
GRANT SELECT ON dbo.course_modules_overview TO manager;
GRANT SELECT ON dbo.course_revenue TO manager;
GRANT SELECT ON dbo.future_course_lesson_enrollment TO manager;
GRANT SELECT ON dbo.general_revenue TO manager;
GRANT SELECT ON dbo.student_internship_status TO manager;
GRANT SELECT ON dbo.study_class_meeting_schedule TO manager;
GRANT SELECT ON dbo.study_online_meeting_schedule TO manager;
GRANT SELECT ON dbo.study_programme TO manager;
GRANT SELECT ON dbo.study_revenue TO manager;
GRANT SELECT ON dbo.unpaid_attendance TO manager;
GRANT SELECT ON dbo.unpaid_orders TO manager;
GRANT SELECT ON dbo.upcoming_activities TO manager;
GRANT SELECT ON dbo.upcoming_webinars TO manager;
GRANT SELECT ON dbo.user_roles TO manager;
GRANT SELECT ON dbo.vw_course_attendance TO manager;
GRANT SELECT ON dbo.vw_study_attendance TO manager;
GRANT SELECT ON dbo.vw_user_activity TO manager;
GRANT SELECT ON dbo.vw_webinar_attendance TO manager;
GRANT SELECT ON dbo.webinar_revenue TO manager;
GRANT EXECUTE ON dbo.add_city TO manager;
GRANT EXECUTE ON dbo.add_country TO manager;
GRANT EXECUTE ON dbo.add_employee TO manager;
GRANT EXECUTE ON dbo.add_language TO manager;
GRANT EXECUTE ON dbo.add_state TO manager;
GRANT EXECUTE ON dbo.add_translator TO manager;
GRANT EXECUTE ON dbo.add_translator_languages TO manager;
GRANT EXECUTE ON dbo.add_user TO manager;
GRANT EXECUTE ON dbo.add_webinar TO manager;
GRANT EXECUTE ON dbo.add_webinar_attendance TO manager;
GRANT EXECUTE ON dbo.add_webinar_translation TO manager;
GRANT EXECUTE ON dbo.AddOrderDetail TO manager;
GRANT EXECUTE ON dbo.AddSemester TO manager;
GRANT EXECUTE ON dbo.AddStudentAttendance TO manager;
GRANT EXECUTE ON dbo.AddStudy TO manager;
GRANT EXECUTE ON dbo.AddStudyOnlineMeeting TO manager;
GRANT EXECUTE ON dbo.AddSubject TO manager;
```

```
GRANT EXECUTE ON dbo.CreateConsent TO manager;
GRANT EXECUTE ON dbo.GetStudentsByStudyId TO manager;
GRANT EXECUTE ON dbo.GetStudyAttendanceByMeetingId TO manager;
GRANT EXECUTE ON dbo.GetStudyMeetings TO manager;
GRANT EXECUTE ON dbo.GetStudyProgram TO manager;
GRANT EXECUTE ON dbo.GetSubjectAttendanceReport TO manager;
GRANT EXECUTE ON dbo.ProcessPayment TO manager;
```

8 Generowanie danych

Do generowania danych wykorzystujemy program RedGate Data Generator w wersji trial. Jest on dostępny do pobrania pod adresem `https://www.red-gate.com/products/sql-data-generator/`. Uzupełniamy zestaw danych dla każdej kolumny. Dla każdej tabeli generujemy kilka tysięcy rekordów.

W celu wygenerowania danych testowych przy użyciu SQL Data Generator 4, najpierw łączymy się z bazą danych, wskazując docelowy serwer oraz schemat. Po nawiązaniu połączenia, program automatycznie wczytuje strukturę bazy, identyfikując tabele oraz ich kolumny. Następnie definiujemy reguły generowania danych dla poszczególnych kolumn, korzystając z dostępnych w programie predefiniowanych generatorów (np. liczby całkowite, tekst, daty) lub dostosowuję je według własnych potrzeb. Następnie dane są wstawiane do wybranych tabel w bazie za pomocą zautomatyzowanego procesu, który zachowuje relacje i warunki integralności danych, uwzględniając klucze obce i ograniczenia unikalności.

9 Indeksy

Indeksy dla kluczy głównych są dodwana automatycznie przez SQL Server podczas tworzenia tabeli. Należy dodać indeksy dla kluczy obcych:

```
CREATE INDEX idx_course_module_type_id ON course_module (course_module_type_id);
CREATE INDEX idx_course_id_course_module ON course_module (course_id);
CREATE INDEX idx_course_id_course_students ON course_students (course_id);
CREATE INDEX idx_course_id_course_teachers ON course_teachers (course_id);
CREATE INDEX idx_teacher_id_course_teachers ON course_teachers (teacher_id);
CREATE INDEX idx_course_id_order_courses ON order_courses (course_id);
CREATE INDEX idx_coordinator_id_course ON course (coordinator_id);
CREATE INDEX idx_order_detail_id_order_courses ON order_courses (order_detail_id);
CREATE INDEX idx_order_detail_id_order_study ON order_study (order_detail_id);
CREATE INDEX idx_order_detail_id_order_webinars ON order_webinars (order_detail_id);
CREATE INDEX idx_order_id_order_detail ON order_detail (order_id);
CREATE INDEX idx_study_id_order_study ON order_study (study_id);
CREATE INDEX idx_order_id_payment ON payment (order_id);
CREATE INDEX idx_state_id_city ON city (state_id);
CREATE INDEX idx_coordinator_id_webinar ON webinar (coordinator_id);
```

```
CREATE INDEX idx_user_id_course_attendance ON course_attendance (user_id);
CREATE INDEX idx_lesson_id_course_attendance ON course_attendance (lesson_id);
CREATE INDEX idx_module_id_course_lesson ON course_lesson (module_id);
CREATE INDEX idx_teacher_id_course_lesson ON course_lesson (teacher_id);
CREATE INDEX idx_course_lesson_id_translation ON course_lesson_translation (course_lesson_id
CREATE INDEX idx_translator_id_course_lesson_translation ON course_lesson_translation (trans
CREATE INDEX idx_language_id_course_lesson_translation ON course_lesson_translation (languag
CREATE INDEX idx_meeting_room_id_stationary_lesson ON course_stationary_lesson (meeting_room
CREATE INDEX idx_course_lesson_id_online_async_lesson ON course_online_async_lesson (course_
CREATE INDEX idx_course_lesson_id_online_lesson ON course_online_lesson (course_lesson_id);
CREATE INDEX idx_course_lesson_id_stationary_lesson ON course_stationary_lesson (course_less
CREATE INDEX idx_user_id_course_students ON course_students (user_id);
CREATE INDEX idx_user_id_employee ON employee (user_id);
CREATE INDEX idx_order_detail_id_exception ON exception (order_detail_id);
CREATE INDEX idx_study_id_internship ON internship (study_id);
CREATE INDEX idx_language_id_webinar_translation ON webinar_translation (language_id);
CREATE INDEX idx_order_detail_type_id_order_detail ON order_detail (order_detail_type_id);
CREATE INDEX idx_user_id_order ON "order" (user_id);
CREATE INDEX idx_webinar_id_order_webinars ON order_webinars (webinar_id);
CREATE INDEX idx_study_id_semester ON semester (study_id);
CREATE INDEX idx_country_id_state ON state (country_id);
CREATE INDEX idx_subject_id_student_grade ON student_grade (subject_id);
CREATE INDEX idx_user_id_student_grade ON student_grade (user_id);
CREATE INDEX idx_internship_id_student_presence ON internship_student_presence (internship_i
CREATE INDEX idx_user_id_internship_presence ON internship_student_presence (user_id);
CREATE INDEX idx_user_id_study_attendance ON study_attendance (user_id);
CREATE INDEX idx_classroom_id_study_class_meeting ON study_class_meeting (classroom_id);
CREATE INDEX idx_study_meeting_id_study_class_meeting ON study_class_meeting (study_meeting_
CREATE INDEX idx_coordinator_id_study ON study (coordinator_id);
CREATE INDEX idx_meeting_id_study_attendance ON study_attendance (meeting_id);
CREATE INDEX idx_subject_id_study_meeting ON study_meeting (subject_id);
CREATE INDEX idx_language_id_study_meeting_translation ON study_meeting_translation (languag
CREATE INDEX idx_study_meeting_id_translation ON study_meeting_translation (study_meeting_id
CREATE INDEX idx_translator_id_study_meeting_translation ON study_meeting_translation (trans
CREATE INDEX idx_study_meeting_id_online_meeting ON study_online_meeting (study_meeting_id);
CREATE INDEX idx_study_id_user_study ON user_study (study_id);
CREATE INDEX idx_employee_id_study_teacher ON study_teacher (employee_id);
CREATE INDEX idx_study_id_study_teacher ON study_teacher (study_id);
CREATE INDEX idx_user_id_user_study ON user_study (user_id);
CREATE INDEX idx_semester_id_subject ON subject (semester_id);
CREATE INDEX idx_supervisor_id_subject ON subject (supervisor_id);
CREATE INDEX idx_user_id_translator ON translator (user_id);
CREATE INDEX idx_language_id_translator ON translator (language_id);
CREATE INDEX idx_city_id_user ON "user" (city_id);
CREATE INDEX idx_country_id_user ON "user" (country_id);
CREATE INDEX idx_role_id_user_role ON user_role (role_id);
```

```
CREATE INDEX idx_user_id_user_role ON user_role (user_id);
CREATE INDEX idx_state_id_user ON "user" (state_id);
CREATE INDEX idx_user_id_webinar_attendance ON webinar_attendance (user_id);
CREATE INDEX idx_webinar_id_webinar_attendance ON webinar_attendance (webinar_id);
CREATE INDEX idx_webinar_id_webinar_user ON webinar_user (webinar_id);
CREATE INDEX idx_user_id_webinar_user ON webinar_user (user_id);
CREATE INDEX idx_translator_id_webinar_translation ON webinar_translation (translator_id);
CREATE INDEX idx_webinar_id_webinar_translation ON webinar_translation (webinar_id);
```

Tabela study_attendance ma najwięcej rekordów dlatego na niej będziemy sprawdzać działanie indeksów. Wykonujemy polecenie:

```
SELECT * FROM study_attendance
WHERE user_id = 902
```

Następnie dodajemy indeks:

```
CREATE INDEX study_attendance__user_id ON study_attendance (user_id);
```

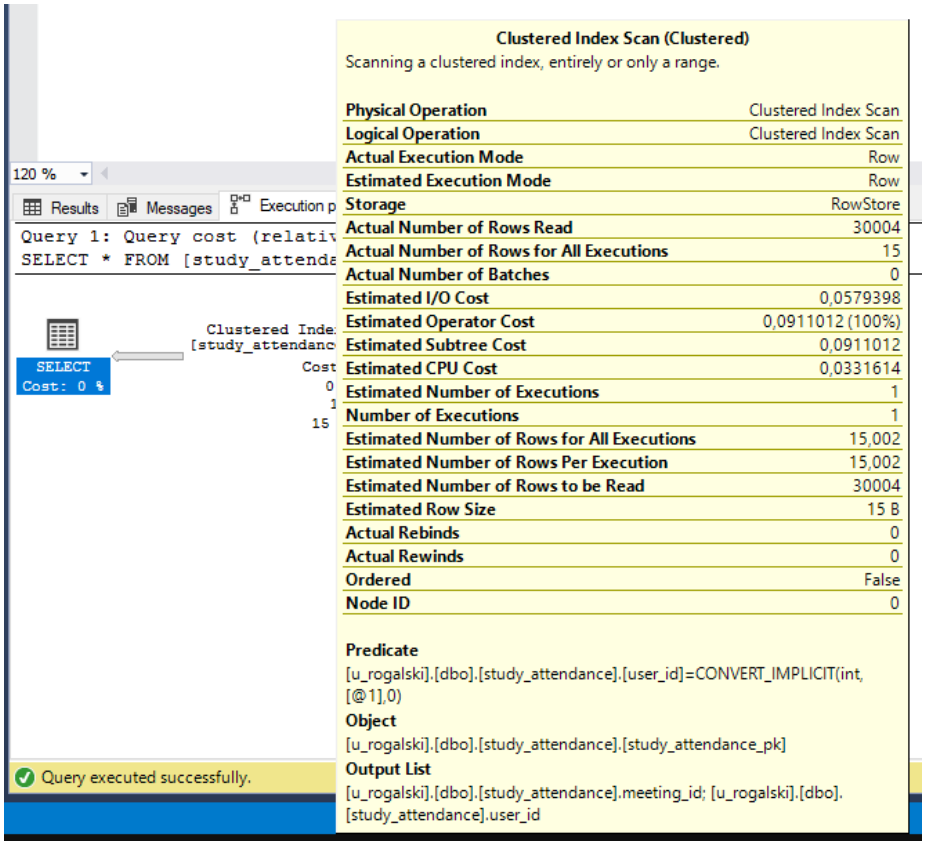


Figure 3: Wynik przed dodaniem indeksu.

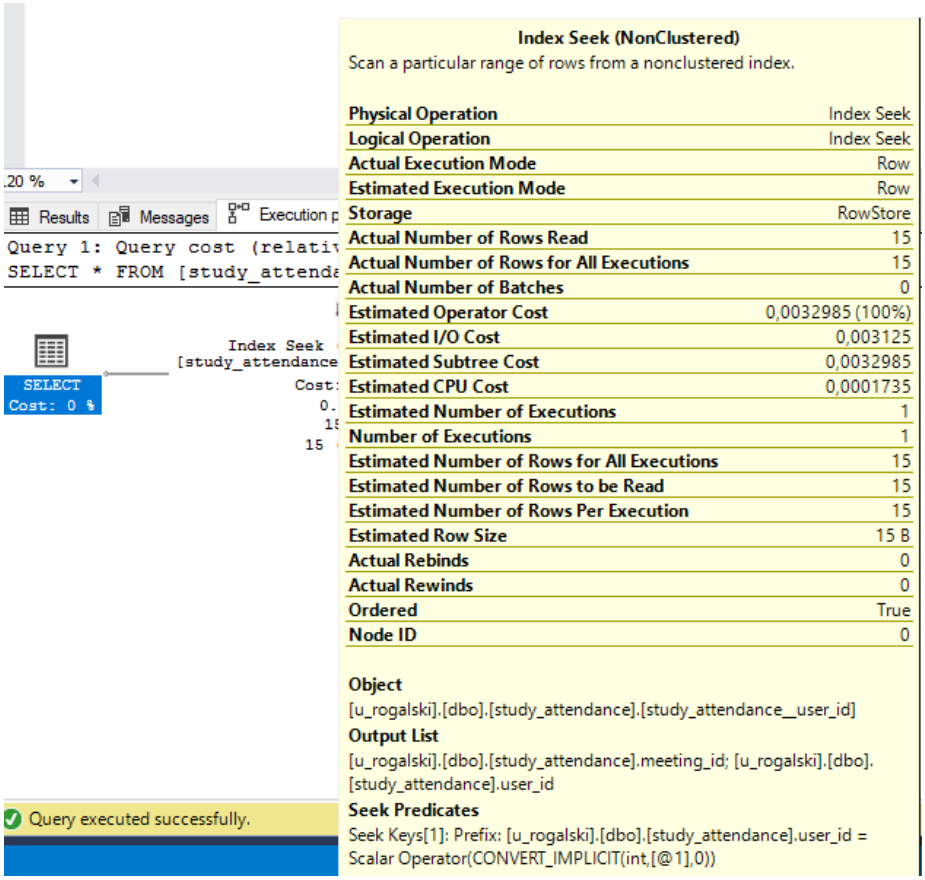
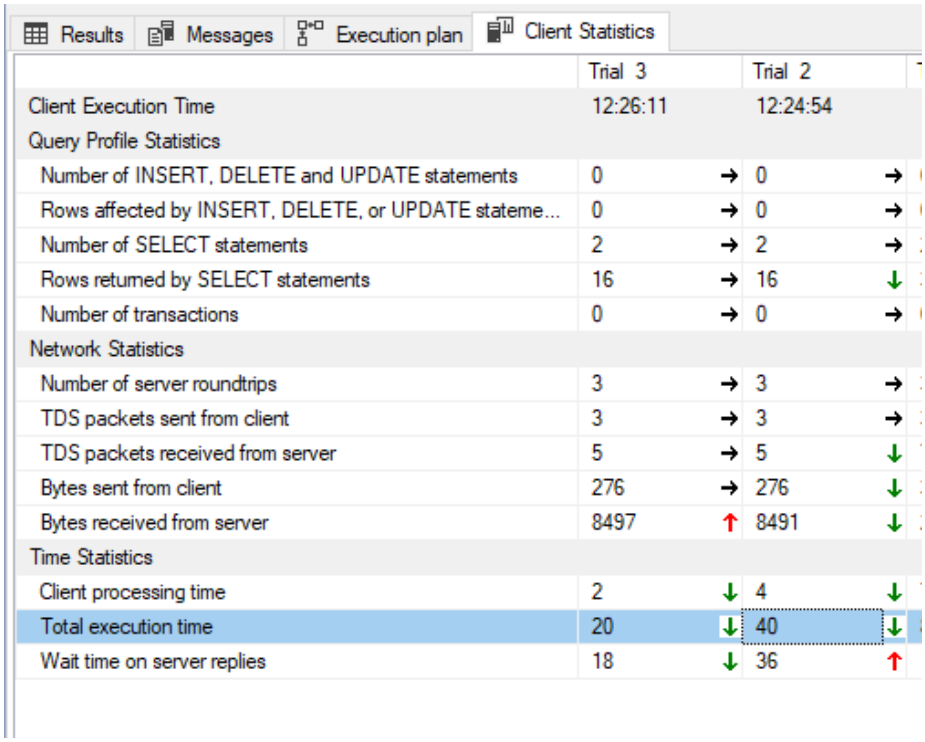


Figure 4: Wynik po dodaniu indeksu.

Zdjęcie nr 5 przedstawia porównanie czasów wykonania poleceń przed dodaniem indeksu (kolumna "Trial 2") oraz po dodaniu indeksu (kolumna "Trial 3").



	Trial 3	Trial 2
Client Execution Time	12:26:11	12:24:54
Query Profile Statistics		
Number of INSERT, DELETE and UPDATE statements	0 →	0 →
Rows affected by INSERT, DELETE, or UPDATE stateme...	0 →	0 →
Number of SELECT statements	2 →	2 →
Rows returned by SELECT statements	16 →	16 ↓
Number of transactions	0 →	0 →
Network Statistics		
Number of server roundtrips	3 →	3 →
TDS packets sent from client	3 →	3 →
TDS packets received from server	5 →	5 ↓
Bytes sent from client	276 →	276 ↓
Bytes received from server	8497 ↑	8491 ↓
Time Statistics		
Client processing time	2 ↓	4 ↓
Total execution time	20 ↓	40 ↓
Wait time on server replies	18 ↓	36 ↑

Figure 5: Porównanie czasów przed i po.