Podstawy baz danych
PROJEKT
"Kursy i szkolenia"
2023/2024

Stworzone przez: Przemysław Popowski oraz Szymon Kubiczek.

Spis treści

Widoki

Procedury

Funkcje

Opis funkcji systemu

Typy użytkowników:

-Podstawowy (osoba korzystająca z usług dydaktycznych serwisu)
-Nauczyciel (uogólnione pojęcie osoby, która wykłada kursy, webinary lub lekcje)
-Tłumacz (osoba z zewnątrz tłumacząca pojedyncze formy nauczania na zlecenie)
-Księgowy (osoba nadzorująca płatności; może nakładać kary pieniężne)
-Manager (administrator systemu)

Funkcje użytkowników:

-*Podstawowy*: Może rozpocząć studia, wziąć udział w webinarach, kursach i pojedynczych wykupionych lekcjach, zobaczyć historię wydarzeń w których brał udział (oraz zobaczyć te, które ukończył z powodzeniem), przeglądać nadchodzące wydarzenia, zobaczyć listę certyfikatów za ukończone wydarzenia, zobaczyć swój plan lekcji (w przypadku rozpoczętych studiów), zobaczyć kalendarz z wydarzeniami, które zostały opłacone. Może opłacić dane rzeczy poprzez zewnętrzny serwis płatnościowy.

-*Nauczyciel*: Może zobaczyć kalendarz z prowadzonymi przez siebie kursami, webinarami, czy lekcjami, może wystawić oceny końcowe, może zobaczyć dane o wykładanej formie nauki i jej tłumaczeniu (jeśli to konieczne), może wystawiać obecność, może zdecydować o przydzieleniu certyfikatu, poprosić o tłumacza dla swoich zajęć. Może zobaczyć statystyki obecności na swoim przedmiocie.

-*Tłumacz:* Może zobaczyć kalendarz z tłumaczonymi przez siebie kursami, webinarami, otrzymywać propozycje tłumaczenia danych form nauczania i wyrażać na nie zgodę.

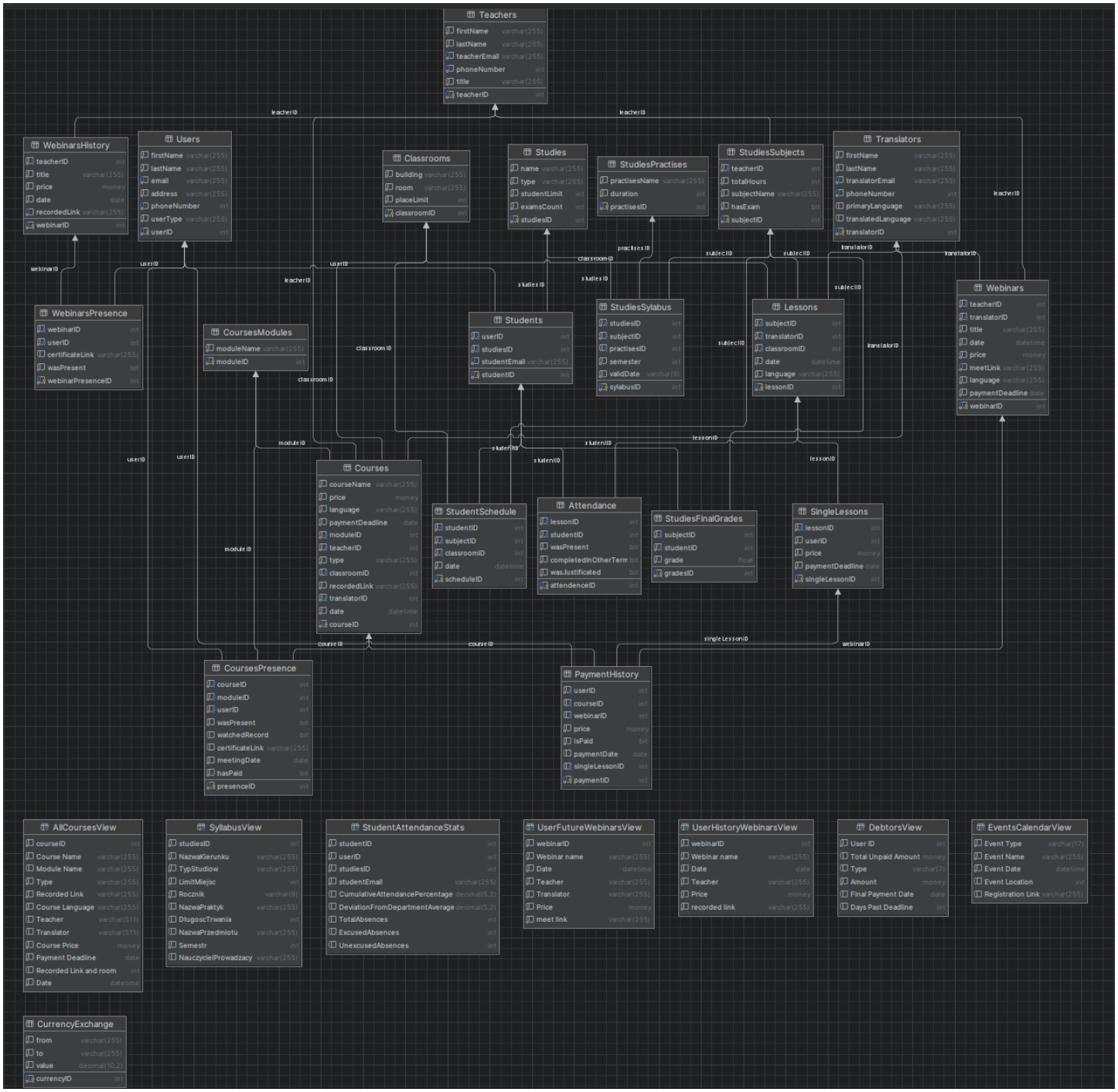
-*Księgowy*: Ma podgląd do historii płatności, listy niezapłaconych rzeczy, statystyk finansowych i ma możliwość generowania automatycznych raportów. Może narzucić kary pieniężne dla użytkowników nie spełniających regulaminu opłat.

-*Manager:* Ma podgląd do listy pracowników i ich danych komunikacyjnych, Widzi kalendarz wszystkich wydarzeń które już się odbyły, tak jak i tych przyszłych, może dodawać i usuwać pracowników i nadawać im określone uprawnienia. Może wysyłać automatyczne maile do wybranych pracowników/ wybranego typu pracowników. Może zobaczyć statystyki obecności wszystkich studentów danego kierunku.

Funkcje systemowe:

-Wyświetlanie nadchodzących wydarzeń określonym typom użytkowników
-Wyświetlanie historii wydarzeń
-Generowanie raportów finansowych
-Generowanie raportów obecności studentów na danych przedmiotach
-Wyliczanie salda użytkownika

Schemat bazy danych


Teachers
firstName varchar(255)
lastName varchar(255)
teacherEmail varchar(255)
phoneNumber int
title varchar(255)
teacherID int
WebinarsHistory
teacherID int
title varchar(255)
price money
date date
recordedLink varchar(255)
webinarID int
Users
firstName varchar(255)
lastName varchar(255)
email varchar(255)
address varchar(255)
phoneNumber int
userType varchar(255)
userID int
Classrooms
building varchar(255)
room varchar(255)
placeLimit int
classroomID int
Studies
name varchar(255)
type varchar(255)
studentLimit int
examsCount int
studiesID int
StudiesPractises
practisesName varchar(255)
duration int
practisesID int
StudiesSubjects
teacherID int
totalHours int
subjectName varchar(255)
hasExam bit
subjectID int
Translators
firstName varchar(255)
lastName varchar(255)
translatorEmail varchar(255)
phoneNumber int
primaryLanguage varchar(255)
translatedLanguage varchar(255)
translatorID int
WebinarsPresence
webinarID int
userID int
certificateLink varchar(255)
wasPresent bit
webinarPresenceID int
CoursesModules
moduleName varchar(255)
moduleID int
Students
userID int
studiesID int
studentEmail varchar(255)
studentID int
StudiesSyllabus
studiesID int
subjectID int
practisesID int
semester int
validDate varchar(9)
syllabusID int
Lessons
subjectID int
translatorID int
classroomID int
date datetime
language varchar(255)
lessonID int
Webinars
teacherID int
translatorID int
title varchar(255)
date datetime
price money
meetLink varchar(255)
language varchar(255)
paymentDeadline date
webinarID int
Courses
courseName varchar(255)
price money
language varchar(255)
paymentDeadline date
moduleID int
teacherID int
type varchar(255)
classroomID int
recordedLink varchar(255)
translatorID int
date datetime
courseID int
StudentSchedule
studentID int
subjectID int
classroomID int
date datetime
scheduleID int
Attendance
lessonID int
studentID int
wasPresent bit
completedInOtherTerm bit
wasJustificated bit
attendenceID int
StudiesFinalGrades
subjectID int
studentID int
grade float
gradesID int
SingleLessons
lessonID int
userID int
price money
paymentDeadline date
singleLessonID int
CoursesPresence
courseID int
moduleID int
userID int
wasPresent bit
watchedRecord bit
certificateLink varchar(255)
meetingDate date
hasPaid bit
presenceID int
PaymentHistory
userID int
courseID int
webinarID int
price money
isPaid bit
paymentDate date
singleLessonID int
paymentID int
AllCoursesView
courseID int
Course Name varchar(255)
Module Name varchar(255)
Type varchar(255)
Recorded Link varchar(255)
Course Language varchar(255)
Teacher varchar(511)
Translator varchar(511)
Course Price money
Payment Deadline date
Recorded Link and room int
Date datetime
SyllabusView
studiesID int
NazwaKierunku varchar(255)
TypStudiow varchar(255)
LimitMiejsc int
Rocznik varchar(9)
NazwaPraktyk varchar(255)
DlugoscTrwania int
NazwaPrzedmiotu varchar(255)
Semestr int
NauczycielProwadzacy varchar(255)
StudentAttendanceStats
studentID int
userID int
studiesID int
studentEmail varchar(255)
CumulativeAttendancePercentage decimal(5,2)
DeviationFromDepartmentAverage decimal(5,2)
TotalAbsences int
ExcusedAbsences int
UnexcusedAbsences int
UserFutureWebinarsView
webinarID int
Webinar name varchar(255)
Date datetime
Teacher varchar(255)
Translator varchar(255)
Price money
meet link varchar(255)
UserHistoryWebinarsView
webinarID int
Webinar name varchar(255)
Date date
Teacher varchar(255)
Price money
recorded link varchar(255)
DebtorsView
User ID int
Total Unpaid Amount money
Type varchar(7)
Amount money
Final Payment Date date
Days Past Deadline int
EventsCalendarView
Event Type varchar(17)
Event Name varchar(255)
Event Date datetime
Event Location int
Registration Link varchar(255)
CurrencyExchange
from varchar(255)
to varchar(255)
value decimal(10,2)
currencyID int

Opisy tabel

Tabela Users - reprezentuje użytkowników serwisu w naszej bazie danych
Klucz główny: userID
Imię: firstName
Nazwisko: lastName
Adres e-mail: email
Adres zamieszkania: address
Numer telefonu: phoneNumber
Typ użytkownika w systemie: userType
Warunki integralnościowe:

- *Adres email jest unikalny oraz musi zawierać "@"*

```
    CONSTRAINT ValidEmail CHECK (Email LIKE '%@%'),
    CONSTRAINT UniqueEmail UNIQUE (Email),
```

- *Numer telefonu jest unikalny i składa się z 9 cyfr z zakresu 0-9.*

```
- CONSTRAINT ValidPhoneNumber CHECK (phoneNumber LIKE
  '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
- CONSTRAINT UniquePhoneNumber UNIQUE (phoneNumber),
```

```
create table dbo.Users
(
   userID      int                          not null,
   firstName   varchar(255)                 not null,
   lastName    varchar(255)                 not null,
   email       varchar(255)                 not null,
   address     varchar(255)                 not null,
   phoneNumber int                          not null,
   userType    varchar(255) default 'user' not null,
     constraint Users_pk primary key (userID),
     constraint UniqueEmail unique (email),
     constraint ValidEmail check ([email] like '%@%'),
     constraint UniquePhoneNumber unique (phoneNumber),
   - CONSTRAINT ValidPhoneNumber CHECK (phoneNumber LIKE '[0-9][0-9][0-
     9][0-9][0-9][0-9][0-9][0-9][0-9]')
)
```

Tabela Students - reprezentuje studentów serwisu w naszej bazie danych
Klucz główny: studentID
Klucze obce: userID, studiesID
Studencki adres e-mail: studentEmail
Warunki integralnościowe:

- *Adres email jest unikalny oraz kończy się na "@student.projekt.edu.pl"*

```
- CONSTRAINT validStudentEmail CHECK (studentEmail
  LIKE '%@student.projekt.edu.pl'),
- CONSTRAINT uniqueStudentEmail UNIQUE (studentEmail),
```

```
create table dbo.Students
(
   studentID    int          not null
       constraint Students_pk
           primary key,
   userID       int          not null
       constraint Students_Users
           references dbo.Users,
   studiesID    int          not null
       constraint Students_Studies
           references dbo.Studies,
   studentEmail varchar(255) not null
       constraint uniqueStudentEmail
           unique
       constraint validStudentEmail
           check ([studentEmail] like
'%@student.projekt.edu.pl')
)
```

Tabela Webinars - reprezentuje webinaria w naszej bazie danych

Klucz główny: webinarID
Klucze obce: teacherID, translatorID
Tytuł webinaru: title
Data odbycia się: date
Cena: price
Link do spotkania: meetLink
Język w którym się odbędzie: language
Ostateczna data opłacenia uczestnictwa: paymentDeadline
Warunki integralnościowe:

- *Data odbycia się webinaru jest w formacie RRRR-MM-DD oraz GG:MM.*

```
constraint check_date
   check CONVERT([varchar],[date],120)=[date]
```

- *Cena musi zawierać wyłącznie liczbę z zaokrągleniem do dwóch miejsc po przecinku.*

```
constraint check_price
   check (price>=0 AND price%0.01=0)
```

- *Link jest unikalny dla każdego webinaru.*

```
constraint uniqueLink
   unique (meetLink)
```

```
create table dbo.Webinars
(
   webinarID       int          not null
       constraint Webinars_pk
           primary key,
   teacherID       int          not null
       constraint Webinars_Teachers
           references dbo.Teachers,
   translatorID    int          not null
       constraint Webinars_Translators
           references dbo.Translators,
   title           varchar(255) not null,
   date            datetime     not null
       constraint check_date
           check (CONVERT([varchar], [date], 120) = [date]),
   price           money        not null
       constraint check_price
           check ([price] >= 0 AND [price] % 0.01 = 0),
   meetLink        varchar(255) not null
       constraint uniqueLink
           unique,
   language        varchar(255) not null,
   paymentDeadline date         not null
)
```

Tabela WebinarsPresence - zapisuje informacje o uczestnictwie użytkowników w webinarach.
Klucz główny: webinarPresenceID
Klucze obce: webinarID, userID
Link do certyfikatu: certificateLink
Obecność użytkownika: wasPresent

```
create table dbo.WebinarsPresence
(
    webinarPresenceID int not null
        constraint WebinarsPresence_pk
            primary key,
    webinarID         int not null
        constraint WebinarsPresence_Webinars_webinarID_fk
            references dbo.Webinars,
    userID            int not null
        constraint WebinarsPresence_Users_userID_fk
            references dbo.Users,
    certificateLink   varchar(255),
    wasPresent        bit not null
)
```

Tabela WebinarsHistory - reprezentuje w naszej bazie danych historię odbytych webinarów
Klucz główny: webinarID
Klucz obcy: teacherID
Nazwa webinaru: title
Cena: price
Data odbycia się: date
Link do nagrania: recordedLink
Warunki integralnościowe:

- *Data kiedy odbył się webinaru jest w formacie RRRR-MM-DD oraz GG:MM.*

```
constraint check_WebinarHistoryDate
   check (CONVERT([varchar],[date],120)=[date])
```

- *Cena musi zawierać wyłącznie liczbę z zaokrągleniem do dwóch miejsc po przecinku.*

```
constraint check_WebinarHistoryPrice
   check (price>=0 AND price%0.01=0)
```

- *Link jest unikalny dla każdego webinaru.*

```
constraint uniqueWebinarHistoryLink
   unique (recordedLink)
```

```
create table dbo.WebinarsHistory
(
   webinarID    int          not null
       constraint WebinarsHistory_pk
           primary key
       constraint WebinarsHistory_Webinars
           references dbo.Webinars,
   teacherID    int          not null
       constraint WebinarsHistory_Teachers
           references dbo.Teachers,
   title        varchar(255) not null,
   price        money        not null
       constraint check_WebinarHistoryPrice
           check ([price] >= 0 AND [price] % 0.01 = 0),
   date         date         not null
       constraint check_WebinarHistoryDate
           check (CONVERT([varchar], [date], 120) = [date]),
   recordedLink varchar(255) not null
       constraint uniqueWebinarHistoryLink
           unique
)
```

Tabela Teachers - reprezentuje nauczycieli w naszej bazie danych
Klucz główny: teacherID
Imię: firstName
Nazwisko: lastName
Nauczycielski e-mail: teacherEmail
Numer telefonu: phoneNumber
Tytuł zawodowy: title
Warunki integralnościowe:

- *Adres email jest unikalny oraz kończy się na "@teacher.projekt.edu.pl"*

```
- CONSTRAINT validTeacherEmail CHECK (teacherEmail LIKE
  '%@teacher.projekt.edu.pl'),
- CONSTRAINT uniqueTeacherEmail UNIQUE (teacherEmail),
```

- *Numer telefonu jest unikalny i składa się z 9 cyfr z zakresu 0-9.*

```
- CONSTRAINT ValidTeacherPhoneNumber CHECK (phoneNumber
  LIKE '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
- CONSTRAINT UniqueTeacherPhoneNumber UNIQUE
  (phoneNumber),
```

```
create table dbo.Teachers
(
   teacherID    int          not null
       constraint Teachers_pk
           primary key,
   firstName    varchar(255) not null,
   lastName     varchar(255) not null,
   teacherEmail varchar(255) not null
       constraint uniqueTeacherEmail
           unique
       constraint validTeacherEmail
           check ([teacherEmail] like '%@teacher.projekt.edu.pl'),
   phoneNumber  int          not null
       constraint uniqueTeacherPhoneNumber
           unique
       constraint validTeacherPhoneNumber
           check ([phoneNumber] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9][0-9]'),
   title        varchar(255) not null
)
```

Tabela Translators - reprezentuje osoby zajmujące się tłumaczeniem zajęć w naszej bazie danych

Klucz główny: translatorID
Imię: firstName
Nazwisko: lastName
E-mail tłumacza: translatorEmail
Numer telefonu: phoneNumber
Język główny: primaryLanguage
Język, na który tłumaczy: translatedLanguage
Warunki integralnościowe:

- *Adres email jest unikalny oraz kończy się na "@translator.projekt.edu.pl"*

```
- CONSTRAINT validTranslatorEmail CHECK (translatorEmail LIKE
  '%@translator.projekt.edu.pl'),
- CONSTRAINT uniqueTranslatorEmail UNIQUE (translatorEmail),
```

- *Numer telefonu jest unikalny i składa się z 9 cyfr z zakresu 0-9.*

```
- CONSTRAINT ValidTranslatorPhoneNumber CHECK (phoneNumber LIKE
  '[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
- CONSTRAINT UniqueTranslatorPhoneNumber UNIQUE (phoneNumber),
```

```
create table dbo.Translators
(
   translatorID       int          not null
       constraint Translators_pk
           primary key,
   firstName          varchar(255) not null,
   lastName           varchar(255) not null,
   translatorEmail    varchar(255) not null
       constraint uniqueTranslatorEmail
           unique
       constraint validTranslatorEmail
           check ([translatorEmail] like '%@translator.projekt.edu.pl'),
   phoneNumber        int          not null
       constraint uniqueTranslatorPhoneNumber
           unique
       constraint validTranslatorPhoneNumber
           check ([phoneNumber] like '[0-9][0-9][0-9][0-9][0-9][0-9][0-
9][0-9][0-9]'),
   primaryLanguage    varchar(255),
   translatedLanguage varchar(255)
)
```

Tabela Studies - reprezentuje kierunki studiów w naszej bazie danych
Klucz główny: studiesID
Nazwa kierunku: name
Typ studiowania: type
Limit studentów: studentLimit
Liczba egzaminów: examsCount
Warunki integralnościowe:

- *Typ studiów może być tylko: "stacjonarne", "niestacjonarne" lub "hybrydowe".*

```
constraint checkStudiesType
   check (type IN
('stacjonarne','niestacjonarne','hybrydowe'))
```

```
create table dbo.Studies
(
   studiesID    int           not null
       constraint Studies_pk
           primary key,
   name         varchar(255)  not null,
   type         varchar(255)  not null
       constraint checkStudiesType
           check ([type] = 'hybrydowe' OR [type] =
'niestacjonarne' OR [type] = 'stacjonarne'),
   studentLimit int           not null,
   examsCount   int default 0 not null
)
```

Tabela StudiesPractises - reprezentuje obowiązkowe praktyki na studiach w naszej bazie danych

Klucz główny: practisesID

Nazwa praktyk: practisesName

Długość trwania: duration

```
create table dbo.StudiesPractises
(
   practisesID   int          not null
       constraint StudiesPractises_pk
           primary key,
   practisesName varchar(255) not null,
   duration      int          not null
)
```

Tabela StudiesSylabus - reprezentuje sylabus kierunków na studiach w naszej bazie danych
Klucz główny: sylabusID
Klucze obce: studiesID, subjectID, practisesID
Numer semestru: semester
Rok od którego do którego obowiązuje dany sylabus: validDate
Warunki integralnościowe:

- *Semestr jest liczbą naturalną z zakresu od 1 do 10.*

```
constraint validSemester
   check (semester BETWEEN 1 AND 10)
```

- *Rocznik jest w formacie XXXX/YYYY, gdzie XXXX to rok rozpoczęcia studiów, natomiast YYYY to rok zakończenia.*

```
constraint check_validDate
   check (validDate LIKE '[1-2][0-9][0-9][0-9]/[1-
2][0-9][0-9][0-9]')
```

```
create table dbo.StudiesSylabus
(
   studiesID   int        not null
       constraint StudiesSylabus_Studies
           references dbo.Studies,
   subjectID   int        not null
       constraint StudiesSylabus_StudiesSubjects
           references dbo.StudiesSubjects,
   practisesID int
       constraint StudiesSylabus_StudiesPractices
           references dbo.StudiesPractises,
   semester    int        not null
       constraint validSemester
           check ([semester] >= 1 AND [semester] <= 10),
   validDate   varchar(9) not null
       constraint check_validDate
           check ([validDate] like '[1-2][0-9][0-9][0-9]/[1-
2][0-9][0-9][0-9]'),
   sylabusID   int        not null
       constraint StudiesSylabus_pk
           primary key
)
```

Tabela StudiesSubjects - reprezentuje przedmioty na studiach w naszej bazie danych

Klucz główny: subjectID

Klucz obcy: teacherID

Ilość godzin trwania przedmiotu: totalHours

Nazwa przedmiotu: subjectName

Czy posiada egzamin: hasExam

```
create table dbo.StudiesSubjects
(
   subjectID   int           not null
       constraint StudiesSubjects_pk
           primary key,
   teacherID   int           not null
       constraint StudiesSubjects_Teachers
           references dbo.Teachers,
   totalHours  int           not null,
   subjectName varchar(255)  not null,
   hasExam     bit default 0 not null
)
```

Tabela StudiesFinalGrades - reprezentuje oceny końcowe danych studentów z danych przedmiotów.

Klucz główny: gradesID

Klucze obce: subjectID, studentID

Ocena studenta z przedmiotu: grade

Warunki integralnościowe:

- *Grade jest oceną, która jest liczbą ze zbioru {2, 3, 3.5, 4, 4.5, 5}*

```
constraint checkGrade
   check ([grade] = 5 OR [grade] = 4.5 OR [grade] = 4 OR
[grade] = 3.5 OR [grade] = 3 OR [grade] = 2)
```

```
create table dbo.StudiesFinalGrades
(
   gradesID  int   not null
       constraint StudiesFinalGrades_pk
           primary key,
   subjectID int   not null
       constraint
StudiesFinalGrades_StudiesSubjects_subjectID_fk
           references dbo.StudiesSubjects,
   studentID int   not null
       constraint
StudiesFinalGrades_Students_studentID_fk
           references dbo.Students,
   grade     float not null
       constraint checkGrade
           check ([grade] = 5 OR [grade] = 4.5 OR
[grade] = 4 OR [grade] = 3.5 OR [grade] = 3 OR [grade] =
2)
)
```

Tabela StudentSchedule - reprezentuje plan zajęć studenta w naszej bazie danych

Klucz główny: scheduleID

Klucze obce: studentID, subjectID, classroomID

Termin odbycia się zajęć: date

Warunki integralnościowe:

- *Data zajęć jest w formacie RRRR-MM-DD, GG-MM.*

```
constraint validScheduleDate
   check (CONVERT([varchar],[date],120)=[date])
```

```
create table dbo.StudentSchedule
(
   studentID   int      not null
       constraint
StudentSchedule_Students_studentID_fk
           references dbo.Students,
   subjectID   int      not null
       constraint StudentSchedule_StudiesSubjects
           references dbo.StudiesSubjects,
   classroomID int      not null
       constraint StudentSchedule_Classrooms
           references dbo.Classrooms,
   date        datetime not null
       constraint validScheduleDate
           check (CONVERT([varchar], [date], 120) =
[date]),
   scheduleID  int      not null
       constraint StudentSchedule_pk
           primary key
)
```

Tabela Attendance - reprezentuje obecność studentów na zajęciach w naszej bazie danych
Klucz główny: attendenceID
Klucze obce: lessonID, studentID
Czy student był obecny: wasPresent
Czy student odrobił zajęcia: completedInOtherTerm
Czy student został usprawiedliwiony: wasJustificated

```
create table dbo.Attendance
(
   lessonID             int not null
       constraint Attendance_Lessons
           references dbo.Lessons,
   studentID            int not null
       constraint Attendance_Students
           references dbo.Students,
   wasPresent           bit not null,
   completedInOtherTerm bit not null,
   wasJustificated      bit not null,
   attendenceID         int not null
       constraint Attendance_pk
           primary key
)
```

Tabela Courses - reprezentuje pojedyńcze kursy w naszej bazie danych.
Klucz główny: courseID
Klucze obce: moduleID, teacherID, classroomID, translatorID
Nazwa kursu: courseName
Cena kursu: price
Język w którym się odbywa: language
Ostateczna data opłacenia: paymentDeadline
Typ odbywania się kursu: type
Link do nagrania: recordedLink
Data obywania się kursu: date
Warunki integralnościowe:

- *Data zajęć jest w formacie RRRR-MM-DD, GG-MM.*

```
constraint validCourseDate
   check (CONVERT([varchar],[date],120)=[date])
```

- *Typ kursu może być tylko: “stacjonarne”, “on-line synchroniczne”, “on-line asynchroniczne” lub “hybrydowe”.*

```
constraint checkCourseType
   check (type IN ('stacjonarne','on-line asynchroniczne','on-line
synchroniczne','hybrydowe'))
```

```
create table dbo.Courses
(
   courseID        int          not null
       constraint Courses_pk
           primary key,
   courseName      varchar(255) not null,
   price           money        not null,
   language        varchar(255) not null,
   paymentDeadline date         not null,
   moduleID        int          not null
       constraint Courses_CoursesModules_moduleID_fk
           references dbo.CoursesModules,
   teacherID       int          not null
       constraint Courses_Teachers_teacherID_fk
          references dbo.Teachers,
   type            varchar(255) not null,
   classroomID     int
       constraint Courses_Classrooms_classroomID_fk
          references dbo.Classrooms,
   recordedLink    varchar(255) not null,
   translatorID    int          not null
       constraint Courses_Translators_translatorID_fk
          references dbo.Translators,
   date            datetime     not null
)
```

Tabela CoursesPresence - przetrzymuje dane o obecności użytkownika na kursie.
Klucz główny: presenceID
Klucze obce: courseID, moduleID, userID
Czy użytkownik był obecny: wasPresent
Czy użytkownik obejrzał nagranie: watchedRecord
Link do certyfikatu: certificateLink
Data spotkania: meetingDate
Czy użytkownik zapłacił za kurs: hasPaid

```
create table dbo.CoursesPresence
(
   presenceID      int  not null
       constraint CoursesPresence_pk
           primary key,
   courseID        int  not null
       constraint CoursesPresence_Courses
           references dbo.Courses,
   moduleID        int  not null
       constraint CoursesPresence_CoursesModules
           references dbo.CoursesModules,
   userID          int  not null
       constraint CoursesPresence_Users
           references dbo.Users,
   wasPresent      bit  not null,
   watchedRecord   bit,
   certificateLink varchar(255),
   meetingDate     date not null,
   hasPaid         bit  not null
)
```

Tabela CoursesModules - posiada dane o obecności użytkownika na kursie.
Klucz główny: moduleID
Nazwa modułu: moduleName

```
create table dbo.CoursesModules
(
   moduleID   int          not null
       constraint CoursesModules_pk
           primary key,
   moduleName varchar(255) not null
)
```

Tabela SingleLessons - posiada dane o pojedynczych wykupionych lekcjach.
Klucz główny: singleLessonID
Klucze obce: lessonID, userID
Cena za lekcję: price
Ostateczna data opłacenia: paymentDeadline

```
create table dbo.SingleLessons
(
   lessonID        int   not null
       constraint SingleLessons_Lessons_lessonID_fk
           references dbo.Lessons,
   userID          int   not null,
   price           money not null,
   paymentDeadline date  not null,
   singleLessonID  int   not null
       constraint SingleLessons_pk
           primary key
)
```

Tabela PaymentHistory - reprezentuje historię płatności za oferowane usługi w naszej bazie danych

Klucz główny: paymentID

Klucze obce: singleLessonID, userID, courseID, webinarID

Koszt: price

Czy zostało opłacone: isPaid

Data opłacenia: paymentDate

Warunki integralnościowe:

-zawsze jeden z kluczów obcych powinien być różny od null

```
constraint checkAtLeastOneNotNull
   check ((singleLessonID IS NOT NULL) OR
   (courseID IS NOT NULL) OR
   (webinarID IS NOT NULL))
```

```
create table dbo.PaymentHistory
(
   paymentID      int   not null
       constraint PaymentHistory_pk
           primary key,
   userID         int   not null
       constraint PaymentHistory_Users
           references dbo.Users,
   courseID       int
       constraint PaymentHistory_Courses
           references dbo.Courses,
   webinarID      int
       constraint PaymentHistory_Webinars
           references dbo.Webinars,
   price          money not null,
   isPaid         bit   not null,
   paymentDate    date,
   singleLessonID int
       constraint PaymentHistory_SingleLessons_singleLessonID_fk
           references dbo.SingleLessons,
   constraint checkAtLeastOneNotNull
       check ([singleLessonID] IS NOT NULL OR [courseID] IS NOT NULL OR
[webinarID] IS NOT NULL)
)
```

Tabela Lessons - reprezentuje lekcje w naszej bazie danych

Klucz główny: lessonID

Klucze obce: subjectID, translatorID, classroomID

Data zajęć: date

Język, w którym się odbywają: language

Warunki integralnościowe:

-Data zajęć jest w formacie RRRR-MM-DD, GG-MM.

```
constraint validLessonDate
   check (CONVERT([varchar],[date],120)=[date])
```

```
create table dbo.Lessons
(
   lessonID     int          not null
       constraint Lessons_pk
           primary key,
   subjectID    int          not null
       constraint Lessons_StudiesSubjects
           references dbo.StudiesSubjects,
   translatorID int          not null
       constraint Lessons_Translators
           references dbo.Translators,
   classroomID  int          not null
       constraint Lessons_Classrooms
           references dbo.Classrooms,
   date         datetime     not null
       constraint validLessonDate
           check (CONVERT([varchar], [date], 120) =
[date]),
   language     varchar(255) not null
)
```

Tabela CurrencyExchange - reprezentuje wymianę walut w naszej bazie danych

Klucz główny: currencyID
Z jakiej waluty chcemy zmienić: from
Na jaką walutę chcemy zmienić: to
Przelicznik: value

```
create table dbo.CurrencyExchange
(
   currencyID int             not null
       constraint CurrencyExchange_pk
           primary key,
   [from]     varchar(255)    not null,
   [to]       varchar(255)    not null,
   value      decimal(10, 2) not null
)
```

Tabela Classrooms - reprezentuje pomieszczenia do przeprowadzania zajęć w naszej bazie danych

Klucz główny: classroomID

Oznaczenie budynku: building

Oznaczenie sali: room

Limit miejsc w sali: placeLimit

Warunki integralnościowe:

- *Informacja o budynku jest w formacie X-YY, gdzie X jest dowolną wielką literą alfabetu rzymskiego, a YY jest dowolną liczbą naturalną.*

```
constraint validBuilding
   check (building LIKE '[A-Z]-[0-9][0-9]')
```

- *Numer sali jest w formacie X.YY, gdzie X to liczba naturalna określająca piętro, natomiast YY to liczba naturalna określająca salę*

```
constraint validRoom
   check (room LIKE '[0-9].[0-9][0-9]')
```

```
create table dbo.Classrooms
(
   classroomID int          not null
       constraint Classrooms_pk
           primary key,
   building    varchar(255) not null
       constraint validBuilding
           check ([building] like '[A-Z]-[0-9][0-
9]'),
   room        varchar(255) not null
       constraint validRoom
           check ([room] like '[0-9].[0-9][0-9]'),
   placeLimit  int          not null
)
```

Widoki

1. Widok przeszłych webinarów

Jego celem jest pokazanie webinarów, które już się odbyły

- *Nazwa webinaru*
- *data*
- *nauczyciel*
- *tłumacz*
- *cena*
- *link do nagrania*

```
CREATE VIEW UserHistoryWebinarsView AS
SELECT
  webinarID,
  HistoryWebinars.title AS 'Webinar name',
  date AS 'Date',
  Teachers.lastName AS 'Teacher',
  price AS 'Price',
  ISNULL(recordedLink, '') AS 'recorded link'
FROM
  (SELECT * FROM WebinarsHistory WHERE date <= GETDATE())
AS HistoryWebinars
JOIN
  Teachers ON HistoryWebinars.teacherID =
Teachers.teacherID
go
```

2. Widok przyszłych webinarów

Jego celem jest pokazanie zarówno przyszłych webinarów

- *Nazwa webinaru*
- *data*
- *nauczyciel*
- *tłumacz*
- *cena*
- *link do spotkania*

```
CREATE VIEW UserFutureWebinarsView AS
SELECT
  webinarID,
  AllWebinars.title AS 'Webinar name',
  date AS 'Date',
  Teachers.lastName AS 'Teacher',
  Translators.lastName AS 'Translator',
  price AS 'Price',
  ISNULL(meetLink, '') AS 'meet link'
FROM
  (SELECT * FROM Webinars WHERE date >= GETDATE()) AS
AllWebinars
JOIN
  Teachers ON AllWebinars.teacherID = Teachers.teacherID
JOIN
  Translators ON AllWebinars.translatorID =
Translators.translatorID
go
```

3. Widok danych statystycznych o obecności studentów

Widok ma dla każdego studenta pokazywać szereg danych statystycznych odnośnie jego uczestnictwa w zajęciach

- *dotychczasowa frekwencja (procentowa)*
- *odchylenie od średniej osób na kierunku*
- *całkowita ilość nieobecności*
- *ilość godzin usprawiedliwionych*
- *ilość godzin nieusprawiedliwionych*

```
CREATE VIEW StudentAttendanceStats AS
SELECT
  S.studentID,
  S.userID,
  S.studiesID,
  S.studentEmail,
  ROUND(CAST(SUM(IIF(A.wasPresent = 1, 1, 0)) * 100.0 /
COUNT(A.lessonID) AS DECIMAL(5,2)), 2) AS
'CumulativeAttendancePercentage',
  ROUND(CAST(SUM(IIF(A.wasPresent = 1, 1, 0)) * 100.0 /
NULLIF(COUNT(DISTINCT S.studentID), 0) -
AVG(IIF(A.wasPresent = 1, 1, 0)) * 100.0 AS DECIMAL(5,2)),
2) AS 'DeviationFromDepartmentAverage',
  SUM(IIF(A.wasPresent = 0, 1, 0)) AS 'TotalAbsences',
  SUM(IIF(A.wasPresent = 0 AND (A.completedInOtherTerm = 1
OR A.wasJustificated = 1), 1, 0)) AS 'ExcusedAbsences',
  SUM(IIF(A.wasPresent = 0 AND A.completedInOtherTerm = 0
AND A.wasJustificated = 0, 1, 0)) AS 'UnexcusedAbsences'
FROM
  Students S
LEFT JOIN
  Attendance A ON S.studentID = A.studentID
GROUP BY
  S.studentID, S.userID, S.studiesID, S.studentEmail
go
```

4. Widok Sylabusów (wszyscy - wszystkie)

Wyświetl “Sylabus” danego kierunku czyli:

- *Nazwa kierunku*
- *Typ studiów (stacjonarny/hybrydowy/niestacjonarny)*
- *Limit miejsc*
- *Rocznik*
- *Przedmioty na danym kierunku:*
  - *Nazwa przedmiotu*
  - *Semestr na którym się odbywa*
  - *Imię i nazwisko nauczyciela prowadzącego*
- *Praktyki na danym kierunku:*
  - *Nazwa praktyk*
  - *Długość trwania*

```
CREATE VIEW SyllabusView AS
SELECT
  S.studiesID,
  S.name AS 'NazwaKierunku',
  S.type AS 'TypStudiow',
  S.studentLimit AS 'LimitMiejsc',
  SS.validDate AS 'Rocznik',
  SP.practisesName AS 'NazwaPraktyk',
  SP.duration AS 'DlugoscTrwania',
  SB.subjectName AS 'NazwaPrzedmiotu',
  SS.semester AS 'Semestr',
  T.lastName AS 'NauczycielProwadzacy'
FROM
  Studies S
JOIN
  StudiesSylabus SS ON S.studiesID = SS.studiesID
JOIN
  StudiesPractises SP ON SS.practisesID = SP.practisesID
LEFT JOIN
  StudiesSubjects SB ON SS.subjectID = SB.subjectID
LEFT JOIN
  Teachers T ON SB.teacherID = T.teacherID;
```

5. Widok Dłużników (księgowy - wszyscy)

Wyświetl użytkowników i ich dane, którzy zapisali się na kurs/webinar/lekcję i jeszcze nie zapłacili.

- *ID użytkownika*
- *Suma niezapłaconych kwot + rozwinięcie na poszczególne:*
  - *Typ (kurs/webinar/lekcja)*
  - *Kwota*
  - *Ostateczna data opłacenia*
  - *Ilość dni od deadline'u opłaty*

```
CREATE VIEW DebtorsView AS
SELECT
    p.userID AS 'User ID',
    SUM(p.price) AS 'Total Unpaid Amount',
    CASE
        WHEN p.courseID IS NOT NULL THEN 'Course'
        WHEN p.webinarID IS NOT NULL THEN 'Webinar'
        WHEN p.lessonID IS NOT NULL THEN 'Lesson'
    END AS 'Type',
    p.price AS 'Amount',
    p.paymentDate AS 'Final Payment Date',
    DATEDIFF(DAY, p.paymentDate, GETDATE()) AS 'Days Past
Deadline'
FROM
    PaymentHistory p
WHERE
    p.isPaid = 0
GROUP BY
    p.userID, CASE
                    WHEN p.courseID IS NOT NULL THEN
'Course'
                    WHEN p.webinarID IS NOT NULL THEN
'Webinar'
                    WHEN p.lessonID IS NOT NULL THEN
'Lesson'
                END, p.price, p.paymentDate;
```

6. Widok wszystkich kursów (wszyscy - wszystko)

Wyświetl dostępne kursy oraz link do nagrań tych, które się już odbyły,

- *Nazwa modułu*
- *Nazwa kursu*
- *Typ (stacjonarny/hybrydowy/niestacjonarny)*
  - *Gdy niestacjonarny -> link do nagrania*
  - *Gdy stacjonarny -> dane o sali zajęć*
  - *Gdy hybrydowy -> oba*
- *Informacje o języku w którym będzie się odbywał kurs:*
  - *Język główny*
  - *Ewentualne dane tłumacza*
- *Informacje o nauczycielu prowadzącym kurs*
- *Cena kursu + deadline opłaty*
- *Możliwość sprawdzenia obecności*

```
CREATE VIEW AllCoursesView AS
SELECT
  c.courseID,
  c.courseName AS 'Course Name',
  cm.moduleName AS 'Module Name',
  c.type AS 'Type',
  c.recordedLink AS 'Recorded Link',
  c.language AS 'Course Language',
  t.firstName + ' ' + t.lastName AS 'Teacher',
  tr.firstName + ' ' + tr.lastName AS 'Translator',
  c.price AS 'Course Price',
  c.paymentDeadline AS 'Payment Deadline',
  CASE
      WHEN c.type = 'on-line synchroniczne' THEN c.recordedLink
      WHEN c.type = 'on-line asynchroniczne' THEN c.recordedLink
      WHEN c.type = 'stacjonarne' THEN c.classroomID
      WHEN c.type = 'hybrydowe' THEN
          CASE
              WHEN c.recordedLink IS NOT NULL THEN c.recordedLink
              ELSE c.classroomID
          END
  END AS 'Recorded Link and room',
  c.date AS 'Date'
FROM
  Courses c
LEFT JOIN
  CoursesModules cm ON c.moduleID = cm.moduleID
LEFT JOIN
  Teachers t ON c.teacherID = t.teacherID
LEFT JOIN
  Translators tr ON c.translatorID = tr.translatorID;
```

7.Widok kalendarza wydarzeń (Wszyscy- wszystko)

Celem widoku jest pokazanie danych wszystkich wydarzeń odbywających się dzisiejszego dnia

- *typ wydarzenia (webinar/pojedyńcza płatna lekcja/ kursy stacjonarne)*
- *nazwa wydarzenia*
- *data*
- *miejsce spotkania*
- *link do zapisów*

```
CREATE VIEW EventsCalendarView AS
SELECT 'Webinar' AS 'Event Type', w.title AS 'Event Name',
w.date AS 'Event Date', NULL AS 'Event Location', w.meetLink AS
'Registration Link'
FROM Webinars w
WHERE CONVERT(DATE, w.date) = CONVERT(DATE, GETDATE())


UNION ALL


SELECT 'Paid Lesson' AS 'Event Type', 'Lesson' AS 'Event Name',
l.date AS 'Event Date', NULL AS 'Event Location', NULL AS
'Registration Link'
FROM SingleLessons sl
JOIN Lessons l ON sl.lessonID = l.lessonID
WHERE CONVERT(DATE, l.date) = CONVERT(DATE, GETDATE())


UNION ALL


SELECT 'Stationary Course' AS 'Event Type', c.courseName AS
'Event Name', c.date AS 'Event Date', c.classroomID AS 'Event
Location', NULL AS 'Registration Link'
FROM Courses c
JOIN CoursesModules cm ON c.moduleID = cm.moduleID
WHERE CONVERT(DATE, c.date) = CONVERT(DATE, GETDATE())
go
```

Procedury

AddUser - dodaje użytkownika do serwisu

```
CREATE PROCEDURE AddUser
(
  @firstName VARCHAR(255),
  @lastName VARCHAR(255),
  @email VARCHAR(255),
  @address VARCHAR(255),
  @phoneNumber INT
)
AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
  IF EXISTS(
      SELECT *
      FROM Users
      WHERE @firstName = firstName
      AND @lastName = lastName
      AND @email = email
      AND @address = address
      AND @phoneNumber = phoneNumber
  )
  BEGIN
      ;THROW 52000, N'User is already in database', 1
  END
  DECLARE @userID INT
  SELECT @userID = ISNULL(MAX(userID), 0) + 1
  FROM Users
  INSERT INTO Users(userID, firstName, lastName, email, address,
phoneNumber)
  VALUES (@userID, @firstName, @lastName, @email, @address, @phoneNumber)
  END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding User: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddTeacher - dodaje nowego nauczyciela

```
CREATE PROCEDURE AddTeacher (
   @firstName varchar(255),
   @lastName varchar(255),
   @email varchar(255),
   @phoneNumber int,
   @title varchar(255)
)
AS
BEGIN
   SET NOCOUNT ON
   BEGIN TRY
   IF EXISTS(
       SELECT *
       FROM Teachers
       WHERE @firstName = firstName
       AND @lastName = lastName
       AND @email = email
       AND @phoneNumber = phoneNumber
       AND @title = title
   )
   BEGIN
       ;
       THROW 52000, N'Teacher is already in database', 1
   END
   DECLARE @teacherID INT
   SELECT @teacherID = ISNULL(MAX(teacherID), 0) + 1
   FROM Teachers
   INSERT INTO
Teachers(teacherID,firstName,lastName,email,phoneNumber,title)
   VALUES
(@teacherID,@firstName,@lastName,@email,@phoneNumber,@title)
   END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding Teacher: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddPractise - dodaje praktyki do naszej bazy danych

```
CREATE PROCEDURE AddPractise
(
  @practisesName varchar(255),
  @duration INT
)
AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
  IF EXISTS(
      SELECT *
      FROM StudiesPractices
      WHERE @practisesName = practisesName
  )
  BEGIN
      ;
      THROW 52000, N'Practise is already in database', 1
  END
  DECLARE @practisesID INT
  SELECT @practisesID = ISNULL(MAX(practisesID), 0) + 1
  FROM StudiesPractices
  INSERT INTO StudiesPractices(practisesID, practisesName,
duration)
  VALUES (@practisesID, @practisesName, @duration)
  END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding Practise: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddClassroom - dodaje nową salę zajęć

```
CREATE PROCEDURE AddClassroom
(
  @building varchar(255),
  @room varchar(255),
  @placeLimit int
)
AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
  IF EXISTS(
      SELECT *
      FROM Classrooms
      WHERE @building = building
      AND @room = room
  )
  BEGIN
      ;
      THROW 52000, N'Classroom is already in database', 1
  END
  DECLARE @classroomID INT
  SELECT @classroomID = ISNULL(MAX(classroomID), 0) + 1
  FROM Classrooms
  INSERT INTO Classrooms(classroomID, building, room,
placeLimit)
  VALUES (@classroomID, @building, @room, @placeLimit)
  END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding Classroom: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddStudies - dodaje nowy kierunek studiów

```
CREATE PROCEDURE AddStudies
(
  @name varchar(255),
  @type varchar(255),
  @studentLimit int
)
AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
  IF EXISTS(
      SELECT *
      FROM Studies
      WHERE @name = name
      AND @type = type
  )
  BEGIN
      ;
      THROW 52000, N'Studies are already in database', 1
  END
  DECLARE @studiesID INT
  SELECT @studiesID = ISNULL(MAX(studiesID), 0) + 1
  FROM Studies
  INSERT INTO Studies(studiesID, name, type, studentLimit)
  VALUES (@studiesID, @name, @type, @studentLimit)
  END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding Studies: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddTranslator - dodaje nowego tłumacza do bazy

```
CREATE PROCEDURE AddTranslator
(
  @firstName varchar(255),
  @lastName varchar(255),
  @email varchar(255),
  @phoneNumber int
)
AS
BEGIN
  SET NOCOUNT ON
  BEGIN TRY
  IF EXISTS(
      SELECT *
      FROM Translators
      WHERE @firstName = firstName
       AND @lastName = lastName
       AND @email = email
       AND @phoneNumber = phoneNumber
  )
  BEGIN
      ;
      THROW 52000, N'Translator is already in database', 1
  END
  DECLARE @translatorID INT
  SELECT @translatorID = ISNULL(MAX(translatorID), 0) + 1
  FROM Translators
  INSERT INTO Translators(translatorID, firstName, lastName,
email, phoneNumber)
  VALUES (@translatorID, @firstName, @lastName, @email,
@phoneNumber)
  END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding Translator: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddWebinar - dodaje nowy webinar

```
CREATE PROCEDURE AddWebinar
(
   @teacherID int,
   @translatorID int,
   @title varchar(255),
   @date date,
   @price money,
   @meetLink varchar(255),
   @language varchar(255),
   @paymentDeadline date
)
AS
BEGIN
 SET NOCOUNT ON
 BEGIN TRY
 IF EXISTS(
     SELECT *
     FROM Webinars
     WHERE @title = title
      AND @date = date
 )
 BEGIN
     ;
     THROW 52000, N'Webinar is already in database', 1
 END
 DECLARE @webinarID INT
 SELECT @webinarID = ISNULL(MAX(webinarID), 0) + 1
 FROM Webinars
 INSERT INTO Webinars(webinarID, teacherID, translatorID, title, date, price, meetLink,
language, paymentDeadline)
 VALUES (@webinarID, @teacherID, @translatorID, @title, @date, @price, @meetLink,
@language, @paymentDeadline)
 END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding Webinar: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddCourse - dodaje nowy kurs do tabeli Courses oraz nazwę modułu do CoursesModules jeżeli jeszcze taka nie istnieje

```
CREATE PROCEDURE AddCourse
 @courseName VARCHAR(255),
 @price MONEY,
 @language VARCHAR(255),
 @paymentDeadline DATE,
 @moduleName VARCHAR(255),
 @teacherID INT,
 @translatorID INT,
 @classroomID INT,
 @type VARCHAR(255),
 @recordedLink VARCHAR(255),
 @date DATE
AS
BEGIN
 DECLARE @NewCourseID INT
 DECLARE @NewModuleID INT
   SELECT @NewModuleID = moduleID
   FROM CoursesModules
   WHERE moduleName = @moduleName
   IF @NewModuleID IS NULL
       BEGIN
           SELECT @NewModuleID = ISNULL(MAX(moduleID), 0) + 1 FROM
CoursesModules
           INSERT INTO CoursesModules (moduleID, moduleName)
           VALUES (@NewModuleID, @moduleName)
      END
   SELECT @NewCourseID = ISNULL(MAX(courseID), 0) + 1 FROM Courses
       INSERT INTO Courses (courseID, courseName, price, language,
paymentDeadline, moduleID, teacherID, type, classroomID, recordedLink,
translatorID, date)
       VALUES (@NewCourseID, @courseName, @price, @language, @paymentDeadline,
@NewModuleID, @teacherID, @type, @classroomID, @recordedLink, @translatorID,
@date)
END
```

AddGrade - dodaje ocenę studenta do tabeli StudiesFinalGrades

```
CREATE PROCEDURE AddGrade
(
 @subjectID int,
 @studentID int,
 @grade float
)
AS
BEGIN
 SET NOCOUNT ON
 BEGIN TRY
 IF EXISTS(
     SELECT *
     FROM StudiesFinalGrades
     WHERE @studentID = studentID
     AND @subjectID = subjectID
 )
 BEGIN
     ;
     THROW 52000, N'Grade is already in database', 1
 END
 DECLARE @gradesID INT
 SELECT @gradesID = ISNULL(MAX(gradesID), 0) + 1
 FROM StudiesFinalGrades
 INSERT INTO StudiesFinalGrades(gradesID, subjectID, studentID,
grade)
 VALUES (@gradesID, @subjectID, @studentID, @grade)
 END TRY
BEGIN CATCH
DECLARE @msg nvarchar(2048)
=N'Error adding Grade: ' + ERROR_MESSAGE();
THROW 52000, @msg, 1
END CATCH
END
```

AddCoursePresence - dodaje obecność użytkownika na kursie do tabeli CoursesPresence

```
CREATE PROCEDURE AddCoursePresence
   @courseID int,
   @userID int,
   @wasPresent bit,
   @watchedRecord bit,
   @certificateLink varchar(255),
   @hasPaid bit
AS
BEGIN
   DECLARE @moduleID int
   SELECT @moduleID = CoursesModules.moduleID
   FROM CoursesModules
   INNER JOIN Courses ON Courses.moduleID =
CoursesModules.moduleID
   WHERE courseID = @courseID
   DECLARE @date date
   SELECT @date = date
   FROM Courses
   WHERE courseID = @courseID
   DECLARE @PresenceID int
   BEGIN
       SELECT @PresenceID = ISNULL(MAX(presenceID), 0) + 1 FROM
CoursesPresence
       INSERT INTO CoursesPresence (presenceID, courseID,
moduleID, userID, wasPresent, watchedRecord, certificateLink,
meetingDate, hasPaid)
       VALUES (@PresenceID, @courseID, @moduleID, @userID,
@wasPresent, @watchedRecord, @certificateLink, @date, @hasPaid)
   END
END
```

ModifyPermission - modyfikuje uprawnienia użytkownika w systemie

```
CREATE PROCEDURE ModifyPermission
   @firstName varchar(255),
   @lastName varchar(255),
   @email varchar(255),
   @newPermission varchar(255)
AS
BEGIN
   SET NOCOUNT ON
   UPDATE Users
   SET userType = @newPermission
   WHERE FirstName = @firstName AND LastName = @lastName AND
Email = @email
END
```

AddWebinarPresence - dodaje obecność użytkownika na webinarze do tabeli WebinarPresence

```
CREATE PROCEDURE AddWebinarPresence
   @webinarID int,
   @userID int,
   @certificateLink varchar(255),
   @wasPresent bit
AS
BEGIN
   DECLARE @webinarPresenceID int
   BEGIN
       SELECT @webinarPresenceID =
ISNULL(MAX(webinarpresenceID), 0) + 1 FROM WebinarsPresence
       INSERT INTO WebinarsPresence (webinarPresenceID,
webinarID, userID, certificateLink, wasPresent)
       VALUES (@webinarPresenceID, @webinarID, @userID,
@certificateLink, @wasPresent)
   END
END
```

AddLessonPresence - dodaje obecność studenta na zajęciach do tabeli Attendance

```
CREATE PROCEDURE AddLessonPresence
   @lessonID int,
   @studentID int,
   @wasPresent bit,
   @completedInOtherTerm bit,
   @wasJustificated bit
AS
BEGIN
   DECLARE @attendanceID int
   BEGIN
       SELECT @attendanceID = ISNULL(MAX(attendenceID), 0) + 1
FROM Attendance
       INSERT INTO Attendance (attendenceID, lessonID,
studentID, wasPresent, completedInOtherTerm, wasJustificated)
       VALUES (@attendanceID, @lessonID, @studentID,
@wasPresent, @completedInOtherTerm, @wasJustificated)
   END
END
```

AddLesson - dodaje zajęcia do tabeli Lessons

```
CREATE PROCEDURE AddLesson
   @subjectID int,
   @translatorID int,
   @classroomID int,
   @date date,
   @language varchar(255)
AS
BEGIN
   DECLARE @lessonID int
   BEGIN
       SELECT @lessonID = ISNULL(MAX(lessonID), 0) + 1 FROM
Lessons
       INSERT INTO Lessons (lessonID, subjectID, translatorID,
classroomID, date, language)
       VALUES (@lessonID, @subjectID, @translatorID,
@classroomID, @date, @language)
   END
END
```

AddSingleLesson - dodaje pojedyncze zajęcia, które zostały wykupione przez użytkownika do tabeli SingleLessons

```
CREATE PROCEDURE AddSingleLesson
   @lessonID int,
   @userID int,
   @price money,
   @paymentDeadline date
AS
BEGIN
   DECLARE @singleLessonID int
   BEGIN
       SELECT @singleLessonID = ISNULL(MAX(singleLessonID), 0) +
1 FROM SingleLessons
       INSERT INTO SingleLessons (singleLessonID, lessonID,
userID, price, paymentDeadline)
       VALUES (@singleLessonID, @lessonID, @userID, @price,
@paymentDeadline)
   END
END
```

AddStudiesSubject - dodaje przedmiot nauczania na studiach do tabeli StudiesSubjects

```
CREATE PROCEDURE AddStudiesSubject
   @teacherID int,
   @totalHours int,
   @subjectName varchar(255),
   @hasExam bit
AS
BEGIN
   DECLARE @subjectID int
   BEGIN
       SELECT @subjectID = ISNULL(MAX(subjectID), 0) + 1 FROM
StudiesSubjects
       INSERT INTO StudiesSubjects (subjectID, teacherID,
totalHours, subjectName, hasExam)
       VALUES (@subjectID, @teacherID, @totalHours,
@subjectName, @hasExam)
   END
END
```

AddPayment - dodaje płatność do tabeli PaymentHistory (podaje się ID usługi którą dana płatność obowiązuje, w pozostałych musi być null)

```
CREATE PROCEDURE AddPayment
   @userID int,
   @courseID int,
   @webinarID int,
   @singleLessonID int,
   @price money,
   @isPaid bit,
   @paymentDate date
AS
BEGIN
   DECLARE @paymentID int
   BEGIN
       SELECT @paymentID = ISNULL(MAX(paymentID), 0) + 1 FROM
PaymentHistory
       INSERT INTO PaymentHistory (paymentID, userID, courseID,
webinarID, singleLessonID, price, isPaid, paymentDate)
       VALUES (@paymentID, @userID, @courseID, @webinarID,
@singleLessonID, @price, @isPaid, @paymentDate)
   END
END
```

AddSylabus - dodaje przedmioty do sylabusu kierunku

```
CREATE PROCEDURE AddToSylabus
   @studiesName varchar(255),
   @studiesType varchar(255),
   @subjectName varchar(255),
   @practisesName varchar(255),
   @semester int,
   @startDate date
AS
   BEGIN
   DECLARE @sylabusID int
   DECLARE @studiesID int
   DECLARE @subjectID int
   DECLARE @practisesID int

   SELECT @studiesID = studiesID
   FROM Studies
   WHERE name = @studiesName
     AND type = @studiesType

   SELECT @subjectID = subjectID
   FROM StudiesSubjects
   WHERE subjectName = @subjectName

   SELECT @practisesID = practisesID
   FROM StudiesPractises
   WHERE practisesName = @practisesName

   SELECT @sylabusID = ISNULL(MAX(sylabusID), 0) + 1
   FROM StudiesSylabus
   INSERT INTO StudiesSylabus (studiesID, subjectID,
practisesID, semester, year, sylabusID)
   VALUES (@studiesID, @subjectID, @practisesID, @semester,
@startDate, @sylabusID)
END
```

Funkcje

FillAttendanceForStudent- wypełnia obecność studenta na lekcji o wybranym ID, o ile osoba o tym imieniu i nazwisku istnieje. Jeśli obecność tego studenta została już wystawiona podczas tych zajęć, wywołanie funkcji nadpisze rekord z nią.

```
CREATE FUNCTION dbo.FillAttendanceForStudent
(
    @firstName NVARCHAR(255),
    @lastName NVARCHAR(255),
    @lessonID INT,
    @wasPresent BIT,
    @completedInOtherTerm BIT,
    @wasJustificated BIT
)
RETURNS INT
AS
BEGIN
    DECLARE @studentID INT;
    SELECT @studentID = s.studentID
    FROM Students s
    JOIN Users u ON s.userID = u.userID
    WHERE u.firstName = @firstName
        AND u.lastName = @lastName;
    IF @studentID IS NOT NULL
    BEGIN
        DECLARE @existingAttendanceID INT;
        SELECT @existingAttendanceID = attendenceID
        FROM Attendance
        WHERE lessonID = @lessonID
            AND studentID = @studentID;
        IF @existingAttendanceID IS NOT NULL
        BEGIN
            UPDATE Attendance
            SET
                wasPresent = @wasPresent,
                completedInOtherTerm = @completedInOtherTerm,
                wasJustificated = @wasJustificated
            WHERE attendenceID = @existingAttendanceID;
            RETURN @existingAttendanceID;
        END
        ELSE
        BEGIN
            INSERT INTO Attendance (lessonID, studentID, wasPresent, completedInOtherTerm,
wasJustificated)
            VALUES (@lessonID, @studentID, @wasPresent, @completedInOtherTerm, @wasJustificated);
            RETURN SCOPE_IDENTITY();
        END
    END
    ELSE
    BEGIN
        RETURN -1; -- Zwróć -1, jeśli student nie został znaleziony
    END
END;
```

makeLessonAvailableToBuy- dodaje lekcję do SingleLessons, co czyni ją możliwą do wykupienia przez użytkownika niebędącego studentem.

```
CREATE FUNCTION dbo.makeLessonAvailableToBuy
(
    @lessonID INT,
    @userID INT,
    @price DECIMAL(10, 2),
    @paymentDeadline DATE
)
RETURNS INT
AS
BEGIN
    IF NOT EXISTS (SELECT 1 FROM SingleLessons WHERE lessonID = @lessonID AND userID =
@userID)
    BEGIN
        INSERT INTO SingleLessons (lessonID, userID, price, paymentDeadline)
        VALUES (@lessonID, @userID, @price, @paymentDeadline);

        RETURN SCOPE_IDENTITY();
    END
    ELSE
    BEGIN
        RETURN -1;
    END
END;
```

getUserActivities- Zwraca tabelę wszystkich działań, w których brał udział użytkownik o ID równym @userID

```
CREATE FUNCTION getUserActivities(@userID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT 'Course' AS 'Activity Type', c.courseName AS 'Activity Name',
cp.meetingDate AS 'Activity Date', CASE WHEN ph.isPaid = 1 THEN 'Paid' ELSE 'Not
Paid' END AS 'Payment Status'
    FROM CoursesPresence cp
    JOIN Courses c ON cp.courseID = c.courseID
    LEFT JOIN PaymentHistory ph ON cp.courseID = ph.courseID AND ph.userID = @userID

    UNION ALL

    SELECT 'Webinar' AS 'Activity Type', w.title AS 'Activity Name', w.date AS
'Activity Date', CASE WHEN ph.isPaid = 1 THEN 'Paid' ELSE 'Not Paid' END AS 'Payment
Status'
    FROM WebinarsPresence wp
    JOIN Webinars w ON wp.webinarID = w.webinarID
    LEFT JOIN PaymentHistory ph ON wp.webinarID = ph.webinarID AND ph.userID = @userID

    UNION ALL

    SELECT 'Lesson' AS 'Activity Type', l.subjectID AS 'Activity Name', l.date AS
'Activity Date', CASE WHEN ph.isPaid = 1 THEN 'Paid' ELSE 'Not Paid' END AS 'Payment
Status'
    FROM Lessons l
    JOIN SingleLessons sl ON l.lessonID = sl.lessonID
    LEFT JOIN PaymentHistory ph ON sl.lessonID = ph.lessonID AND ph.userID = @userID
    WHERE sl.userID = @userID
);
```

getTeacherWebinars - funkcja która zwraca dane o wszystkich webinarach przypiętych do konkretnego nauczyciela

```
CREATE FUNCTION getTeacherWebinars(@teacherID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        'Past Webinar' AS 'Webinar Type',
        wh.webinarID,
        wh.title AS 'Webinar Title',
        wh.price AS 'Webinar Price',
        wh.date AS 'Webinar Date',
        wh.recordedLink AS 'Recorded Link'
    FROM
        WebinarsHistory wh
    WHERE
        wh.teacherID = @teacherID

    UNION ALL

    SELECT
        'Future Webinar' AS 'Webinar Type',
        w.webinarID,
        w.title AS 'Webinar Title',
        w.price AS 'Webinar Price',
        w.date AS 'Webinar Date',
        NULL AS 'Recorded Link'
    FROM
        Webinars w
    WHERE
        w.teacherID = @teacherID
);
```

getTranslatorWebinars - funkcja która zwraca dane o wszystkich przyszłych webinarach przypiętych do konkretnego tłumacza. W przeciwieństwie do odpowiednika tej funkcji dla nauczycieli nie zawiera przeszłych webinarów, bo nie powinno być to przydatne.

```
CREATE FUNCTION GetTranslatorWebinars(@translatorID INT)
RETURNS TABLE
AS
RETURN
(
   SELECT
       'Future Webinar' AS 'Webinar Type',
       w.webinarID,
       w.title AS 'Webinar Title',
       w.price AS 'Webinar Price',
       w.date AS 'Webinar Date',
       NULL AS 'Recorded Link'
   FROM
       Webinars w
   WHERE
       w.translatorID = @translatorID
);
```

getTranslatorCourses - zwraca wszystkie kursy, które są tłumaczone przez osobę @translatorID

```
CREATE FUNCTION GetTranslatorCourses(@translatorID INT)
RETURNS TABLE
AS
RETURN
(
   SELECT
       'Future Webinar' AS 'Webinar Type',
       c.courseID,
       c.courseName AS 'Webinar Title',
       c.price AS 'Webinar Price'
   FROM
       Courses c
   WHERE
       c.translatorID = @translatorID
);
```

getTeacherCourses - zwraca wszystkie kursy, które prowadzi @teacherID

```
CREATE FUNCTION GetTeacherCourses(@teacherID INT)
RETURNS TABLE
AS
RETURN
(
   SELECT
       'Future Webinar' AS 'Webinar Type',
       c.courseID,
       c.courseName AS 'Webinar Title',
       c.price AS 'Webinar Price'
   FROM
       Courses c
   WHERE
       c.teacherID = @teacherID
);
```

Triggery

countExamsTrigger - trigger odpowiedzialny za automatyczne wyliczanie liczby egzaminów na danych studiach i uzupełnianie tej informacji w tabeli Studies

```
CREATE TRIGGER countExamsTrigger
ON StudiesSubjects
AFTER INSERT
AS
BEGIN
   SET NOCOUNT ON;
   DECLARE @subjectID INT;
   DECLARE @hasExam BIT;
   SELECT @subjectID = subjectID, @hasExam = hasExam
   FROM inserted;
   IF @hasExam = 1
   BEGIN
       UPDATE Studies
       SET examsCount = examsCount + 1
       WHERE subjectID = @subjectID;
   END;
END;
```

afterDeleteWebinarTrigger - trigger, który po usunięciu z bazy webinaru automatycznie przerzuca go do historii.

```
CREATE TRIGGER AfterDeleteWebinarTrigger
ON Webinars
AFTER DELETE
AS
BEGIN
   SET NOCOUNT ON;
   DECLARE @webinarID int
   DECLARE @teacherID int
   DECLARE @translatorID int
   DECLARE @translatorID text
   DECLARE @title text
   DECLARE @date date
   DECLARE @price money
   DECLARE @recordedLink text
   SELECT @webinarID=webinarID, @teacherID=teacherID, @title=title, @date=date,
@price=price, @recordedLink=ISNULL(meetLink, '') AS recordedLink
   FROM deleted;
   INSERT INTO WebinarsHistory (webinarID, teacherID, title, date, price,
recordedLink)
   VALUES (@webinarID, @teacherID, @title, @date, @price, @recordedLink)

END;
```

Indeksy

Teacher

```
CREATE UNIQUE INDEX Teacher ON Teachers (teacherID)
```

Subject

```
CREATE UNIQUE INDEX [Subject] ON StudiesSubjects(subjectID)
```

Classroom

```
CREATE UNIQUE INDEX Classroom ON Classrooms(classroomID)
```

Translator

```
CREATE UNIQUE INDEX Translator ON Translators(translatorID)
```

Lesson

```
CREATE UNIQUE INDEX Lesson ON Lessons(lessonID)
```

CoursesModule

```
CREATE UNIQUE INDEX CoursesModule ON CoursesModules(moduleID)
```

Practices

```
CREATE UNIQUE INDEX Practices ON StudiesPractices(practisesID)
```

Studies

```
CREATE UNIQUE INDEX Studies ON Studies(studiesID)
```

Course

```
CREATE UNIQUE INDEX Course ON Courses(courseID)
```

SingleLesson

```
CREATE UNIQUE INDEX SingleLesson ON SingleLessons(lessonID)
```

User

```
CREATE UNIQUE INDEX [User] ON Users(userID)
```

Webinar

```
CREATE UNIQUE INDEX Webinar ON Webinars(webinarID)
```

Sylabus

```
CREATE UNIQUE INDEX Sylabus ON StudiesSylabus(sylabusID)
```

Student

```
CREATE UNIQUE INDEX Student ON Students(studentID)
```

Payment

```
CREATE UNIQUE INDEX Payment ON PaymentHistory(paymentID)
```

WebinarsPresence

```
CREATE UNIQUE INDEX WebinarsPresence ON WebinarsPresence(webinarPresenceID)
```

PastWebinar

```
CREATE UNIQUE INDEX PastWebinar ON WebinarsHistory(webinarID)
```

Schedule

```
CREATE UNIQUE INDEX Schedule ON StudentSchedule(scheduleID)
```

FinalGrade

```
CREATE UNIQUE INDEX FinalGrade ON StudiesFinalGrades(gradesID)
```

CurrencyExchange

```
CREATE UNIQUE INDEX CurrencyExchange ON CurrencyExchange(currencyID)
```

Attendance

```
CREATE UNIQUE INDEX Attendance ON Attendance(attendenceID)
```

Uprawnienia

Podstawowy użytkownik

```
CREATE ROLE [user]

GRANT SELECT ON UserFuturelWebinarsView TO accountant
GRANT SELECT ON UserPastlWebinarsView TO accountant
GRANT SELECT ON SyllabusView TO [user]
GRANT SELECT ON AllCoursesView TO [user]
GRANT SELECT ON EventsCallendarView TO [user]
GRANT EXECUTE ON getTeacherWebinars TO [user]
GRANT EXECUTE ON getTranslatorWebinars TO [user]
GRANT EXECUTE ON getTranslatorCourses TO [user]
GRANT EXECUTE ON getTeacherCourses TO [user]
GRANT EXECUTE ON countExamsTrigger TO [user]
```

Nauczyciel

```
CREATE ROLE teacher

GRANT SELECT ON UserFuturelWebinarsView TO accountant
GRANT SELECT ON UserPastlWebinarsView TO accountant
GRANT SELECT ON StudentAttendanceStats TO teacher
GRANT SELECT ON SyllabusView TO teacher
GRANT SELECT ON AllCoursesView TO teacher
GRANT SELECT ON EventsCallendarView TO teacher
GRANT EXECUTE ON AddGrade TO teacher
GRANT EXECUTE ON AddCoursePresence TO teacher
GRANT EXECUTE ON AddWebinarPresence TO teacher
GRANT EXECUTE ON AddLessonPresence TO teacher
GRANT EXECUTE ON FillAttendanceForStudent TO teacher
GRANT EXECUTE ON getUserActivities TO teacher
GRANT EXECUTE ON getTeacherWebinars TO teacher
GRANT EXECUTE ON getTranslatorWebinars TO teacher
GRANT EXECUTE ON getTranslatorCourses TO teacher
GRANT EXECUTE ON getTeacherCourses TO teacher
GRANT EXECUTE ON countExamsTrigger TO teacher
```

Tłumacz

```
CREATE ROLE translator

GRANT SELECT ON UserFuturelWebinarsView TO accountant
GRANT SELECT ON UserPastlWebinarsView TO accountant
GRANT SELECT ON SyllabusView TO translator
GRANT SELECT ON AllCoursesView TO translator
GRANT SELECT ON EventsCallendarView TO translator
GRANT EXECUTE ON AddWebinarPresence TO translator
GRANT EXECUTE ON getTeacherWebinars TO translator
GRANT EXECUTE ON getTranslatorWebinars TO translator
GRANT EXECUTE ON getTranslatorCourses TO translator
GRANT EXECUTE ON getTeacherCourses TO translator
GRANT EXECUTE ON countExamsTrigger TO translator
```

Księgowy

```
CREATE ROLE accountant

GRANT SELECT ON UserFuturelWebinarsView TO accountant
GRANT SELECT ON UserPastlWebinarsView TO accountant
GRANT SELECT ON SyllabusView TO accountant
GRANT SELECT ON AllCoursesView TO accountant
GRANT SELECT ON EventsCallendarView TO accountant
GRANT SELECT ON DebtorsView TO accountant
GRANT EXECUTE ON AddWebinarPresence TO accountant
GRANT EXECUTE ON getTeacherWebinars TO accountant
GRANT EXECUTE ON getTranslatorWebinars TO accountant
GRANT EXECUTE ON getTranslatorCourses TO accountant
GRANT EXECUTE ON getTeacherCourses TO accountant
GRANT EXECUTE ON countExamsTrigger TO accountant
```

Manager

```
CREATE ROLE manager
GRANT ALL PRIVILEGES TO manager
```