Bazy danych projekt

Kamil Lesiński, Maciej Michoń, Ignacy Siklucki

Spis treści

Użytkownicy

1. **Użytkownicy niezarejestrowanych**
2. **Użytkownicy zarejestrowani**
3. **Uczestnik webinaru**
4. **Uczestnik kursu**
5. **Uczestnik studiów**
6. **Koordynator studiów**
7. **Koordynator kursu**
8. **Koordynator webinaru**
9. **Koordynator praktyk**
10. **Tłumacz**
11. **Pracownicy sekretariatu**
12. **Nauczyciel**
13. **Dyrektor Szkoły**
14. **Administrator systemu**
15. **Administrator studiów**

1. Użytkownik niezarejestrowany
- rejestracja (założenie konta)
- przegląd ofert kursów, webinarów i studiów

2. Użytkownik zarejestrowany
- dodawanie produktów do koszyka
- zapisanie się na poszczególne usługi
- zapłacenie za wybrane usługi (całość lub zaliczka przy kursach)
- przegląd ofert kursów, webinarów i studiów
- dostęp do darmowych webinarów
- zapis na pojedyncze spotkania studyjne
- możliwość rezygnacji z danej usługi
- usunięcie swojego konta

3. Uczestnik webinaru (użytkownik zarejestrowany)
- dostęp do zakupionych webinarów
- dostęp do nagrań webinarów

4. Uczestnik kursu (użytkownik zarejestrowany)
- dostęp do nagrań (dla kursów on-line asynchronicznie i hybrydowych)

5. Uczestnik studiów (użytkownik zarejestrowany)
- dostęp do sylabusa
- widok harmonogramu zajęć
- możliwość odrobienia nieobecności po ustaleniu z nauczycielem przedmiotu
- dostęp do nagrań spotkań online

- widok swoich ocen

6. Koordynator studiów

- ustalenie sylabusu
- ustalenie harmonogramu (termin i data spotkań)
- ustalenie formy spotkań (stacjonarne/on-line/hybrydowe)
- ustala przedmioty studiów
- ustala język przedmiotów
- przypisanie tłumacza
- widok frekwencji uczestników
- ustalenie limitu miejsc
- dostęp do listy obecności wszystkich użytkowników
- dostęp do informacji o odbyciu praktyk
- wystawianie informacji o zaliczeniach

7. Koordynator kursu

-ustalenie sylabusu
-ustalenie harmonogramu (termin kursu, godziny i dni spotkań)
-ustalenie języka kursu
-przypisanie tłumacza
-ustalenie modułów

- stacjonarne
  -przypisanie sali zajęciowej
  -ustalenie limitu miejsc
  -przypisanie nauczycieli prowadzących
- on-line synchroniczne
  -przypisanie nauczycieli prowadzących
- on-line asynchroniczne
  -udostępnienie nagrań lub zlecenie nagrania spotkań przez nauczycieli
- hybrydowe
  -przypisanie sali zajęciowej
  -ustalenie limitu miejsc
  -przypisanie nauczycieli prowadzących
  -udostępnienie nagrań lub zlecenie nagrania spotkań przez nauczycieli

- generowanie listy obecności
- wystawia zaliczenie

8. Koordynator webinaru

-dodawanie, usuwanie webinaru
-ustalenie terminu webinaru
-ustalenie języka
-przypisanie tłumacza
-ustalenie nauczyciela prowadzącego (sam może prowadzić)
-dodanie nagrania
-udostępnienie nagrania

9. Koordynator praktyk

- wystawienie zaliczenia praktyk

10. Tłumacz

- podaje język który tłumaczy
- dostęp do zajęć w których tłumaczy

11. Pracownicy sekretariatu

- generowanie raportów finansowych (dla webinarów/kursów/studiów)
- generowanie listy dłużników (osób, które skorzystały z usług, ale nie uiściły opłaty)
- generowanie raportu dotyczącego frekwencji na zakończonych już wydarzeniach

12. Nauczyciel

- naucza na przedmiocie (kursy lub studia) lub może prowadzić spotkanie na webinarze
- wystawia oceny (swojego przedmiotu studiów)
- aktualizuje listę obecności
- generowanie listy obecności
- wystawia zaliczenie przedmiotu (studia i kursy)

13. Dyrektor Szkoły

- decyduje zgodę na płatność odroczoną
- usunięcie studenta z listy

14. Administrator systemu

- tworzenie backupów bazy danych
- zarządzanie kontami użytkowników

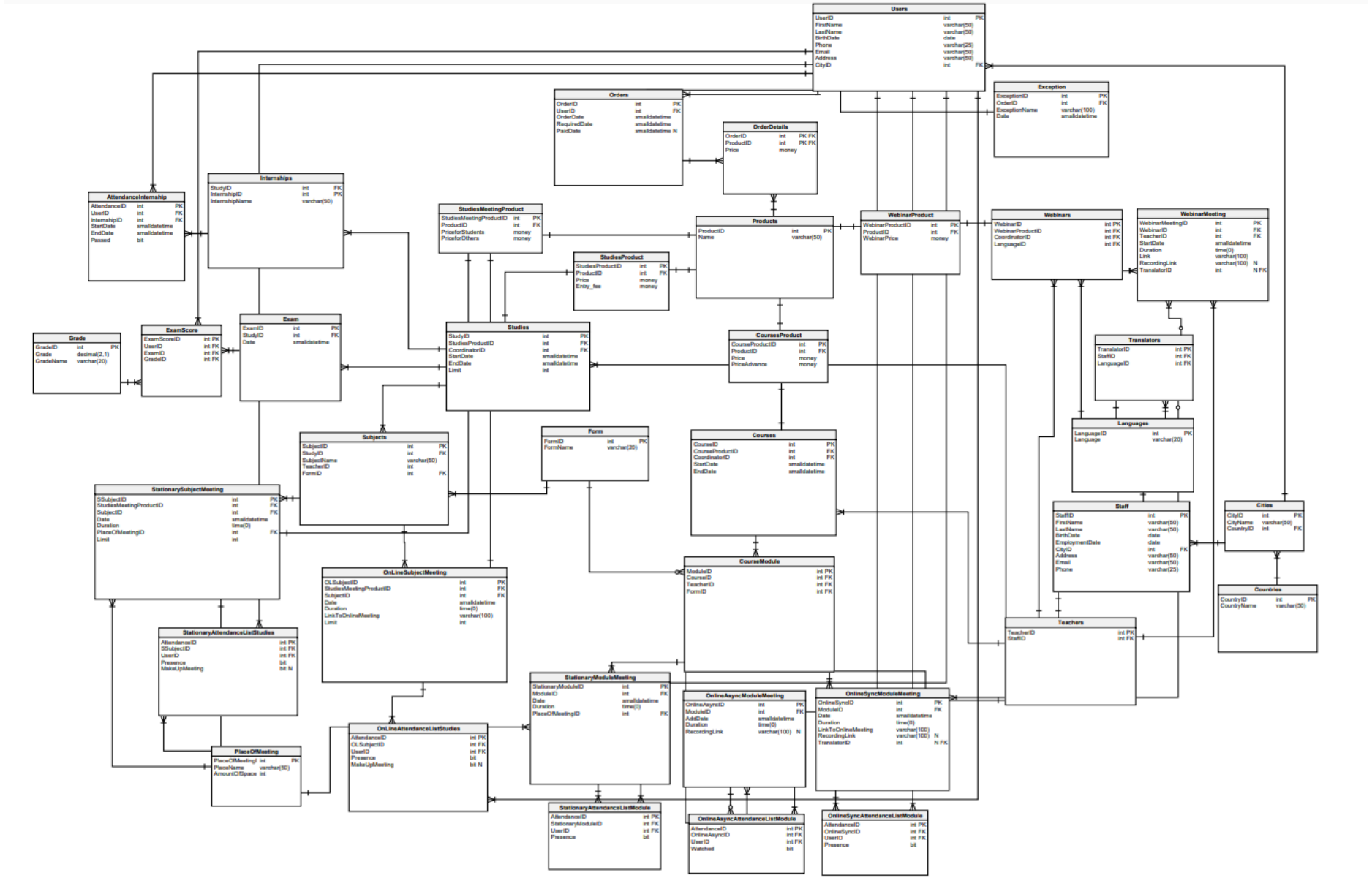
15. Administrator studiów

- modyfikacja harmonogramu studiów
- wydawanie dyplomów
- usuwanie webinarów
- generowanie raportu bilokacji (listy osób, które są zapisane na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo)

Funkcje Systemu

- obliczanie kosztu zamówienia
- sprawdzenie poprawności ilości uczestników do limitu miejsc
- informacja zwrotna o statusie płatności
- generowanie linku płatności
- przechowywanie informacji o wyjątkach w sprawach płatności

Diagram


Users
UserID int PK
FirstName varchar(50)
LastName varchar(50)
BirthDate date
Phone varchar(25)
Email varchar(50)
Address varchar(50)
CityID int FK
Orders
OrderID int PK
UserID int FK
OrderDate smalldatetime
RequiredDate smalldatetime
PaidDate smalldatetime N
OrderDetails
OrderID int PK FK
ProductID int PK FK
Price money
Exception
ExceptionID int PK
OrderID int FK
ExceptionName varchar(100)
Date smalldatetime
Internships
StudyID int FK
InternshipID int PK
InternshipName varchar(50)
AttendanceInternship
AttendanceID int PK
UserID int FK
InternshipID int FK
StartDate smalldatetime
EndDate smalldatetime
Passed bit
StudiesMeetingProduct
StudiesMeetingProductID int PK
ProductID int FK
PriceforStudents money
PriceforOthers money
StudiesProduct
StudiesProductID int PK
ProductID int FK
Price money
Entry_fee money
Products
ProductID int PK
Name varchar(50)
WebinarProduct
WebinarProductID int PK
ProductID int FK
WebinarPrice money
Webinars
WebinarID int PK
WebinarProductID int FK
CoordinatorID int FK
LanguageID int FK
WebinarMeeting
WebinarMeetingID int PK
WebinarID int FK
TeacherID int FK
StartDate smalldatetime
Duration time(0)
Link varchar(100)
RecordingLink varchar(100) N
TranslatorID int N FK
Translators
TranslatorID int PK
StaffID int FK
LanguageID int FK
Languages
LanguageID int PK
Language varchar(20)
Grade
GradeID int PK
Grade decimal(2,1)
GradeName varchar(20)
ExamScore
ExamScoreID int PK
UserID int FK
ExamID int FK
GradeID int FK
Exam
ExamID int PK
StudyID int FK
Date smalldatetime
Studies
StudyID int PK
StudiesProductID int FK
CoordinatorID int FK
StartDate smalldatetime
EndDate smalldatetime
Limit int
CoursesProduct
CourseProductID int PK
ProductID int FK
Price money
PriceAdvance money
Subjects
SubjectID int PK
StudyID int FK
SubjectName varchar(50)
TeacherID int
FormID int FK
Form
FormID int PK
FormName varchar(20)
Courses
CourseID int PK
CourseProductID int FK
CoordinatorID int FK
StartDate smalldatetime
EndDate smalldatetime
Staff
StaffID int PK
FirstName varchar(50)
LastName varchar(50)
BirthDate date
EmploymentDate date
CityID int FK
Address varchar(50)
Email varchar(50)
Phone varchar(25)
Cities
CityID int PK
CityName varchar(50)
CountryID int FK
Countries
CountryID int PK
CountryName varchar(50)
StationarySubjectMeeting
SSubjectID int PK
StudiesMeetingProductID int FK
SubjectID int FK
Date smalldatetime
Duration time(0)
PlaceOfMeetingID int FK
Limit int
OnLineSubjectMeeting
OLSubjectID int PK
StudiesMeetingProductID int FK
SubjectID int FK
Date smalldatetime
Duration time(0)
LinkToOnlineMeeting varchar(100)
Limit int
CourseModule
ModuleID int PK
CourseID int FK
TeacherID int FK
FormID int FK
Teachers
TeacherID int PK
StaffID int FK
StationaryAttendanceListStudies
AttendanceID int PK
SSubjectID int FK
UserID int FK
Presence bit
MakeUpMeeting bit N
PlaceOfMeeting
PlaceOfMeetingId int PK
PlaceName varchar(50)
AmountOfSpace int
OnLineAttendanceListStudies
AttendanceID int PK
OLSubjectID int FK
UserID int FK
Presence bit
MakeUpMeeting bit N
StationaryModuleMeeting
StationaryModuleID int PK
ModuleID int FK
Date smalldatetime
Duration time(0)
PlaceOfMeetingID int FK
OnlineAsyncModuleMeeting
OnlineAsyncID int PK
ModuleID int FK
AddDate smalldatetime
Duration time(0)
RecordingLink varchar(100) N
OnlineSyncModuleMeeting
OnlineSyncID int PK
ModuleID int FK
Date smalldatetime
Duration time(0)
LinkToOnlineMeeting varchar(100)
RecordingLink varchar(100) N
TranslatorID int N FK
StationaryAttendanceListModule
AttendanceID int PK
StationaryModuleID int FK
UserID int FK
Presence bit
OnlineAsyncAttendanceListModule
AttendanceID int PK
OnlineAsyncID int FK
UserID int FK
Watched bit
OnlineSyncAttendanceListModule
AttendanceID int PK
OnlineSyncID int FK
UserID int FK
Presence bit

Opisy tabel

Tabela Users

Określa użytkowników bazy danych

Klucz główny: UserID
Klucze obce: CityID (z City)

UserID – Unikalne ID użytkownika
FirstName – imię użytkownika
LastName – nazwisko użytkownika
BirthDate – data urodzenia użytkownika
Phone – numer telefonu użytkownika
Email - email użytkownika
Address – adres zamieszkania użytkownika
CityID – ID miasta, z którego pochodzi użytkownik

```
-- Table: Users
CREATE TABLE Users (
    UserID int  NOT NULL,
    FirstName varchar(50)  NOT NULL,
    LastName varchar(50)  NOT NULL,
    BirthDate date  NOT NULL,
    Phone varchar(25)  NOT NULL,
    Email varchar(50)  NOT NULL,
    Address varchar(50)  NOT NULL,
    CityID int  NOT NULL,
    CONSTRAINT UsersEmailValid CHECK (Email LIKE '%@%' + '.' +
'%'),
    CONSTRAINT UsersPhoneValid CHECK (Phone LIKE
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
    CONSTRAINT UsersBirthDateValid CHECK (BirthDate <
GETDATE()),
    CONSTRAINT UsersEmailUnique UNIQUE (Email),
    CONSTRAINT UsersPhoneUnique UNIQUE (Phone),
    CONSTRAINT Users_pk PRIMARY KEY  (UserID)
);


-- Reference: Users_Cities (table: Users)
ALTER TABLE Users ADD CONSTRAINT Users_Cities
    FOREIGN KEY (CityID)
    REFERENCES Cities (CityID);
```

Tabela Staff

Zawiera dane wspólne dla pracowników bazy danych

Klucz główny: StaffID
Klucze obce: CityID (z City)

StaffID – unikalne ID pracownika
FirstName – imię pracownika
LastName – nazwisko pracownika
BirthDate – data urodzenia pracownika
EmploymentDate – data zatrudnienia pracownika
CityID – ID miasta, z którego pochodzi pracownik
Phone – numer telefonu pracownika
Address – adres zamieszkania pracownika
Email - email pracownika

```
CREATE TABLE Staff (
   StaffID int  NOT NULL,
   FirstName varchar(50)  NOT NULL,
   LastName varchar(50)  NOT NULL,
   BirthDate date  NOT NULL,
   EmploymentDate date  NOT NULL,
   CityID int  NOT NULL,
   Address varchar(50)  NOT NULL,
   Email varchar(50)  NOT NULL,
   Phone varchar(25)  NOT NULL,
   CONSTRAINT StaffEmailValid CHECK (Email LIKE '%@%' + '.' +
'%'),
   CONSTRAINT StaffPhoneValid CHECK (Phone LIKE
'+[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'),
   CONSTRAINT StaffUniqueEmail UNIQUE(Email),
   CONSTRAINT StaffUniquePhone UNIQUE(Phone),
   CONSTRAINT Staff_pk PRIMARY KEY  (StaffID)
);

-- Reference: Staff_Cities (table: Staff)
ALTER TABLE Staff ADD CONSTRAINT Staff_Cities
   FOREIGN KEY (CityID)
   REFERENCES Cities (CityID);
```

Tabela Teachers

Określa wszystkich nauczycieli bazy danych

Klucz główny: TeacherID
Klucze obce: StaffID (z Staff)

TeacherID – unikalne ID nauczyciela
StaffID – ID pracownika

```
-- Table: Teachers
CREATE TABLE Teachers (
    TeacherID int  NOT NULL,
    StaffID int  NOT NULL,
    CONSTRAINT Teachers_pk PRIMARY KEY  (TeacherID)
);
-- Reference: Staff_Teachers (table: Teachers)
ALTER TABLE Teachers ADD CONSTRAINT Staff_Teachers
    FOREIGN KEY (StaffID)
    REFERENCES Staff (StaffID);
```

Tabela Translators

Określa wszystkich tłumaczy bazy danych

Klucz główny: TranslatorID
Klucze obce: StaffID (z Staff), LanguageID (z Languages)

TranslatorID – unikalne ID tłumacza
TeacherID –ID nauczyciela
StaffID – ID pracownika

```
-- Table: Translators
CREATE TABLE Translators (
    TranslatorID int  NOT NULL,
    StaffID int  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT Translators_pk PRIMARY KEY  (TranslatorID)
);
-- Reference: Translator_Languages (table: Translators)
ALTER TABLE Translators ADD CONSTRAINT Translator_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);


-- Reference: Translator_Staff (table: Translators)
ALTER TABLE Translators ADD CONSTRAINT Translator_Staff
    FOREIGN KEY (StaffID)
    REFERENCES Staff (StaffID);
```

Tabela Languages

Zawiera informacje o językach

Klucz główny: LanguageID

LanguageID – unikalne ID języka
Language – nazwa języka

```
CREATE TABLE Languages (

   LanguageID int  NOT NULL,

   Language varchar(20)  NOT NULL,

   CONSTRAINT Languages_pk PRIMARY KEY  (LanguageID)

);
```

Tabela Cities

Zawiera informacje o miastach

Klucz główny: CityID
Klucze obce: CountryID (z Country)

CityID – unikalne ID miasta
CityName – nazwa miasta
CountryID – ID państwa

```
-- Table: Cities

CREATE TABLE Cities (

   CityID int  NOT NULL,

   CityName varchar(50)  NOT NULL,

   CountryID int  NOT NULL,

   CONSTRAINT Cities_pk PRIMARY KEY  (CityID)

);

-- Reference: Cities_Countries (table: Cities)

ALTER TABLE Cities ADD CONSTRAINT Cities_Countries

   FOREIGN KEY (CountryID)

   REFERENCES Countries (CountryID);
```

Tabela Countries

Zawiera informacje o państwach

Klucz główny: CountryID

CountryID – unikalne ID państwa
CountryName – nazwa państwa

```
-- Table: Countries

CREATE TABLE Countries (

   CountryID int  NOT NULL,

   CountryName varchar(50)  NOT NULL,

   CONSTRAINT Countries_pk PRIMARY KEY  (CountryID),

   CONSTRAINT CountryNameUnique UNIQUE (CountryName)

);
```

Tabela Webinars

Zawiera informacje o webinarach

Klucz główny: WebinarID
Klucze obce: WebinarProductID (z WebinarProducts), CoordinatorID (z Teachers), LanguageID (z Languages)

WebinarID – unikalne ID webinaru
WebinarProductID – ID produktu, do którego przypisany jest webinar
CoordinatorID – ID koordynatora webinaru
LanguageID – ID języka, w jakim prowadzony jest webinar

```
-- Table: Webinars
CREATE TABLE Webinars (
    WebinarID int  NOT NULL,
    WebinarProductID int  NOT NULL,
    CoordinatorID int  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT Webinars_pk PRIMARY KEY  (WebinarID)
);

-- Reference: Webinars_Coordinator (table: Webinars)
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Coordinator
    FOREIGN KEY (CoordinatorID)
    REFERENCES Teachers (TeacherID);

-- Reference: Webinars_Languages (table: Webinars)
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);

-- Reference: Webinars_WebinarProduct (table: Webinars)
ALTER TABLE Webinars ADD CONSTRAINT Webinars_WebinarProduct
    FOREIGN KEY (WebinarProductID)
    REFERENCES WebinarProduct (WebinarProductID);
```

Tabela WebinarProduct

Zawiera informacje o cenie webinarów

Klucz główny: WebinarProductID
Klucze obce: ProductID (z tabeli Products)

WebinarProductID - ID webinar produktu
ProductID - ID produktu
WebinarPrice - cena webinaru

```
-- Table: WebinarProduct

CREATE TABLE WebinarProduct (

   WebinarProductID int  NOT NULL,

   ProductID int  NOT NULL,

   WebinarPrice money  NOT NULL DEFAULT 0,

   CONSTRAINT WPPriceValid CHECK (WebinarPrice>=0),

   CONSTRAINT WebinarProduct_pk PRIMARY KEY  (WebinarProductID)

);

-- Reference: WebinarProduct_Products (table: WebinarProduct)

ALTER TABLE WebinarProduct ADD CONSTRAINT WebinarProduct_Products

   FOREIGN KEY (ProductID)

   REFERENCES Products (ProductID);
```

Tabela WebinarMeeting

Zawiera informacje o każdym spotkaniu dla danego webinara

Klucz główny: WebinarMeetingID
Klucze obce: WebinarID (z Webinars), TeacherID (z Teachers), TranslatorID (z Translator)

WebinarMeetingID – ID spotkania
WebinarID –ID webinaru
TeacherID – ID nauczyciela, który prowadzi webinar
StartDate – data rozpoczęcia się webinaru
Duration – czas trwania webinaru
Link – link do spotkania na webinar
RecordingLink – link do nagrania spotkania
TranslatorID – ID translatora, który może tłumaczyć

```
-- Table: WebinarMeeting

CREATE TABLE WebinarMeeting (

   WebinarMeetingID int  NOT NULL,

   WebinarID int  NOT NULL,

   TeacherID int  NOT NULL,

   StartDate smalldatetime  NOT NULL,

   Duration time(0)  NOT NULL,

   Link varchar(100)  NOT NULL,

   RecordingLink varchar(100)  NULL,

   TranslatorID int  NULL,

   CONSTRAINT WMLinkValid CHECK (Link LIKE 'http://%' or Link LIKE
'https://%'),

   CONSTRAINT WMLinkRecordingValid CHECK (RecordingLink LIKE
'http://%' or RecordingLink LIKE 'https://%'),

   CONSTRAINT WMDurationValid CHECK (Duration>'00:00:00'),

   CONSTRAINT WMStartDate CHECK (StartDate>=GETDATE()),

   CONSTRAINT WebinarMeeting_pk PRIMARY KEY  (WebinarMeetingID)

);
```

```sql
-- Reference: WebinarMeeting_Teachers (table: WebinarMeeting)
ALTER TABLE WebinarMeeting ADD CONSTRAINT WebinarMeeting_Teachers
    FOREIGN KEY (TeacherID)
    REFERENCES Teachers (TeacherID);

-- Reference: WebinarMeeting_Translator (table: WebinarMeeting)
ALTER TABLE WebinarMeeting ADD CONSTRAINT
WebinarMeeting_Translator
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);

-- Reference: WebinarMeeting_Webinars (table: WebinarMeeting)
ALTER TABLE WebinarMeeting ADD CONSTRAINT WebinarMeeting_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES Webinars (WebinarID);
```

Tabela Courses

Zawiera informacje o kursach znajdujących się w ofercie

Klucz główny: CourseID
Klucze obce: CoordinatorID (z Teachers (TeacherID)), ProductID (z Products), FormID (z Form)

CourseID - ID kursu
CourseProductID - ID produktu, do którego przypisany jest kurs
CoordinatorID - ID koordynatora
StartDate - data rozpoczęcia kursu
EndDate - data zakończenia kursu

```
-- Table: Courses
CREATE TABLE Courses (
    CourseID int  NOT NULL,
    CourseProductID int  NOT NULL,
    CoordinatorID int  NOT NULL,
    StartDate smalldatetime  NOT NULL,
    EndDate smalldatetime  NOT NULL,
    CONSTRAINT CourseDateValid CHECK (StartDate<EndDate),
    CONSTRAINT Courses_pk PRIMARY KEY  (CourseID),
    CONSTRAINT CoursesValidDate CHECK (StartDate>=GETDATE())
);


-- Reference: CoursesProduct_Products (table: CoursesProduct)
ALTER TABLE CoursesProduct ADD CONSTRAINT CoursesProduct_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);

-- Reference: Courses_CoursesProduct (table: Courses)
ALTER TABLE Courses ADD CONSTRAINT Courses_CoursesProduct
    FOREIGN KEY (CourseProductID)
    REFERENCES CoursesProduct (CourseProductID);

-- Reference: Courses_Teachers (table: Courses)
ALTER TABLE Courses ADD CONSTRAINT Courses_Teachers
    FOREIGN KEY (CoordinatorID)
    REFERENCES Teachers (TeacherID);
```

Tabela CoursesProduct

Zawiera informacje o cenie kursów

Klucz główny: CourseProductID
Klucze obce: ProductID (z Products)

CourseProductID - ID kurs produktu
ProductID - ID produktu
Price - cena za cały kurs
PriceAdvance - kwota zaliczki, żeby zamówić produkt

```
-- Table: CoursesProduct
CREATE TABLE CoursesProduct (
    CourseProductID int  NOT NULL,
    ProductID int  NOT NULL,
    Price money  NOT NULL,
    PriceAdvance money  NOT NULL,
    CONSTRAINT CoursePriceValid CHECK (Price>0),
    CONSTRAINT CoursePriceAdvanceValid CHECK (PriceAdvance>0),
    CONSTRAINT CoursesProduct_pk PRIMARY KEY  (CourseProductID)
);
-- Reference: CoursesProduct_Products (table: CoursesProduct)
ALTER TABLE CoursesProduct ADD CONSTRAINT CoursesProduct_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

Tabela CourseModule

Zawiera moduły przypisane do danych kursów

Klucz główny: ModuleID
Klucze obce: CourseID (z Courses), TeacherID (z Teachers), FormID (z Form)

ModuleID - ID kurs produktu
CourseID - ID kursu
TeacherID - ID nauczyciela
Form - forma danego modulu

```
CREATE TABLE CourseModule (
    ModuleID int  NOT NULL,
    CourseID int  NOT NULL,
    TeacherID int  NOT NULL,
    FormID int  NOT NULL,
    CONSTRAINT CourseModule_pk PRIMARY KEY  (ModuleID)
);

-- Reference: CourseModule_Form (table: CourseModule)
ALTER TABLE CourseModule ADD CONSTRAINT CourseModule_Form
    FOREIGN KEY (FormID)
    REFERENCES Form (FormID);
-- Reference: OnLineModuleSync_Courses (table: CourseModule)
ALTER TABLE CourseModule ADD CONSTRAINT OnLineModuleSync_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
-- Reference: Teachers_CourseModule (table: CourseModule)
ALTER TABLE CourseModule ADD CONSTRAINT Teachers_CourseModule
    FOREIGN KEY (TeacherID)
    REFERENCES Teachers (TeacherID);
```

Tabela OnlineAsyncModuleMeeting

Zawiera listę spotkań asynchronicznych danego modułu

Klucz główny: OnlineAsyncID
Klucze obce: ModuleID (z CourseModule)

OnlineAsyncID - ID spotkania asynchronicznego
ModuleID - ID modułu
AddDate - data dodania
Duration - czas trwania nagrania
RecordingLink - nagranie do oglądnięcia

```
CREATE TABLE OnlineAsyncModuleMeeting (

   OnlineAsyncID int  NOT NULL,

   ModuleID int  NOT NULL,

   AddDate smalldatetime  NOT NULL Default GETDATE(),

   Duration time(0)  NOT NULL,

   RecordingLink varchar(100)  NULL,

   CONSTRAINT OAMMDurationValid CHECK (Duration>'00:00:00'),

   CONSTRAINT OAMMAddDateValid CHECK (AddDate>=GETDATE()),

   CONSTRAINT OnlineAsyncModuleMeeting_pk PRIMARY KEY
(OnlineAsyncID)

);

-- Reference: OnlineAsyncModuleMeeting_CourseModule (table:
OnlineAsyncModuleMeeting)

ALTER TABLE OnlineAsyncModuleMeeting ADD CONSTRAINT
OnlineAsyncModuleMeeting_CourseModule

   FOREIGN KEY (ModuleID)

   REFERENCES CourseModule (ModuleID);
```

Tabela OnlineAsyncAttendanceListModule

Lista obecności osób, które obejrzały dane nagranie

Klucz główny: AttendanceID
Klucze obce: UserID (z Users), OnlineAsyncID (z OnlineAsyncModuleMeeting)

AttendanceID - ID uczestnictwa
OnlineAsyncID - ID spotkania
UserID - ID uczesnika
Watched - informacja czy osoba oglądnęła nagranie

```
CREATE TABLE OnlineAsyncAttendanceListModule (
    AttendanceID int  NOT NULL,
    OnlineAsyncID int  NOT NULL,
    UserID int  NOT NULL,
    Watched bit  NOT NULL DEFAULT 0,
    CONSTRAINT OnlineAsyncAttendanceListModule_pk PRIMARY KEY
(AttendanceID)
);

-- Reference:
OnlineAsyncAttendanceListModule_OnlineAsyncModuleMeeting (table:
OnlineAsyncAttendanceListModule)
ALTER TABLE OnlineAsyncAttendanceListModule ADD CONSTRAINT
OnlineAsyncAttendanceListModule_OnlineAsyncModuleMeeting
    FOREIGN KEY (OnlineAsyncID)
    REFERENCES OnlineAsyncModuleMeeting (OnlineAsyncID);

-- Reference: OnlineAsyncAttendanceListModule_Users (table:
OnlineAsyncAttendanceListModule)
ALTER TABLE OnlineAsyncAttendanceListModule ADD CONSTRAINT
OnlineAsyncAttendanceListModule_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Tabela OnlineSyncModuleMeeting

Zawiera listę spotkań synchronicznych danego modułu

Klucz główny: OnlineSyncID
Klucze obce: ModuleID (z CourseModule), TranslatorID (z Translators)

OnlineSyncID - ID spotkania synchronicznego
ModuleID - ID modułu
Date - data spotkania
Duration - czas trwania spotkania
LinkToOnlineMeeting - link do spotkania
RecordingLink - nagranie zapisane po spotkaniu
TranslatorID - ID translatora

```
-- Table: OnlineSyncModuleMeeting
CREATE TABLE OnlineSyncModuleMeeting (
    OnlineSyncID int  NOT NULL,
    ModuleID int  NOT NULL,
    Date smalldatetime  NOT NULL,
    Duration time(0)  NOT NULL,
    LinkToOnlineMeeting varchar(100)  NOT NULL,
    RecordingLink varchar(100)  NULL,
    TranslatorID int  NULL,
    CONSTRAINT WMLinkValid CHECK (LinkToOnlineMeeting LIKE 'http://%'
or LinkToOnlineMeeting LIKE 'https://%'),
    CONSTRAINT WMLinkRecordingValid CHECK (RecordingLink LIKE
'http://%' or RecordingLink LIKE 'https://%'),
    CONSTRAINT OSMMDurationValid CHECK (Duration>'00:00:00'),
    CONSTRAINT OnlineSyncModuleMeeting_pk PRIMARY KEY  (OnlineSyncID)
);

-- Reference: OnlineSyncModuleMeeting_CourseModule (table:
OnlineSyncModuleMeeting)
ALTER TABLE OnlineSyncModuleMeeting ADD CONSTRAINT
OnlineSyncModuleMeeting_CourseModule
    FOREIGN KEY (ModuleID)
    REFERENCES CourseModule (ModuleID);

-- Reference: Translators_OnlineSyncModuleMeeting (table:
OnlineSyncModuleMeeting)
ALTER TABLE OnlineSyncModuleMeeting ADD CONSTRAINT
Translators_OnlineSyncModuleMeeting
    FOREIGN KEY (TranslatorID)
    REFERENCES Translators (TranslatorID);
```

Tabela OnlineSyncAttendanceListModule

Lista obecności osób, które uczestniczyły w danym spotkaniu modułu

Klucz główny: AttendanceID
Klucze obce: UserID (z Users), OnlineSyncID (z OnlineAsyncModuleMeeting)

AttendanceID - ID uczestnictwa
OnlineSyncID - ID spotkania
UserID - ID uczesnika
Presence - informacja czy osoba była na spotkaniu

```
-- Table: OnlineSyncAttendanceListModule

CREATE TABLE OnlineSyncAttendanceListModule (

   AttendanceID int  NOT NULL,

   OnlineSyncID int  NOT NULL,

   UserID int  NOT NULL,

   Presence bit  NOT NULL DEFAULT 0,

   CONSTRAINT OnlineSyncAttendanceListModule_pk PRIMARY KEY
(AttendanceID)

);

-- Reference: OnlineSyncAttendanceListModule_OnlineSyncModuleMeeting
(table: OnlineSyncAttendanceListModule)

ALTER TABLE OnlineSyncAttendanceListModule ADD CONSTRAINT
OnlineSyncAttendanceListModule_OnlineSyncModuleMeeting

   FOREIGN KEY (OnlineSyncID)

   REFERENCES OnlineSyncModuleMeeting (OnlineSyncID);



-- Reference: OnlineSyncAttendanceListModule_Users (table:
OnlineSyncAttendanceListModule)

ALTER TABLE OnlineSyncAttendanceListModule ADD CONSTRAINT
OnlineSyncAttendanceListModule_Users

   FOREIGN KEY (UserID)

   REFERENCES Users (UserID);
```

Tabela StationaryModuleMeeting

Zawiera listę spotkań stacjonarnych danego modułu
Klucz główny: StationaryModuleID
Klucze obce: ModuleID (z CourseModule), PlaceOfMeetingID (z PlaceOfMeeting)

StationaryModuleID - ID spotkania stacjonarnego
ModuleID - ID modułu
Date - data spotkania
Duration - czas trwania spotkania
PlaceOfMeetingID - ID miejsca spotkania

```
-- Table: StationaryModuleMeeting
CREATE TABLE StationaryModuleMeeting (
    StationaryModuleID int  NOT NULL,
    ModuleID int  NOT NULL,
    Date smalldatetime  NOT NULL,
    Duration time(0)  NOT NULL,
    PlaceOfMeetingID int  NOT NULL,
    CONSTRAINT SMMDurationValid CHECK (Duration>'00:00:00'),
    CONSTRAINT SMMDateValid CHECK (Date>GETDATE()),
    CONSTRAINT StationaryModuleMeeting_pk PRIMARY KEY
(StationaryModuleID)
);
-- Reference: PlaceOfMeeting_StationaryModuleMeeting (table:
StationaryModuleMeeting)
ALTER TABLE StationaryModuleMeeting ADD CONSTRAINT
PlaceOfMeeting_StationaryModuleMeeting
    FOREIGN KEY (PlaceOfMeetingID)
    REFERENCES PlaceOfMeeting (PlaceOfMeetingID);

-- Reference: StationaryModule_CourseModule (table:
StationaryModuleMeeting)
ALTER TABLE StationaryModuleMeeting ADD CONSTRAINT
StationaryModule_CourseModule
    FOREIGN KEY (ModuleID)
    REFERENCES CourseModule (ModuleID);
```

Tabela StationaryAttendanceListModule

Lista obecności osób, które uczestniczyły w danym spotkaniu modułu

Klucz główny: AttendanceID
Klucze obce: UserID (z Users), StationaryModuleID (z OnlineAsyncModuleMeeting)

AttendanceID - ID uczestnictwa
StationaryModuleID - ID spotkania
UserID - ID uczesnika
Presence - informacja czy osoba była na spotkaniu

```
-- Table: StationaryAttendanceListModule
CREATE TABLE StationaryAttendanceListModule (
    AttendanceID int  NOT NULL,
    StationaryModuleID int  NOT NULL,
    UserID int  NOT NULL,
    Presence bit  NOT NULL DEFAULT 0,
    CONSTRAINT StationaryAttendanceListModule_pk PRIMARY KEY
(AttendanceID)
);
-- Reference: StationaryAttendanceListModule_StationaryModuleMeeting
(table: StationaryAttendanceListModule)
ALTER TABLE StationaryAttendanceListModule ADD CONSTRAINT
StationaryAttendanceListModule_StationaryModuleMeeting
    FOREIGN KEY (StationaryModuleID)
    REFERENCES StationaryModuleMeeting (StationaryModuleID);

-- Reference: StationaryAttendanceListModule_Users (table:
StationaryAttendanceListModule)
ALTER TABLE StationaryAttendanceListModule ADD CONSTRAINT
StationaryAttendanceListModule_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Tabela Studies

Zawiera informacje o studiach znajdujących się w ofercie

Klucz główny: StudyID
Klucze obce: StudiesProductID (z StudiesProduct), CoordinatorID (z Teachers)
StudyID - ID studiów
StudiesProductID - ID produktu, do którego przypisany jest studiów
CoordinatorID - ID koordynatora studiów
StartDate - data rozpoczęcia studiów
EndDate - data zakończenia studiów
Limit - maksymalna liczba studentów

```
-- Table: Studies
CREATE TABLE Studies (
    StudyID int  NOT NULL,
    StudiesProductID int  NOT NULL,
    CoordinatorID int  NOT NULL,
    StartDate smalldatetime  NOT NULL,
    EndDate smalldatetime  NOT NULL,
    Limit int  NOT NULL,
    CONSTRAINT StudiesLimitValid CHECK (Limit>0),
    CONSTRAINT StudiesStartDateValid CHECK (StartDate>=GETDATE()),
    CONSTRAINT StudiesDateValid CHECK (StartDate>EndDate),
    CONSTRAINT Studies_pk PRIMARY KEY  (StudyID)
);

-- Reference: Coordinator_Teachers (table: Studies)
ALTER TABLE Studies ADD CONSTRAINT Coordinator_Teachers
    FOREIGN KEY (CoordinatorID)
    REFERENCES Teachers (TeacherID);
-- Reference: Studies_StudiesProduct (table: Studies)
ALTER TABLE Studies ADD CONSTRAINT Studies_StudiesProduct
    FOREIGN KEY (StudiesProductID)
    REFERENCES StudiesProduct (StudiesProductID);
```

Tabela StudiesProduct

Zawiera informacje o cenie studiów

Klucz główny: StudiesProductID
Klucze obce: ProductID (z Products)

StudiesProductID - ID studiów produktu
ProductID - ID produktu
Price - cena za całe studia
Entry_fee - czesne potrzebne do zapisania się na studia

```
-- Table: StudiesProduct
CREATE TABLE StudiesProduct (
    StudiesProductID int  NOT NULL,
    ProductID int  NOT NULL,
    Price money  NOT NULL,
    Entry_fee money  NOT NULL,
    CONSTRAINT StudiesPriceValid CHECK (Price>0),
    CONSTRAINT Entry_feeValid CHECK (Entry_fee>0),
    CONSTRAINT StudiesProduct_pk PRIMARY KEY  (StudiesProductID)
);
-- Reference: StudiesProduct_Products (table: StudiesProduct)
ALTER TABLE StudiesProduct ADD CONSTRAINT StudiesProduct_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

Tabela StudiesMeetingProduct

Zawiera informacje o cenie za pojedyńcze spotkanie na studium

Klucz główny: StudiesMeetingProductID
Klucze obce: ProductID (z Products)

StudiesMeetingProductID - ID pojedyńczego spotkania produktu
ProductID - ID produktu
PriceforStudents - cena dla osób uczestniczących stale na zajęcia
PriceforOthers - cena dla pozostałych osób

```
-- Table: StudiesMeetingProduct

CREATE TABLE StudiesMeetingProduct (

   StudiesMeetingProductID int  NOT NULL,

   ProductID int  NOT NULL,

   PriceforStudents money  NOT NULL,

   PriceforOthers money  NOT NULL,

   CONSTRAINT StudiesMeetingPriceSValid CHECK (PriceforStudents>0),

   CONSTRAINT StudiesMeetingPriceOValid CHECK (PriceforOthers>0),

   CONSTRAINT StudiesMeetingProduct_pk PRIMARY KEY
(StudiesMeetingProductID)

);

-- Reference: StudiesMeetingProduct_Products (table:
StudiesMeetingProduct)

ALTER TABLE StudiesMeetingProduct ADD CONSTRAINT
StudiesMeetingProduct_Products

   FOREIGN KEY (ProductID)

   REFERENCES Products (ProductID);
```

Tabela Subjects

Zawiera informacje o wszystkich przedmiotach wchodzących w skład studiów

Klucz główny: SubjectID
Klucze obce: StudyID (z Studies), FormID (z Form)
SubjectID - ID przedmiotu
StudyID - ID studiów do których należy przedmiot
SubjectName - nazwa przedmiotu
TeacherID - ID nauczyciela prowadzącego przedmiot
FormID - forma przedmiotu

```
-- Table: Subjects
CREATE TABLE Subjects (
    SubjectID int  NOT NULL,
    StudyID int  NOT NULL,
    SubjectName varchar(50)  NOT NULL,
    TeacherID int  NOT NULL,
    FormID int  NOT NULL,
    CONSTRAINT SubjectID PRIMARY KEY  (SubjectID)
);

-- Reference: Subject_Studies (table: Subjects)
ALTER TABLE Subjects ADD CONSTRAINT Subject_Studies
    FOREIGN KEY (StudyID)
    REFERENCES Studies (StudyID);

-- Reference: Subjects_Form (table: Subjects)
ALTER TABLE Subjects ADD CONSTRAINT Subjects_Form
    FOREIGN KEY (FormID)
    REFERENCES Form (FormID);
```

Tabela StationarySubjectMeeting

Zawiera informacje o lekcjach stacjonarnych na studiach
Klucz główny: SSubjectID
Klucze obce: SubjectID (z Subjects), StudiesMeetingProductID (z StudiesMeetingProduct), PlaceOfMeetingID (z PlaceOfMeeting)
SSubjectID - ID spotkania
StudiesMeetingProductID - ID spotkania produktu
SubjectID - ID przedmiotu w ramach którego odbywa się spotkanie
Date - Data spotkania
Duration - czas trwania spotkania
PlaceOfMeetingID - ID miejsca spotkania
Limit - limit osób na spotkaniu

```
-- Table: StationarySubjectMeeting
CREATE TABLE StationarySubjectMeeting (
    SSubjectID int  NOT NULL,
    StudiesMeetingProductID int  NOT NULL,
    SubjectID int  NOT NULL,
    Date smalldatetime  NOT NULL,
    Duration time(0)  NOT NULL,
    PlaceOfMeetingID int  NOT NULL,
    Limit int  NOT NULL,
    CONSTRAINT SSMDurationValid CHECK (Duration>'00:00:00'),
    CONSTRAINT SSMLimitValid CHECK (Limit>0),
    CONSTRAINT SSMDateValid CHECK (Date>GETDATE()),
    CONSTRAINT StationarySubjectMeeting_pk PRIMARY KEY
(SSubjectID)
);

-- Reference: PlaceOfMeeting_StationarySubjectMeeting (table:
StationarySubjectMeeting)
ALTER TABLE StationarySubjectMeeting ADD CONSTRAINT
PlaceOfMeeting_StationarySubjectMeeting
    FOREIGN KEY (PlaceOfMeetingID)
    REFERENCES PlaceOfMeeting (PlaceOfMeetingID);
-- Reference: StationarySubjectMeeting_StudiesMeetingProduct
(table: StationarySubjectMeeting)
ALTER TABLE StationarySubjectMeeting ADD CONSTRAINT
StationarySubjectMeeting_StudiesMeetingProduct
    FOREIGN KEY (StudiesMeetingProductID)
    REFERENCES StudiesMeetingProduct (StudiesMeetingProductID);

-- Reference: StationarySubjectMeeting_Subjects (table:
StationarySubjectMeeting)
ALTER TABLE StationarySubjectMeeting ADD CONSTRAINT
StationarySubjectMeeting_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES Subjects (SubjectID);
```

Tabela OnlineSubjectMeeting

Zawiera informacje o lekcjach zdalnych na studiach

Klucz główny: OLSubjectID
Klucze obce: SubjectID (z Subject), StudiesMeetingProductID (z StudiesMeetingProduct),
OLSubjectID - ID spotkania
StudiesMeetingProductID - ID spotkania produktu
SubjectID - ID przedmiotu w ramach którego odbywa się spotkanie
Date - data spotkania
Duration - czas trwania spotkania
LinkToOnlineMeeting - link do spotkania
Limit - limit osób na spotkaniu

```
-- Table: OnLineSubjectMeeting
CREATE TABLE OnLineSubjectMeeting (
    OLSubjectID int  NOT NULL,
    StudiesMeetingProductID int  NOT NULL,
    SubjectID int  NOT NULL,
    Date smalldatetime  NOT NULL,
    Duration time(0)  NOT NULL,
    LinkToOnlineMeeting varchar(100)  NOT NULL,
    Limit int  NOT NULL,
    CONSTRAINT OSMLimit CHECK (Limit>0),
    CONSTRAINT OSMDurationValid CHECK (Duration>'00:00:00'),
    CONSTRAINT OnLineSubjectMeeting_pk PRIMARY KEY  (OLSubjectID)
);

-- Reference: OnLineSubjectMeeting_StudiesMeetingProduct (table:
OnLineSubjectMeeting)
ALTER TABLE OnLineSubjectMeeting ADD CONSTRAINT
OnLineSubjectMeeting_StudiesMeetingProduct
    FOREIGN KEY (StudiesMeetingProductID)
    REFERENCES StudiesMeetingProduct (StudiesMeetingProductID);

-- Reference: OnLineSubjectMeeting_Subjects (table:
OnLineSubjectMeeting)
ALTER TABLE OnLineSubjectMeeting ADD CONSTRAINT
OnLineSubjectMeeting_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES Subjects (SubjectID);
```

Tabela StationaryAttendanceListStudies

Lista obecności dla stacjonarnych spotkań studiów
klucz główny: AttendanceID
klucze obce: SSubjectID (z StationarySubjectMeeting), UserID (z Users)

AttendanceID - ID listy obecności
SSubjectID - stacjonarne zajęcia, do których lista jest przypisana
UserID - ID studenta
Presence - czy student był obecny
MakeUpMeeting - czy student odrobił zajęcia

```
-- Table: StationaryAttendanceListStudies
CREATE TABLE StationaryAttendanceListStudies (
    AttendanceID int  NOT NULL,
    SSubjectID int  NOT NULL,
    UserID int  NOT NULL,
    Presence bit  NOT NULL DEFAULT 0,
    MakeUpMeeting bit  NULL,
    CONSTRAINT StationaryAttendanceListStudies_pk PRIMARY KEY
(AttendanceID)
);


-- Reference:
StationaryAttendanceListStudies_StationarySubjectMeeting (table:
StationaryAttendanceListStudies)
ALTER TABLE StationaryAttendanceListStudies ADD CONSTRAINT
StationaryAttendanceListStudies_StationarySubjectMeeting
    FOREIGN KEY (SSubjectID)
    REFERENCES StationarySubjectMeeting (SSubjectID);

-- Reference: StationaryAttendanceListStudies_Users (table:
StationaryAttendanceListStudies)
ALTER TABLE StationaryAttendanceListStudies ADD CONSTRAINT
StationaryAttendanceListStudies_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Tabela OnLineAttendanceListStudies

lista obecności dla stacjonarnych spotkań studiów
AttendanceID (klucz główny) - ID listy obecności
OLSubjectID (klucz obcy z OnLineSubjectMeeting) - stacjonarne zajęcia, do których lista jest przypisana
UserID (klucz obcy z Users) - ID studenta
Presence - czy student był obecny
MakeUpMeeting - czy student odrobił zajęcia

```
-- Table: OnLineAttendanceListStudies
CREATE TABLE OnLineAttendanceListStudies (
    AttendanceID int  NOT NULL,
    OLSubjectID int  NOT NULL,
    UserID int  NOT NULL,
    Presence bit  NOT NULL DEFAULT 0,
    MakeUpMeeting bit  NULL,
    CONSTRAINT OnLineAttendanceListStudies_pk PRIMARY KEY
(AttendanceID)
);

-- Reference: OnLineAttendanceListStudies_OnLineSubjectMeeting
(table: OnLineAttendanceListStudies)
ALTER TABLE OnLineAttendanceListStudies ADD CONSTRAINT
OnLineAttendanceListStudies_OnLineSubjectMeeting
    FOREIGN KEY (OLSubjectID)
    REFERENCES OnLineSubjectMeeting (OLSubjectID);

-- Reference: OnLineAttendanceListStudies_Users (table:
OnLineAttendanceListStudies)
ALTER TABLE OnLineAttendanceListStudies ADD CONSTRAINT
OnLineAttendanceListStudies_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Tabela Internships

Zawiera informacje o prowadzonych praktykach

Klucz główny: InternshipID
Klucze obce: StudyID (z Studies)
StudyID - ID studiów w ramach których odbywają się praktyki
InternshipID - ID praktyk
InternshipName - nazwa praktyk

```
-- Table: Internships
CREATE TABLE Internships (
    StudyID int  NOT NULL,
    InternshipID int  NOT NULL,
    InternshipName varchar(50)  NOT NULL,
    CONSTRAINT Internships_pk PRIMARY KEY  (InternshipID)
);

-- Reference: Internships_Studies (table: Internships)
ALTER TABLE Internships ADD CONSTRAINT Internships_Studies
    FOREIGN KEY (StudyID)
    REFERENCES Studies (StudyID);
```

Tabela AttendanceInternship

Zawiera informacje o zaliczenia praktyk dla danych studentów
Klucz główny: AttendanceID
Klucze obce: InternshipsID (z Internship), UserID (z Users)

AttendanceID (klucz główny) - ID zaliczenia praktyk
UserID (klucz obcy z Users) - ID studenta
InternshipID (klucz obcy z Internships) - ID praktyk
StartDate - data rozpoczęcia się praktyk
EndDate - data zakończenia się praktyk
Passed - czy student zaliczył praktyki

```
-- Table: AttendanceInternship
CREATE TABLE AttendanceInternship (
    AttendanceID int  NOT NULL,
    UserID int  NOT NULL,
    InternshipID int  NOT NULL,
    StartDate smalldatetime  NOT NULL,
    EndDate smalldatetime  NOT NULL,
    Passed bit  NOT NULL DEFAULT 0,
    CONSTRAINT AttendanceValidDate CHECK (StartDate>EndDate),
    CONSTRAINT AttendanceInternship_pk PRIMARY KEY  (AttendanceID),
    CONSTRAINT AttendanceValidStartDate CHECK (StartDate>=GETDATE())
);


-- Reference: AttendanceInternship_Internships (table:
AttendanceInternship)
ALTER TABLE AttendanceInternship ADD CONSTRAINT
AttendanceInternship_Internships
    FOREIGN KEY (InternshipID)
    REFERENCES Internships (InternshipID);

-- Reference: AttendanceInternship_Users (table:
AttendanceInternship)
ALTER TABLE AttendanceInternship ADD CONSTRAINT
AttendanceInternship_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Tabela Form

Opisuje formy zajęć
Klucz główny: FormID

FormID - ID formy zajęć
FormName - nazwa formy zajęć

```
-- Table: Form
CREATE TABLE Form (
    FormID int  NOT NULL,
    FormName varchar(20)  NOT NULL,
    CONSTRAINT Form_pk PRIMARY KEY  (FormID)
    CONSTRAINT FormNameUnique UNIQUE (FormName)
);
```

Tabela Exam

Zawiera informacje o danym egzaminie końcowym dla danych studiów

Klucz główny: ExamID
Klucze obce: StudyID (z Study)

ExamID –ID egzaminu końcowego
StudyID – ID studiów
Date – data odbycia się egzaminu

```
-- Table: Exam
CREATE TABLE Exam (
    ExamID int  NOT NULL,
    StudyID int  NOT NULL,
    Date smalldatetime  NOT NULL,
    CONSTRAINT Exam_pk PRIMARY KEY  (ExamID),
    CONSTRAINT CoursesValidDate CHECK (Date>GETDATE())
);

-- Reference: Exam_Studies (table: Exam)
ALTER TABLE Exam ADD CONSTRAINT Exam_Studies
    FOREIGN KEY (StudyID)
    REFERENCES Studies (StudyID);
```

Tabela ExamScore

Zawiera wyniki egzaminu
Klucz główny: ExamScoreID
Klucze obce: UserID (z Users), ExamID (z Exam), GradeID (z Grade)

ExamScoreID - ID wyniku
UserID - ID studenta
ExamID - ID egzaminu
GradeID - ID oceny

```
-- Table: ExamScore
CREATE TABLE ExamScore (
    ExamScoreID int  NOT NULL,
    UserID int  NOT NULL,
    ExamID int  NOT NULL,
    GradeID int  NOT NULL,
    CONSTRAINT ExamScore_pk PRIMARY KEY  (ExamScoreID)
);


-- Reference: ExamScore_Exam (table: ExamScore)
ALTER TABLE ExamScore ADD CONSTRAINT ExamScore_Exam
    FOREIGN KEY (ExamID)
    REFERENCES Exam (ExamID);

-- Reference: ExamScore_Grade (table: ExamScore)
ALTER TABLE ExamScore ADD CONSTRAINT ExamScore_Grade
    FOREIGN KEY (GradeID)
    REFERENCES Grade (GradeID);

-- Reference: ExamScore_Users (table: ExamScore)
ALTER TABLE ExamScore ADD CONSTRAINT ExamScore_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Tabela Grade

Zawiera wszystkie możliwe do uzyskania oceny z egzaminu

Klucz główny: GradeID

GradeID - ID oceny
Grade - ocena
GradeName - nazwa oceny

```
-- Table: Grade
CREATE TABLE Grade (
    GradeID int  NOT NULL,
    Grade decimal(2,1)  NOT NULL,
    GradeName varchar(20)  NOT NULL,
    CONSTRAINT GradeValid CHECK (Grade>=2 AND Grade<=5),
    CONSTRAINT Grade_pk PRIMARY KEY  (GradeID)
);
```

Tabela Products

Zawiera wszystkie produkty które są możliwe do kupienia dla użytkowników
Klucz główny: ProductID

ProductID - ID produktu
Name - nazwa produktu

```
-- Table: Products
CREATE TABLE Products (
    ProductID int  NOT NULL,
    Name varchar(50)  NOT NULL,
    CONSTRAINT Products_pk PRIMARY KEY  (ProductID)
);
```

Tabela OrderDetails

Zawiera informacje o przedmiotach kupionych w poszczególnych zamówieniach.

Klucz główny: OrderID, ProductID
Klucze obce: OrderID (z Orders), ProductID (z Products)
OrderID - ID zamówienia
ProductID - ID zamówionego produktu
Price - ilość wpłaconego za dany produkt kwoty

```
-- Table: OrderDetails
CREATE TABLE OrderDetails (
    OrderID int  NOT NULL,
    ProductID int  NOT NULL,
    Price money  NOT NULL,
    CONSTRAINT OrderDetailsPriceValid CHECK (Price>0),
    CONSTRAINT OrderDetails_pk PRIMARY KEY  (OrderID,ProductID)
);


-- Reference: OrderDetails_Orders (table: OrderDetails)
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);

-- Reference: OrderDetails_Products (table: OrderDetails)
ALTER TABLE OrderDetails ADD CONSTRAINT OrderDetails_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
```

Tabela Orders

Zawiera informacje o wszystkich zamówieniach

Klucz główny: OrderID
Klucze obce: UserID (z Users)
OrderID - ID zamówienia
UserID - ID użytkownika który złożył zamówienia
OrderDate - data złożenia zamówienia
RequiredDate - data do kiedy trzeba zapłacić
PaidDate - data zapłacenia za wszystkie produkty

```
-- Table: Orders
CREATE TABLE Orders (
    OrderID int  NOT NULL,
    UserID int  NOT NULL,
    OrderDate smalldatetime  NOT NULL DEFAULT GETDATE(),
    RequiredDate smalldatetime  NOT NULL,
    PaidDate smalldatetime  NULL,
    CONSTRAINT OrdersRequiredDateValid CHECK
(RequiredDate>OrderDate),
    CONSTRAINT Orders_pk PRIMARY KEY  (OrderID)
);

-- Reference: Transactions_Users (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT Transactions_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Tabela Exception

Zawiera informacje o zamówieniach które zostały odroczone

Klucz główny:ExceptionID
Klucze obce: OrderID (z Orders)
ExceptionID - ID odroczenia
ExceptionName - przyczyna odroczenia
OrderID - ID zamówienia którego płatność została odroczona
Date - data przesunięcia płatności

```
-- Table: Exception
CREATE TABLE Exception (
    ExceptionID int  NOT NULL,
    OrderID int  NOT NULL,
    ExceptionName varchar(100)  NOT NULL,
    Date smalldatetime  NOT NULL,
    CONSTRAINT Exception_pk PRIMARY KEY  (ExceptionID)
);


-- Reference: Exception_Orders (table: Exception)
ALTER TABLE Exception ADD CONSTRAINT Exception_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders (OrderID);
```

Tabela PlaceOfMeeting

Zawiera informacje o salach spotkań

Klucz główny: PlaceOfMeetingID

PlaceOfMeetingID - ID sali
PlaceName - nazwa sali
AmountOfSpace - ilość miejsca

```
-- Table: PlaceOfMeeting
CREATE TABLE PlaceOfMeeting (
    PlaceOfMeetingID int  NOT NULL,
    PlaceName varchar(50)  NOT NULL,
    AmountOfSpace int  NOT NULL,
    CONSTRAINT AmountOfSpaceValid CHECK (AmountOfSpace>0),
    CONSTRAINT PlaceOfMeeting_pk PRIMARY KEY  (PlaceOfMeetingID)
);
```

Widoki

AttendanceList (IS)

Wyświetla listę obecności dla zakończonych wydarzeń

```
alter view AttendanceList as
   select 'studies' as CourseType, 'stationary' as MeetingType, al.SSubjectID as
ID, (select Date
           from StationarySubjectMeeting m
           where m.SSubjectID=al.SSubjectID) 'Date',
       (select FirstName
        from Users u
        where u.UserID=al.UserID) 'FirstName',
       (select LastName
        from Users u
        where u.UserID=al.UserID) 'LastName',
       Presence
   from StationaryAttendanceListStudies al
   where (select Date
           from StationarySubjectMeeting m
           where m.SSubjectID=al.SSubjectID)<GETDATE()
   union
   select 'studies' as CourseType, 'online' as MeetingType, al.OLSubjectID as ID,
(select Date
           from OnLineSubjectMeeting m
           where m.OLSubjectID=al.OLSubjectID) 'Date',
       (select FirstName
        from Users u
        where u.UserID=al.UserID) 'FirstName',
       (select LastName
        from Users u
        where u.UserID=al.UserID) 'LastName',
       Presence
   from OnLineAttendanceListStudies al
   where (select Date
           from OnLineSubjectMeeting m
           where m.OLSubjectID=al.OLSubjectID)<GETDATE()
   union
   select 'course' as CourseType, 'stationary' as MeetingType,
al.StationaryModuleID as ID, (select Date
           from StationaryModuleMeeting m
           where m.StationaryModuleID=al.StationaryModuleID) 'Date',
       (select FirstName
        from Users u
        where u.UserID=al.UserID) 'FirstName',
       (select LastName
        from Users u
        where u.UserID=al.UserID) 'LastName',
       Presence
   from StationaryAttendanceListModule al
   where (select Date
           from StationaryModuleMeeting m
           where m.StationaryModuleID=al.StationaryModuleID)<GETDATE()
   union
   select 'course' as CourseType, 'online async' as MeetingType, al.OnlineAsyncID
as ID, (select AddDate
           from OnlineAsyncModuleMeeting m
```

```
          where m.OnlineAsyncID=al.OnlineAsyncID) 'Date',
       (select FirstName
        from Users u
        where u.UserID=al.UserID) 'FirstName',
       (select LastName
        from Users u
        where u.UserID=al.UserID) 'LastName',
       Watched
   from OnlineAsyncAttendanceListModule al
   union
   select 'course' as CourseType, 'online sync' as MeetingType, al.OnlineSyncID as
ID, (select Date
          from OnlineSyncModuleMeeting m
          where m.OnlineSyncID=al.OnlineSyncID) 'Date',
       (select FirstName
        from Users u
        where u.UserID=al.UserID) 'FirstName',
       (select LastName
        from Users u
        where u.UserID=al.UserID) 'LastName',
       al.Presence
   from OnlineSyncAttendanceListModule al
   where (select Date
          from OnlineSyncModuleMeeting m
          where m.OnlineSyncID=al.OnlineSyncID)<GETDATE()
```

AttendancePerSubject (IS)

Wyświetla procent obecności dla każdego przedmiotu z podziałem na moduły

```
create view AttendancePerSubject
as
   select distinct s.SubjectID, s.SubjectName, 'on-site' as module,
round(cast(sum(cast(Presence as int)) as float)/cast(count(Presence) as float)*100,
2) as AttendancePercentage
   from Subjects s
   join StationarySubjectMeeting m on m.SubjectID=s.SubjectID
   join StationaryAttendanceListStudies al on al.SSubjectID=m.SSubjectID
   group by s.SubjectID, s.SubjectName
   union
   select distinct s.SubjectID, s.SubjectName, 'online' as module,
round(cast(sum(cast(Presence as int)) as float)/cast(count(Presence) as float)*100,
2) as AttendancePercentage
   from Subjects s
   join OnLineSubjectMeeting m on m.SubjectID=s.SubjectID
   join OnLineAttendanceListStudies al on al.OLSubjectID=m.OLSubjectID
   group by s.SubjectID, s.SubjectName
```

AttendancePerSubjectTotal (IS)

Wyświetla procent łącznej obecności dla każdego przedmiotu

```
create view AttendancePerSubjectTotal
as
   select distinct s.SubjectID, s.SubjectName, round(cast(sum(cast(al.Presence as
int)+isnull(cast(al2.Presence as int),0)) as
float)/cast(count(al.Presence)+isnull(count(al2.Presence),0) as float)*100, 2) as
AttendancePercentage
   from Subjects s
   join StationarySubjectMeeting m on m.SubjectID=s.SubjectID
   join StationaryAttendanceListStudies al on al.SSubjectID=m.SSubjectID
   left join OnLineSubjectMeeting m2 on m2.SubjectID=s.SubjectID
   left join OnLineAttendanceListStudies al2 on al2.OLSubjectID=m2.OLSubjectID
   group by s.SubjectID, s.SubjectName
```

CourseAttendanceTotal (IS)

Wyświetla łączną obecność dla każdego kursu

```
create view CourseAttendanceTotal
as
   select c.CourseID, p.Name, round(cast(sum(isnull(cast(al.Presence as
int),0)+isnull(cast(al2.Presence as int),0)+isnull(cast(al3.Watched as int),0)) as
float)/cast(isnull(count(al.Presence),0)+isnull(count(al2.Presence),0)+isnull(count
(al3.Watched),0) as float)*100, 2) as AttendancePercentage
   from Courses c
   join CoursesProduct cp on cp.CourseProductID=c.CourseProductID
   join Products p on p.ProductID=cp.ProductID
   join CourseModule cm on cm.CourseID=c.CourseID
   left join StationaryModuleMeeting m on m.ModuleID=cm.ModuleID
   left join StationaryAttendanceListModule al on
al.StationaryModuleID=m.StationaryModuleID
   left join OnlineSyncModuleMeeting m2 on m2.ModuleID=cm.ModuleID
   left join OnlineSyncAttendanceListModule al2 on al2.OnlineSyncID=m2.OnlineSyncID
   left join OnlineAsyncModuleMeeting m3 on m3.ModuleID=cm.ModuleID
   left join OnlineAsyncAttendanceListModule al3 on
al3.OnlineAsyncID=m3.OnlineAsyncID
   group by c.CourseID, p.Name
   having
cast(isnull(count(al.Presence),0)+isnull(count(al2.Presence),0)+isnull(count(al3.Wa
tched),0) as float)<>0
```

Unpaid3DaysLeft (IS)

Wyświetla zamówienia, które muszą zostać opłacone w ciągu 3 dni

```
create view Unpaid3DaysLeft as
select od.OrderID, od.ProductID, RequiredDate
   from Orders o
            join OrderDetails od on od.OrderID = o.OrderID
            join Products p on p.ProductID = od.ProductID
   where PaidDate is null
     and GETDATE() >= DATEADD(day, -3, RequiredDate)
     and not exists(select *
                    from Exception e
                    where e.OrderID = od.OrderID)
```

Students (IS)

Wyświetla wszystkich studentów

```
create view Students
as
   select s.StudyID, p.Name, u.UserID, u.FirstName, u.LastName
   from Users u
   join Orders o on o.UserID=u.UserID
   join OrderDetails od on od.OrderID=o.OrderID
   join Products p on od.ProductID=p.ProductID
   join StudiesProduct sp on sp.ProductID=p.ProductID
   join Studies s on s.StudiesProductID=sp.StudiesProductID
   where exists (select * from StudiesProduct sp where sp.ProductID=od.ProductID)
```

ExamGrades (IS)

Wyświetla wszystkie oceny z egzaminów

```
create view ExamGrades
as
   select e.Date, u.FirstName, u.LastName, Grade
   from ExamScore es
   join Users u on u.UserID=es.UserID
   join Grade g on g.GradeID=es.GradeID
   join Exam e on e.ExamID=es.ExamID
```

GradeCount (IS)

Dla każdego egzaminu wyświetla liczbę zdobytych ocen z podziałem na oceny

```
create view GradeCount
as
   select es.ExamID, Grade, count(Grade) as GradeCount
   from ExamScore es
   join Grade g on g.GradeID=es.GradeID
   join Exam e on e.ExamID=es.ExamID
   group by es.ExamID, Grade
   order by es.ExamID offset 0 rows
```

ExamsPassed (IS)

Wyświetla studentów i egzaminy, które zaliczyli

```
create view ExamsPassed
as
   select u.UserID, u.FirstName, u.LastName, Grade, e.ExamID, e.Date
   from ExamScore es
   join Users u on u.UserID=es.UserID
   join Grade g on g.GradeID=es.GradeID
   join Exam e on e.ExamID=es.ExamID
   where Grade>2
```

ExamsNotPassed (IS)

Wyświetla studentów i egzaminy, których nie zaliczyli

```
create view ExamsNotPassed
as
   select u.UserID, u.FirstName, u.LastName, Grade, e.ExamID, e.Date
   from ExamScore es
   join Users u on u.UserID=es.UserID
   join Grade g on g.GradeID=es.GradeID
   join Exam e on e.ExamID=es.ExamID
   where Grade=2
```

StudentInternships (IS)

Wyświetla dane studentów i praktyk dla odbytych praktyk

```
create view StudentInternships
as
   select u.UserID, u.FirstName, u.LastName, ai.StartDate, ai.EndDate,
i.InternshipID, s.StudyID
   from Internships i
   join AttendanceInternship ai on ai.InternshipID=i.InternshipID
   join Studies s on s.StudyID=i.StudyID
   join Users u on u.UserID=ai.UserID
```

StudentCount (IS)

Dla każdych studiów wyświetla liczbę studentów oraz procent zajętych miejsc na studiach

```
alter view StudentCount
as
   select s.StudyID, p.Name, count(u.UserID) as StudentCount, Limit,
round(cast(count(u.UserID) as float)/cast(Limit as float)*100, 2) as
SpotsTakenPercentage
   from Users u
   join Orders o on o.UserID=u.UserID
   join OrderDetails od on od.OrderID=o.OrderID
   join Products p on od.ProductID=p.ProductID
   join StudiesProduct sp on sp.ProductID=p.ProductID
   join Studies s on s.StudiesProductID=sp.StudiesProductID
   where exists (select * from StudiesProduct sp where sp.ProductID=od.ProductID)
   group by s.StudyID, p.Name, s.Limit
   order by SpotsTakenPercentage desc offset 0 rows
```

TranslatorCount (IS)

Dla każdego języka wyświetla liczbę tłumaczy

```
create view TranslatorCount
as
   select Language, count(TranslatorID) as TranslatorCount
   from Languages l
   join Translators t on t.LanguageID=l.LanguageID
   group by Language
   order by TranslatorCount desc offset 0 rows
```

UsersPerCountry (IS)

Wyświetla liczbę użytkowników z poszczególnych krajów, oraz łączną sumę

```
create view UsersPerCountry
as
   select isnull(cu.CountryName, 'Total') as CountryName, count(cu.CountryName) as
UserCount
   from Users u
   join Cities c on c.CityID=u.CityID
   join Countries cu on cu.CountryID=c.CityID
   group by cu.CountryName with rollup
```

StaffPerCountry (IS)

Wyświetla liczbę pracowników z poszczególnych krajów, oraz łączną sumę

```
create view StaffPerCountry
as
   select isnull(cu.CountryName, 'Total') as CountryName, count(cu.CountryName) as
StaffCount
   from Staff u
   join Cities c on c.CityID=u.CityID
   join Countries cu on cu.CountryID=c.CityID
   group by cu.CountryName with rollup
```

UsersPerAge (IS)

Wyświetla rozpiskę wieku użytkowników

```
create view UsersPerAge
as
   select datediff(year, BirthDate, getdate()) as Age, count(datediff(year,
BirthDate, getdate())) as AgeCount
   from Users u
   group by datediff(year, BirthDate, getdate())
   order by Age offset 0 rows
```

StaffPerAge (IS)

Wyświetla rozpiskę wieku pracowników

```
create view StaffPerAge
as
   select datediff(year, BirthDate, getdate()) as Age, count(datediff(year,
BirthDate, getdate())) as AgeCount
   from Staff u
   group by datediff(year, BirthDate, getdate())
   order by Age offset 0 rows
```

EmploymentTime (IS)

Wyświetla czas zatrudnienia w miesiącach każdego pracownika

```
create view EmploymentTime
as
   select StaffID, FirstName, LastName, datediff(month, EmploymentDate, getdate())
as EmploymentTime
   from Staff s
```

TranslatorsView (IS)

Wyświetla tłumaczy oraz języki, które tłumaczą

```
create view TranslatorsView
as
   select s.StaffID, FirstName, LastName, Language
   from Translators t
   join Staff s on s.StaffID=t.StaffID
   join Languages l on l.LanguageID=t.LanguageID
```

OrdersAll (IS)

Wyświetla wszystkie zamówienia

```
alter view OrdersAll
as
   select o.UserID, o.OrderID, p.ProductID, p.Name, dbo.GetPrice(p.ProductID) as
Price, OrderDate, RequiredDate
   from Orders o
   join OrderDetails od on od.OrderID=o.OrderID
   join Products p on p.ProductID=od.ProductID
```

RemainingPlacesStudies (MM)

Wyświetla ilość wolnego miejsca na studiach

```
create view RemainingPlacesStudies as
select StudyID, Products.Name,Limit-count(UserID) as wolne_miejsca
from Studies
inner join dbo.StudiesProduct SP on Studies.StudiesProductID =
SP.StudiesProductID
inner join Products on SP.ProductID = Products.ProductID
inner join OrderDetails on Products.ProductID =
OrderDetails.ProductID
inner join Orders on OrderDetails.OrderID = Orders.OrderID
group by StudyID, Name, Limit
```

MandatoryAttendance (MM)

Wyświetla wszystkie spotkania na których dany użytkownik powinien być. Widok pomocniczy do BilocationReport

```
Create View MandatoryAttendance as
with webinarFirst as (SELECT UserID
as userID
                              , OD.ProductID
as ProductID
                              , StartDate
as Start
                              , DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00',
Duration), StartDate) as 'End'
                              , ROW_NUMBER() OVER (PARTITION BY UserID,
OD.ProductID
          ORDER BY StartDate)
AS seq
                        from Orders
                                    inner join dbo.OrderDetails OD on
Orders.OrderID = OD.OrderID
                                    inner join Products on OD.ProductID =
Products.ProductID
                                    inner join WebinarProduct on
Products.ProductID = WebinarProduct.ProductID
                                    inner join Webinars on
WebinarProduct.WebinarProductID = Webinars.WebinarProductID
                                    inner join WebinarMeeting on
Webinars.WebinarID = WebinarMeeting.WebinarID)
   select UserID,
          ProductID,
          Start,
          [End]
   from webinarFirst
   where seq = 1
   union
   select UserID,
          OD.ProductID,
          Date                                                    as
Start,
          DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', Duration), Date) as
'End'
   from Orders
            inner join dbo.OrderDetails OD on Orders.OrderID = OD.OrderID
            inner join Products on OD.ProductID = Products.ProductID
            inner join dbo.StudiesProduct SP on Products.ProductID =
SP.ProductID
            inner join Studies on SP.StudiesProductID =
Studies.StudiesProductID
            inner join Subjects on Studies.StudyID = Subjects.StudyID
            inner join StationarySubjectMeeting on Subjects.SubjectID =
StationarySubjectMeeting.SubjectID
   union
   select UserID,
```

```
          OD.ProductID,
          Date                                                              as
Start,
          DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', Duration), Date) as
'End'
   from Orders
            inner join dbo.OrderDetails OD on Orders.OrderID = OD.OrderID
            inner join Products on OD.ProductID = Products.ProductID
            inner join dbo.StudiesProduct SP on Products.ProductID =
SP.ProductID
            inner join Studies on SP.StudiesProductID =
Studies.StudiesProductID
            inner join Subjects on Studies.StudyID = Subjects.StudyID
            inner join OnLineSubjectMeeting on Subjects.SubjectID =
OnLineSubjectMeeting.SubjectID
   union
   select UserID,
          OD.ProductID,
          Date                                                              as
Start,
          DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', Duration), Date) as
'End'
   from Orders
            inner join dbo.OrderDetails OD on Orders.OrderID = OD.OrderID
            inner join Products on OD.ProductID = Products.ProductID
            inner join CoursesProduct on Products.ProductID =
CoursesProduct.ProductID
            inner join Courses on CoursesProduct.CourseProductID =
Courses.CourseProductID
            inner join CourseModule on Courses.CourseID =
CourseModule.CourseID
            inner join StationaryModuleMeeting on CourseModule.ModuleID =
StationaryModuleMeeting.ModuleID
   union
   select UserID,
          OD.ProductID,
          Date                                                              as
Start,
          DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', Duration), Date) as
'End'
   from Orders
            inner join dbo.OrderDetails OD on Orders.OrderID = OD.OrderID
            inner join Products on OD.ProductID = Products.ProductID
            inner join CoursesProduct on Products.ProductID =
CoursesProduct.ProductID
            inner join Courses on CoursesProduct.CourseProductID =
Courses.CourseProductID
            inner join CourseModule on Courses.CourseID =
CourseModule.CourseID
            inner join OnlineSyncModuleMeeting on CourseModule.ModuleID =
OnlineSyncModuleMeeting.ModuleID
```

BilocationReport (MM)

Wyświetla użytkowników dla których spotkania kolidują czasowo.

```
Create View BilocationReport as
select distinct u.UserID, u.FirstName, u.LastName, A.Start,
A.[End], p.Name
   from MandatoryAttendance A
            inner join Users as u on A.UserID = u.UserID
            inner join MandatoryAttendance AA on A.UserID =
AA.UserID AND
                                                  A.Start <
AA.[End]
       AND A.[End] > AA.Start and A.ProductID != AA.ProductID
            inner join Products as p on A.ProductID = p.ProductID
```

WCSPrices (MM)

Wyświetla produkty, które są webinarami,kursami i studiami, koszt całkowity i minimalny koszt do zrealizowania zamówienia

```
CREATE VIEW WCSPrices as
   select Products.ProductID,Products.Name,SP.Price as 'full
price', SP.Entry_fee as 'min_price_to_order_product' from Products
   inner join StudiesProduct SP on Products.ProductID =
SP.ProductID
   union
   select Products.ProductID,Products.Name,CP.Price as 'full
price', CP.PriceAdvance as 'min_price_to_order_product' from
Products
   inner join CoursesProduct CP on Products.ProductID =
CP.ProductID
   union
   select Products.ProductID,Products.Name,WP.WebinarPrice as
'full price', WP.WebinarPrice as 'min_price_to_order_product' from
Products
   inner join WebinarProduct WP on Products.ProductID =
WP.ProductID
```

ClientStats (MM)

Wyświetla użytkowników, ich ilość zamówień oraz ogólna liczba wpłat

```
CREATE VIEW ClientStats as
   select u.UserID,
          u.FirstName,
          u.LastName,
          count(distinct Orders.OrderID) as 'times_ordered',
          ISNULL(sum(OrderDetails.Price),0)        as 'wpłaty'
   from Users as u
            left join Orders on u.UserID = Orders.UserID
            left join OrderDetails on Orders.OrderID =
OrderDetails.OrderID
   group by u.UserID, u.FirstName, u.LastName
```

CurrentOffer (MM)

Wyświetla produkty, które dopiero mają się zacząć

```
create view CurrentOffer as
select Products.ProductID,Products.Name,'A' as 'For who',SP.Price
as 'full price', SP.Entry_fee as 'min_price_to_order_product' from
Products
inner join StudiesProduct SP on Products.ProductID = SP.ProductID
inner join dbo.Studies S on SP.StudiesProductID =
S.StudiesProductID
where StartDate>GETDATE()
union
select Products.ProductID,Products.Name,'A',CP.Price as 'full
price', CP.PriceAdvance as 'min_price_to_order_product' from
Products
inner join CoursesProduct CP on Products.ProductID = CP.ProductID
inner join Courses on CP.CourseProductID = Courses.CourseProductID
where StartDate>GETDATE()
union
select Products.ProductID,Products.Name,'A',WP.WebinarPrice as
'full price', WP.WebinarPrice as 'min_price_to_order_product' from
Products
inner join WebinarProduct WP on Products.ProductID = WP.ProductID
inner join Webinars on WP.WebinarProductID =
Webinars.WebinarProductID
inner join WebinarMeeting on Webinars.WebinarID =
WebinarMeeting.WebinarID
where StartDate>GETDATE()
union
select Products.ProductID,Products.Name,'S',SMP.PriceforStudents
as 'full price', SMP.PriceforStudents as
'min_price_to_order_product' from Products
inner join StudiesMeetingProduct SMP on Products.ProductID =
SMP.ProductID
inner join dbo.OnLineSubjectMeeting OLSM on
SMP.StudiesMeetingProductID = OLSM.StudiesMeetingProductID
where Date>GETDATE()
union
select Products.ProductID,Products.Name,'A',SMP.PriceforOthers as
'full price', SMP.PriceforOthers as 'min_price_to_order_product'
from Products
inner join StudiesMeetingProduct SMP on Products.ProductID =
SMP.ProductID
inner join dbo.OnLineSubjectMeeting OLSM on
SMP.StudiesMeetingProductID = OLSM.StudiesMeetingProductID
where Date>GETDATE()
union
```

```
select Products.ProductID,Products.Name,'S',SMP.PriceforStudents
as 'full price', SMP.PriceforStudents as
'min_price_to_order_product' from Products
inner join StudiesMeetingProduct SMP on Products.ProductID =
SMP.ProductID
inner join dbo.StationarySubjectMeeting SSM on
SMP.StudiesMeetingProductID = SSM.StudiesMeetingProductID
where Date>GETDATE()
union
select Products.ProductID,Products.Name,'A',SMP.PriceforOthers as
'full price', SMP.PriceforOthers as 'min_price_to_order_product'
from Products
inner join StudiesMeetingProduct SMP on Products.ProductID =
SMP.ProductID
inner join dbo.StationarySubjectMeeting SSM on
SMP.StudiesMeetingProductID = SSM.StudiesMeetingProductID
where Date>GETDATE()
```

AllFutureAndPresentMeetings (MM)

Wyswietla wszystkie spotkania, które się teraz odbywają lub będą w przyszłości

```
create view AllFutureAndPresentMeetings as
select * from MandatoryAttendance
where GETDATE()<[Start] OR ([Start]<=GETDATE() AND
[End]>=GETDATE())
```

FutureEvents (KL)

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

```
CREATE VIEW FutureEvents as
SELECT
   p.Name,
   CASE
       WHEN COUNT(od.ProductID) IS NULL THEN 0
       ELSE COUNT(od.ProductID)
   END AS ilosc_wystapien
FROM
   Products AS p
LEFT JOIN
   OrderDetails AS od ON p.ProductID = od.ProductID
GROUP BY
   p.Name
```

Financial Reports (KL)

Zestawienie przychodów dla każdego webinaru/kursu/studium.

```
CREATE VIEW Financial Reports as
select p.ProductID,p.Name,isnull(sum(od.Price),0) as przychód
from Products as p
left join OrderDetails as od
on od.ProductID=p.ProductID
group by p.ProductID,p.Name
```

CourseStructure (KL)

Struktura każdego kursu.

```
Create View CourseStructure as
select p.ProductID,
          p.Name,
          cp.Price,
          cp.PriceAdvance,
          c.StartDate,
          c.EndDate,
          cm.ModuleID,
          f.FormName              as ModuleForm,
          'Stacjonarne'           as MeetingForm,
          smm.StationaryModuleID as 'IDSpotkania',
          smm.Date                as 'Date meeting/add',
          smm.Duration            as 'Czas_trwania'
   from Products as p
            inner join CoursesProduct as cp on p.ProductID =
cp.ProductID
            inner join Courses as c on cp.CourseProductID =
c.CourseProductID
            inner join CourseModule as cm on c.CourseID =
cm.CourseID
            inner join Form as f on cm.FormID = f.FormID
            inner join StationaryModuleMeeting as smm on
smm.ModuleID = cm.ModuleID
   union
   --online sync
   select p.ProductID,
          p.Name,
          cp.Price,
          cp.PriceAdvance,
          c.StartDate,
          c.EndDate,
          cm.ModuleID,
          f.FormName              as ModuleForm,
          'Online Synchroniczne' as MeetingForm,
          osmm.OnlineSyncID       as 'IDSpotkania',
          osmm.Date               as 'Date meeting/add',
          osmm.Duration           as 'Czas_trwania'
   from Products as p
```

```
            inner join CoursesProduct as cp on p.ProductID =
cp.ProductID
            inner join Courses as c on cp.CourseProductID =
c.CourseProductID
            inner join CourseModule as cm on c.CourseID =
cm.CourseID
            inner join Form as f on cm.FormID = f.FormID
            inner join OnlineSyncModuleMeeting as osmm on
osmm.ModuleID = cm.ModuleID
   union
   -- online async
   select p.ProductID,
          p.Name,
          cp.Price,
          cp.PriceAdvance,
          c.StartDate,
          c.EndDate,
          cm.ModuleID,
          f.FormName               as ModuleForm,
          'Online Asynchroniczne' as MeetingForm,
          oasmm.OnlineAsyncID     as 'IDSpotkania',
          oasmm.AddDate           as 'Date meeting/add',
          oasmm.Duration          as 'Czas_trwania'
   from Products as p
            inner join CoursesProduct as cp on p.ProductID =
cp.ProductID
            inner join Courses as c on cp.CourseProductID =
c.CourseProductID
            inner join CourseModule as cm on c.CourseID =
cm.CourseID
            inner join Form as f on cm.FormID = f.FormID
            inner join OnlineAsyncModuleMeeting as oasmm on
oasmm.ModuleID = cm.ModuleID
```

WebinarStructure (KL)

Struktura wszystkich webinarów.

```
Create View WebinarStructure as
SELECT
   w.WebinarID,
   p.Name AS 'WebinarName',
   wm.WebinarMeetingID,
   wm.StartDate,
   wm.Duration
FROM
   Products AS p
   INNER JOIN WebinarProduct AS wp ON wp.ProductID = p.ProductID
   INNER JOIN Webinars AS w ON w.WebinarProductID =
wp.WebinarProductID
   INNER JOIN WebinarMeeting AS wm ON wm.WebinarID = w.WebinarID
```

StudiesStructure (KL)

Struktura wszystkich studiów.

```
CREATE VIEW StudiesStructure AS
SELECT p.ProductID AS 'Studies number', p.Name, stud.StartDate AS
'Studies start date', stud.EndDate AS 'Studies end date',
sub.SubjectID, sub.SubjectName, ssm.SSubjectID AS 'Number of
meeting', 'stationary' AS 'Form',
                  ssm.Date AS 'Date of meeting', ssm.Duration,
sp.Price AS 'Studies price', smp.PriceforOthers AS 'Meeting price'
FROM     Products AS p INNER JOIN
                  StudiesProduct AS sp ON sp.ProductID =
p.ProductID INNER JOIN
                  Studies AS stud ON stud.StudiesProductID =
sp.StudiesProductID INNER JOIN
                  Subjects AS sub ON sub.StudyID = stud.StudyID
INNER JOIN
                  StationarySubjectMeeting AS ssm ON ssm.SubjectID
= sub.SubjectID INNER JOIN
                  StudiesMeetingProduct AS smp ON
smp.StudiesMeetingProductID = ssm.StudiesMeetingProductID
UNION
SELECT p.ProductID AS 'Studies number', p.Name, stud.StartDate AS
'Studies start date', stud.EndDate AS 'Studies end date',
sub.SubjectID, sub.SubjectName, osm.OLSubjectID AS 'Number of
meeting', 'online' AS 'Form',
                  osm.Date AS 'Date of meeting', osm.Duration,
sp.Price AS 'Studies price', smp.PriceforOthers AS 'Meeting price'
FROM     Products AS p INNER JOIN
                  StudiesProduct AS sp ON sp.ProductID =
p.ProductID INNER JOIN
                  Studies AS stud ON stud.StudiesProductID =
sp.StudiesProductID INNER JOIN
                  Subjects AS sub ON sub.StudyID = stud.StudyID
INNER JOIN
                  OnLineSubjectMeeting AS osm ON osm.SubjectID =
sub.SubjectID INNER JOIN
                  StudiesMeetingProduct AS smp ON
smp.StudiesMeetingProductID = osm.StudiesMeetingProductID
```

AllOrders (KL)

Koszyk zakupowy.

```
CREATE VIEW AllOrders as
SELECT o.OrderID,
          u.FirstName + ' ' + u.LastName AS [User],
          o.OrderDate,
          o.RequiredDate,
          o.PaidDate,
          p.Name                         AS [Product Name],
          od.Price                       AS Paid,
          wp.WebinarPrice                as Price
   FROM dbo.Orders AS o
            INNER JOIN dbo.Users AS u ON o.UserID = u.UserID
            INNER JOIN dbo.OrderDetails AS od ON od.OrderID =
o.OrderID
            INNER JOIN dbo.Products AS p ON p.ProductID =
od.ProductID
            inner join dbo.WebinarProduct as wp on wp.ProductID =
p.ProductID
   union
   SELECT o.OrderID,
          u.FirstName + ' ' + u.LastName AS [User],
          o.OrderDate,
          o.RequiredDate,
          o.PaidDate,
          p.Name                         AS [Product Name],
          od.Price                       AS Paid,
          cp.Price                       as Price
   FROM dbo.Orders AS o
            INNER JOIN dbo.Users AS u ON o.UserID = u.UserID
            INNER JOIN dbo.OrderDetails AS od ON od.OrderID =
o.OrderID
            INNER JOIN dbo.Products AS p ON p.ProductID =
od.ProductID
            inner join dbo.CoursesProduct as cp on cp.ProductID =
p.ProductID
   union
   SELECT o.OrderID,
          u.FirstName + ' ' + u.LastName AS [User],
          o.OrderDate,
          o.RequiredDate,
          o.PaidDate,
          p.Name                         AS [Product Name],
          od.Price                       AS Paid,
          sp.Price                       as Price
   FROM dbo.Orders AS o
```

```
            INNER JOIN dbo.Users AS u ON o.UserID = u.UserID
            INNER JOIN dbo.OrderDetails AS od ON od.OrderID =
o.OrderID
            INNER JOIN dbo.Products AS p ON p.ProductID =
od.ProductID
            inner join dbo.StudiesProduct as sp on sp.ProductID =
p.ProductID
   union
   SELECT o.OrderID,
          u.FirstName + ' ' + u.LastName AS [User],
          o.OrderDate,
          o.RequiredDate,
          o.PaidDate,
          p.Name                          AS [Product Name],
          od.Price                        AS Paid,
          smp.PriceforOthers              as Price
   FROM dbo.Orders AS o
            INNER JOIN dbo.Users AS u ON o.UserID = u.UserID
            INNER JOIN dbo.OrderDetails AS od ON od.OrderID =
o.OrderID
            INNER JOIN dbo.Products AS p ON p.ProductID =
od.ProductID
            inner join dbo.StudiesMeetingProduct as smp on
smp.ProductID = p.ProductID
```

CourseUsersPassed (KL)

Widok osób które zaliczyły kursy.

```
CREATE VIEW CourseUsersPassed AS
WITH ModuleCompletion AS (
    SELECT
        UserID,
        smm.ModuleID,
        cm.CourseID,
        CAST(SUM(IIF(Presence = 1, 1, 0)) AS FLOAT) /
CAST(COUNT(Presence) AS FLOAT) AS CompletionRate
    FROM
        StationaryAttendanceListModule AS salm
        INNER JOIN StationaryModuleMeeting AS smm ON
salm.StationaryModuleID = smm.StationaryModuleID
        INNER JOIN CourseModule AS cm ON cm.ModuleID = smm.ModuleID

    GROUP BY
        UserID,
        smm.ModuleID,
        cm.CourseID

    UNION

    SELECT
        UserID,
        oamm.ModuleID,
        cm.CourseID,
        CAST(SUM(IIF(Watched = 1, 1, 0)) AS FLOAT) /
CAST(COUNT(Watched) AS FLOAT) AS CompletionRate
    FROM
        OnlineAsyncAttendanceListModule AS oalm
        INNER JOIN OnlineAsyncModuleMeeting AS oamm ON
oamm.OnlineAsyncID = oalm.OnlineAsyncID
        INNER JOIN CourseModule AS cm ON cm.ModuleID =
oamm.ModuleID
    GROUP BY
        UserID,
        oamm.ModuleID,
        cm.CourseID

    UNION

    SELECT
        UserID,
        osmm.ModuleID,
        cm.CourseID,
```

```
        CAST(SUM(IIF(Presence = 1, 1, 0)) AS FLOAT) /
CAST(COUNT(Presence) AS FLOAT) AS CompletionRate
    FROM
        OnlineSyncAttendanceListModule AS osalm
        INNER JOIN OnlineSyncModuleMeeting AS osmm ON
osmm.OnlineSyncID = osalm.OnlineSyncID
        INNER JOIN CourseModule AS cm ON cm.ModuleID =
osmm.ModuleID
    GROUP BY
        UserID,
        osmm.ModuleID,
        cm.CourseID
)
SELECT
    mc.CourseID,
    p.Name,
    u.UserID,
    u.FirstName+ ' '+u.LastName as 'UserName',
    AVG(CompletionRate) AS AverageCompletionRate,
    CASE WHEN AVG(CompletionRate) >= 0.8 THEN 'Zdał' ELSE 'Nie
zdał' END AS CourseOutcome
FROM
    ModuleCompletion as mc
inner join Users as u on mc.UserID=u.UserID
inner join Courses as c on c.CourseID=mc.CourseID
inner join CoursesProduct as cp on
cp.CourseProductID=c.CourseProductID
inner join Products as p on p.ProductID=cp.ProductID
GROUP BY
    mc.CourseID,
    p.Name,
    u.UserID,
    u.FirstName+ ' '+u.LastName
```

Procedury

AddStudy (IS)

Dodaje studium
Argumenty:

- Name - nazwa (opis) studiów
- CoordinatorID - ID koordynatora
- StartDate - data rozpoczęcia studiów
- EndDate - data zakończenia studiów
- Limit - limit miejsc na studiach
- Price - cena studiów
- Entry_fee - wartość zaliczki

```
create procedure AddStudy
   @Name varchar(50),
   @CoordinatorID int,
   @StartDate smalldatetime,
   @EndDate smalldatetime,
   @Limit int,
   @Price money,
   @Entry_fee money
as begin
   set nocount on

   if not exists(
       select *
       from Teachers
       where TeacherID=@CoordinatorID
   ) begin
       ;throw 52000, 'Cannot add study course: coordinator id does not exist', 1
   end
   if @EndDate <= @StartDate begin
       ;throw 52000, 'Cannot add study course: end date must be after start date',
1
   end
   if @Limit <= 0 begin
       ;throw 52000, 'Cannot add study course: limit must be greater than 0', 1
   end
   if @Price <= 0 begin
       ;throw 52000, 'Cannot add study course: price must be greater than 0', 1
   end
   if @Entry_fee <= 0 begin
       ;throw 52000, 'Cannot add study course: entry fee must be greater than 0', 1
   end

   declare @StudyID int = (select isnull(max(StudyID), 0) + 1 from Studies)
   declare @StudiesProductID int = (select isnull(max(StudiesProductID), 0) + 1
from StudiesProduct)
   declare @ProductID int = (select isnull(max(ProductID), 0) + 1 from Products)

   insert into Products(ProductID, Name)
       values (@ProductID, @Name)
```

```
   insert into StudiesProduct(StudiesProductID, ProductID, Price, Entry_fee)
       values (@StudiesProductID, @ProductID, @Price, @Entry_fee)
   insert into Studies(StudyID, StudiesProductID, CoordinatorID, StartDate,
EndDate, Limit)
       values (@StudyID, @StudiesProductID, @CoordinatorID, @StartDate, @EndDate,
@Limit)
end
```

ModifyLimit (IS)

Zmienia limit miejsc na studiach
Argumenty:

- StudyID - ID studium
- Limit - nowy limit

```
CREATE procedure ModifyLimit
   @StudyID int,
   @Limit int
as begin
   set nocount on

   if not exists(
       select *
       from Studies
       where StudyID=@StudyID
   ) begin
       ;throw 52000, 'Cannot modify limit: study course does not exist', 1
   end
   if @Limit <= 0 begin
       ;throw 52000, 'Cannot modify limit: limit must be greater than 0', 1
   end

   update Studies
       set Limit=@Limit
       where StudyID=@StudyID
end
```

AddCourse (IS)

Dodaje kurs
Argumenty:

- Name - nazwa (opis) kursu
- CoordinatorID - ID koordynatora
- StartDate - data rozpoczęcia kursu
- EndDate - data zakończenia kursu
- Price - cena studiów
- PriceAdvance - wartość zaliczki

```
create procedure AddCourse
   @Name varchar(50),
   @CoordinatorID int,
   @StartDate smalldatetime,
   @EndDate smalldatetime,
   @Price money,
   @PriceAdvance money
as begin
   set nocount on

   if not exists(
       select *
       from Teachers
       where TeacherID=@CoordinatorID
   ) begin
       ;throw 52000, 'Cannot add course: coordinator id does not exist', 1
   end
   if @EndDate <= @StartDate begin
       ;throw 52000, 'Cannot add course: end date must be after start date', 1
   end
   if @Price <= 0  or @PriceAdvance <= 0 begin
       ;throw 52000, 'Cannot add course: price must be greater than 0', 1
   end

   declare @CourseID int = (select isnull(max(CourseID), 0) + 1 from Courses)
   declare @CourseProductID int = (select isnull(max(CourseProductID), 0) + 1 from
CoursesProduct)
   declare @ProductID int = (select isnull(max(ProductID), 0) + 1 from Products)

   insert into Products(ProductID, Name)
       values (@ProductID, @Name)
   insert into CoursesProduct(CourseProductID, ProductID, Price, PriceAdvance)
       values (@CourseProductID, @ProductID, @Price, @PriceAdvance)
   insert into Courses(CourseID, CourseProductID, CoordinatorID, StartDate,
EndDate)
       values (@CourseID, @CourseProductID, @CoordinatorID, @StartDate, @EndDate)
end
```

AddModule (IS)

Dodaje moduł do kursu
Argumenty:

- CourseID - ID kursu
- TeacherID - ID prowadzącego moduł
- FormID - ID formy modułu

```
create procedure AddModule
   @CourseID int,
   @TeacherID int,
   @FormID int
as begin
   set nocount on

   if not exists(
       select *
       from Courses
       where CourseID=@CourseID
   ) begin
       ;throw 52000, 'Cannot add module: course does not exist', 1
   end
   if not exists(
       select *
       from Teachers
       where TeacherID=@TeacherID
   ) begin
       ;throw 52000, 'Cannot add module: teacher does not exist', 1
   end
   if not exists(
       select *
       from Form
       where FormID=@FormID
   ) begin
       ;throw 52000, 'Cannot add module: form does not exist', 1
   end

   declare @ModuleID int = (select isnull(max(ModuleID), 0) + 1 from CourseModule)

   insert into CourseModule(ModuleID, CourseID, TeacherID, FormID)
       values (@ModuleID, @CourseID, @TeacherID, @FormID)
end
```

AddSubject (IS)

Dodaje przedmiot do studium
Argumenty:

- StudyID - ID studiów
- SubjectName - nazwa przedmiotu
- TeacherID - ID prowadzącego przedmiot
- FormID - ID formy przedmiotu

```
create procedure AddSubject
   @StudyID int,
   @SubjectName varchar(50),
   @TeacherID int,
   @FormID int
as begin
   set nocount on

   if not exists(
       select *
       from Studies
       where StudyID=@StudyID
   ) begin
       ;throw 52000, 'Cannot add subject: study course does not exist', 1
   end
   if not exists(
       select *
       from Teachers
       where TeacherID=@TeacherID
   ) begin
       ;throw 52000, 'Cannot add course: teacher does not exist', 1
   end
   if not exists(
       select *
       from Form
       where FormID=@FormID
   ) begin
       ;throw 52000, 'Cannot add course: form does not exist', 1
   end

   declare @SubjectID int = (select isnull(max(SubjectID), 0) + 1 from Subjects)

   insert into Subjects(SubjectID, StudyID, SubjectName, TeacherID, FormID)
       values (@SubjectID, @StudyID, @SubjectName, @TeacherID, @FormID)
end
```

AddException (IS)

Dodaje wyjątek do zamówienia
Argumenty:

- OrderID - ID zamówienia
- Description - opis wyjątku
- NewDate - nowa wymagana data płatności

```
create procedure AddException
   @OrderID int,
   @Description varchar(100),
   @NewDate smalldatetime
as begin
   set nocount on

   if not exists(
       select *
       from Orders
       where OrderID=@OrderID
   ) begin
       ;throw 52000, 'Cannot add exception: order does not exist', 1
   end

   declare @OldDate smalldatetime = (select RequiredDate from Orders where
OrderID=@OrderID)

   if @NewDate < GETDATE() or @NewDate <= @OldDate begin
       ;throw 52000, 'Cannot add exception: the new date cannot be before the old
or the current date', 1
   end

   declare @ExceptionID int = (select isnull(max(ExceptionID), 0) + 1 from
Exception)

   insert into Exception(ExceptionID, OrderID, ExceptionName, Date)
       values (@ExceptionID, @OrderID, @Description, @NewDate)
end
```

AddUser (MM)

Dodaje webinar.
Argumenty:

- @FirstName - imię użytkownika
- @LastName - nazwisko użytkownika
- @BirthDate - data urodzenia
- @Phone - numer telefonu
- @Email - email użytkownika
- @Address - adres zamieszkania
- @CityID - ID miasta

```
CREATE PROCEDURE AddUser    @FirstName varchar(50),
                            @LastName varchar(50),
                            @BirthDate date,
                            @Phone varchar(50),
                            @Email varchar(50),
                            @Address varchar(50),
                            @CityID INT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS (
       SELECT *
       FROM Cities
       WHERE CityID = @CityID
           )
       BEGIN
           THROW 52000, N'Dane miasto nie zostało wpisane', 1;
       END
       BEGIN TRAN
       DECLARE @UserID INT
       SELECT @UserID = ISNULL(MAX(UserID), 0) + 1
       FROM Users
       INSERT INTO Users(UserID, FirstName, LastName, BirthDate,
Phone, Email, Address, CityID) VALUES
(@UserID,@FirstName,@LastName,@BirthDate,@Phone,@Email,@Address,@C
ityID)
       COMMIT TRAN
   END TRY
   BEGIN CATCH
       IF @@TRANCOUNT > 0
           ROLLBACK TRAN
       DECLARE @msg NVARCHAR(2048) = N'Błąd dodawania usera:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddWebinar (MM)

Dodaje webinar.
Argumenty:

- @webinarName - nazwa (opis) webinaru
- @CoordinatorID - ID koordynatora
- @Language - język webinaru
- @price - cena webinaru

```
CREATE PROCEDURE AddWebinar @webinarName varchar(255),
                            @CoordinatorID INT,
                            @Language varchar(255),
                            @price money
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF EXISTS(
           SELECT *
               FROM Products
               WHERE Name = @webinarName
               )
           BEGIN
               ;
               THROW 52000, N'Dany webinar juz istnieje', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM Languages
               WHERE Language = @Language
               )
           BEGIN
               ;
               THROW 52000, N'Dany język nie istnieje', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM Teachers
               WHERE TeacherID = @CoordinatorID
               )
           BEGIN
               ;
               THROW 52000, N'Dany koordynator nie istnieje', 1
           END
       BEGIN TRAN
       DECLARE @ProductID INT
       SELECT @ProductID = ISNULL(MAX(ProductID), 0) + 1
       FROM Products
       INSERT INTO Products(ProductID, Name) VALUES (@ProductID,@webinarName)
       DECLARE @webinarProductID INT
       SELECT @webinarProductID = ISNULL(MAX(WebinarProductID), 0) + 1
       FROM WebinarProduct
       INSERT INTO WebinarProduct(WebinarProductID,ProductID,WebinarPrice) VALUES
(@webinarProductID,@ProductID,@price)
       DECLARE @WebinarID INT
       SELECT @WebinarID = ISNULL(MAX(WebinarID), 0) + 1
       FROM Webinars
       DECLARE @LanguageID INT
```

```
       Select @LanguageID = LanguageID
       from Languages
       where Language=@Language
       INSERT INTO Webinars(WebinarID,webinarProductID,CoordinatorID,LanguageID)
VALUES (@WebinarID,@webinarProductID,@CoordinatorID,@LanguageID)
       COMMIT TRAN
   END TRY
   BEGIN CATCH
       IF @@TRANCOUNT > 0
           ROLLBACK TRAN
       DECLARE @msg NVARCHAR(2048) = N'Blad dodawania webinaru:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

DeleteWebinar (MM)

Usuwa webinar.
Argumenty:

- @WebinarID - id webinaru

```
CREATE PROCEDURE DeleteWebinar   @WebinarID INT
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Webinars
               WHERE WebinarID = @WebinarID
               )
           BEGIN
               ;
               THROW 52000, N'Dany webinar nie istnieje', 1
           END

       BEGIN TRAN
       DECLARE @WebinarProductDelete INT
       Select @WebinarProductDelete=WebinarProductID from Webinars
       where WebinarID=@WebinarID
       DELETE FROM WebinarMeeting where WebinarID=@WebinarID
       DELETE FROM Webinars where WebinarID=@WebinarID
       DELETE FROM WebinarProduct where
WebinarProductID=@WebinarProductDelete
       COMMIT TRAN
   END TRY
   BEGIN CATCH
       IF @@TRANCOUNT > 0
          ROLLBACK TRAN
       DECLARE @msg NVARCHAR(2048) = N'Błąd usuwania webinaru:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddWebinarMeeting (MM)

Dodaje spotkanie do danego webinaru.
Argumenty:

- @webinarID - ID webinaru
- @date - data wydarzenia
- @duration - czas trwania
- @TranslatorID - ID tłumacza
- @teacherID - ID nauczyciela
- @link - link do spotkania

```
CREATE PROCEDURE AddWebinarMeeting @webinarID INT,
                          @date smalldatetime,
                          @duration varchar(255),
                          @TranslatorID INT = NULL,
                          @teacherID INT,
                          @link varchar(255)

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Webinars
               WHERE WebinarID = @webinarID
               )
           BEGIN
               ;
               THROW 52000, N'Dany webinar nie istnieje ', 1
           END
       IF ISNULL(@date,'2000-01-01') < GETDATE()
           BEGIN
               ;
               THROW 52000, N'Niepoprawna data rozpoczęcia', 1
           END
       IF ISNULL(@duration,'00:00:00') < '00:00:00'
           BEGIN
               ;
               THROW 52000, N'Niepoprawny czas trwania', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM Teachers
               WHERE TeacherID = @teacherID
               )
           BEGIN
               ;
               THROW 52000, N'Dany nauczyciel nie istnieje', 1
           END
       If ISNULL(@TranslatorID,NULL) IS NOT NULL
           IF NOT EXISTS(
           SELECT *
               FROM Translators
               WHERE TranslatorID = @TranslatorID
               )
           BEGIN
```

```
            ;
            THROW 52000, N'Dany translator nie istnieje', 1
        END
    IF EXISTS(
        select *
        from WebinarStructure
        where WebinarID=@webinarID and
              DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', @duration),
@date)>StartDate and @date <DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', Duration),
StartDate)
    )
        BEGIN
            ;
            THROW 52000, N'Dane spotkanie koliduje juz z innym istniejacym', 1
        END
    DECLARE @WebinarMeetingID INT
    SELECT @WebinarMeetingID = ISNULL(MAX(WebinarMeetingID), 0) + 1
    FROM WebinarMeeting
    insert into WebinarMeeting (WebinarMeetingID, WebinarID, TeacherID,
StartDate, Duration, Link, RecordingLink,
                                TranslatorID) values
(@WebinarMeetingID,@webinarID,@teacherID,@date,@duration,@link,NULL,@TranslatorID)

   END TRY
   BEGIN CATCH

    DECLARE @msg NVARCHAR(2048) = N'Bład dodawania spotkania webinaru:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
    THROW 52000,@msg, 1;
   END CATCH
END
```

AddCountry (MM)

Dodaje państwo:
Argumenty:

- @countryName - nazwa kraju

```
CREATE PROCEDURE AddCountry @countryName varchar(255)
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF EXISTS(
           SELECT *
               FROM Countries
               WHERE CountryName = @countryName
               )
           BEGIN
               ;
               THROW 52000, N'Panstwo juz istnieje', 1
           END
       DECLARE @CountryID INT
       SELECT @CountryID = ISNULL(MAX(CountryID), 0) + 1
       FROM Countries
       INSERT INTO Countries(CountryID,CountryName) VALUES
(@CountryID,@countryName)
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Błąd dodawania kraju:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddExam (MM)

Dodaje egzamin do danych studiów:
Argumenty:

- @ExamID - ID egzaminu
- @StudyID - ID studiów
- @date - data egzaminu

```
CREATE PROCEDURE AddExam @ExamID INT,
                         @StudyID INT,
                         @date smalldatetime

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Studies
               WHERE StudyID = @StudyID
               )
           BEGIN
               ;
               THROW 52000, N'Dane studia nie istnieja', 1
           END
       IF EXISTS(
           SELECT *
               FROM Exam
               WHERE StudyID = @StudyID
               )
           BEGIN
               ;
               THROW 52000, N'Dane studia posiadaja juz przypisany egzamin', 1
           END
       INSERT INTO Exam(ExamID, StudyID, Date) VALUES (@ExamID,@StudyID,@date)
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Bład dodawania egzaminu:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
```

```
END
```

AddExamScore (MM)

Dodaje egzamin do danych studiów:
Argumenty:

- @ExamID - ID egzaminu
- @UserID - ID studenta
- @GradeID - ID oceny

```
CREATE PROCEDURE AddExamScore @ExamID INT,
                              @UserID INT,
                              @GradeID INT

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Exam
               WHERE ExamID = @ExamID
               )
           BEGIN
               ;
               THROW 52000, N'Dany egzamin nie istnieja', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM Users
                WHERE Users.UserID IN (Select UserID from Students
               where (Select StudyID from Exam where
ExamID=@ExamID)=Students.StudyID) and Users.UserID=@UserID

               )
           BEGIN
               ;
               THROW 52000, N'Dany student nie istnieje na tych
studiach', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM Grade
               WHERE GradeID = @GradeID
               )
           BEGIN
               ;
               THROW 52000, N'Dana ocena nie istnieja', 1
           END
       DECLARE @ExamScoreID INT
       SELECT @ExamScoreID = ISNULL(MAX(ExamScoreID), 0) + 1
```

```
       FROM ExamScore
       INSERT INTO ExamScore(ExamScoreID, UserID, ExamID, GradeID)
VALUES (@ExamScoreID,@UserID,@ExamID,@GradeID)
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Blad dodawania egzaminu:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddOrder (MM)

Dodaje zamówienie:
Argumenty:

- @UserID - ID użytkownika
- @PaidDate - data zapłacenia wszystkiego, jeśli zapłacił za dane zamówienie

```
CREATE PROCEDURE AddOrder   @UserID INT,
                           @PaidDate smalldatetime = NULL

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Users
               WHERE UserID = @UserID
               )
           BEGIN
               ;
               THROW 52000, N'Dany uzytkownik nie istnieje', 1
           END

       DECLARE @OrderID INT
       SELECT @OrderID = ISNULL(MAX(OrderID), 0) + 1
       FROM Orders
       INSERT INTO Orders(OrderID, UserID, OrderDate, RequiredDate, PaidDate)
VALUES (@OrderID,@UserID,GETDATE(),DATEADD(day, 30, GETDATE()),@PaidDate)
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Blad dodawania orderu:' + CHAR(13)+CHAR(10)
+ ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddProduct (MM)

Dodaje zamówienie:
Argumenty:

- @OrderID - ID zamówienia
- @ProductID - ID produktu
- @money - kwota za dany produkt

```
CREATE PROCEDURE AddProduct   @OrderID INT,
                          @ProductID INT,
                          @money money

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Orders
               WHERE OrderID = @OrderID
               )
           BEGIN
               ;
               THROW 52000, N'Dany order nie istnieje', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM Products
               WHERE ProductID = @ProductID
               )
           BEGIN
               ;
               THROW 52000, N'Dany produkt nie istnieje', 1
           END

       INSERT INTO OrderDetails(OrderID, ProductID, Price) VALUES
(@OrderID,@ProductID,@money)
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Błąd dodawania orderu:' + CHAR(13)+CHAR(10)
+ ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddStationaryModuleMeeting (MM)

Dodaje spotkanie stacjonarne do danego typu modułu :
Argumenty:

- @ModuleID - ID modułu
- @Duration - czas trwania spotkania
- @date - data spotkania
- @PlaceOfMeetingID - ID miejsca spotkania

```
CREATE PROCEDURE AddStationaryModuleMeeting   @ModuleID INT,
                                      @Duration time,
                                      @date smalldatetime,
                                      @PlaceOfMeetingID INT

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM CourseModule
               WHERE ModuleID = @ModuleID and (FormID=1 or FormID=4)
               )
           BEGIN
               ;
               THROW 52000, N'Dany modul nie istnieje w takiej formie', 1
           END
       IF ISNULL(@duration,'00:00:00') < '00:00:00'
           BEGIN
               ;
               THROW 52000, N'Niepoprawny czas trwania', 1
           END
       IF ISNULL(@date,'2000-01-01') < GETDATE()
           BEGIN
               ;
               THROW 52000, N'Niepoprawna data spotkania', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM PlaceOfMeeting
               WHERE PlaceOfMeetingID = @PlaceOfMeetingID
               )
           BEGIN
               ;
               THROW 52000, N'Dane miejsce nie istnieje', 1
           END
       IF EXISTS(
           select *
           from CourseStructure
           where ModuleID=@ModuleID and
                 DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', @Duration),
@date)>[Date meeting/add] and @date <DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00',
Czas_trwania), [Date meeting/add])

       )
           BEGIN
               ;
               THROW 52000, N'Dane spotkanie koliduje juz z innym istniejacym', 1
```

```
           END
       DECLARE @StationaryModuleID INT
       SELECT @StationaryModuleID = ISNULL(MAX(StationaryModuleID), 0) + 1
       FROM StationaryModuleMeeting
       INSERT INTO StationaryModuleMeeting(StationaryModuleID, ModuleID, Date,
Duration, PlaceOfMeetingID) VALUES
(@StationaryModuleID,@ModuleID,@date,@duration,@PlaceOfMeetingID)
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Blad dodawania spotkania:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
go
```

AddOnlineAsyncModuleMeeting (MM)

Dodaje spotkanie online asynchroniczne do danego typu modułu:
Argumenty:

- @ModuleID - ID modułu
- @Duration - czas trwania spotkania
- @RecordingLink - data spotkania

```
CREATE PROCEDURE AddOnlineAsyncModuleMeeting   @ModuleID INT,
                                               @Duration time,
                                               @RecordingLink varchar(255)

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM CourseModule
               WHERE ModuleID = @ModuleID and (FormID=2 or
FormID=4)
               )
           BEGIN
               ;
               THROW 52000, N'Dany modul nie istnieje w takiej
formie', 1
           END
       IF ISNULL(@duration,'00:00:00') < '00:00:00'
           BEGIN
               ;
               THROW 52000, N'Niepoprawny czas trwania', 1
           END
       BEGIN TRAN
       DECLARE @AsyncModuleID INT
       SELECT @AsyncModuleID = ISNULL(MAX(OnlineAsyncID), 0) + 1
       FROM OnlineAsyncModuleMeeting
       INSERT INTO OnlineAsyncModuleMeeting(OnlineAsyncID,
ModuleID, AddDate, Duration, RecordingLink) VALUES
(@AsyncModuleID,@ModuleID,GETDATE(),@Duration,@RecordingLink)
       COMMIT TRAN
   END TRY
   BEGIN CATCH
       IF @@TRANCOUNT > 0
          ROLLBACK TRAN
       DECLARE @msg NVARCHAR(2048) = N'Błąd dodawania spotkania:'
+ CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddOnlineSyncModuleMeeting (MM)

Dodaje spotkanie online synchroniczne do danego typu modułu:
Argumenty:

- @ModuleID - ID modułu
- @Duration - czas trwania spotkania
- @date - data spotkania
- @TranslatorID - ID translatora
- @link - link do spotkania

```
CREATE PROCEDURE AddOnlineSyncModuleMeeting   @ModuleID INT,
                                    @Duration time,
                                    @date smalldatetime,
                                    @TranslatorID INT = NULL,
                                    @link varchar(255)
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM CourseModule
               WHERE ModuleID = @ModuleID and (FormID=3 or FormID=4)
               )
           BEGIN
               ;
               THROW 52000, N'Dany modul nie istnieje w takiej formie', 1
           END
       IF ISNULL(@duration,'00:00:00') < '00:00:00'
           BEGIN
               ;
               THROW 52000, N'Niepoprawny czas trwania', 1
           END
       IF ISNULL(@date,'2000-01-01') < GETDATE()
           BEGIN
               ;
               THROW 52000, N'Niepoprawna data spotkania', 1
           END

       If ISNULL(@TranslatorID,NULL) IS NOT NULL
           IF NOT EXISTS(
           SELECT *
               FROM Translators
               WHERE TranslatorID = @translatorID
               )
           BEGIN
               ;
               THROW 52000, N'Dany translator nie istnieje', 1
           END
       IF EXISTS(
           select *
           from CourseStructure
           where ModuleID=@ModuleID and
                 DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00', @Duration),
@date)>[Date meeting/add] and @date <DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00',
Czas_trwania), [Date meeting/add])

       )
```

```
            BEGIN
                ;
                THROW 52000, N'Dane spotkanie koliduje juz z innym istniejacym', 1
            END
        DECLARE @OnlineSyncModuleID INT
        SELECT @OnlineSyncModuleID = ISNULL(MAX(OnlineSyncID), 0) + 1
        FROM OnlineSyncModuleMeeting
        INSERT INTO OnlineSyncModuleMeeting(OnlineSyncID, ModuleID, Date, Duration,
LinkToOnlineMeeting, RecordingLink, TranslatorID) VALUES
(@OnlineSyncModuleID,@ModuleID,@date,@Duration,@link,NULL,@TranslatorID)
    END TRY
    BEGIN CATCH
        DECLARE @msg NVARCHAR(2048) = N'Bład dodawania spotkania:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

ModifyPrice (MM)

Edytuje już zapłaconą kwotę za dany produkt:
Argumenty:

- @ProductID - ID produktu
- @OrderID - ID zamowienia
- @money - kwota zaplacona za dany produkt

```
CREATE PROCEDURE ModifyPrice
   @ProductID int,
   @OrderID int,
   @money money
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           select *
               from OrderDetails
            where ProductID=@ProductID and OrderID=@OrderID)
           BEGIN
               ;
               THROW 52000, N'Dany order nie istnieje z danym
produktem', 1
           END
       IF @money>dbo.GetPrice(@ProductID)
           BEGIN
               set @money=dbo.GetPrice(@ProductID);
           end
       update OrderDetails
       set Price=@money
       where ProductID=@ProductID and OrderID=@OrderID

   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Bład zmiany kwoty:' +
CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddStationarySubjectMeeting (MM)

Edytuje już zapłaconą kwotę za dany produkt:
Argumenty:

- @SubjectID - ID zajęć
- @date - date spotkania
- @moneys - cena za spotkanie pojedyńcze dla studenta
- @moneyo - cena za spotkanie pojedyńcze dla pozostałych
- @meetingName - nazwa spotkania
- @limit - ilość osób możliwa w danym spotkaniu
- @PlaceOfMeetingID - id miejsca spotkania
- @duration - czas trwania spotkania

```
CREATE PROCEDURE AddStationarySubjectMeeting
                                        @SubjectID INT,
                                         @date smalldatetime,
                                         @duration time,
                                         @PlaceOfMeetingID INT,
                                         @limit INT,
                                         @meetingName varchar(100),
                                         @moneys money,
                                         @moneyo money

AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Subjects
               WHERE SubjectID = @SubjectID
               )
           BEGIN
               ;
               THROW 52000, N'Dany przedmiot nie istnieje', 1
           END
       IF ISNULL(@duration,'00:00:00') < '00:00:00'
           BEGIN
               ;
               THROW 52000, N'Niepoprawny czas trwania', 1
           END
       IF ISNULL(@date,'2000-01-01') < GETDATE()
           BEGIN
               ;
               THROW 52000, N'Niepoprawna data spotkania', 1
           END
       IF NOT EXISTS(
           SELECT *
               FROM PlaceOfMeeting
```

```
                WHERE PlaceOfMeetingID = @PlaceOfMeetingID
                )
            BEGIN
                ;
                THROW 52000, N'Dane miejsce nie istnieje', 1
            END
        IF EXISTS(
            select *
            from StudiesStructure
            where SubjectID=@SubjectID and
                  DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00',
@Duration), @date)>[Date of meeting] and @date <DATEADD(SECOND,
DATEDIFF(SECOND, '00:00:00', Duration), [Date of meeting])

        )
            BEGIN
                ;
                THROW 52000, N'Dane spotkanie koliduje juz z innym
istniejacym', 1
            END
        Begin TRAN
        DECLARE @ProductID INT
        SELECT @ProductID = ISNULL(MAX(ProductID), 0) + 1
        FROM Products
        insert into Products(ProductID, Name) VALUES
(@ProductID,@meetingName)

        DECLARE @StudiesMeetingProductID INT
        SELECT @StudiesMeetingProductID =
ISNULL(MAX(StudiesMeetingProductID), 0) + 1
        FROM StudiesMeetingProduct
        insert into StudiesMeetingProduct(StudiesMeetingProductID,
ProductID, PriceforStudents, PriceforOthers) VALUES
(@StudiesMeetingProductID,@ProductID,@moneys,@moneyo)

        DECLARE @SSubjectID INT
        SELECT @SSubjectID = ISNULL(MAX(SSubjectID), 0) + 1
        FROM StationarySubjectMeeting
        INSERT INTO StationarySubjectMeeting(SSubjectID,
StudiesMeetingProductID, SubjectID, Date, Duration,
PlaceOfMeetingID, Limit) VALUES
(@SSubjectID,@StudiesMeetingProductID,@SubjectID,@date,@duration,@
PlaceOfMeetingID,@limit)
        COMMIT TRAN
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRAN
```

```
        DECLARE @msg NVARCHAR(2048) = N'Blad dodawania spotkania:'
+ CHAR(13)+CHAR(10) + ERROR_MESSAGE();
        THROW 52000,@msg, 1;
    END CATCH
END
```

AddOnLineSubjectMeeting (MM)

Edytuje już zapłaconą kwotę za dany produkt:
Argumenty:

- @SubjectID - ID zajęć
- @date - date spotkania
- @moneys - cena za spotkanie pojedyńcze dla studenta
- @moneyo - cena za spotkanie pojedyńcze dla pozostałych
- @meetingName - nazwa spotkania
- @limit - ilość osób możliwa w danym spotkaniu
- @link - link do spotkania online
- @duration - czas trwania spotkania

```
CREATE PROCEDURE AddOnLineSubjectMeeting   @SubjectID INT,
                                           @date smalldatetime,
                                           @duration time,
                                           @link varchar(255),
                                           @moneys money,
                                           @moneyo money,
                                           @limit INT,
                                           @meetingName varchar(100)
AS
BEGIN
   SET NOCOUNT ON;
   BEGIN TRY
       IF NOT EXISTS(
           SELECT *
               FROM Subjects
               WHERE SubjectID = @SubjectID
               )
           BEGIN
               ;
               THROW 52000, N'Dany przedmiot nie istnieje', 1
           END
       IF ISNULL(@duration,'00:00:00') < '00:00:00'
           BEGIN
               ;
               THROW 52000, N'Niepoprawny czas trwania', 1
           END
       IF ISNULL(@date,'2000-01-01') < GETDATE()
           BEGIN
               ;
               THROW 52000, N'Niepoprawna data spotkania', 1
           END

       IF EXISTS(
           select *
           from StudiesStructure
           where SubjectID=@SubjectID and
```

```
                DATEADD(SECOND, DATEDIFF(SECOND, '00:00:00',
@Duration), @date)>[Date of meeting] and @date <DATEADD(SECOND,
DATEDIFF(SECOND, '00:00:00', Duration), [Date of meeting])

       )
           BEGIN
               ;
               THROW 52000, N'Dane spotkanie koliduje juz z innym
istniejacym', 1
           END
       Begin TRAN
       DECLARE @ProductID INT
       SELECT @ProductID = ISNULL(MAX(ProductID), 0) + 1
       FROM Products
       insert into Products(ProductID, Name) VALUES
(@ProductID,@meetingName)

       DECLARE @StudiesMeetingProductID INT
       SELECT @StudiesMeetingProductID =
ISNULL(MAX(StudiesMeetingProductID), 0) + 1
       FROM StudiesMeetingProduct
       insert into StudiesMeetingProduct(StudiesMeetingProductID,
ProductID, PriceforStudents, PriceforOthers) VALUES
(@StudiesMeetingProductID,@ProductID,@moneys,@moneyo)

       DECLARE @OLSubjectID INT
       SELECT @OLSubjectID = ISNULL(MAX(OLSubjectID), 0) + 1
       FROM OnLineSubjectMeeting
       INSERT INTO OnLineSubjectMeeting(OLSubjectID,
StudiesMeetingProductID, SubjectID, Date, Duration,
LinkToOnlineMeeting, Limit) VALUES
(@OLSubjectID,@StudiesMeetingProductID,@SubjectID,@date,@duration,
@link,@limit)
       COMMIT TRAN
   END TRY
   BEGIN CATCH
       DECLARE @msg NVARCHAR(2048) = N'Bład dodawania spotkania:'
+ CHAR(13)+CHAR(10) + ERROR_MESSAGE();
       THROW 52000,@msg, 1;
   END CATCH
END
```

AddStaff (KL)

Dodaje pracownika szkoły:
Argumenty:

- @FirstName - imię,
- @LastName -nazwisko,
- @BirthDate - data urodzenia,
- @EmploymentDate -data zatrudnienia,
- @CityID - numer miasta,
- @Address -adres zamieszkania,
- @Email - email,
- @Phone -numer telefonu

```
CREATE PROCEDURE [dbo].[AddStaff]
   @FirstName VARCHAR(50),
   @LastName VARCHAR(50),
   @BirthDate DATE,
   @EmploymentDate DATE,
   @CityID INT,
   @Address VARCHAR(50),
   @Email VARCHAR(50),
   @Phone VARCHAR(25)
AS
BEGIN

   -- Sprawdzenie, czy dane miasto istnieje
   IF NOT EXISTS (
       SELECT *
       FROM Cities
       WHERE CityID = @CityID
   )
   BEGIN
       THROW 52000, N'Dane miasto nie zostało wpisane', 1;
   END

   BEGIN TRAN


   DECLARE @StaffID INT;
   SET @StaffID= (SELECT ISNULL(MAX(StaffID),0)+1 from Staff)


   INSERT INTO Staff (StaffID,FirstName, LastName, BirthDate, EmploymentDate,
CityID, Address, Email, Phone)
   VALUES (@StaffID,@FirstName, @LastName, @BirthDate, @EmploymentDate, @CityID,
@Address, @Email, @Phone)

   COMMIT TRAN
END;
```

AddTeacher (KL)

Dodaje nauczyciela:
Argumenty:

- @FirstName - imię,
- @LastName -nazwisko,
- @BirthDate - data urodzenia,
- @EmploymentDate -data zatrudnienia,
- @CityID - numer miasta,
- @Address -adres zamieszkania,
- @Email - email,
- @Phone -numer telefonu

```
create procedure AddTeacher
@FirstName VARCHAR(50),
@LastName VARCHAR(50),
@BirthDate DATE,
@EmploymentDate DATE,
@CityID INT,
@Address VARCHAR(50),
@Email VARCHAR(50),
@Phone VARCHAR(25)
AS
BEGIN

   IF NOT EXISTS (
       SELECT *
       FROM Cities
       WHERE CityID = @CityID
   )
   BEGIN
       THROW 52000, N'Dane miasto nie zostało wpisane', 1;
   END

   BEGIN TRAN


   DECLARE @StaffID INT;
   SET @StaffID= (SELECT ISNULL(MAX(StaffID),0)+1 from Staff)


   INSERT INTO Staff (StaffID,FirstName, LastName, BirthDate, EmploymentDate,
CityID, Address, Email, Phone)
   VALUES (@StaffID,@FirstName, @LastName, @BirthDate, @EmploymentDate, @CityID,
@Address, @Email, @Phone)

   DECLARE @TeacherID INT;
   SET @TeacherID= (SELECT ISNULL(MAX(TeacherID),0)+1 from Teachers)


   INSERT INTO Teachers(StaffID,TeacherID)
   VALUES (@StaffID,@TeacherID)

   COMMIT TRAN
END;
```

AddTranslator (KL)

Dodaje tłumacza:
Argumenty:

- @FirstName - imię,
- @LastName -nazwisko,
- @BirthDate - data urodzenia,
- @EmploymentDate -data zatrudnienia,
- @CityID - numer miasta,
- @Address -adres zamieszkania,
- @Email - email,
- @Phone -numer telefonu,
- @LanguageID - numer języka

```
CREATE procedure AddTranslator
@FirstName VARCHAR(50),
@LastName VARCHAR(50),
@BirthDate DATE,
@EmploymentDate DATE,
@CityID INT,
@Address VARCHAR(50),
@Email VARCHAR(50),
@Phone VARCHAR(25),
@LanguageID INT
AS
BEGIN
   IF NOT EXISTS (
       SELECT *
       FROM Cities
       WHERE CityID = @CityID
   )
   BEGIN
       THROW 52000, N'Dane miasto nie zostało wpisane', 1;
   END
   IF NOT EXISTS (
       SELECT *
       FROM Languages
       WHERE LanguageID = @LanguageID
   )
   BEGIN
       THROW 52000, N'Dany język nie został wpisany', 1;
   END
   BEGIN TRAN

   DECLARE @StaffID INT;
   SET @StaffID= (SELECT ISNULL(MAX(StaffID),0)+1 from Staff)

   INSERT INTO Staff (StaffID,FirstName, LastName, BirthDate, EmploymentDate,
CityID, Address, Email, Phone)
   VALUES (@StaffID,@FirstName, @LastName, @BirthDate, @EmploymentDate, @CityID,
@Address, @Email, @Phone)
   DECLARE @TranslatorID INT;
   SET @TranslatorID= (SELECT ISNULL(MAX(TranslatorID),0)+1 from Translators)
   INSERT INTO Translators(StaffID,TranslatorID,LanguageID)
   VALUES (@StaffID,@TranslatorID,@LanguageID)
   COMMIT TRAN
END;
```

AddInternship (KL)

Dodaje praktyki:
Argumenty:

- @StudyID -numer studiów,
- @InternshipName -nazwa praktyk

```
Create procedure AddInternship
@StudyID INT,
@InternshipName VARCHAR(50)
AS
BEGIN

   IF NOT EXISTS (
       SELECT *
       FROM Studies
       WHERE StudyID = @StudyID
   )
   BEGIN
       THROW 52000, N'Dane studia nie istnieją', 1;
   END

   BEGIN TRAN

   DECLARE @InternshipID INT;
   SET @InternshipID= (SELECT ISNULL(MAX(InternshipID),0)+1 from
Internships)


   INSERT INTO Internships(StudyID,InternshipID,InternshipName)
   VALUES (@StudyID,@InternshipID, @InternshipName)

   COMMIT TRAN
END;
```

AddCity (KL)

Dodaje miasto:
Argumenty:

- @CityName - nazwa miasta,
- @CountryID - numer ID państwa

```
CREATE PROCEDURE AddCity
   @CityName VARCHAR(50),
   @CountryID INT
AS
BEGIN

   -- Sprawdzenie, czy dane panstwo istnieje
   IF NOT EXISTS (
       SELECT *
       FROM Countries
       WHERE CountryID = @CountryID
   )
   BEGIN
       THROW 52000, N'Dane panstwo nie zostało wpisane', 1;
   END

   BEGIN TRAN


   DECLARE @CityID INT;
   SET @CityID= (SELECT ISNULL(MAX(CityID),0)+1 from Cities)

   insert into Cities values
   (@CityID,@CityName,@CountryID)

   COMMIT TRAN
END;
```

AddCourseAttendance (KL)

Uzupełnia listę obecności:
Argumenty:

- @UserID - numer ID użytkownika,
- @CourseID - numer ID kursu
- @Form - numer ID formy spotkania (stacjonarne, asynchroniczne, synchroniczne)
- @MeetingID - numer spotkania

```
CREATE PROCEDURE AddCourseAttendance
   @UserID INT,
   @CourseID INT,
   @Form INT, -- Forma spotkania
   @MeetingID INT
AS
BEGIN
   -- Sprawdzenie czy forma jest z dobrego przedziału
   IF @Form NOT BETWEEN 1 AND 3
   BEGIN
       THROW 52000, N'Forma zajęć spoza przedziału', 1;
   END

   -- Sprawdzenie czy istnieje użytkownik
   IF NOT EXISTS (SELECT * FROM Users WHERE UserID = @UserID)
   BEGIN
       THROW 52000, N'Dany użytkownik nie istnieje', 1;
   END

   -- Sprawdzenie czy istnieje kurs
   IF NOT EXISTS (SELECT * FROM Courses WHERE CourseID = @CourseID)
   BEGIN
       THROW 52000, N'Dany kurs nie istnieje', 1;
   END

   -- Sprawdzenie czy istnieje spotkanie
   IF dbo.ifExistsMeetingCourse(@MeetingID, @Form) = 0
   BEGIN
       THROW 52000, N'Dane spotkanie nie odbyło się', 1;
   END

   -- Sprawdzenie, czy użytkownik jest zapisany na kurs
   IF NOT EXISTS (
       SELECT 1
       FROM Orders AS o
       INNER JOIN OrderDetails AS od ON od.OrderID = o.OrderID
       INNER JOIN Users AS u ON u.UserID = o.UserID
       INNER JOIN Products AS p ON p.ProductID = od.ProductID
```

```
        INNER JOIN CoursesProduct AS cp ON cp.ProductID =
p.ProductID
        INNER JOIN Courses AS c ON c.CourseProductID =
cp.CourseProductID
        WHERE u.UserID = @UserID AND c.CourseID = @CourseID AND
o.PaidDate IS NOT NULL
    )
    BEGIN
        THROW 52000, N'Dany użytkownik nie jest zapisany na dany
kurs', 1;
    END

    -- Dodanie rekordu do odpowiedniej tabeli zgodnie z formą
    BEGIN TRAN
    DECLARE @AttendanceID INT;

    IF @Form = 1
    BEGIN
        SET @AttendanceID = (SELECT ISNULL(MAX(AttendanceID), 0) + 1
FROM StationaryAttendanceListModule);
        INSERT INTO StationaryAttendanceListModule VALUES
(@AttendanceID, @MeetingID, @UserID, 1);
    END

    IF @Form = 2
    BEGIN
        SET @AttendanceID = (SELECT ISNULL(MAX(AttendanceID), 0) + 1
FROM OnlineAsyncAttendanceListModule);
        INSERT INTO OnlineAsyncAttendanceListModule VALUES
(@AttendanceID, @MeetingID, @UserID, 1);
    END

    IF @Form = 3
    BEGIN
        SET @AttendanceID = (SELECT ISNULL(MAX(AttendanceID), 0) + 1
FROM OnlineSyncAttendanceListModule);
        INSERT INTO OnlineSyncAttendanceListModule VALUES
(@AttendanceID, @MeetingID, @UserID, 1);
    END

    COMMIT TRAN
END;
```

AddStudiesAttendance (KL)

Dodaje obecność do spotkania na studiach:
Argumenty:

- @UserID - numer ID użytkownika,
- @FormID - numer ID formy spotkania (stacjonarne, online)
- @MeetingID - numer ID spotkania
- @Presence - obecny (1) / nieobecny (0)

```
CREATE PROCEDURE [dbo].[AddStudiesAttendance]
   @UserID INT,
   @Form INT, -- Forma spotkania (stationary,online)
   @MeetingID INT,
   @Presence BIT
AS
BEGIN
   -- Sprawdzenie czy forma jest z dobrego przedziału
   IF @Form NOT BETWEEN 1 AND 2
   BEGIN
       THROW 52000, N'Forma zajęć spoza przedziału', 1;
   END

   -- Sprawdzenie czy istnieje użytkownik
   IF NOT EXISTS (SELECT * FROM Users WHERE UserID = @UserID)
   BEGIN
       THROW 52000, N'Dany użytkownik nie istnieje', 1;
   END

   -- Sprawdzenie czy istnieje spotkanie
   IF dbo.ifExistsMeetingStudies(@MeetingID,@Form) = 0
   BEGIN
       THROW 52000, N'Dane spotkanie nie odbyło się', 1;
   END

   -- Sprawdzenie, czy użytkownik jest zapisany na spotkanie
  IF dbo.ifRegisteredStudies(@UserID,@MeetingID,@Form) = 0
   BEGIN
       THROW 52000, N'Dany użytkownik nie jest zapisany na dane
spotkanie', 1;
   END

   -- Dodanie rekordu do odpowiedniej tabeli zgodnie z formą
   BEGIN TRAN
   DECLARE @AttendanceID INT;

   IF @Form = 1
```

```
   BEGIN
       SET @AttendanceID = (SELECT ISNULL(MAX(AttendanceID), 0) + 1
FROM StationaryAttendanceListStudies);
       INSERT INTO StationaryAttendanceListStudies VALUES
(@AttendanceID, @MeetingID, @UserID, @Presence,NULL);
   END

   IF @Form = 2
   BEGIN
       SET @AttendanceID = (SELECT ISNULL(MAX(AttendanceID), 0) + 1
FROM OnLineAttendanceListStudies);
       INSERT INTO OnLineAttendanceListStudies VALUES
(@AttendanceID, @MeetingID, @UserID, @Presence,NULL);
   END

   COMMIT TRAN
END;
```

Funkcje

WebinarOrderCount (IS)

Zwraca ilość zamówień danego webinaru
Argumenty:

- WebinarID - ID webinaru

```
create function WebinarOrderCount(@WebinarID int)
   returns int
as begin
   return (
       select count(od.ProductID)
       from OrderDetails od
       join WebinarProduct wp on wp.ProductID = od.ProductID
       join Webinars w on w.WebinarProductID = wp.WebinarProductID
       where WebinarID=@WebinarID
   )
end
```

GetStudyVacancies (IS)

Zwraca liczbę wolnych miejsc dla danych studiów
Argumenty:

- StudyID - ID studiów

```
create function GetStudyVacancies(@StudyID int)
   returns int
as begin
   return (
       select Limit-count(od.ProductID)
       from Studies s
       join StudiesProduct sp on sp.StudiesProductID=s.StudiesProductID
       join OrderDetails od on od.ProductID=sp.ProductID
       where s.StudyID=@StudyID
       group by StudyID, Limit
   )
end
```

GetMeetingVacancies (IS)

Zwraca liczbę wolnych miejsc dla danego spotkania
Argumenty:

- StudiesMeetingProductID - ID spotkania

```
CREATE function GetMeetingVacancies(@StudiesMeetingProductID int)
   returns int
as begin
   return (
       case
           when exists(
               select *
               from StationarySubjectMeeting m
               where m.StudiesMeetingProductID=@StudiesMeetingProductID
           ) then (
               select Limit-count(od.ProductID)
               from StationarySubjectMeeting m
               join StudiesProduct sp on
sp.StudiesProductID=m.StudiesMeetingProductID
               join OrderDetails od on od.ProductID=sp.ProductID
               where m.StudiesMeetingProductID=@StudiesMeetingProductID
               group by SSubjectID, Limit
           )
           else (
               select Limit-count(od.ProductID)
               from OnLineSubjectMeeting m
               join StudiesProduct sp on
sp.StudiesProductID=m.StudiesMeetingProductID
               join OrderDetails od on od.ProductID=sp.ProductID
               where m.StudiesMeetingProductID=@StudiesMeetingProductID
               group by OLSubjectID, Limit
               )
       end
   )
end
```

GetGrades (IS)

Zwraca wartości ocen studenta
Argumenty:

- UserID - ID studenta

```
create function GetGrades(@UserID int)
   returns varchar(100)
as begin
   return (
       select STRING_AGG(Grade, ', ')
       from Grade g
       join ExamScore es on es.GradeID=g.GradeID
       where UserID=@UserID
   )
end
```

GetPrice (IS)

Zwraca zwykłą cenę produktu
Argumenty:

- ProductID - ID produktu

```
create function GetPrice(@ProductID int)
   returns money
as begin
   return (
       case
           when exists(
               select * from StudiesProduct
               where ProductID=@ProductID
           ) then (
               select Price from StudiesProduct
               where ProductID=@ProductID
           )
           when exists(
               select * from StudiesMeetingProduct
               where ProductID=@ProductID
           ) then (
               select PriceForOthers from StudiesMeetingProduct
               where ProductID=@ProductID
           )
           when exists(
               select * from CoursesProduct
               where ProductID=@ProductID
           ) then (
               select Price from CoursesProduct
               where ProductID=@ProductID
           )
           when exists(
               select * from WebinarProduct
               where ProductID=@ProductID
           ) then (
               select WebinarPrice from WebinarProduct
               where ProductID=@ProductID
           )
       end
   )
end
```

GetPriceAlt (IS)

Zwraca alternatywną cenę/wartość dla produktu lub null jeśli produkt jej nie ma

- dla studiów - wejściowe
- dla pojedynczych zajęć ze studiów - cena dla studentów
- dla kursów - wartość zaliczki

Argumenty:

- ProductID - ID produktu

```
create function GetPriceAlt(@ProductID int)
   returns money
as begin
   return (
       case
            when exists(
                select * from StudiesProduct
                where ProductID=@ProductID
            ) then (
                select Entry_fee from StudiesProduct
                where ProductID=@ProductID
            )
            when exists(
                select * from StudiesMeetingProduct
                where ProductID=@ProductID
            ) then (
                select PriceforStudents from StudiesMeetingProduct
                where ProductID=@ProductID
            )
            when exists(
                select * from CoursesProduct
                where ProductID=@ProductID
            ) then (
                select PriceAdvance from CoursesProduct
                where ProductID=@ProductID
            )
       end
   )
end
```

GetMaxPriceOfProduct (MM)

Zwraca tabele z najdroższymi produktami typu webinar,kurs,studia
Argumenty:

- @amount - ilosc najdroższych produktów

```
CREATE FUNCTION GetMaxPriceOfProduct(@amount int)
   RETURNS TABLE AS
       RETURN
   SELECT Distinct TOP (@amount) WCS.ProductID,WCS.Name,WCS.[full
price],WCS.min_price_to_order_product
   FROM WCSPrices as WCS
   order by WCS.[full price]
```

GetClientsOrderedMoreThanXTimes (MM)

Zwraca tabele użytkowników, którzy zamówili X razy
Argumenty:

- @amount - ilosc zamowień

```
CREATE FUNCTION GetClientsOrderedMoreThanXTimes(@amount int)
   RETURNS TABLE AS
       RETURN
           SELECT *
       FROM ClientStats
WHERE times_ordered > @amount
```

GetBestProducts (MM)

Zwraca tabele z najbardziej dochodowymi produktami
Argumenty:

- @input - ilość

```
CREATE FUNCTION GetBestProducts(@input int)
   RETURNS table AS
       RETURN
           SELECT DISTINCT TOP (@input) P.Name, RP.przychód
           FROM Products P
           INNER JOIN RaportyFinansowe RP on P.ProductID =
RP.ProductID
           ORDER BY RP.przychód
```

GetMinPriceOfProduct (MM)

Zwraca tabele z najtańszymi produktami typu webinar,kurs,studia
Argumenty:

- @amount - ilosc najtańszych produktów

```
CREATE FUNCTION GetMinPriceOfProduct(@amount int)
   RETURNS TABLE AS
       RETURN
   SELECT Distinct TOP (@amount) WCS.ProductID,WCS.Name,WCS.[full
price],WCS.min_price_to_order_product
   FROM WCSPrices as WCS
   order by WCS.[full price] desc
```

GetProductsSoldAtLeastXTimes (KL)

Zwraca pozycje, które sprzedały się więcej niż przyjętą jako argument liczbę razy.

```
Create FUNCTION GetProductsSoldAtLeastXTimes(@input int)
RETURNS table
AS return
select p.ProductID,p.Name, count(*) as 'liczba sprzedanych
produktow'
from Products as p
inner join OrderDetails as od
on od.ProductID=p.ProductID
group by p.ProductID,p.Name having count(*)>@input
```

GetSylabusForStudies (MM)

Zwraca tablice zawierająca sylabus danych studiów.

```
Create FUNCTION GetSylabusForStudies(@studyID int)
RETURNS table
AS return
select s.StudyID,s.SubjectName as NazwaPrzedmiotu
from Subjects as s
where StudyID=@studyID
```

GetValueOfOrdersOnDay (KL)

Zwraca wartość zamówień podczas przyjętego jako argument dnia

```
CREATE FUNCTION getValueOfOrdersOnDay(@date date)
RETURNS int
AS
BEGIN
RETURN (select
Case
   when not exists(select * from Orders as o where
o.OrderDate=@date) then 0
   else
(select sum(od.Price)
from Orders as o
inner join OrderDetails as od
on o.OrderID=od.OrderID
where o.OrderDate=@date
group by o.OrderDate
)
end as 'cena za dany dzien')
END
```

GetPriceOfOrder (MM)

Zwraca wartość podanego koszyka jaką ma dany koszyk.

```
CREATE function GetPriceOfOrder(@OrderID int)
   returns money
as begin
   return (
       select sum(Price) as 'Cena za koszyk' from OrdersAll
       where OrderID=@OrderID
   )
end
```

WebinarInfo (KL)

Zwraca dokładne informacje na temat wybranego Webinaru.

```
CREATE FUNCTION WebinarInfo(@input int)
RETURNS table
AS return
select w.WebinarID,p.Name as
'WebinarName',w.CoordinatorID,s.FirstName+' '+s.LastName as
'Coordinator Name',l.Language from Products as p
inner join WebinarProduct as wp on wp.ProductID=p.ProductID
inner join Webinars as w on w.WebinarProductID=wp.WebinarProductID
inner join Teachers as t on t.TeacherID=w.CoordinatorID
inner join Staff as s on s.StaffID=t.StaffID
inner join Languages as l on l.LanguageID=w.LanguageID
where w.WebinarID=@input
```

CourseInfo (MM)

Zwraca dokładne informacje na temat wybranego kursu.

```
CREATE FUNCTION WebinarInfo(@input int)
RETURNS table
AS return
select w.WebinarID,p.Name as
'WebinarName',w.CoordinatorID,s.FirstName+' '+s.LastName as
'Coordinator Name',l.Language from Products as p
inner join WebinarProduct as wp on wp.ProductID=p.ProductID
inner join Webinars as w on w.WebinarProductID=wp.WebinarProductID
inner join Teachers as t on t.TeacherID=w.CoordinatorID
inner join Staff as s on s.StaffID=t.StaffID
inner join Languages as l on l.LanguageID=w.LanguageID
where w.WebinarID=@input
```

StudyInfo (MM)

Zwraca dokładne informacje na temat wybranego studium.

```
CREATE FUNCTION StudyInfo(@input int)
RETURNS table
AS return
select * from StudiesStructure
where [Studies number]=@input
```

ifExistsMeetingCourse (KL)

Funkcja pomocnicza do widoku addCourseAttendance. Sprawdza czy istnieje spotkanie o podanym indeksie.

```
CREATE FUNCTION [dbo].[ifExistsMeetingCourse](@meetingID INT,
@formID INT)
RETURNS BIT
AS
BEGIN
   DECLARE @result BIT;

   SET @result = (
       CASE
           WHEN @formID = 1 THEN
               CASE
                   WHEN EXISTS (
                       SELECT 1
                       FROM CourseModule AS cm
                       INNER JOIN StationaryModuleMeeting AS smm
ON smm.ModuleID = cm.ModuleID
                       WHERE smm.StationaryModuleID = @meetingID
AND smm.Date < GETDATE()
                   ) THEN 1
                   ELSE 0
               END
           WHEN @formID = 2 THEN
               CASE
                   WHEN EXISTS (
                       SELECT 1
                       FROM CourseModule AS cm
                       INNER JOIN OnlineAsyncModuleMeeting AS oamm
ON oamm.ModuleID = cm.ModuleID
                       WHERE oamm.OnlineAsyncID = @meetingID AND
oamm.AddDate < GETDATE()
                   ) THEN 1
                   ELSE 0
               END
           WHEN @formID = 3 THEN
               CASE
                   WHEN EXISTS (
                       SELECT 1
                       FROM CourseModule AS cm
                       INNER JOIN OnlineSyncModuleMeeting AS osmm
ON osmm.ModuleID = cm.ModuleID
```

```
                        WHERE osmm.OnlineSyncID = @meetingID AND
osmm.Date < GETDATE()
                    ) THEN 1
                    ELSE 0
                END
            ELSE 0
        END
    );

    RETURN @result;
END;
```

GetBasket (MM)

Funkcja zwraca dany koszyk.

```
Create FUNCTION GetBasket(@orderID int)
RETURNS table
AS return
select * from AllOrders
where OrderID=@orderID
```

GetBasket (MM)

Funkcja zwraca ile dany uzytkownik musi zapłacić za dane zamówienie

```
CREATE function HowMuchNeedUserPayForBasket(@OrderID int)
   returns table
as
   return (
       select [User] as 'kupujacy', sum(Price)-sum(Paid) as 'Cena
za koszyk' from AllOrders
       where OrderID=@OrderID
   )
```

.

GetListOfAllParticipants (MM)

Funkcja zwraca listę osób na przyszłe i teraźniejsze spotkania.

```
Create FUNCTION GetListOfParticipants(@ProductID int)
RETURNS table
AS return
select distinct FirstName,LastName from
AllFutureAndPresentMeetings
inner join Users on
Users.UserID=AllFutureAndPresentMeetings.UserID
where ProductID=@ProductID
```

ifExistsMeetingStudies (KL)

Funkcja sprawdza czy istnieje dane spotkanie na studiach.

```
CREATE FUNCTION [dbo].[ifExistsMeetingStudies](@meetingID INT,
@formID INT)
RETURNS BIT
AS
BEGIN
   DECLARE @result BIT;

   SET @result = (
       CASE
           WHEN @formID = 1 THEN
               CASE
                   WHEN EXISTS (
                       SELECT 1
                       FROM StationarySubjectMeeting AS ssm
                       WHERE ssm.SSubjectID= @meetingID AND
ssm.Date < GETDATE()
                   ) THEN 1
                   ELSE 0
               END
           WHEN @formID = 2 THEN
               CASE
                   WHEN EXISTS (
                       SELECT 1
                       FROM OnLineSubjectMeeting as osm
                       WHERE osm.OLSubjectID = @meetingID AND
osm.Date < GETDATE()
                   ) THEN 1
                   ELSE 0
               END
           ELSE 0
       END
   );

   RETURN @result;
END;
```

ifRegisteredStudies (KL)

Funkcja sprawdza czy użytkownik jest zapisany na dane spotkanie na studiach.

```
CREATE FUNCTION [dbo].[ifRegisteredStudies](@UserID INT, @MeetingID
INT,@FormID INT)
RETURNS BIT
AS
BEGIN
   DECLARE @result BIT;

   SET @result = (
       CASE
           WHEN @formID = 1 THEN
               CASE
                   WHEN EXISTS (
                       select u.UserID as us from
StationarySubjectMeeting as ssm
                   inner join StudiesMeetingProduct as smp on
smp.StudiesMeetingProductID=ssm.StudiesMeetingProductID
                   inner join Products as p on
p.ProductID=smp.ProductID
                   inner join OrderDetails as od on
od.ProductID=p.ProductID
                   inner join Orders as o on o.OrderID=od.OrderID
                   inner join Users as u on u.UserID=o.UserID
                   where u.UserID=@UserID and o.PaidDate is not
NULL
                   union
                   select u.UserID as us from
StationarySubjectMeeting as ssm
                   inner join Subjects as s on
s.SubjectID=ssm.SubjectID
                   inner join Studies as stud on
stud.StudyID=s.StudyID
                   inner join StudiesProduct as sp on
sp.StudiesProductID=stud.StudiesProductID
                   inner join Products as p on
p.ProductID=sp.ProductID
                   inner join OrderDetails as od on
od.ProductID=p.ProductID
                   inner join Orders as o on o.OrderID=od.OrderID
                   inner join Users as u on u.UserID=o.UserID
```

```
                    where u.UserID=@UserID and o.PaidDate is not
NULL
                     ) THEN 1
                     ELSE 0
                END
            WHEN @formID = 2 THEN
                CASE
                     WHEN EXISTS (
                         select u.UserID as us from
OnLineSubjectMeeting as osm
                    inner join StudiesMeetingProduct as smp on
smp.StudiesMeetingProductID=osm.StudiesMeetingProductID
                    inner join Products as p on
p.ProductID=smp.ProductID
                    inner join OrderDetails as od on
od.ProductID=p.ProductID
                    inner join Orders as o on o.OrderID=od.OrderID
                    inner join Users as u on u.UserID=o.UserID
                    where u.UserID=@UserID and o.PaidDate is not
NULL
                    union
                    select u.UserID as us from OnLineSubjectMeeting
as osm
                    inner join Subjects as s on
s.SubjectID=osm.SubjectID
                    inner join Studies as stud on
stud.StudyID=s.StudyID
                    inner join StudiesProduct as sp on
sp.StudiesProductID=stud.StudiesProductID
                    inner join Products as p on
p.ProductID=sp.ProductID
                    inner join OrderDetails as od on
od.ProductID=p.ProductID
                    inner join Orders as o on o.OrderID=od.OrderID
                    inner join Users as u on u.UserID=o.UserID
                    where u.UserID=@UserID and o.PaidDate is not
NULL
                     ) THEN 1
                     ELSE 0
                END
            ELSE 0
        END
    );

    RETURN @result;
END;
```

ifPassedModule (KL)

Funkcja sprawdza czy dana osoba zaliczyła dany moduł.

```
CREATE FUNCTION [dbo].[ifPassedModule](@UserID INT, @ModuleID INT)
RETURNS BIT
AS
BEGIN
   DECLARE @result BIT;

   declare @att INT;
   declare @amount INT;

   set @att=(
   select count(*) from StationaryAttendanceListModule as salm
   where salm.UserID=@UserID and salm.StationaryModuleID in (
   select smm.StationaryModuleID from StationaryModuleMeeting as
smm
   where smm.ModuleID=@ModuleID)
   )

   set @amount=(

   select count(*) from CourseModule as cm
   inner join StationaryModuleMeeting as smm on
smm.ModuleID=cm.ModuleID
   where cm.ModuleID=@ModuleID
   )


   IF @att = @amount
       SET @result = 1;
   ELSE
       SET @result = 0;

   RETURN @result;
END;
```

Triggery

ExceptionValidDate

Sprawdza, czy data wyjątku nie jest wcześniejsza niż data zamówienia

```
create trigger ExceptionValidDate
   on Exception
   for insert, update
as begin
   if (select Date from inserted) <= (select RequiredDate from Orders where
OrderID=(select OrderID from inserted)) begin
       raiserror('Date cannot be earlier than order''s RequiredDate', 16, 1)
       rollback transaction
   end
end
```

CourseValidDateSMeeting

Sprawdza, czy data spotkania jest między datą rozpoczęcia i zakończenia kursu

```
create trigger CourseValidDateSMeeting
   on StationaryModuleMeeting
   for insert, update
as begin
   if (select Date from inserted) not between (select StartDate from Courses where
CourseID=(select CourseID from inserted join CourseModule cm on cm.ModuleID=(select
ModuleID from inserted))) and (select EndDate from Courses where CourseID=(select
CourseID from inserted join CourseModule cm on cm.ModuleID=(select ModuleID from
inserted))) begin
       raiserror('Date must be withing course''s start and end dates', 16, 1)
       rollback transaction
   end
end
```

CourseValidDateOAMeeting

Sprawdza, czy data spotkania jest między datą rozpoczęcia i zakończenia kursu

```
create trigger CourseValidDateOAMeeting
   on OnlineAsyncModuleMeeting
   for insert, update
as begin
   if (select AddDate from inserted) not between (select StartDate from Courses
where CourseID=(select CourseID from inserted join CourseModule cm on
cm.ModuleID=(select ModuleID from inserted))) and (select EndDate from Courses
where CourseID=(select CourseID from inserted join CourseModule cm on
cm.ModuleID=(select ModuleID from inserted))) begin
       raiserror('Date must be withing course''s start and end dates', 16, 1)
       rollback transaction
   end
end
```

CourseValidDateOSMeeting

Sprawdza, czy data spotkania jest między datą rozpoczęcia i zakończenia kursu

```
create trigger CourseValidDateOSMeeting
   on OnlineSyncModuleMeeting
   for insert, update
as begin
   if (select Date from inserted) not between (select StartDate from Courses where
CourseID=(select CourseID from inserted join CourseModule cm on cm.ModuleID=(select
ModuleID from inserted))) and (select EndDate from Courses where CourseID=(select
CourseID from inserted join CourseModule cm on cm.ModuleID=(select ModuleID from
inserted))) begin
       raiserror('Date must be withing course''s start and end dates', 16, 1)
       rollback transaction
   end
end
```

StudiesValidDateSMeeting

Sprawdza, czy data spotkania jest między datą rozpoczęcia i zakończenia studiów

```
create trigger StudiesValidDateSMeeting
   on StationarySubjectMeeting
   for insert, update
as begin
   if (select Date from inserted) not between (select StartDate from Studies where
StudyID=(select StudyID from inserted join Subjects s on s.SubjectID=(select
SubjectID from inserted))) and (select EndDate from Studies where StudyID=(select
StudyID from inserted join Subjects s on s.SubjectID=(select SubjectID from
inserted))) begin
       raiserror('Date must be withing study course''s start and end dates', 16, 1)
       rollback transaction
   end
end
go
```

StudiesValidDateOMeeting

Sprawdza, czy data spotkania jest między datą rozpoczęcia i zakończenia studiów

```
create trigger StudiesValidDateOMeeting
   on OnLineSubjectMeeting
   for insert, update
as begin
   if (select Date from inserted) not between (select StartDate from Studies where
StudyID=(select StudyID from inserted join Subjects s on s.SubjectID=(select
SubjectID from inserted))) and (select EndDate from Studies where StudyID=(select
StudyID from inserted join Subjects s on s.SubjectID=(select SubjectID from
inserted))) begin
       raiserror('Date must be withing study course''s start and end dates', 16, 1)
       rollback transaction
   end
end
```

PaidDateAfterPayingAllProducts

Sprawdza, czy wszystkie produkty zostały zapłacone w danym zamówieniu. Jeśli tak to ustawia PaidDate w tablicy order.

```
CREATE TRIGGER PaidDateAfterPayingAllProducts
ON OrderDetails
AFTER UPDATE
AS
BEGIN
   DECLARE @OrderPaid INT = (Select  inserted.OrderID from
inserted)
   DECLARE @MoneyPaid MONEY = (Select sum(Price) from OrderDetails
where OrderID=@OrderPaid)
   IF @MoneyPaid=dbo.GetPriceOfOrder(@OrderPaid)
   BEGIN
       update Orders
       set PaidDate=GETDATE()
       where OrderID=@OrderPaid
   end
END;
```

OnStudyLimitChange

Blokuje zmianę limitu miejsc gdy ilość osób zapisanych na studia jest większa niż wartość zmiany Limitu

```
create trigger OnStudyLimitChange on Studies for update
as
begin
if (select count(UserID) from Students where
Students.StudyID=(select StudyID from inserted)) > (select Limit
from inserted)
begin
RAISERROR(N'Nie można zmodyfikować danego limitu', 11, 1)
rollback transaction
end
end
```

Indeksy

Tabela Countries

```
create unique index Countries_pk on Countries (CountryID)
create unique index CountryNameUnique on Countries (CountryName)
```

Tabela Cities

```
create unique index Cities_pk on Cities (CityID)
```

Tabela Form

```
create unique index Form_pk on Form (FormID)
create unique index FormNameUnique on Form (FormName)
```

Tabela Grade

```
create unique index Grade_pk on Grade (GradeID)
```

Tabela Languages

```
create unique index Languages_pk on Languages (LanguageID)
```

Tabela PlaceOfMeeting

```
create unique index PlaceOfMeeting_pk on PlaceOfMeeting (PlaceOfMeetingID)
```

Tabela Products

```
create unique index Products_pk on Products (ProductID)
```

Tabela CoursesProduct

```
create unique index CoursesProduct_pk on CoursesProduct (CourseProductID)
```

Tabela Staff

```
create unique index Staff_pk on Staff (StaffID)
create unique index PhoneUniqueStaff on Staff (Phone)
create unique index EmailUniqueStaff on Staff (Email)
```

Tabela StudiesMeetingProduct

```
create unique index StudiesMeetingProduct_pk on StudiesMeetingProduct
(StudiesMeetingProductID)
```

Tabela StudiesProduct

```
create unique index StudiesProduct_pk on StudiesProduct (StudiesProductID)
```

Tabela Teachers

```
create unique index Teachers_pk on Teachers (TeacherID)
```

Tabela Courses

```
create unique index Courses_pk on Courses (CourseID)
```

Tabela CourseModule

```
create unique index CourseModule_pk on CourseModule (ModuleID)
```

Tabela OnlineAsyncModuleMeeting

```
create unique index OnlineAsyncModuleMeeting_pk on OnlineAsyncModuleMeeting
(OnlineAsyncID)
```

Tabela RecordingLinkUnique

```
create unique index RecordingLinkUnique on OnlineAsyncModuleMeeting (RecordingLink)
```

Tabela StationaryModuleMeeting

```
create unique index StationaryModuleMeeting_pk on StationaryModuleMeeting
(StationaryModuleID)
```

Tabela Studies

```
create unique index Studies_pk on Studies (StudyID)
```

Tabela Exam

```
create unique index Exam_pk on Exam (ExamID)
```

Tabela Internships

```
create unique index Internships_pk on Internships (InternshipID)
```

Tabela Subjects

```
create unique index Subjects_pk on Subjects (SubjectID)
```

Tabela OnLineSubjectMeeting

```
create unique index OnLineSubjectMeeting_pk on OnLineSubjectMeeting (OLSubjectID)
```

Tabela StationarySubjectMeeting

```
create unique index StationarySubjectMeeting_pk on StationarySubjectMeeting
(SSubjectID)
```

Tabela Translators

```
create unique index Translators_pk on Translators (TranslatorID)
```

Tabela OnlineSyncModuleMeeting

```
create unique index OnlineSyncModuleMeeting_pk on OnlineSyncModuleMeeting
(OnlineSyncID)
```

Tabela RecordingLinkUnique

```
create unique index RecordingLinkUnique on OnlineSyncModuleMeeting (RecordingLink)
```

Tabela Users

```
create unique index Users_pk on Users (UserID)
create unique index PhoneUnique on Users (Phone)
create unique index EmailUnique on Users (Email)
```

Tabela AttendanceInternship

```
create unique index AttendanceInternship_pk on AttendanceInternship (AttendanceID)
```

Tabela ExamScore

```
create unique index ExamScore_pk on ExamScore (ExamScoreID)
```

Tabela OnLineAttendanceListStudies

```
create unique index OnLineAttendanceListStudies_pk on OnLineAttendanceListStudies
(AttendanceID)
```

Tabela OnlineAsyncAttendanceListModule

```
create unique index OnlineAsyncAttendanceListModule_pk on
OnlineAsyncAttendanceListModule (AttendanceID)
```

Tabela OnlineSyncAttendanceListModule

```
create unique index OnlineSyncAttendanceListModule_pk on
OnlineSyncAttendanceListModule (AttendanceID)
```

Tabela Orders

```
create unique index Orders_pk on Orders (OrderID)
```

Tabela Exception

```
create unique index Exception_pk on Exception (ExceptionID)
```

Tabela OrderDetails

```
create unique index OrderDetails_pk on OrderDetails (OrderID, ProductID)
```

Tabela StationaryAttendanceListModule

```
create unique index StationaryAttendanceListModule_pk on
StationaryAttendanceListModule (AttendanceID)
```

Tabela StationaryAttendanceListStudies

```
create unique index StationaryAttendanceListStudies_pk on
StationaryAttendanceListStudies (AttendanceID)
```

Tabela WebinarProduct

```
create unique index WebinarProduct_pk on WebinarProduct (WebinarProductID)
```

Tabela Webinars

```
create unique index Webinars_pk on Webinars (WebinarID)
```

Tabela WebinarMeeting

```
create unique index WebinarMeeting_pk on WebinarMeeting (WebinarMeetingID)
```

Uprawnienia

Administrator

```
create role Admin
grant all privileges on u_siklucki.dbo to admin
```

Użytkownik

```
create role Uzytkownik
grant select on CurrentOffer to Uzytkownik
grant execute on AddOrder to Uzytkownik
grant execute on AddProduct to Uzytkownik
grant select on WebinarInfo to Uzytkownik
grant select on CourseInfo to Uzytkownik
grant select on StudyInfo to Uzytkownik
grant select on GetBasket to Uzytkownik
grant select on HowMuchNeedUserPayForBasket to Uzytkownik
grant select on WebinarStructure to Uzytkownik
grant select on CourseStructure to Uzytkownik
grant select on StudiesStructure to Uzytkownik
```

Koordynator Studiów

```
create role StudiesCoordinator
grant execute on AddExam to StudiesCoordinator
grant execute on AddExamScore to StudiesCoordinator
grant execute on AddInternship to StudiesCoordinator
grant execute on AddStationaryModuleMeeting to StudiesCoordinator
grant execute on AddStudiesAttendance to StudiesCoordinator
grant execute on AddStudy to StudiesCoordinator
grant execute on AddSubject to StudiesCoordinator
grant execute on GetGrades to StudiesCoordinator
grant execute on GetMeetingVacancies to StudiesCoordinator
grant execute on GetStudyVacancies to StudiesCoordinator
grant select on GetSylabusForStudies to StudiesCoordinator
grant execute on ModifyLimit to StudiesCoordinator
grant execute on ifExistsMeetingStudies to StudiesCoordinator
grant execute on ifRegisteredStudies to StudiesCoordinator

grant select on AttendanceInternship to StudiesCoordinator
grant select on Exam to StudiesCoordinator
grant select on ExamScore to StudiesCoordinator
```

```
grant select on Form to StudiesCoordinator
grant select on Grade to StudiesCoordinator
grant select on Internships to StudiesCoordinator
grant select on Languages to StudiesCoordinator
grant select on OnLineAttendanceListStudies to StudiesCoordinator
grant select on OnLineSubjectMeeting to StudiesCoordinator
grant select on PlaceOfMeeting to StudiesCoordinator
grant select on StationaryAttendanceListStudies to
StudiesCoordinator
grant select on StationarySubjectMeeting to StudiesCoordinator
grant select on Studies to StudiesCoordinator
grant select on StudiesMeetingProduct to StudiesCoordinator
grant select on StudiesProduct to StudiesCoordinator
grant select on Subjects to StudiesCoordinator
grant select on Teachers to StudiesCoordinator
grant select on Translators to StudiesCoordinator


grant select on AttendanceList to StudiesCoordinator
grant select on AttendancePerSubject to StudiesCoordinator
grant select on AttendancePerSubjectTotal to StudiesCoordinator
grant select on ExamGrades to StudiesCoordinator
grant select on ExamsNotPassed to StudiesCoordinator
grant select on ExamsPassed to StudiesCoordinator
grant select on GradeCount to StudiesCoordinator
grant select on RemainingPlacesStudies to StudiesCoordinator
grant select on StudentCount to StudiesCoordinator
grant select on StudentInternships to StudiesCoordinator
grant select on Students to StudiesCoordinator
grant select on StudiesStructure to CourseCoordinator
```

Koordynator kursów

```
create role CourseCoordinator
grant execute on AddCourse to CourseCoordinator
grant execute on AddCourseAttendance to CourseCoordinator
grant execute on AddModule to CourseCoordinator
grant execute on AddOnlineAsyncModuleMeeting to CourseCoordinator
grant execute on AddOnlineSyncModuleMeeting to CourseCoordinator
grant execute on AddStationaryModuleMeeting to CourseCoordinator
grant execute on ifExistsMeetingCourse to CourseCoordinator
grant execute on ifExistsMeetingStudies to CourseCoordinator

grant select on CourseModule to CourseCoordinator
grant select on Courses to CourseCoordinator
grant select on CoursesProduct to CourseCoordinator
grant select on Form to CourseCoordinator
```

```
grant select on Languages to CourseCoordinator
grant select on OnlineAsyncAttendanceListModule to CourseCoordinator
grant select on OnlineAsyncModuleMeeting to CourseCoordinator
grant select on OnlineSyncAttendanceListModule to CourseCoordinator
grant select on OnlineSyncModuleMeeting to CourseCoordinator
grant select on PlaceOfMeeting to CourseCoordinator
grant select on StationaryAttendanceListModule to CourseCoordinator
grant select on StationaryModuleMeeting to CourseCoordinator
grant select on Translators to CourseCoordinator

grant select on CourseAttendanceTotal to CourseCoordinator
grant select on CourseStructure to CourseCoordinator
```

Koordynator Webinarów

```
create role WebinarCoordinator
grant execute on AddWebinar to WebinarCoordinator
grant execute on AddWebinarMeeting to WebinarCoordinator
grant select on WebinarInfo to WebinarCoordinator
grant execute on WebinarOrderCount to WebinarCoordinator

grant select on Languages to WebinarCoordinator
grant select on Translators to WebinarCoordinator
grant select on WebinarMeeting to WebinarCoordinator
grant select on WebinarProduct to WebinarCoordinator
grant select on Webinars to WebinarCoordinator

grant select on WebinarStructure to WebinarCoordinator
```

Sekretarz

```
create role Secretary
grant execute on AddException to Secretary
grant execute on AddOrder to Secretary
grant execute on AddProduct to Secretary
grant execute on AddStaff to Secretary
grant execute on AddTeacher to Secretary
grant execute on AddTranslator to Secretary
grant execute on AddUser to Secretary
grant select on GetBasket to Secretary

grant select on DebtorList to Secretary
grant select on EmploymentTime to Secretary
grant select on Financial_Reports to Secretary
grant select on OrdersAll to Secretary
grant select on Unpaid3DaysLeft to Secretary
```

```
grant select on AttendanceList to Secretary
grant select on AttendancePerSubject to Secretary
grant select on AttendancePerSubjectTotal to Secretary
grant select on ifPassedModule to Secretary

grant select on Cities to Secretary
grant select on Countries to Secretary
grant select on CoursesProduct to Secretary
grant select on Exception to Secretary
grant select on Languages to Secretary
grant select on OrderDetails to Secretary
grant select on Orders to Secretary
grant select on PlaceOfMeeting to Secretary
grant select on Products to Secretary
grant select on Staff to Secretary
grant select on StudiesMeetingProduct to Secretary
grant select on StudiesProduct to Secretary
grant select on Teachers to Secretary
grant select on Translators to Secretary
grant select on Users to Secretary
grant select on WebinarProduct to Secretary
```

Nauczyciel

```
create role Teacher
grant select on AttendanceInternship to Teacher
grant select on CourseModule to Teacher
grant select on Courses to Teacher
grant select on Exam to Teacher
grant select on Grade to Teacher
grant select on OnLineAttendanceListStudies to Teacher
grant select on OnLineSubjectMeeting to Teacher
grant select on OnlineAsyncAttendanceListModule to Teacher
grant select on OnlineAsyncModuleMeeting to Teacher
grant select on OnlineSyncAttendanceListModule to Teacher
grant select on OnlineSyncModuleMeeting to Teacher
grant select on StationaryAttendanceListModule to Teacher
grant select on StationaryAttendanceListStudies to Teacher
grant select on StationaryModuleMeeting to Teacher
grant select on StationarySubjectMeeting to Teacher
grant select on Studies to Teacher
grant select on Subjects to Teacher
grant select on Teachers to Teacher
grant select on Translators to Teacher
grant select on WebinarMeeting to Teacher
grant select on Webinars to Teacher
```

```
grant select on AttendanceList to Teacher
grant select on ExamGrades to Teacher
grant select on ExamsNotPassed to Teacher
grant select on ExamsPassed to Teacher
grant select on GradeCount to Teacher
grant select on MandatoryAttendance to Teacher
grant select on StudentCount to Teacher
grant select on Students to Teacher
grant select on TranslatorsView to Teacher

grant execute on AddCourseAttendance to Teacher
grant execute on AddExamScore to Teacher
grant execute on AddOnlineAsyncModuleMeeting to Teacher
grant execute on AddOnlineSyncModuleMeeting to Teacher
grant execute on AddStationaryModuleMeeting to Teacher
grant execute on AddStudiesAttendance to Teacher
grant execute on AddWebinarMeeting to Teacher
grant execute on GetGrades to Teacher
```

Dyrektor

```
create role HeadTeacher

grant execute on AddException to HeadTeacher
grant execute on AddUser to HeadTeacher

grant select on DebtorList to HeadTeacher
grant select on EmploymentTime to HeadTeacher
grant select on Financial_Reports to HeadTeacher
grant select on OrdersAll to HeadTeacher
grant select on Unpaid3DaysLeft to HeadTeacher
grant select on AttendanceList to HeadTeacher
grant select on AttendancePerSubject to HeadTeacher
grant select on AttendancePerSubjectTotal to HeadTeacher

grant select on Cities to HeadTeacher
grant select on Countries to HeadTeacher
grant select on CoursesProduct to HeadTeacher
grant select on Exception to HeadTeacher
grant select on Languages to HeadTeacher
grant select on OrderDetails to HeadTeacher
grant select on Orders to HeadTeacher
grant select on PlaceOfMeeting to HeadTeacher
grant select on Products to HeadTeacher
grant select on Staff to HeadTeacher
```

```
grant select on StudiesMeetingProduct to HeadTeacher
grant select on StudiesProduct to HeadTeacher
```