Projekt bazy danych

Podstawy Baz Danych 2023/24

Jakub Konopka, Norbert Dziwak, Mateusz Bobula

1. Użytkownicy:

- dyrektor szkoły
- administrator
- wykładowca
- tłumacz
- koordynator przedmiotów
- student
- niezarejestrowany użytkownik
- księgowa

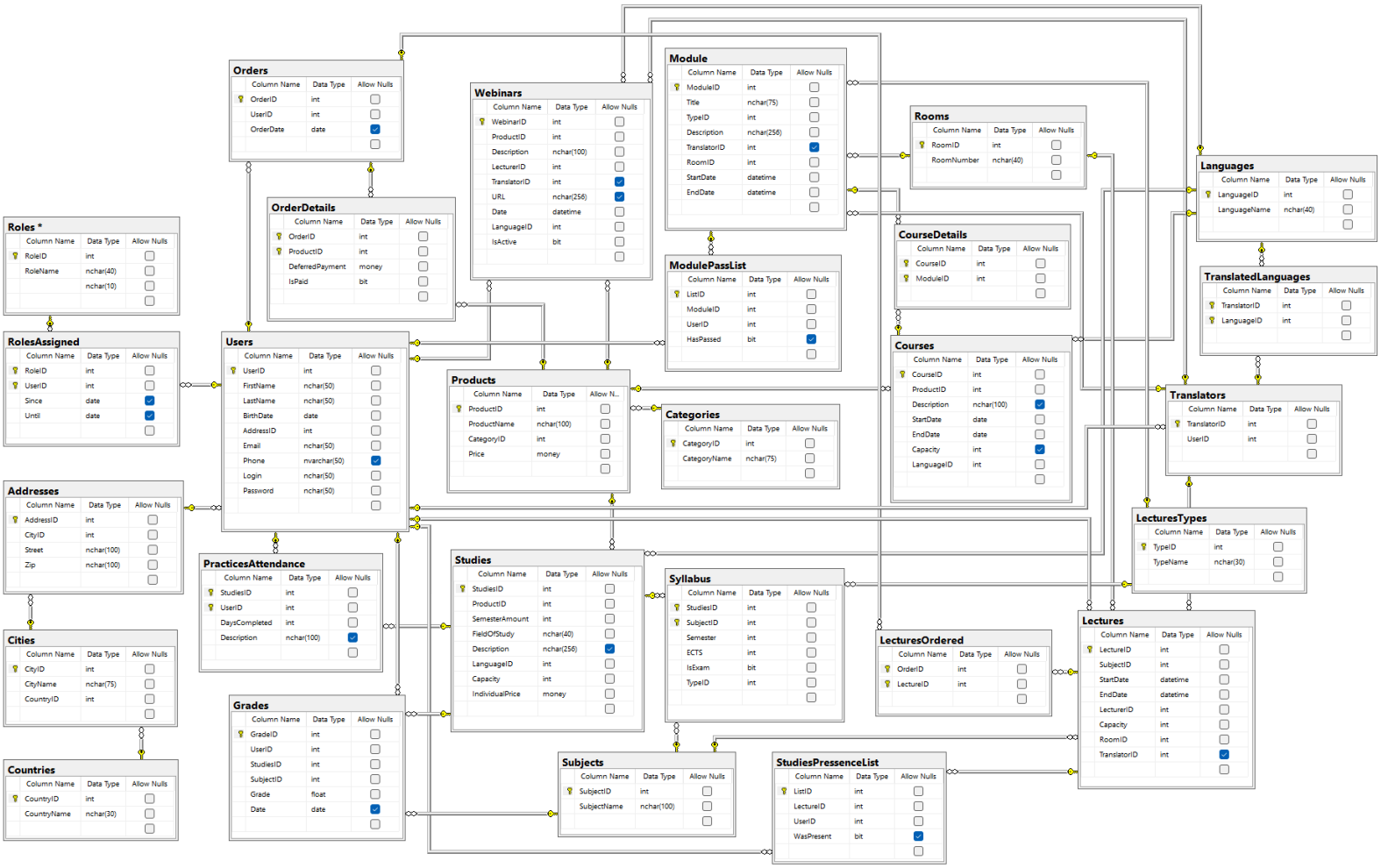
2. Funkcje

Funkcje	Uprawnieni użytkownicy
Modyfikowanie dostępności nagrań	- system
Dodawanie nagrań z webinarów	- wykładowca
Usuwanie nagrań z webinarów	- administrator
Wyświetlenie dostępnych ofert webinarów / kierunków studiów / sylabusa	- wszyscy
Odtworzenie nagrania darmowego webinaru	- każdy zalogowany użytkownik, będący uczestnikiem tego webinaru
Wykupienie dostępu do webinaru	- student (zarejestrowany użytkownik)
Odtworzenie nagrania płatnego webinaru	- zalogowany użytkownik, będący uczestnikiem tego webinaru (ma potwierdzenie zapłaty)
Założenie konta	- system
Logowanie/wylogowywanie	- wszyscy
przypomnienie hasła	- system
zmiana loginu/hasła	- system
Wykupienie kursu	- student

Dodawanie / usuwanie ofert kursów	- koordynator przedmiotów
Wyświetlanie informacji o kursie (obecności, % zaliczenia)	- student - wykładowca
Nie zaliczenie kursu uczestnikowi	- wykładowca
Rezygnacja z kursu	- student
Odrabianie zajęć	- student
Wyszukanie zajęć o podobnej tematyce	- koordynator przedmiotów
Zapis na studia/praktyki	- student
Rezygnacja ze studiów	- student
Aktualizacja sylabusa	- koordynator przedmiotów
Dodanie nowego kierunku studiów	- dyrektor szkoły - administrator
Tworzenie / usuwanie harmonogramu studiów	- koordynator przedmiotów
Modyfikowanie harmonogramu studiów	- koordynator przedmiotów
Wyświetlenie planu zajęć	- student - wykładowca - tłumacz
Przydzielanie tłumacza do wykładu	- koordynator przedmiotów
Nie zaliczenie przedmiotu/praktyk/egzaminu studentowi	- wykładowca
Zapis na pojedynczy etap studiów	- student
Koszyk (dodawanie / usuwanie / płatność)	- student
Generowanie linku do płatności oraz informacja zwrotna o statusie zapłaty	- system
Dodanie użytkownika do listy dłużników	- system
weryfikacja poprawności zapisu na studia/praktyki	- system
przypisywanie cech modułów	- koordynator przedmiotów
Kontrola statusu zaliczenia praktyk przez studenta	- wykładowca
Aktualizacja listy studentów	- administrator

Zarządzanie wykładowcami i ich przypisanie do kursów, szkoleń, studiów	- koordynator przedmiotów
Generowanie raportów finansowych	- księgowa
Zarządzanie limitami miejsc na kursy, szkolenia, studia	- koordynator przedmiotów
Wystawianie ocen za ukończenie kursów/studiów	- wykładowca

3. Schemat bazy danych



4. Opis tabel

Tabela Users

Zawiera informacje o użytkownikach

Klucz główny: UserID

Klucze obce: AddressID, RoleID

Warunki integralności:

poprawna data urodzenia:

```
CONSTRAINT
[CK_UserBirthDate] CHECK  ((datepart(year,[birthdate])>(1900)
AND [birthdate]<getdate()))
```

poprawny email:

```
CONSTRAINT [CK_Email] CHECK  (([Email] like '%@%'))
```

poprawny numer telefonu:

```
CONSTRAINT [CK_Phone] 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]'))
```

unikatowy login:

```
CONSTRAINT [UQ_Login] UNIQUE NONCLUSTERED
```

unikatowy email:

```
CONSTRAINT [UQ_Email] UNIQUE NONCLUSTERED
```

ID użytkownika	UserID
ID roli użytkownika	RoleID
Imię użytkownika	FirstName
Nazwisko użytkownika	LastName
Data urodzenia się użytkownika	BirthDate
ID adresu użytkownika	AddressID
Email	Email

ID użytkownika	UserID
ID roli użytkownika	RoleID
Imię użytkownika	FirstName
Telefon do użytkownika	Phone
Login użytkownika używany przy logowaniu	Login
Hasło użytkownika używane przy logowaniu	Password

```
CREATE TABLE [dbo].[Users](
  [UserID] [int] IDENTITY(1,1) NOT NULL,
  [FirstName] [nchar](50) NOT NULL,
  [LastName] [nchar](50) NOT NULL,
  [BirthDate] [date] NOT NULL,
  [AddressID] [int] NOT NULL,
  [Email] [nchar](50) NOT NULL,
  [Phone] [nvarchar](50) NULL,
  [Login] [nchar](50) NOT NULL,
  [Password] [nchar](50) NOT NULL,
 CONSTRAINT [PK_Users] PRIMARY KEY CLUSTERED
(
  [UserID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UQ_Email] UNIQUE NONCLUSTERED
(
  [Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UQ_Login] UNIQUE NONCLUSTERED
(
  [Login] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT
[FK_Users_Addresses] FOREIGN KEY([AddressID])
REFERENCES [dbo].[Addresses] ([AddressID])
GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [FK_Users_Addresses]
GO

ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT [CK_Email] CHECK
(([Email] like '%@%'))
GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CK_Email]
GO

ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT [CK_Phone] CHECK
(([Phone] like
N'[+][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CK_Phone]
GO

ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT
[CK_UserBirthDate] CHECK  ((datepart(year,[birthdate])>(1900) AND
[birthdate]<getdate()))
GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [CK_UserBirthDate]
GO
```

Tabela Roles

Słownik ról

Klucz główny: RoleID

ID roli	RoleID
Nazwa roli	RoleName

```
CREATE TABLE [dbo].[Roles](
    [RoleID] [int] IDENTITY(1,1) NOT NULL,
    [RoleName] [nchar](40) NOT NULL,
 CONSTRAINT [PK_Roles] PRIMARY KEY CLUSTERED
(
    [RoleID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela RolesAssigned

Który użytkownik ma jaką rolę

Klucz główny: RoleID, UserID

Klucz obcy: RoleID, UserID

Warunki integralności:

poprawne daty Since oraz Until:

```
CONSTRAINT [CK_AssignedDate] CHECK  (([since]<=[until] AND
datepart(year,[since])>(1950) AND
datepart(year,[since])<=datepart(year,getdate()) AND
datepart(year,[until])<=datepart(year,getdate())))
```

ID roli	RoleID
ID użytkonika	UserID
Data nadania uprawnień **Default GETDATE()**	Since
Data kiedy użytkownik straci uprawnienia	Until

```
CREATE TABLE [dbo].[RolesAssigned](
    [RoleID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [Since] [date] NULL,
    [Until] [date] NULL,
 CONSTRAINT [PK_RolesAssigned] PRIMARY KEY CLUSTERED
(
    [RoleID] ASC,
    [UserID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[RolesAssigned] ADD  CONSTRAINT
[DF_RolesAssigned_Since]  DEFAULT (getdate()) FOR [Since]
GO
```

```
ALTER TABLE [dbo].[RolesAssigned]  WITH CHECK ADD  CONSTRAINT
[FK_RolesAssigned_Roles] FOREIGN KEY([RoleID])
REFERENCES [dbo].[Roles] ([RoleID])
GO

ALTER TABLE [dbo].[RolesAssigned] CHECK CONSTRAINT
[FK_RolesAssigned_Roles]
GO

ALTER TABLE [dbo].[RolesAssigned]  WITH CHECK ADD  CONSTRAINT
[FK_RolesAssigned_Users] FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[RolesAssigned] CHECK CONSTRAINT
[FK_RolesAssigned_Users]
GO

ALTER TABLE [dbo].[RolesAssigned]  WITH CHECK ADD  CONSTRAINT
[CK_AssignedDate] CHECK  (([since]<[until] AND
datepart(year,[since])>(1950) AND
datepart(year,[since])<=datepart(year,getdate()) AND
datepart(year,[until])<=datepart(year,getdate())))
GO

ALTER TABLE [dbo].[RolesAssigned] CHECK CONSTRAINT [CK_AssignedDate]
GO
```

Tabela Addresses

Zawiera informacje o adresach

Klucz główny: AddressID

Klucze obce: CityID

ID adresu	AddressID
ID miasta danego adresu	CityID
Ulica danego adresu	Street
Kod pocztowy danego adresu	Zip

```
CREATE TABLE [dbo].[Addresses](
    [AddressID] [int] IDENTITY(1,1) NOT NULL,
    [CityID] [int] NOT NULL,
    [Street] [nchar](100) NOT NULL,
    [Zip] [nchar](100) NOT NULL,
 CONSTRAINT [PK_Addresses] PRIMARY KEY CLUSTERED
 (
    [AddressID] ASC
 )WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
 OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
 OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
 ) ON [PRIMARY]
 GO

 ALTER TABLE [dbo].[Addresses]  WITH CHECK ADD  CONSTRAINT
 [FK_Addresses_Cities] FOREIGN KEY([CityID])
 REFERENCES [dbo].[Cities] ([CityID])
 GO

 ALTER TABLE [dbo].[Addresses] CHECK CONSTRAINT [FK_Addresses_Cities]
 GO
```

Tabela Cities

Słownik miast

Klucz główny: CityID

Klucze obce: CountryID

Warunki integralności:

poprawna nazwa miasta:

```
CONSTRAINT [CK_CityName]
 CHECK  ((NOT [CityName] like '%[^a-zA-Z ]%'))
```

ID miasta	CityID
Nazwa miasta	CityName
ID państwa, w którym to miasto się znajduje	CountryID

```
CREATE TABLE [dbo].[Cities](
   [CityID] [int] IDENTITY(1,1) NOT NULL,
   [CityName] [nchar](75) NOT NULL,
   [CountryID] [int] NOT NULL,
CONSTRAINT [PK_Cities] PRIMARY KEY CLUSTERED
(
   [CityID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT
[FK_Cities_Countries] FOREIGN KEY([CountryID])
REFERENCES [dbo].[Countries] ([CountryID])
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [FK_Cities_Countries]
GO

ALTER TABLE [dbo].[Cities]  WITH CHECK ADD  CONSTRAINT [CK_CityName]
CHECK  ((NOT [CityName] like '%[^a-zA-Z ]%'))
```

```
GO

ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [CK_CityName]
GO
```

Tabela Countries

Słownik państw

Klucz główny: CountryID

Warunki integralności:

poprawna nazwa państwa:

```
CONSTRAINT
 [CK_CountryName] CHECK  ((NOT [CountryName] like '%[^a-zA-Z
]%'))
```

ID państwa	CountryID
Nazwa państwa	CountryName

```
CREATE TABLE [dbo].[Countries](
   [CountryID] [int] IDENTITY(1,1) NOT NULL,
   [CountryName] [nchar](30) NOT NULL,
CONSTRAINT [PK_Countries] PRIMARY KEY CLUSTERED
(
   [CountryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Countries]  WITH CHECK ADD  CONSTRAINT
[CK_CountryName] CHECK  ((NOT [CountryName] like '%[^a-zA-Z
]%'))
GO

ALTER TABLE [dbo].[Countries] CHECK CONSTRAINT [CK_CountryName]
GO
```

Tabela Orders

Zawiera złożone zamówienia

Klucz główny: OrderID

Klucze obce: UserID

Warunki integralności:

data złożenia zamówienia wcześniejsza niż aktualna data i rok nie mniejszy niż 2022:

```
CONSTRAINT [CK_OrderDate] CHECK  (([orderDate]<getdate() AND
datepart(year,[orderdate])>=(2022)))
```

ID zamówienia	OrderID
ID użytkownika, który złożył to zamówienie	UserID
Data złożenia zamówienia **Default: getdate()**	OrderDate

```
CREATE TABLE [dbo].[Orders](
    [OrderID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NOT NULL,
    [OrderDate] [date] NULL,
 CONSTRAINT [PK_Orders] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Orders] ADD  CONSTRAINT [DF_Orders_OrderDate]
DEFAULT (getdate()) FOR [OrderDate]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT
[FK_Orders_Users] FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO
```

```
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Users]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_OrderDate]
CHECK  (([orderDate]<getdate() AND datepart(year,[orderdate])>=(2022)))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_OrderDate]
GO
```

Tabela OrderDetails

Zawiera szczegóły zamówień

Klucz główny: OrderID, ProductID

Klucze obce: OrderID, ProductID

Warunki integralności:

odroczona zapłata >= 0:

```
CONSTRAINT [CK_DeferredPayment] CHECK  (([deferredpayment]>=(0)))
```

ID zamówienia	OrderID
ID produktu	ProductID
Odroczona zapłata **Default 0**	DeferredPayment
Czy już opłacone **Default 0**	IsPaid

```
CREATE TABLE [dbo].[OrderDetails](
  [OrderID] [int] NOT NULL,
  [ProductID] [int] NOT NULL,
  [DeferredPayment] [money] NOT NULL,
  [IsPaid] [bit] NOT NULL,
 CONSTRAINT [PK_OrderDetails] PRIMARY KEY CLUSTERED
(
  [OrderID] ASC,
  [ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderDetails] ADD  CONSTRAINT
[DF_OrderDetails_OrderID]  DEFAULT ((0)) FOR [OrderID]
GO
```

```
ALTER TABLE [dbo].[OrderDetails] ADD  CONSTRAINT
[DF_OrderDetails_DeferredPayment]  DEFAULT ((0)) FOR
[DeferredPayment]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[FK_OrderDetails_Orders]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderDetails_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[FK_OrderDetails_Products]
GO

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT
[CK_DeferredPayment] CHECK  (([deferredpayment]>=(0)))
GO

ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[CK_DeferredPayment]
GO
```

Tabela Products

Zawiera dostępne produkty (webinary/kursy/studia)

Klucz główny: ProductID

Klucze obce: CategoryID

Warunki integralności:

cena >= 0:

```
CONSTRAINT [CK_ProductPrice] CHECK  (([price]>=(0)))
```

ID produktu	ProductID
Nazwa produktu	ProductName
Kategoria produktu	CategoryID
Cena produktu	Price

```
CREATE TABLE [dbo].[Products](
   [ProductID] [int] IDENTITY(1,1) NOT NULL,
   [ProductName] [nchar](100) NOT NULL,
   [CategoryID] [int] NOT NULL,
   [Price] [money] NOT NULL,
CONSTRAINT [PK_Products] PRIMARY KEY CLUSTERED
(
   [ProductID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT
[FK_Products_Categories] FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Categories] ([CategoryID])
GO

ALTER TABLE [dbo].[Products] CHECK CONSTRAINT
[FK_Products_Categories]
GO
```

```
ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT
[CK_ProductPrice] CHECK  (([price]>=(0)))
GO

ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [CK_ProductPrice]
GO
```

Tabela Categories

Słownik kategorii (webinar/studia/kurs)

Klucz główny: CategoryID

ID kategorii	CategoryID
Nazwa kategorii	CategoryName

```
CREATE TABLE [dbo].[Categories](
   [CategoryID] [int] IDENTITY(1,1) NOT NULL,
   [CategoryName] [nchar](75) NOT NULL,
CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
   [CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela Webinars

Zawiera informacje o webinarach

Klucz główny: WebinarID

Klucze obce: ProductID, LanguageID, TranslatorID

Warunki integralności:

dobra data:

```
CONSTRAINT [CK_WebinarDate] CHECK
((datepart(year,[date])>=(2022) AND
datepart(year,[date])<=(datepart(year,getdate())+(1))))
```

poprawny URL:

```
CONSTRAINT [CK_URL] CHECK  (([url] like 'https://%'))
```

ID webinaru	WebinarID
ID produktu	ProductID
Tematyka webinaru	Description
ID wykładowcy	LecturerID
ID tłumacza	TranslatorID
Link do nagrania z webinaru	URL
Data odbycia się webinaru	Date
ID języka w jakim prowadzony jest webinar	LanguageID
Czy nagranie jest jeszcze aktywne	IsActive

```
CREATE TABLE [dbo].[Webinars](
    [WebinarID] [int] IDENTITY(1,1) NOT NULL,
    [ProductID] [int] NOT NULL,
    [Description] [nchar](100) NOT NULL,
    [LecturerID] [int] NOT NULL,
    [TranslatorID] [int] NULL,
    [URL] [nchar](256) NULL,
    [Date] [datetime] NOT NULL,
    [LanguageID] [int] NOT NULL,
    [IsActive] [bit] NOT NULL,
```

```
 CONSTRAINT [PK_Webinars] PRIMARY KEY CLUSTERED
(
    [WebinarID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT
[FK_Webinars_Languages] FOREIGN KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
GO

ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [FK_Webinars_Languages]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT
[FK_Webinars_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [FK_Webinars_Products]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT
[FK_Webinars_Translators] FOREIGN KEY([TranslatorID])
REFERENCES [dbo].[Translators] ([TranslatorID])
GO

ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [FK_Webinars_Translators]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT
[FK_Webinars_Users] FOREIGN KEY([LecturerID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [FK_Webinars_Users]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT [CK_URL] CHECK
(([url] like 'https://%'))
```

```
GO

ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [CK_URL]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT
[CK_WebinarDate] CHECK  ((datepart(year,[date])>=(2022) AND
datepart(year,[date])<=(datepart(year,getdate())+(1))))
GO

ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [CK_WebinarDate]
GO
```

Tabela Courses

Zawiera informacje o kursach

Klucz główny: CourseID

Klucze obce: ProductID, LanguageID

Warunki integralności:

liczba początkowych miejsc > 0:

```
CONSTRAINT [CK_Capacity] CHECK  (([Capacity]>(0)))
```

poprawna data:

```
CONSTRAINT [CK_Date] CHECK  (([enddate]>[startdate] AND
datepart(year,[startdate])>=(2022) AND
datepart(year,[enddate])<=(datepart(year,getdate())+(1))))
```

ID kursu	CourseID
ID produktu	ProductID
Tematyka kursu	Description
Data startu kursu	StartDate
Data końca kursu	EndDate
Liczba osób mogących zapisać się na dany kurs	Capacity
ID języka w jakim prowadzony jest kurs Default 1 - polski	LanguageID

```
CREATE TABLE [dbo].[Courses](
    [CourseID] [int] IDENTITY(1,1) NOT NULL,
    [ProductID] [int] NOT NULL,
    [Description] [nchar](100) NULL,
    [StartDate] [date] NOT NULL,
    [EndDate] [date] NOT NULL,
    [Capacity] [int] NULL,
    [LanguageID] [int] NOT NULL,
 CONSTRAINT [PK_Courses] PRIMARY KEY CLUSTERED
(
```

```
    [CourseID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Courses] ADD  CONSTRAINT
[DF_Courses_LanguageID]  DEFAULT ((1)) FOR [LanguageID]
GO

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT
[FK_Courses_Languages] FOREIGN KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
GO

ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT
[FK_Courses_Languages]
GO

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT
[FK_Courses_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT
[FK_Courses_Products]
GO

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT
[CK_Capacity] CHECK  (([Capacity]>(0)))
GO

ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [CK_Capacity]
GO

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CK_Date]
CHECK  (([enddate]>[startdate] AND
datepart(year,[startdate])>=(2022) AND
datepart(year,[enddate])<=(datepart(year,getdate())+(1))))
GO

ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [CK_Date]
```

```
GO
```

Tabela CourseDetails

Zawiera informacje o szczegółach danego kursu

Klucz główny: CourseID, ModuleID

Klucze obce: CourseID, ModuleID

ID kursu	CourseID
ID modułu podpiętego do danego kursu	ModuleID

```
CREATE TABLE [dbo].[CourseDetails](
    [CourseID] [int] NOT NULL,
    [ModuleID] [int] NOT NULL,
 CONSTRAINT [PK_CourseDetails] PRIMARY KEY CLUSTERED
(
    [CourseID] ASC,
    [ModuleID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CourseDetails]  WITH CHECK ADD  CONSTRAINT
[FK_CourseDetails_Courses] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Courses] ([CourseID])
GO

ALTER TABLE [dbo].[CourseDetails] CHECK CONSTRAINT
[FK_CourseDetails_Courses]
GO

ALTER TABLE [dbo].[CourseDetails]  WITH CHECK ADD  CONSTRAINT
[FK_CourseDetails_Module] FOREIGN KEY([ModuleID])
REFERENCES [dbo].[Module] ([ModuleID])
GO

ALTER TABLE [dbo].[CourseDetails] CHECK CONSTRAINT
[FK_CourseDetails_Module]
```

```
GO
```

Tabela Module

Zawiera informacje o module

Klucz główny: ModuleID

Klucze obce: TypeID, TranslatorID, RoomID

Warunki integralności:

poprawna data:

```
CONSTRAINT [CK_StartModuleDate] CHECK  (([EndDate]>[StartDate]
AND datepart(year,[StartDate])>=(2022) AND
datepart(year,[EndDate])<=(datepart(year,getdate())+(1))))
```

ID modułu	ModuleID
Nazwa modułu	Title
ID typu kursu	TypeID
Opis modułu	Description
ID tłumacza	TranslatorID
ID pokoju	RoomID
Data rozpoczęcia modułu	StartDate
Data zakończenia modułu	EndDate

```
CREATE TABLE [dbo].[Module](
  [ModuleID] [int] IDENTITY(1,1) NOT NULL,
  [Title] [nchar](75) NOT NULL,
  [TypeID] [int] NOT NULL,
  [Description] [nchar](256) NOT NULL,
  [TranslatorID] [int] NULL,
  [RoomID] [int] NOT NULL,
  [StartDate] [date] NOT NULL,
  [EndDate] [date] NOT NULL,
 CONSTRAINT [PK_Module] PRIMARY KEY CLUSTERED
(
  [ModuleID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
```

```
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Module]  WITH CHECK ADD  CONSTRAINT
[FK_Module_LecturesTypes] FOREIGN KEY([TypeID])
REFERENCES [dbo].[LecturesTypes] ([TypeID])
GO

ALTER TABLE [dbo].[Module] CHECK CONSTRAINT [FK_Module_LecturesTypes]
GO

ALTER TABLE [dbo].[Module]  WITH CHECK ADD  CONSTRAINT
[FK_Module_Rooms] FOREIGN KEY([RoomID])
REFERENCES [dbo].[Rooms] ([RoomID])
GO

ALTER TABLE [dbo].[Module] CHECK CONSTRAINT [FK_Module_Rooms]
GO

ALTER TABLE [dbo].[Module]  WITH CHECK ADD  CONSTRAINT
[FK_Module_Translators] FOREIGN KEY([TranslatorID])
REFERENCES [dbo].[Translators] ([TranslatorID])
GO

ALTER TABLE [dbo].[Module] CHECK CONSTRAINT [FK_Module_Translators]
GO

ALTER TABLE [dbo].[Module]  WITH CHECK ADD  CONSTRAINT
[CK_StartModuleDate] CHECK  (([EndDate]>[StartDate] AND
datepart(year,[StartDate])>=(2022) AND
datepart(year,[EndDate])<=(datepart(year,getdate())+(1))))
GO

ALTER TABLE [dbo].[Module] CHECK CONSTRAINT [CK_StartModuleDate]
GO
```

Tabela Studies

Zawiera informacje o studiach

Klucz główny: StudiesID

Klucze obce: ProductID, LanguageID

Warunki integralności:

poprawna liczba semestrów:

```
CONSTRAINT [CK_SemesterAmount] CHECK  (([SemesterAmount]>(0) AND
[SemesterAmount]<=(16)))
```

początkowa liczba miejsc > 0:

```
CONSTRAINT [CK_StudiesCapacity] CHECK  (([capacity]>(0)))
```

poprawna cena:

```
CONSTRAINT [CK_StudiesIndividualPrice] CHECK
(([individualprice]>=(0)))
```

ID studiów	StudiesID
ID produktu	ProductID
Liczba semestrów	SemesterAmount
Kierunek studiów	FieldOfStudy
Opis studiów	Description
ID języka w jakim prowadzone są studia	LanguageID
Liczba osób mogących zapisać się na dane studia	Capacity
Cena pojedynczego spotkania	IndividualPrice

```
CREATE TABLE [dbo].[Studies](
    [StudiesID] [int] IDENTITY(1,1) NOT NULL,
    [ProductID] [int] NOT NULL,
    [SemesterAmount] [int] NOT NULL,
    [FieldOfStudy] [nchar](40) NOT NULL,
    [Description] [nchar](256) NULL,
    [LanguageID] [int] NOT NULL,
    [Capacity] [int] NOT NULL,
```

```
    [IndividualPrice] [money] NOT NULL,
 CONSTRAINT [PK_Studies] PRIMARY KEY CLUSTERED
(
    [StudiesID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Studies] ADD  CONSTRAINT
[DF_Studies_LanguageID]  DEFAULT ((1)) FOR [LanguageID]
GO

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT
[FK_Studies_Languages] FOREIGN KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
GO

ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT
[FK_Studies_Languages]
GO

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT
[FK_Studies_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT
[FK_Studies_Products]
GO

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT
[CK_SemesterAmount] CHECK  (([SemesterAmount]>(0) AND
[SemesterAmount]<=(16)))
GO

ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT [CK_SemesterAmount]
GO

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT
[CK_StudiesCapacity] CHECK  (([capacity]>(0)))
GO
```

```
ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT [CK_StudiesCapacity]
GO

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT
[CK_StudiesIndividualPrice] CHECK  (([individualprice]>=(0)))
GO

ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT
[CK_StudiesIndividualPrice]
GO
```

Tabela PracticesAttendance

Lista odbytych dni praktyk przez studentów

Klucz główny: StudiesID, UserID

Klucze obce: StudiesID, UserID

Warunki integralności:

poprawna liczba dni praktyk:

```
CONSTRAINT [CK_PracticeDaysCompleted] CHECK
(([dayscompleted]>=(0) AND [dayscompleted]<=(28)))
```

ID studiów	StudiesID
ID studenta	UserID
Ukończone dni przez studenta **Default 0**	DaysCompleted
Opis praktyk, na które uczęszcza student	Description

```
CREATE TABLE [dbo].[PracticesAttendance](
    [StudiesID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [DaysCompleted] [int] NOT NULL,
    [Description] [nchar](100) NOT NULL,
 CONSTRAINT [PK_PracticesAttendance] PRIMARY KEY CLUSTERED
(
    [StudiesID] ASC,
    [UserID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[PracticesAttendance] ADD  CONSTRAINT
[DF_PracticesAttendance_StudiesID]  DEFAULT ((0)) FOR [StudiesID]
GO
```

```
ALTER TABLE [dbo].[PracticesAttendance]  WITH CHECK ADD
CONSTRAINT [FK_PracticesAttendance_Studies] FOREIGN
KEY([StudiesID])
REFERENCES [dbo].[Studies] ([StudiesID])
GO

ALTER TABLE [dbo].[PracticesAttendance] CHECK CONSTRAINT
[FK_PracticesAttendance_Studies]
GO

ALTER TABLE [dbo].[PracticesAttendance]  WITH CHECK ADD
CONSTRAINT [FK_PracticesAttendance_Users] FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[PracticesAttendance] CHECK CONSTRAINT
[FK_PracticesAttendance_Users]
GO

ALTER TABLE [dbo].[PracticesAttendance]  WITH CHECK ADD
CONSTRAINT [CK_PracticeDaysCompleted] CHECK
(([dayscompleted]>=(0) AND [dayscompleted]<=(14)))
GO

ALTER TABLE [dbo].[PracticesAttendance] CHECK CONSTRAINT
[CK_PracticeDaysCompleted]
GO
```

Tabela Syllabus

Zawiera informacje o danym kierunku studiów

Klucz główny: StudiesID, SubjectID

Klucze obce: StudiesID, SubjectID, TypeID

Warunki integralności:

poprawny numer semestru:

```
CONSTRAINT [CK_Semester] CHECK  (([Semester]>=(0) AND
[Semester]<=(16)))
```

liczba ECTS > 0:

```
CONSTRAINT [CK_ECTS] CHECK  (([ECTS]>(0)))
```

ID studiów	StudiesID
ID przedmiotu	SubjectID
Semestr, na którym przedmiot występuje	Semester
Liczba ECTS za dany przedmiot	ECTS
Określa czy na koniec semestru jest egzamin	IsExam
ID typu spotkań	TypeID

```
CREATE TABLE [dbo].[Syllabus](
    [StudiesID] [int] NOT NULL,
    [SubjectID] [int] NOT NULL,
    [Semester] [int] NOT NULL,
    [ECTS] [int] NOT NULL,
    [IsExam] [bit] NOT NULL,
    [TypeID] [int] NOT NULL,
 CONSTRAINT [PK_Syllabus] PRIMARY KEY CLUSTERED
(
    [StudiesID] ASC,
    [SubjectID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
```

```
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Syllabus] ADD  CONSTRAINT [DF_Syllabus_IsExam]
DEFAULT ((0)) FOR [IsExam]
GO

ALTER TABLE [dbo].[Syllabus]  WITH CHECK ADD  CONSTRAINT
[FK_Syllabus_LecturesTypes] FOREIGN KEY([TypeID])
REFERENCES [dbo].[LecturesTypes] ([TypeID])
GO

ALTER TABLE [dbo].[Syllabus] CHECK CONSTRAINT
[FK_Syllabus_LecturesTypes]
GO

ALTER TABLE [dbo].[Syllabus]  WITH CHECK ADD  CONSTRAINT
[FK_Syllabus_Studies] FOREIGN KEY([StudiesID])
REFERENCES [dbo].[Studies] ([StudiesID])
GO

ALTER TABLE [dbo].[Syllabus] CHECK CONSTRAINT [FK_Syllabus_Studies]
GO

ALTER TABLE [dbo].[Syllabus]  WITH CHECK ADD  CONSTRAINT
[FK_Syllabus_Subjects] FOREIGN KEY([Semester])
REFERENCES [dbo].[Subjects] ([SubjectID])
GO

ALTER TABLE [dbo].[Syllabus] CHECK CONSTRAINT [FK_Syllabus_Subjects]
GO

ALTER TABLE [dbo].[Syllabus]  WITH CHECK ADD  CONSTRAINT [CK_ECTS]
CHECK  (([ECTS]>(0)))
GO

ALTER TABLE [dbo].[Syllabus] CHECK CONSTRAINT [CK_ECTS]
GO

ALTER TABLE [dbo].[Syllabus]  WITH CHECK ADD  CONSTRAINT [CK_Semester]
CHECK  (([Semester]>=(0) AND [Semester]<=(16)))
GO
```

```
ALTER TABLE [dbo].[Syllabus] CHECK CONSTRAINT [CK_Semester]
GO
```

Tabela Subjects

Zawiera informacje o przedmiotach

Klucz główny: SubjectID

ID przedmiotu	SubjectID
Nazwa przedmiotu	SubjectName

```
CREATE TABLE [dbo].[Subjects](
  [SubjectID] [int] IDENTITY(1,1) NOT NULL,
  [SubjectName] [nchar](100) NOT NULL,
 CONSTRAINT [PK_Subjects] PRIMARY KEY CLUSTERED
(
  [SubjectID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela Lectures

Zawiera informacje o wykładach/ćwiczeniach

Klucz główny: LectureID

Klucze obce: UserID, RoomID, TranslatorID, SubjectID

Warunki integralności:

Poprawna data:

```
CONSTRAINT [CK_LectureDates] CHECK  (([enddate]>[startdate] AND
datepart(year,[startdate])>=(2022) AND
datepart(year,[enddate])<=(datepart(year,getdate())+(1))))
```

początkowa liczba miejsc > 0:

```
CONSTRAINT [CK_CapacityLectures] CHECK  (([capacity]>(0)))
```

ID wykładu	LectureID
ID przedmiotu	SubjectID
Data startu wykładu	StartDate
Data końca wykładu	EndDate
ID wykładowcy	LecturerID
Liczba osób mogących uczęszczać na dany wykład	Capacity
ID pokoju	RoomID
ID tłumacza	TranslatorID

```
CREATE TABLE [dbo].[Lectures](
    [LectureID] [int] IDENTITY(1,1) NOT NULL,
    [SubjectID] [int] NOT NULL,
    [StartDate] [datetime] NOT NULL,
    [EndDate] [datetime] NOT NULL,
    [LecturerID] [int] NOT NULL,
    [Capacity] [int] NOT NULL,
    [RoomID] [int] NOT NULL,
    [TranslatorID] [int] NULL,
 CONSTRAINT [PK_Lectures] PRIMARY KEY CLUSTERED
(
```

```
    [LectureID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Lectures]  WITH CHECK ADD  CONSTRAINT
[FK_Lectures_Rooms] FOREIGN KEY([RoomID])
REFERENCES [dbo].[Rooms] ([RoomID])
GO

ALTER TABLE [dbo].[Lectures] CHECK CONSTRAINT [FK_Lectures_Rooms]
GO

ALTER TABLE [dbo].[Lectures]  WITH CHECK ADD  CONSTRAINT
[FK_Lectures_Subjects] FOREIGN KEY([SubjectID])
REFERENCES [dbo].[Subjects] ([SubjectID])
GO

ALTER TABLE [dbo].[Lectures] CHECK CONSTRAINT [FK_Lectures_Subjects]
GO

ALTER TABLE [dbo].[Lectures]  WITH CHECK ADD  CONSTRAINT
[FK_Lectures_Translators] FOREIGN KEY([TranslatorID])
REFERENCES [dbo].[Translators] ([TranslatorID])
GO

ALTER TABLE [dbo].[Lectures] CHECK CONSTRAINT [FK_Lectures_Translators]
GO

ALTER TABLE [dbo].[Lectures]  WITH CHECK ADD  CONSTRAINT
[FK_Lectures_Users] FOREIGN KEY([LecturerID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[Lectures] CHECK CONSTRAINT [FK_Lectures_Users]
GO

ALTER TABLE [dbo].[Lectures]  WITH CHECK ADD  CONSTRAINT
[CK_CapacityLectures] CHECK  (([capacity]>(0)))
GO
```

```
ALTER TABLE [dbo].[Lectures] CHECK CONSTRAINT [CK_CapacityLectures]
GO

ALTER TABLE [dbo].[Lectures]  WITH CHECK ADD  CONSTRAINT
[CK_LectureDates] CHECK  (([enddate]>[startdate] AND
datepart(year,[startdate])>=(2022) AND
datepart(year,[enddate])<=(datepart(year,getdate())+(1))))
GO

ALTER TABLE [dbo].[Lectures] CHECK CONSTRAINT [CK_LectureDates]
GO
```

Tabela LecturesOrdered

Zawiera informacje o zakupionych pojedynczych spotkaniach

Klucz główny: OrderID, LectureID

Klucze obce: OrderID, LectureID

ID zamówienia	OrderID
ID wykładu	LectureID

```
CREATE TABLE [dbo].[LecturesOrdered](
    [OrderID] [int] IDENTITY(1,1) NOT NULL,
    [LectureID] [int] NOT NULL,
 CONSTRAINT [PK_LecturesOrdered] PRIMARY KEY CLUSTERED
(
    [OrderID] ASC,
    [LectureID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[LecturesOrdered]  WITH CHECK ADD  CONSTRAINT
[FK_LecturesOrdered_Lectures] FOREIGN KEY([LectureID])
REFERENCES [dbo].[Lectures] ([LectureID])
GO

ALTER TABLE [dbo].[LecturesOrdered] CHECK CONSTRAINT
[FK_LecturesOrdered_Lectures]
GO

ALTER TABLE [dbo].[LecturesOrdered]  WITH CHECK ADD  CONSTRAINT
[FK_LecturesOrdered_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO
```

```
ALTER TABLE [dbo].[LecturesOrdered] CHECK CONSTRAINT
[FK_LecturesOrdered_Orders]
GOq
```

Tabela StudiesPressenceList

Lista obecności na studiach

Klucz główny: ListID

Klucze obce: LectureID, UserID

ID obecności na liście	ListID
ID wykładu	LectureID
ID użytkownika	UserID
Czy użytkownik był obecny (0 lub 1) Defalut 1	WasPresent

```
CREATE TABLE [dbo].[StudiesPressenceList](
    [ListID] [int] IDENTITY(1,1) NOT NULL,
    [LectureID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [WasPresent] [bit] NULL,
 CONSTRAINT [PK_StudiesPressenceList] PRIMARY KEY CLUSTERED
(
    [ListID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[StudiesPressenceList] ADD  CONSTRAINT
[DF_StudiesPressenceList_WasPresent]  DEFAULT ((1)) FOR [WasPresent]
GO

ALTER TABLE [dbo].[StudiesPressenceList]  WITH CHECK ADD  CONSTRAINT
[FK_StudiesPressenceList_Lectures1] FOREIGN KEY([LectureID])
REFERENCES [dbo].[Lectures] ([LectureID])
GO
```

```
ALTER TABLE [dbo].[StudiesPressenceList] CHECK CONSTRAINT
[FK_StudiesPressenceList_Lectures1]
GO

ALTER TABLE [dbo].[StudiesPressenceList]  WITH CHECK ADD  CONSTRAINT
[FK_StudiesPressenceList_Users] FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[StudiesPressenceList] CHECK CONSTRAINT
[FK_StudiesPressenceList_Users]
GO
```

Tabela Translators

Słownik tłumaczy

Klucz główny: TranslatorID

Klucze obce: UserID

ID tłumacza	TranslatorID
ID użytkownika	UserID

```
CREATE TABLE [dbo].[Translators](
    [TranslatorID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NOT NULL,
 CONSTRAINT [PK_Translators] PRIMARY KEY CLUSTERED
(
    [TranslatorID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Translators]  WITH CHECK ADD  CONSTRAINT
[FK_Translators_Users1] FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[Translators] CHECK CONSTRAINT
[FK_Translators_Users1]
GO
```

Tabela Languages

Słownik języków

Klucz główny: LanguageID

Warunki integralności:

poprawna nazwa języka:

```
CCONSTRAINT [CK_LanguageName] CHECK  (([LanguageName] like
'%[a-zA-Z]%'))
```

ID języka	LanguageID
Język	LanguageName

```
CREATE TABLE [dbo].[Languages](
    [LanguageID] [int] IDENTITY(1,1) NOT NULL,
    [LanguageName] [nchar](40) NOT NULL,
 CONSTRAINT [PK_Languages] PRIMARY KEY CLUSTERED
(
    [LanguageID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Languages]  WITH CHECK ADD  CONSTRAINT
[CK_LanguageName] CHECK  (([LanguageName] like '%[a-zA-Z]%'))
GO

ALTER TABLE [dbo].[Languages] CHECK CONSTRAINT [CK_LanguageName]
GO
```

Tabela TranslatedLanguages

Przypisuje języki do tłumaczy

Klucz główny: TranslatorID, LanguageID

Klucze obce: TranslatorID, LanguageID

ID tłumacza	TranslatorID
ID języka	LanguageID

```
CREATE TABLE [dbo].[TranslatedLanguages](
    [TranslatorID] [int] NOT NULL,
    [LanguageID] [int] NOT NULL,
 CONSTRAINT [PK_TranslatedLanguages] PRIMARY KEY CLUSTERED
(
    [TranslatorID] ASC,
    [LanguageID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[TranslatedLanguages]  WITH CHECK ADD
CONSTRAINT [FK_TranslatedLanguages_Languages] FOREIGN
KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
GO

ALTER TABLE [dbo].[TranslatedLanguages] CHECK CONSTRAINT
[FK_TranslatedLanguages_Languages]
GO

ALTER TABLE [dbo].[TranslatedLanguages]  WITH CHECK ADD
CONSTRAINT [FK_TranslatedLanguages_Translators] FOREIGN
KEY([TranslatorID])
REFERENCES [dbo].[Translators] ([TranslatorID])
GO
```

```
ALTER TABLE [dbo].[TranslatedLanguages] CHECK CONSTRAINT
[FK_TranslatedLanguages_Translators]
GO
```

Tabela Rooms

Słownik sal

Klucz główny: RoomID

ID pokoju	RoomID
Numer pokoju	RoomNumber

```
CREATE TABLE [dbo].[Rooms](
    [RoomID] [int] IDENTITY(1,1) NOT NULL,
    [RoomNumber] [nchar](40) NOT NULL,
 CONSTRAINT [PK_Rooms] PRIMARY KEY CLUSTERED
(
    [RoomID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS =
ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela LecturesTypes

Słownik rodzajów spotkań

Klucz główny: `TypeID`

ID typu zajęć	`TypeID`
Typ zajęć	`TypeName`

```
CREATE TABLE [dbo].[LecturesTypes](
    [TypeID] [int] IDENTITY(1,1) NOT NULL,
    [TypeName] [nchar](30) NOT NULL,
 CONSTRAINT [PK_LecturesTypes] PRIMARY KEY CLUSTERED
(
    [TypeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
```

Tabela Grades

Oceny końcowe na studiach

Klucz główny: GradeID

Klucze obce: UserID, StudiesID, SubjectID

Warunki integralności:

poprawna ocena:

```
CONSTRAINT [CK_Grade] CHECK  (([grade]=(5.0) OR [grade]=(4.5) OR [grade]=(4.0) OR [grade]=(3.5) OR [grade]=(3.0) OR [grade]=(2.5) OR [grade]=(2.0)))
```

poprawna data:

```
CONSTRAINT [CK_GradeDate] CHECK  (([date]<getdate() AND datepart(year,[date])>=(datepart(year,getdate())-(1))))
```

ID oceny	GradeID
ID użytkownika	UserID
ID studiów	StudiesID
ID przedmiotu	SubjectID
Ocena	Grade
Data uzyskania oceny **Default GETDATE()**	Date

```
CREATE TABLE [dbo].[Grades](
    [GradeID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NOT NULL,
    [StudiesID] [int] NOT NULL,
    [SubjectID] [int] NOT NULL,
    [Grade] [float] NOT NULL,
    [Date] [date] NULL,
 CONSTRAINT [PK_Grades] PRIMARY KEY CLUSTERED
(
    [GradeID] ASC
```

```
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Grades] ADD  CONSTRAINT [DF_Grades_Date]  DEFAULT
(getdate()) FOR [Date]
GO

ALTER TABLE [dbo].[Grades]  WITH CHECK ADD  CONSTRAINT
[FK_Grades_Studies] FOREIGN KEY([StudiesID])
REFERENCES [dbo].[Studies] ([StudiesID])
GO

ALTER TABLE [dbo].[Grades] CHECK CONSTRAINT [FK_Grades_Studies]
GO

ALTER TABLE [dbo].[Grades]  WITH CHECK ADD  CONSTRAINT
[FK_Grades_Subjects] FOREIGN KEY([SubjectID])
REFERENCES [dbo].[Subjects] ([SubjectID])
GO

ALTER TABLE [dbo].[Grades] CHECK CONSTRAINT [FK_Grades_Subjects]
GO

ALTER TABLE [dbo].[Grades]  WITH CHECK ADD  CONSTRAINT
[FK_Grades_Users] FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[Grades] CHECK CONSTRAINT [FK_Grades_Users]
GO

ALTER TABLE [dbo].[Grades]  WITH CHECK ADD  CONSTRAINT [CK_Grade] CHECK
(([grade]=(5.0) OR [grade]=(4.5) OR [grade]=(4.0) OR [grade]=(3.5) OR
[grade]=(3.0) OR [grade]=(2.5) OR [grade]=(2.0)))
GO

ALTER TABLE [dbo].[Grades] CHECK CONSTRAINT [CK_Grade]
GO
```

```
ALTER TABLE [dbo].[Grades]  WITH CHECK ADD  CONSTRAINT [CK_GradeDate]
CHECK  (([date]<getdate() AND
datepart(year,[date])>=(datepart(year,getdate())-(1))))
GO

ALTER TABLE [dbo].[Grades] CHECK CONSTRAINT [CK_GradeDate]
GO
```

Tabela ModulePassList

Lista zaliczeń modułów

Klucz główny: ListID

Klucze obce: UserID, ModuleID

ID listy	ListID
ID użytkownika	UserID
ID modułu	ModuleID
Czy moduł zaliczył (0 - nie, 1 - tak) Default 1	HasPassed

```
CREATE TABLE [dbo].[ModulePassList](
    [ListID] [int] IDENTITY(1,1) NOT NULL,
    [ModuleID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [HasPassed] [bit] NULL,
 CONSTRAINT [PK_ModulePassList] PRIMARY KEY CLUSTERED
(
    [ListID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[ModulePassList] ADD  CONSTRAINT
[DF_ModulePassList_HasPassed]  DEFAULT ((1)) FOR [HasPassed]
GO

ALTER TABLE [dbo].[ModulePassList]  WITH CHECK ADD  CONSTRAINT
[FK_ModulePassList_Module] FOREIGN KEY([ModuleID])
REFERENCES [dbo].[Module] ([ModuleID])
GO

ALTER TABLE [dbo].[ModulePassList] CHECK CONSTRAINT
[FK_ModulePassList_Module]
```

```
GO

ALTER TABLE [dbo].[ModulePassList]  WITH CHECK ADD  CONSTRAINT
[FK_ModulePassList_Users] FOREIGN KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[ModulePassList] CHECK CONSTRAINT
[FK_ModulePassList_Users]
GO
```

5. Widoki

Raporty finansowe MB

Zestawienie przychodów dla każdego webinaru/kursu/studium

#1 - Raport finansowy webinaru

```
CREATE VIEW [dbo].[WebinarsFinanceRaport]
AS
SELECT        W.WebinarID, P.ProductName, SUM(P.Price) AS Income
FROM dbo.OrderDetails AS OD
INNER JOIN dbo.Products AS P ON OD.ProductID = P.ProductID
INNER JOIN dbo.Categories AS C ON P.CategoryID = C.CategoryID
INNER JOIN dbo.Webinars AS W ON P.ProductID = W.ProductID
WHERE (C.CategoryName = 'webinar') and
       (OD.IsPaid = 1 or Od.DeferredPayment>0)
GROUP BY W.WebinarID, P.ProductName
GO
```

#2 - Raport finansowy kursu

```
CREATE VIEW [dbo].[CoursesFinanaceRaport]
AS
SELECT CO.CourseID, P.ProductName, SUM(P.Price) AS Income
FROM dbo.OrderDetails AS OD
INNER JOIN dbo.Products AS P ON OD.ProductID = P.ProductID
INNER JOIN dbo.Categories AS CAT ON P.CategoryID = CAT.CategoryID
INNER JOIN dbo.Courses AS CO ON P.ProductID = CO.ProductID
WHERE (CAT.CategoryName = 'kurs') and
      (OD.IsPaid = 1 or Od.DeferredPayment>0)
GROUP BY CO.CourseID, P.ProductName
GO
```

#3 - Raport finansowy studium

```
CREATE VIEW [dbo].[StudiesFinanceRaport]
AS
SELECT S.StudiesID, P.ProductName, SUM(P.Price) AS Income
FROM dbo.OrderDetails AS OD
INNER JOIN dbo.Products AS P ON OD.ProductID = P.ProductID
INNER JOIN dbo.Categories AS C ON P.CategoryID = C.CategoryID
INNER JOIN dbo.Studies AS S ON P.ProductID = S.ProductID
WHERE (C.CategoryName = 'studia') and
      (IsPaid=1 or DeferredPayment>0)
GROUP BY S.StudiesID, P.ProductName
GO
```

Lista dłużników MB

Osoby, które skorzystały z usług,
ale nie uiściły opłat.

Nazwa widoku: DebtorList

```
CREATE VIEW [dbo].[DebtorList]
AS
SELECT u.UserID, u.FirstName, u.LastName,
       SUM(od.DeferredPayment) AS TotalDept
FROM dbo.Orders AS o
INNER JOIN dbo.OrderDetails AS od ON o.OrderID = od.OrderID
INNER JOIN dbo.Users AS u ON u.UserID = o.UserID
WHERE (od.DeferredPayment > 0) AND (od.IsPaid = 0)
GROUP BY u.UserID, u.FirstName, u.LastName
GO
```

Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia ND

(z informacją, czy wydarzenie jest stacjonarnie, czy zdalnie)

Nazwa widoku: ReportForFutureEventsRegistration

```
CREATE VIEW [dbo].[ReportForFutureEventsRegistration]
AS
SELECT  c.CategoryName as Category, P.ProductName, COUNT(OD.OrderID) AS
EnrolledAmount, 'ZDALNIE' AS Location
FROM dbo.Webinars AS W
INNER JOIN dbo.OrderDetails AS OD ON W.ProductID = OD.ProductID
INNER JOIN dbo.Products AS P ON W.ProductID = P.ProductID
INNER JOIN dbo.Categories as c on c.CategoryID = p.CategoryID
WHERE (W.Date > GETDATE())
GROUP BY W.WebinarID, P.ProductName, c.CategoryName

UNION
SELECT c.CategoryName as Category, RTRIM(CAST(P.ProductName AS
NVARCHAR(MAX), c.CategoryName
)) + ' ' + RTRIM(SUB.SubjectName) +
       CAST(L.LectureID as char(10)) AS ProductName,
       COUNT(*) AS EnrolledAmount, 'STACJONARNIE' AS Location
FROM dbo.OrderDetails AS OD
INNER JOIN dbo.Products AS P ON OD.ProductID = P.ProductID
INNER JOIN dbo.Studies AS STU ON P.ProductID = STU.ProductID
INNER JOIN dbo.Syllabus AS SYL ON STU.StudiesID = SYL.StudiesID
INNER JOIN dbo.Subjects AS SUB ON SYL.SubjectID = SUB.SubjectID
INNER JOIN dbo.Lectures AS L ON SUB.SubjectID = L.SubjectID
INNER JOIN dbo.Categories as c on c.CategoryID = p.CategoryID
WHERE (L.StartDate > GETDATE())
GROUP BY RTRIM(CAST(P.ProductName AS NVARCHAR(MAX))) + ' ' +
RTRIM(SUB.SubjectName) +
         CAST(L.LectureID as char(10)), c.CategoryName

UNION
SELECT ca.CategoryName as Category, RTRIM(CAST(P.ProductName AS
NVARCHAR(MAX)))+ ' - ' + M.Title AS ProductName,
       COUNT(OD.OrderID) AS EnrolledAmount,
```

```
	CASE WHEN R.RoomNumber = 'Online' THEN 'ZDALNIE' ELSE
'STACJONARNIE' END AS Location
FROM dbo.Products AS P
INNER JOIN dbo.OrderDetails AS OD ON P.ProductID = OD.ProductID
INNER JOIN dbo.Courses AS C ON P.ProductID = C.ProductID
INNER JOIN dbo.CourseDetails AS CD ON C.CourseID = CD.CourseID
INNER JOIN dbo.Module AS M ON CD.ModuleID = M.ModuleID
INNER JOIN dbo.Rooms AS R ON M.RoomID = R.RoomID
INNER JOIN dbo.Categories as ca on ca.CategoryID = p.CategoryID
WHERE (M.StartDate > GETDATE())
GROUP BY RTRIM(CAST(P.ProductName AS NVARCHAR(MAX)))+ ' - ' + M.Title,
R.RoomNumber, ca.CategoryName
GO
```

Lista obecności dla każdego szkolenia MB

z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie

#1 LecturesPressenceList

```
CREATE VIEW [dbo].[LecturesPressenceList]
AS
SELECT CONCAT('Lecture', CAST(L.LectureID AS char(10))) AS What,
L.StartDate AS Date,
	U.FirstName, U.LastName, P_L.WasPresent
FROM dbo.Users AS U
INNER JOIN dbo.StudiesPressenceList AS P_L ON U.UserID = P_L.UserID
INNER JOIN dbo.Lectures AS L ON P_L.LectureID = L.LectureID
GO
```

#2 ModulesPressenceList

```
CREATE VIEW [dbo].[ModulesPressenceList]
AS
SELECT 'MODULE' + CAST(m.ModuleID AS char(10)) AS What, m.StartDate AS
Date,
        u.FirstName, u.LastName, mpl.HasPassed AS WasPresent
FROM dbo.Module AS m
INNER JOIN dbo.ModulePassList AS mpl ON mpl.ModuleID = m.ModuleID
INNER JOIN dbo.Users AS u ON u.UserID = mpl.UserID
GO
```

Ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach MB

Nazwa widoku: ReportOfGeneralPressence
Oparty o widoki:
- LecturesPressenceList
- ModulesPressenceList

```
CREATE VIEW [dbo].[ReportOfGeneralPressence]
AS
SELECT        What, Date, SUM(CAST(WasPresent AS INT)) AS
PresentAmount, COUNT(WasPresent) - SUM(CAST(WasPresent AS INT)) AS
AbsentAmonut
FROM            dbo.LecturesPressenceList AS L_P_L
WHERE        (Date < GETDATE())
GROUP BY What, Date

UNION
```

```
SELECT        What, Date, SUM(CAST(WasPresent AS INT)) AS
PresentAmount, COUNT(WasPresent) - SUM(CAST(WasPresent AS INT)) AS
AbsentAmonut
FROM            dbo.ModulesPressenceList AS M_P_L
WHERE        (Date < GETDATE())
GROUP BY What, Date
GO
```

Raport bilokacji MB

lista osób, które są zapisane na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo

Oparty o widok:
- KtoWKtorychGodzinach

```
CREATE VIEW [dbo].[ReportOfBilocation]
AS
SELECT t1.userId, t1.StartDate AS StartDate1, t1.EndDate AS EndDate1,
t2.StartDate AS StartDate2, t2.EndDate AS EndDate2,
t1.ProductName AS ProductName1, t2.ProductName AS ProductName2
FROM dbo.KtoWKtorychGodzinach AS t1
INNER JOIN dbo.KtoWKtorychGodzinach AS t2 ON t1.userId = t2.userId AND
                                             t1.ClassID < t2.ClassID
WHERE (t1.StartDate < t2.EndDate) AND (t1.EndDate > t2.StartDate) OR
      (t2.StartDate < t1.EndDate) AND (t2.EndDate > t1.StartDate)
GROUP BY t1.userId, t1.StartDate, t1.EndDate, t2.StartDate, t2.EndDate,
t1.ProductName, t2.ProductName
GO
```

KtoWKtorychGodzinach MB

```
CREATE VIEW [dbo].[KtoWKtorychGodzinach]
AS
with table1 AS  -- Studia daty
```

```
(SELECT o.userId, l.StartDate, l.EndDate,
        'S' + Cast(o.UserID as char(10)) +
        Cast(s.StudiesID as char(10))+Cast(sub.SubjectID as char(10)) +
        CAST(l.LectureID as char(10))as ClassID
FROM Orders o
JOIN OrderDetails od ON od.OrderID = o.OrderID
JOIN Products p ON p.ProductID = od.ProductID
JOIN Studies s ON s.ProductID = p.ProductID
JOIN Syllabus syl ON syl.StudiesID = s.StudiesID
JOIN Subjects sub ON sub.SubjectID = syl.SubjectID
JOIN Lectures l ON l.SubjectID = sub.SubjectID)
,
table2 AS --Moduly daty
(SELECT o.userId, m.StartDate, m.EndDate,
        'M' + Cast(o.UserID as char(10)) +
        CAST(c.CourseID as char(10)) +
        CAST(m.moduleID as char(10))as ClassID
FROM Orders o
JOIN OrderDetails od ON od.OrderID = o.OrderID
JOIN Products p ON p.ProductID = od.ProductID
JOIN Courses c ON c.ProductID = p.ProductID
JOIN CourseDetails cd ON cd.CourseID = c.CourseID
JOIN Module m ON m.ModuleID = cd.ModuleID)
,
table3 AS --Webinary daty
(SELECT o.UserID, w.Date AS StartDate, DATEADD(minute, 90, w.date) AS
EndDate,
        'W' + Cast(o.UserID as char(10)) +
        CAST(w.WebinarID as char(10)) as ClassID
FROM Orders o
JOIN OrderDetails od ON od.OrderID = o.OrderID
JOIN Products p ON p.ProductID = od.ProductID
JOIN Webinars w ON w.ProductID = p.ProductID)


SELECT * FROM table1
UNION
SELECT * FROM table2
UNION
SELECT * FROM table3
GO
```

DostepneOferty MB

Aktualne oferty produktów możliwych do kupienia

```sql
CREATE VIEW [dbo].[DostepneOferty]
AS
with table1 as --Ile ludzi na studiach
(
select ProductID, count(*) as StudentsAmount
from OrderDetails
where ProductID in
    (select ProductID from Products p
    where CategoryID = 2)
    and (IsPaid = 1 or DeferredPayment>0)
group by ProductID
)


SELECT p.ProductID, p.ProductName, p.CategoryID, p.Price
FROM dbo.Products AS p
join Studies s on s.ProductID = p.ProductID
join table1 t on t.productID = p.ProductID
where t.StudentsAmount < Capacity

UNION
SELECT p.ProductID, p.ProductName, p.CategoryID, p.Price
FROM dbo.Products AS p
join Webinars w on w.ProductID = p.ProductID
where w.Date > GETDATE()

UNION
SELECT p.ProductID, p.ProductName, p.CategoryID, p.Price
FROM dbo.Products AS p
join Courses c on c.ProductID = p.ProductID
where c.startDate > DATEADD(day, 3, GETDATE())
GO
```

PrzedawnioneZamowieniaDoUsuniecia MB

Zawiera OrderID które nie mają już sensu kupowania (np. kurs już wystartował lub webinar się już skończył a ktoś nie potwierdził płatności)

```
CREATE VIEW [dbo].[PrzedawnioneZamowieniaDoUsuniecia]
AS
SELECT OrderID
FROM dbo.OrderDetails AS od
WHERE (DeferredPayment = 0) AND (IsPaid = 0) AND
      (ProductID NOT IN
                (SELECT ProductID FROM dbo.DostepneOferty))
GO
```

Koszyk ND

Informacje o zamówionych lecz niezapłaconych produktach

```
CREATE VIEW [dbo].[Koszyk]
AS
SELECT o.UserID, o.OrderID, COUNT(*) AS BuyingAmount, SUM(p.Price) AS
CartValue
FROM  dbo.Orders AS o
INNER JOIN dbo.OrderDetails AS od ON o.OrderID = od.OrderID
INNER JOIN dbo.Products AS p ON p.ProductID = od.ProductID
WHERE (od.DeferredPayment = 0) AND (od.IsPaid = 0)
GROUP BY o.UserID, o.OrderID
GO
```

OplaconeZamowienia MB

Wyświetla informacje o zamowieniach które nie są już w koszyku - zostały zatwierdzone

```
CREATE VIEW [dbo].[OplaconeZamowienia]
AS
SELECT o.UserID, o.OrderID, COUNT(*) AS BuyingAmount,
SUM(p.Price) AS CartValue, od.IsPaid
FROM dbo.Orders AS o
INNER JOIN dbo.OrderDetails AS od ON o.OrderID = od.OrderID
INNER JOIN dbo.Products AS p ON p.ProductID = od.ProductID
WHERE (od.IsPaid = 1) OR (od.DeferredPayment > 0)
GROUP BY o.UserID, o.OrderID, od.IsPaid
GO
```

ListaAktywnychWebinarów ND

Wyświetla informacje o webinarach które nie uległy jeszcze przedawnieniu

```
CREATE VIEW [dbo].[ListaAktywnychWebinarow]
AS
SELECT WebinarID, Description, URL, Date
FROM dbo.Webinars
WHERE (IsActive LIKE 1)
GO
```

OcenyStudentowZaPrzedmiot ND

Wyświetla zdobyte oceny studentów z przedmiotów na które uczęszczają

```
CREATE VIEW [dbo].[OcenyStudentowZaPrzedmiot]
AS
SELECT G.UserID, U.FirstName, U.LastName, STU.FieldOfStudy,
SUB.SubjectName, G.Grade
FROM dbo.Grades AS G
INNER JOIN dbo.Users AS U ON G.UserID = U.UserID
INNER JOIN dbo.Subjects AS SUB ON G.SubjectID = SUB.SubjectID
INNER JOIN dbo.Studies AS STU ON G.StudiesID = STU.StudiesID
GO
```

ZaliczonePraktyki ND

Wyświetla UserID oraz nazwę użytkowników którzy zaliczyli wszystkie dni praktyk

```
CREATE VIEW [dbo].[ZaliczonePraktyki]
AS
SELECT u.UserID, u.FirstName + ' ' + u.LastName AS Name,
pa.DaysCompleted
FROM     dbo.PracticesAttendance AS pa INNER JOIN
                  dbo.Users AS u ON u.UserID = pa.UserID
WHERE   (pa.DaysCompleted = 28)
GO
```

StudentWszystkieZajęcia JK

Wyświetla wszystkie zajęcia studentów na które są zapisani

Oparty o widok:

- StudentWebinary
- StudentModuly
- StudentCwiczenia

```
CREATE VIEW [dbo].[StudentWszystkieZajecia]
AS
SELECT U.UserID, P.ProductName, W.Date, 'Webinar' AS What,
       U2.FirstName + ' ' + U2.LastName AS LecturerName
FROM dbo.Users AS U
INNER JOIN dbo.Orders AS O ON U.UserID = O.UserID
INNER JOIN dbo.OrderDetails AS OD ON O.OrderID = OD.OrderID
INNER JOIN dbo.Products AS P ON OD.ProductID = P.ProductID
INNER JOIN dbo.Webinars AS W ON P.ProductID = W.ProductID
INNER JOIN dbo.Users AS U2 ON W.LecturerID = U2.UserID
UNION
SELECT U.UserID, P.ProductName, M.StartDate AS Date, 'MODULE' AS What,
       'N/A' AS LecturerName
FROM dbo.Users AS U
INNER JOIN dbo.Orders AS O ON U.UserID = O.UserID
INNER JOIN dbo.OrderDetails AS OD ON O.OrderID = OD.OrderID
INNER JOIN dbo.Products AS P ON OD.ProductID = P.ProductID
INNER JOIN dbo.Courses AS C ON P.ProductID = C.ProductID
INNER JOIN dbo.CourseDetails AS CD ON C.CourseID = CD.CourseID
INNER JOIN dbo.Module AS M ON CD.ModuleID = M.ModuleID
UNION
SELECT U.UserID, P.ProductName, L.StartDate AS Date, 'LECTURE' AS What,
       U2.FirstName + ' ' + U2.LastName AS LecturerName
FROM dbo.Users AS U
INNER JOIN dbo.Orders AS O ON U.UserID = O.UserID
INNER JOIN dbo.OrderDetails AS OD ON O.OrderID = OD.OrderID
INNER JOIN dbo.Products AS P ON OD.ProductID = P.ProductID
INNER JOIN dbo.Studies AS STU ON P.ProductID = STU.ProductID
INNER JOIN dbo.Syllabus AS SYL ON STU.StudiesID = SYL.StudiesID
INNER JOIN dbo.Subjects AS SUB ON SYL.Semester = SUB.SubjectID
INNER JOIN dbo.Lectures AS L ON SUB.SubjectID = L.SubjectID
INNER JOIN dbo.Users AS U2 ON L.LecturerID = U2.UserID
GO
```

WykładowcaWszystkieZajecia JK

Wyświetla jakie zajęcia prowadzi dany wykładowca

Oparty o widok:

- WykladowcaWebinary
- WykladowcaCwiczenia

```
CREATE VIEW [dbo].[WykladowcaWszystkieZajecia]
AS
SELECT U.UserID, P.ProductName AS Title, W.Date, 'WEBINAR' AS What
FROM dbo.Users U
INNER JOIN dbo.Webinars W ON U.UserID = W.LecturerID
INNER JOIN dbo.Products P ON W.ProductID = P.ProductID

UNION
SELECT U.UserID, CONCAT('Lecture', CAST(L.LectureID AS char(10))) AS
Title,
       L.StartDate AS Date, 'LECTURE' AS What
FROM dbo.Users AS U
INNER JOIN dbo.Lectures AS L ON U.UserID = L.LecturerID
GO
```

TlumaczWszystkieZajecia JK

Wyświetla jakie zajęcia tłumaczy dany tłumacz

Oparty o widok:

- TlumaczWebinary
- TlumaczModuly
- TlumaczStudia

```
CREATE VIEW [dbo].[TlumaczWszystkieZajecia]
AS
SELECT T.UserID, CONCAT('Lecture', CAST(L.LectureID AS char(10))) AS
Title,
       L.StartDate AS Date, 'LECTURE' AS What
FROM dbo.Translators AS T
INNER JOIN dbo.Lectures AS L ON T.TranslatorID = L.TranslatorID

UNION
SELECT T.UserID, M.Title, M.StartDate, 'MODULE' AS What
FROM  dbo.Products AS P
INNER JOIN dbo.Courses AS C ON P.ProductID = C.ProductID
INNER JOIN dbo.Translators AS T
INNER JOIN dbo.Module AS M ON T.TranslatorID = M.TranslatorID
INNER JOIN dbo.CourseDetails AS CD ON M.ModuleID = CD.ModuleID ON
C.CourseID = CD.CourseID

UNION
SELECT T.UserID, P.ProductName AS Title, W.Date, 'WEBINAR' AS What
FROM dbo.Translators AS T
INNER JOIN dbo.Webinars AS W ON T.TranslatorID = W.TranslatorID
INNER JOIN dbo.Products AS P ON W.ProductID = P.ProductID
GO
```

ProcentZaliczeniaModulow JK

Wyświetla procent zaliczenia wszystkich modułów dla użytkowników

Oparty o widok:

- ModulePassList

```
CREATE VIEW [dbo].[ProcentZaliczeniaModulow]
AS
SELECT mpl.UserID, c.CourseID,
       CAST(SUM(CAST(mpl.HasPassed AS decimal(7, 2))) /
       COUNT(mpl.HasPassed) * 100 AS int) AS ProcentZaliczenia
FROM dbo.ModulePassList AS mpl
INNER JOIN dbo.Module AS m ON m.ModuleID = mpl.ModuleID
INNER JOIN dbo.CourseDetails AS cd ON cd.ModuleID = m.ModuleID
INNER JOIN dbo.Courses AS c ON c.CourseID = cd.CourseID
GROUP BY mpl.UserID, c.CourseID
GO
```

6. Procedury

AddAccountant JK

Dodaje księgową

```
ALTER PROCEDURE [dbo].[AddAccountant]
    @FirstName nchar(50),
    @LastName nchar(50),
    @BirthDate date,
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
    @Zip nchar(100),
    @Email nchar(50),
    @Phone nvarchar(50),
    @Login nchar(50),
    @Password nchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewUserID int;
    EXEC [dbo].[AddUser]
        @FirstName,
        @LastName,
        @BirthDate,
        @CityName,
        @CountryName,
        @Street,
        @Zip,
        @Email,
        @Phone,
        @Login,
        @Password,
        @NewUserID OUTPUT;

    IF @NewUserID IS NOT NULL
    BEGIN
        PRINT 'Added user with userID: ' + CAST(@NewUserID AS
NVARCHAR(10));
        EXEC [dbo].[AssignRoleToUser] @UserID = @NewUserID, @RoleID =
2;
```

```
    END
    ELSE
    BEGIN

        PRINT 'Adding a user failed. No role assigned.';
    END
END
```

AddAdmin JK

Dodaje admina

```
ALTER PROCEDURE [dbo].[AddAdmin]
    @FirstName nchar(50),
    @LastName nchar(50),
    @BirthDate date,
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
    @Zip nchar(100),
    @Email nchar(50),
    @Phone nvarchar(50),
    @Login nchar(50),
    @Password nchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewUserID int;
    EXEC [dbo].[AddUser]
        @FirstName,
        @LastName,
        @BirthDate,
        @CityName,
        @CountryName,
        @Street,
        @Zip,
        @Email,
        @Phone,
        @Login,
        @Password,
        @NewUserID OUTPUT;

    IF @NewUserID IS NOT NULL
```

```
    BEGIN
        PRINT 'Added user with userID: ' + CAST(@NewUserID AS
NVARCHAR(10));
        EXEC [dbo].[AssignRoleToUser] @UserID = @NewUserID, @RoleID =
6;
    END
    ELSE
    BEGIN
        PRINT 'Adding a user failed. No role assigned.';
    END
END
```

AddLecturer JK

Dodaje wykładowcę

```
ALTER PROCEDURE [dbo].[AddLecturer]
    @FirstName nchar(50),
    @LastName nchar(50),
    @BirthDate date,
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
    @Zip nchar(100),
    @Email nchar(50),
    @Phone nvarchar(50),
    @Login nchar(50),
    @Password nchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewUserID int;
    EXEC [dbo].[AddUser]
        @FirstName,
        @LastName,
        @BirthDate,
        @CityName,
        @CountryName,
        @Street,
        @Zip,
        @Email,
        @Phone,
        @Login,
```

```
        @Password,
        @NewUserID OUTPUT;

    IF @NewUserID IS NOT NULL
    BEGIN
        PRINT 'Added user with userID: ' + CAST(@NewUserID AS
NVARCHAR(10));
        EXEC [dbo].[AssignRoleToUser] @UserID = @NewUserID, @RoleID =
5;
    END
    ELSE
    BEGIN
        PRINT 'Adding a user failed. No role assigned.';
    END
END
```

AddPrincipal JK

Dodaje dyrektora

```
ALTER PROCEDURE [dbo].[AddPrincipal]
    @FirstName nchar(50),
    @LastName nchar(50),
    @BirthDate date,
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
    @Zip nchar(100),
    @Email nchar(50),
    @Phone nvarchar(50),
    @Login nchar(50),
    @Password nchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewUserID int;
    EXEC [dbo].[AddUser]
        @FirstName,
        @LastName,
        @BirthDate,
        @CityName,
        @CountryName,
        @Street,
```

```
        @Zip,
        @Email,
        @Phone,
        @Login,
        @Password,
        @NewUserID OUTPUT;

    IF @NewUserID IS NOT NULL
    BEGIN
        PRINT 'Added user with userID: ' + CAST(@NewUserID AS
NVARCHAR(10));
        EXEC [dbo].[AssignRoleToUser] @UserID = @NewUserID, @RoleID =
7;
    END
    ELSE
    BEGIN
        PRINT 'Adding a user failed. No role assigned.';
    END
END
```

AddStudent JK

Dodaje studenta

```
ALTER PROCEDURE [dbo].[AddStudent]
    @FirstName nchar(50),
    @LastName nchar(50),
    @BirthDate date,
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
    @Zip nchar(100),
    @Email nchar(50),
    @Phone nvarchar(50),
    @Login nchar(50),
    @Password nchar(50)
AS
BEGIN
    SET NOCOUNT ON;


    DECLARE @NewUserID int;
    EXEC [dbo].[AddUser]
        @FirstName,
```

```
        @LastName,
        @BirthDate,
        @CityName,
        @CountryName,
        @Street,
        @Zip,
        @Email,
        @Phone,
        @Login,
        @Password,
        @NewUserID OUTPUT;

    IF @NewUserID IS NOT NULL
    BEGIN
        PRINT 'Added user with userID: ' + CAST(@NewUserID AS
NVARCHAR(10));
        EXEC [dbo].[AssignRoleToUser] @UserID = @NewUserID, @RoleID =
1;
    END
    ELSE
    BEGIN

        PRINT 'Adding a user failed. No role assigned.';
    END
END
```

AddSubjectCoordinator JK

Dodaje koordynatora przedmiotów

```
ALTER PROCEDURE [dbo].[AddSubjectCoordinator]
    @FirstName nchar(50),
    @LastName nchar(50),
    @BirthDate date,
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
    @Zip nchar(100),
    @Email nchar(50),
    @Phone nvarchar(50),
    @Login nchar(50),
    @Password nchar(50)
AS
```

```
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewUserID int;
    EXEC [dbo].[AddUser]
        @FirstName,
        @LastName,
        @BirthDate,
        @CityName,
        @CountryName,
        @Street,
        @Zip,
        @Email,
        @Phone,
        @Login,
        @Password,
        @NewUserID OUTPUT;

    IF @NewUserID IS NOT NULL
    BEGIN
        PRINT 'Added user with userID: ' + CAST(@NewUserID AS
NVARCHAR(10));
        EXEC [dbo].[AssignRoleToUser] @UserID = @NewUserID, @RoleID =
3;
    END
    ELSE
    BEGIN
        PRINT 'Adding a user failed. No role assigned.';
    END
END

select * from Roles
```

AddTranslator JK

Dodaje tłumacza

```
ALTER PROCEDURE [dbo].[AddTranslator]
    @FirstName nchar(50),
    @LastName nchar(50),
    @BirthDate date,
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
```

```
    @Zip nchar(100),
    @Email nchar(50),
    @Phone nvarchar(50),
    @Login nchar(50),
    @Password nchar(50)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewUserID int;
    EXEC [dbo].[AddUser]
        @FirstName,
        @LastName,
        @BirthDate,
        @CityName,
        @CountryName,
        @Street,
        @Zip,
        @Email,
        @Phone,
        @Login,
        @Password,
        @NewUserID OUTPUT;

    IF @NewUserID IS NOT NULL
    BEGIN
        PRINT 'Added user with userID: ' + CAST(@NewUserID AS
NVARCHAR(10));
        EXEC [dbo].[AssignRoleToUser] @UserID = @NewUserID, @RoleID =
4;

        INSERT INTO [dbo].[Translators] ([UserID])
        VALUES (@NewUserID);
        PRINT 'Added translator with UserID: ' + CAST(@NewUserID AS
NVARCHAR(10));
    END
    ELSE
    BEGIN
        PRINT 'Adding a user failed. No role assigned.';
    END
END
```

AddCourse JK

Dodaje kurs

```
ALTER PROCEDURE [dbo].[AddCourse]
    @ProductName nchar(100),
    @Description nchar(100),
    @StartDate date,
    @EndDate date,
    @Capacity int,
    @LanguageID int,
    @ModuleIDsList nvarchar(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ProductID int, @CourseID int;

    BEGIN TRANSACTION;

    INSERT INTO [dbo].[Products] ([ProductName], [CategoryID], [Price])
    VALUES (@ProductName, 1, 0);

    SET @ProductID = SCOPE_IDENTITY();

    INSERT INTO [dbo].[Courses] ([ProductID], [Description],
[StartDate], [EndDate], [Capacity], [LanguageID])
    VALUES (@ProductID, @Description, @StartDate, @EndDate, @Capacity,
@LanguageID);

    SET @CourseID = SCOPE_IDENTITY();

    DECLARE @ModuleID int;

    DECLARE @ModuleIDsTable TABLE (ModuleID int);

    INSERT INTO @ModuleIDsTable (ModuleID)
    SELECT value FROM STRING_SPLIT(@ModuleIDsList, ',');

    DECLARE @ModuleTable TABLE (ModuleID int, StartDate datetime,
EndDate datetime)
```

```
    INSERT INTO @ModuleTable (ModuleID, StartDate, EndDate)
    SELECT m.ModuleID, m.StartDate, m.EndDate FROM Module m where
ModuleID in (SELECT * FROM @ModuleIDsTable)

    IF EXISTS (
        SELECT 1
        FROM @ModuleTable t1
        join @ModuleTable t2 on t1.ModuleID <> t2.moduleID
        WHERE (t1.StartDate < t2.EndDate) AND (t1.EndDate >=
t2.EndDate) OR
              (t2.StartDate >= t1.StartDate) AND (t2.StartDate <
t1.EndDate)
    )
    BEGIN
        ROLLBACK TRANSACTION;
        PRINT 'Error: Modules have overlapping time slots.';
        RETURN;
    END


    IF EXISTS (
        SELECT 1
        FROM @ModuleTable mt
        WHERE
            (@StartDate > mt.StartDate)
            OR (@EndDate < mt.EndDate)
    )
    BEGIN
        ROLLBACK TRANSACTION;
        PRINT 'Error: Modules do not fit in course dates.';
        RETURN;
    END

    IF EXISTS (
        SELECT 1
        FROM @ModuleIDsTable mt
        WHERE NOT EXISTS (
            SELECT 1
            FROM [dbo].[CourseDetails] cd
            WHERE cd.ModuleID = mt.ModuleID
        )
    )
    BEGIN
```

```
        INSERT INTO [dbo].[CourseDetails] ([CourseID], [ModuleID])
        SELECT @CourseID, mt.ModuleID
        FROM @ModuleIDsTable mt;
    END
    ELSE
    BEGIN
        ROLLBACK TRANSACTION;
        PRINT 'Error: One or more ModuleIDs do not exist or are already
assigned to other courses.';
        RETURN;
    END

    COMMIT TRANSACTION;

    PRINT 'Course added successfully.';
END
```

AddModule JK

Dodaje moduł

```
ALTER PROCEDURE [dbo].[AddModule]
    @Title nchar(75),
    @TypeID int,
    @Description nchar(256),
    @TranslatorID int,
    @RoomID int,
    @StartDate datetime,
    @EndDate datetime
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO [dbo].[Module] ([Title], [TypeID], [Description],
[TranslatorID], [RoomID], [StartDate], [EndDate])
    VALUES (
        @Title,
        @TypeID,
        @Description,
        @TranslatorID,
        @RoomID,
        @StartDate,
        @EndDate
```

```
    );
END
```

AddWebinar ND

Dodaje webinar

```
ALTER PROCEDURE [dbo].[AddWebinar]
    @Description NCHAR(100),
    @LecturerID INT,
    @TranslatorID INT,
    @URL NCHAR(256),
    @Date DATETIME,
    @LanguageID INT,
    @Price MONEY
AS
BEGIN
    SET NOCOUNT ON;
        DECLARE @IsValidLecturer BIT;
    DECLARE @IsValidTranslator BIT;
    DECLARE @IsValidLanguage BIT;

    SELECT @IsValidLecturer = CASE WHEN EXISTS (SELECT 1 FROM
[dbo].[Users] WHERE [UserID] = @LecturerID) THEN 1 ELSE 0 END;

    SELECT @IsValidTranslator = CASE WHEN EXISTS (SELECT 1 FROM
[dbo].[Translators] WHERE [TranslatorID] = @TranslatorID) THEN 1 ELSE 0
END;

    SELECT @IsValidLanguage = CASE WHEN EXISTS (SELECT 1 FROM
[dbo].[Languages] WHERE [LanguageID] = @LanguageID) THEN 1 ELSE 0 END;


    IF @IsValidLecturer = 1 AND @IsValidTranslator = 1 AND
@IsValidLanguage = 1
    BEGIN

    INSERT INTO [dbo].[Products] (
      [ProductName],
      [CategoryID],
      [Price]
    )
```

```
    VALUES (
      @Description,
      3,
      @Price
    );

    DECLARE @ProductID INT;
    SET @ProductID = SCOPE_IDENTITY();

    INSERT INTO [dbo].[Webinars] (
    [Description],
    [LecturerID],
    [TranslatorID],
    [URL],
    [Date],
    [LanguageID],
    [ProductID],
    [IsActive]
    )
    VALUES (
      @Description,
      @LecturerID,
      @TranslatorID,
      @URL,
      @Date,
      @LanguageID,
      @ProductID,
      1
    );

    END
    ELSE
    BEGIN
        PRINT 'Błąd: Nieprawidłowy wykładowca, tłumacz lub język.';
    END;
END;
```

AddNewStudy ND

Dodaje studia

```
ALTER PROCEDURE [dbo].[AddNewStudy]
    @ProductName nchar(100),
```

```
    @Price money,
    @LanguageID int,
    @SemesterAmount int,
    @FieldOfStudy nchar(40),
    @Description nchar(256),
    @Capacity int,
    @IndividualPrice money
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM dbo.Languages WHERE LanguageID =
@LanguageID)
    BEGIN
        PRINT 'Error: Language with ID ' + CAST(@LanguageID AS
NVARCHAR(10)) + ' does not exist.';
        RETURN;
    END

    INSERT INTO dbo.Products (ProductName, CategoryID, Price)
    VALUES (@ProductName, 2, @Price);
    DECLARE @ProductID int = SCOPE_IDENTITY();

    INSERT INTO dbo.Studies (ProductID, SemesterAmount, FieldOfStudy,
Description, LanguageID, Capacity, IndividualPrice)
    VALUES (@ProductID, @SemesterAmount, @FieldOfStudy, @Description,
@LanguageID, @Capacity, @IndividualPrice);

    PRINT 'Study program added successfully.';
END
```

AddSyllabus ND

Dodaje syllabus

```
ALTER PROCEDURE [dbo].[AddSyllabus]
    @StudiesID INT,
    @SubjectID INT,
    @Semester INT,
    @ECTS INT,
    @IsExam BIT,
    @TypeID INT
```

```
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM dbo.Studies WHERE StudiesID =
@StudiesID)
    BEGIN
        PRINT 'Error: Study with ID ' + CAST(@StudiesID AS
NVARCHAR(10)) + ' does not exist. Check the correctness of the data or
add a new study';
        RETURN;
    END

  IF NOT EXISTS (SELECT 1 FROM dbo.LecturesTypes WHERE TypeID =
@TypeID)
    BEGIN
        PRINT 'Error: Lecture type with ID ' + CAST(@TypeID AS
NVARCHAR(10)) + ' does not exist.';
        RETURN;
    END



    INSERT INTO dbo.Syllabus (StudiesID, SubjectID, Semester, ECTS,
IsExam, TypeID)
    VALUES (@StudiesID, @SubjectID, @Semester, @ECTS, @IsExam,
@TypeID);

    PRINT 'Syllabus entry added successfully.';
END
```

AddNewSubject ND

Dodaje przedmiot

```
ALTER PROCEDURE [dbo].[AddNewSubject]
    @SubjectName nchar(100)
```

```
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (SELECT 1 FROM dbo.Subjects WHERE SubjectName =
@SubjectName)
    BEGIN
        PRINT 'Error: Subject with name ' + @SubjectName + ' already
exists.';
        RETURN;
    END

    INSERT INTO dbo.Subjects (SubjectName)
    VALUES (@SubjectName);

    PRINT 'Subject added successfully.';
END
```

AddNewLecture ND

Dodaje wykład do planu zajęć

```
ALTER PROCEDURE [dbo].[AddNewLecture]
    @SubjectID INT,
    @StartDate DATETIME,
    @EndDate DATETIME,
    @LecturerID INT,
    @Capacity INT,
    @RoomID INT,
    @TranslatorID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM dbo.Subjects WHERE SubjectID =
@SubjectID)
    BEGIN
        PRINT 'Error: Subject with ID ' + CAST(@SubjectID AS
NVARCHAR(10)) + ' does not exist.';
        RETURN;
    END
```

```
    IF NOT EXISTS (SELECT 1 FROM dbo.Users WHERE UserID = @LecturerID)
    BEGIN
        PRINT 'Error: Lecturer with ID ' + CAST(@LecturerID AS
NVARCHAR(10)) + ' does not exist.';
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM dbo.Rooms WHERE RoomID = @RoomID)
    BEGIN
        PRINT 'Error: Room with ID ' + CAST(@RoomID AS NVARCHAR(10)) +
' does not exist.';
        RETURN;
    END

    IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT 1 FROM
dbo.Translators WHERE TranslatorID = @TranslatorID)
    BEGIN
        PRINT 'Error: Translator with ID ' + CAST(@TranslatorID AS
NVARCHAR(10)) + ' does not exist.';
        RETURN;
    END


    INSERT INTO dbo.Lectures (SubjectID, StartDate, EndDate,
LecturerID, Capacity, RoomID, TranslatorID)
    VALUES (@SubjectID, @StartDate, @EndDate, @LecturerID, @Capacity,
@RoomID, @TranslatorID);

    PRINT 'Lecture added successfully.';
END
```

UpdateUserAddress ND

Umożliwia zmianę adresu zamieszkania dla danego użytkownika

```
CREATE PROCEDURE [dbo].[UpdateUserAddress]
    @CityName nchar(75),
    @CountryName nchar(30),
    @Street nchar(100),
    @Zip nchar(100),
    @UserID int OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @CityID int, @CountryID int, @AddressID int;

    SELECT @CityID = [CityID]
    FROM [dbo].[Cities]
    WHERE [CityName] = @CityName;

    IF @CityID IS NULL
    BEGIN

        SELECT @CountryID = [CountryID]
        FROM [dbo].[Countries]
        WHERE [CountryName] = @CountryName;


        IF @CountryID IS NULL
        BEGIN
            INSERT INTO [dbo].[Countries] ([CountryName])
            VALUES (@CountryName);

            SET @CountryID = SCOPE_IDENTITY();
        END


        INSERT INTO [dbo].[Cities] ([CityName], [CountryID])
        VALUES (@CityName, @CountryID);

        SET @CityID = SCOPE_IDENTITY();


        INSERT INTO [dbo].[Addresses] ([CityID], [Street], [Zip])
        VALUES (@CityID, @Street, @Zip);
```

```
        SET @AddressID = SCOPE_IDENTITY();
    END
    ELSE
    BEGIN

        SELECT @AddressID = [AddressID]
        FROM [dbo].[Addresses]
        WHERE [CityID] = @CityID;


        IF @AddressID IS NULL
        BEGIN
            INSERT INTO [dbo].[Addresses] ([CityID], [Street], [Zip])
            VALUES (@CityID, @Street, @Zip);

            SET @AddressID = SCOPE_IDENTITY();
        END
    END

    UPDATE Users
    SET AddressID = @AddressID
    WHERE UserID = @UserID;

END
GO
```

AddGrade ND

Dodaje ocenę dla studenta

```
ALTER PROCEDURE [dbo].[AddGrade]
    @UserID INT,
    @StudiesID INT,
    @SubjectID INT,
    @Grade FLOAT
AS
BEGIN
    SET NOCOUNT ON;

    IF NOT EXISTS (SELECT 1 FROM dbo.Users WHERE UserID = @UserID)
    BEGIN
```

```
        PRINT 'Error: User with ID ' + CAST(@UserID AS NVARCHAR(10)) +
' not found.';
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM dbo.Studies WHERE StudiesID =
@StudiesID)
    BEGIN
        PRINT 'Error: Studies with ID ' + CAST(@StudiesID AS
NVARCHAR(10)) + ' not found.';
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM dbo.Subjects WHERE SubjectID =
@SubjectID)
    BEGIN
        PRINT 'Error: Subject with ID ' + CAST(@SubjectID AS
NVARCHAR(10)) + ' not found.';
        RETURN;
    END

    IF NOT EXISTS (SELECT 1 FROM dbo.GetUserStudies(@UserID) us WHERE
us.StudiesID = @StudiesID)
    BEGIN
        PRINT 'Error: User is not enrolled in the specified subject for
the given studies.';
        RETURN;
    END

    IF @Grade NOT IN (5.0, 4.5, 4.0, 3.5, 3.0, 2.5, 2.0)
    BEGIN
        PRINT 'Error: Invalid grade. Valid grades are 5.0, 4.5, 4.0,
3.5, 3.0, 2.5, 2.0.';
        RETURN;
    END

    INSERT INTO dbo.Grades (UserID, StudiesID, SubjectID, Grade, Date)
    VALUES (@UserID, @StudiesID, @SubjectID, @Grade, GETDATE());

    PRINT 'Grade added successfully.';
END
```

CheckAndUpdateWebinarStatus ND

Umożliwia uaktualnienie webinaru jeśli jego data opublikowania jest starsza niż 30 dni wstecz

```
ALTER PROCEDURE [dbo].[CheckAndUpdateWebinarStatus](@WebinarID INT)
AS
BEGIN
    DECLARE @WebinarDate DATETIME
    DECLARE @CurrentDate DATETIME

    SELECT @WebinarDate = [Date], @CurrentDate = GETDATE()
    FROM dbo.Webinars
    WHERE WebinarID = @WebinarID

    IF DATEDIFF(DAY, @WebinarDate, @CurrentDate) > 30
    BEGIN
        UPDATE dbo.Webinars
        SET URL = NULL, IsActive = 0
        WHERE WebinarID = @WebinarID

        PRINT 'Status webinaru został zaktualizowany.'
    END
    ELSE
    BEGIN
        PRINT 'Webinar musi być aktywny co najmniej 30 dni.'
    END
END
```

AddLanguageToTranslator JK

Przypisuje język tłumaczowi

```
ALTER PROCEDURE [dbo].[AddLanguageToTranslator]
    @UserID int,
    @LanguageName nvarchar(50)
AS
BEGIN
    SET NOCOUNT ON;
```

```
    DECLARE @TranslatorID int;
    DECLARE @LanguageID int;

    SELECT @TranslatorID = [TranslatorID]
    FROM [dbo].[Translators]
    WHERE [UserID] = @UserID;

    IF @TranslatorID IS NOT NULL
    BEGIN
        SELECT @LanguageID = [LanguageID]
        FROM [dbo].[Languages]
        WHERE [LanguageName] = @LanguageName;

        IF @LanguageID IS NULL
        BEGIN
            INSERT INTO [dbo].[Languages] ([LanguageName])
            VALUES (@LanguageName);

            SET @LanguageID = SCOPE_IDENTITY();
        END

        IF NOT EXISTS (
            SELECT 1
            FROM [dbo].[TranslatedLanguages]
            WHERE [TranslatorID] = @TranslatorID AND [LanguageID] =
@LanguageID
        )
        BEGIN
            INSERT INTO [dbo].[TranslatedLanguages] ([TranslatorID],
[LanguageID])
            VALUES (@TranslatorID, @LanguageID);

            PRINT 'Language added to translator successfully.';
        END
        ELSE
        BEGIN
            PRINT 'Translator already knows this language.';
        END
    END
    ELSE
    BEGIN
        PRINT 'User is not a translator.';
    END
```

```
END
```

CreateOrder MB

Tworzy nowe zamówienie

```
ALTER PROCEDURE [dbo].[CreateOrder]
    @UserID INT,
    @ProductIDs NVARCHAR(MAX)
AS
BEGIN
  DECLARE @ListOfProductIDs TABLE (ProductID INT);
  INSERT INTO @ListOfProductIDs (ProductID)
    SELECT value
    FROM STRING_SPLIT(@ProductIDs, ',');

  DECLARE @ProductCount INT;
    DECLARE @UniqueProductCount INT;

    SELECT @ProductCount = COUNT(*) FROM @ListOfProductIDs;
    SELECT @UniqueProductCount = COUNT(DISTINCT ProductID) FROM
@ListOfProductIDs;

  -- Sprawdzenie czy ktoś 2 razy tego samego produktu nie chce kupić
  IF @ProductCount <> @UniqueProductCount
  BEGIN
        PRINT 'Podane ProductID nie są unikalne!';
        RETURN;
    END;

    -- Sprawdzenie poprawność ProductID
  DECLARE @AvailableOffersAmount INT;
  SELECT @AvailableOffersAmount = COUNT(*) FROM dbo.DostepneOferty
                      WHERE ProductID IN
                     (SELECT ProductID FROM @ListOfProductIDs);

    IF @AvailableOffersAmount < @ProductCount
    BEGIN
        PRINT 'Podane ProductID są nieprawidłowe lub nie są dostępne do
kupienia.';
        RETURN;
    END;
```

```
    -- Sprawdzenie, czy użytkownik nie kupił już danego produktu
    IF EXISTS (SELECT 1 FROM dbo.OrderDetails od
                INNER JOIN dbo.DostepneOferty do ON od.ProductID =
do.ProductID
                WHERE od.OrderID IN (SELECT OrderID FROM dbo.Orders
WHERE UserID = @UserID)
                  AND do.ProductID IN (SELECT ProductID FROM
@ListOfProductIDs))
    BEGIN
        PRINT 'Użytkownik już zakupił co najmniej jeden z podanych
produktów.';
        RETURN;
    END;

    -- Dodanie nowego zamówienia do tabeli Orders
    DECLARE @NewOrderID INT;

    INSERT INTO dbo.Orders (UserID, OrderDate)
    VALUES (@UserID, GETDATE());

    SET @NewOrderID = SCOPE_IDENTITY();

    -- Dodanie rekordu/ów do tabeli OrderDetails
    INSERT INTO dbo.OrderDetails (OrderID, ProductID, DeferredPayment,
IsPaid)
    SELECT @NewOrderID, ProductID, 0, 0
    FROM dbo.DostepneOferty
    WHERE ProductID IN (SELECT ProductID FROM @ListOfProductIDs);

    PRINT 'Oferta została utworzona dla użytkownika o ID ' +
CAST(@UserID AS NVARCHAR(10));

END;
```

DeleteOrder MB

Usuwa nieopłacacone jeszcze zamówienia

```
ALTER PROCEDURE [dbo].[DeleteOrder]
    @UserID INT,
    @OrderID INT
```

```
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzenie czy można usunąć zamówienie
    IF EXISTS (
        SELECT 1
        FROM dbo.Koszyk
        WHERE UserID = @UserID AND OrderID = @OrderID
    )
    BEGIN
    DELETE FROM dbo.OrderDetails WHERE OrderID = @OrderID;
        DELETE FROM dbo.Orders WHERE OrderID = @OrderID;
        PRINT 'Order deleted!';
    END
    ELSE
    BEGIN
        PRINT 'Wrong parameters!';
    END
END;
```

MarkAttendanceForModule MB

Sprawdzanie obecności na danym module

```
ALTER PROCEDURE [dbo].[MarkAttendanceForModule]
    @ModuleID INT,
    @AbsentUserIDs NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzenie, czy istnieje moduł o podanym ModuleID
    IF NOT EXISTS (SELECT 1 FROM dbo.Module WHERE ModuleID = @ModuleID)
    BEGIN;
        THROW 50001, 'ModuleID does not exist.', 1;
```

```
        RETURN;
    END

   -- Sprawdzenie, czy istnieje już lista obecności dla tego ModuleID
    IF EXISTS (SELECT 1 FROM dbo.ModulePassList WHERE ModuleID =
@ModuleID)
    BEGIN;
        THROW 50002, 'Attendance list already exists for ModuleID.', 1;
        RETURN;
    END

  -- Konwersja listy ID użytkowników do tabeli tymczasowej
    DECLARE @UserIDsTable TABLE (UserID INT);
    INSERT INTO @UserIDsTable (UserID)
    SELECT CAST(value AS INT)
    FROM STRING_SPLIT(@AbsentUserIDs, ',');

  -- Sprawdzenie, czy wszyscy użytkownicy są zapisani na dany moduł
  IF (SELECT COUNT(UserID) FROM @UserIDsTable)
  !=
  (SELECT COUNT(u.UserID)
    FROM dbo.GetUsersEnrolledInModule(@ModuleID) u
    JOIN @UserIDsTable a ON u.UserID = a.UserID)
    BEGIN;
        THROW 50003, 'Not all users are enrolled in the specified
module.', 1;
        RETURN;
    END;


    -- Dodanie nieobecnych
    INSERT INTO dbo.ModulePassList (ModuleID, UserID, HasPassed)
  SELECT @ModuleID, UserID, 0
  FROM @UserIDsTable;

  -- Dodanie obecnych
  INSERT INTO dbo.ModulePassList (ModuleID, UserID, HasPassed)
  SELECT
    @ModuleID,
    u.UserID,
    1
  FROM
    dbo.GetUsersEnrolledInModule(@ModuleID) u
```

```
  LEFT JOIN
    @UserIDsTable a ON u.UserID = a.UserID
  WHERE
    a.UserID IS NULL;

END;
```

MarkAttendanceForLecture MB

Sprawdzanie obecności na danych ćwiczeniach

```
alter PROCEDURE [dbo].[MarkAttendanceForLecture]
    @LectureID INT,
    @AbsentUserIDs NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzenie, czy istnieją zajęcia o podanym LectureID
    IF NOT EXISTS (SELECT 1 FROM dbo.Lectures WHERE LectureID =
@LectureID)
    BEGIN;
        THROW 50001, 'ID does not exist.', 1;
```

```
        RETURN;
    END

     -- Sprawdzenie, czy istnieje już lista obecności dla tego ModuleID
    IF EXISTS (SELECT 1 FROM StudiesPressenceList WHERE LectureID =
@LectureID)
    BEGIN;
        THROW 50002, 'Attendance list already exists for LectureID.',
1;
        RETURN;
    END

    DECLARE @UserIDsTable TABLE (UserID INT);

    -- Konwersja listy ID użytkowników do tabeli tymczasowej
    IF LEN(@AbsentUserIDs) > 0
    BEGIN
        INSERT INTO @UserIDsTable (UserID)
        SELECT CAST(value AS INT)
        FROM STRING_SPLIT(@AbsentUserIDs, ',');
    END

    -- Sprawdzenie, czy wszyscy użytkownicy są zapisani na dany moduł
    IF (SELECT COUNT(UserID) FROM @UserIDsTable)
    !=
    (SELECT COUNT(u.UserID) FROM
dbo.GetUsersEnrolledOnLecture(@LectureID) u
    JOIN @UserIDsTable a ON u.UserID = a.UserID)
    BEGIN;
        THROW 50003, 'Not all users are enrolled in the specified
module.', 1;
        RETURN;
    END;


    -- Dodanie nieobecnych
    INSERT INTO dbo.StudiesPressenceList (LectureID, UserID,
WasPresent)
    SELECT @LectureID, u.UserID, 0
    FROM @UserIDsTable u;

    -- Dodanie obecnych
    INSERT INTO dbo.StudiesPressenceList(LectureID, UserID, WasPresent)
```

```
    SELECT @LectureID, u.UserID, 1
    FROM dbo.GetUsersEnrolledOnLecture(@LectureID) u
    LEFT JOIN @UserIDsTable a ON u.UserID = a.UserID
    WHERE a.UserID IS NULL;

END;
GO
```

PayForOrder MB

Płatność za zamówienie

```
ALTER PROCEDURE [dbo].[PayForOrder]
    @OrderID INT
AS
BEGIN
    -- Sprawdzanie, czy zamówienie istnieje i nie jest jeszcze opłacone
    IF EXISTS (SELECT 1 FROM dbo.OrderDetails
                        WHERE OrderID = @OrderID AND IsPaid = 0)
    BEGIN
        UPDATE dbo.OrderDetails
        SET IsPaid = 1
        WHERE OrderID = @OrderID AND IsPaid = 0;

        PRINT 'Zamówienie o ID ' + CAST(@OrderID AS NVARCHAR(10)) + '
```

```
        zostało opłacone.';
    END
    ELSE
    BEGIN
        PRINT 'Zamówienie o ID ' + CAST(@OrderID AS NVARCHAR(10)) + '
               nie istnieje lub już zostało opłacone.';
    END
END;
```

PostponePayment MB

Odroczenie zapłaty

```
ALTER PROCEDURE [dbo].[PostponePayment]
    @OrderID INT
AS
BEGIN
    -- Aktualizowanie "OrderDetails" tylko jeśli "DeferredPayment" jest
równe 0 i "IsPaid" jest równe 0
    UPDATE od
    SET od.DeferredPayment = p.Price
    FROM dbo.OrderDetails od
    INNER JOIN dbo.Products p ON od.ProductID = p.ProductID
    WHERE od.OrderID = @OrderID AND od.DeferredPayment = 0 AND
od.IsPaid = 0;
```

```
    PRINT 'Płatność dla zamówienia o ID ' + CAST(@OrderID AS
NVARCHAR(10)) + ' została opóźniona.';

END;
```

RevokeUserRole JK

Cofa role użytkownikowi

```
ALTER PROCEDURE [dbo].[RevokeUserRole]
    @UserID INT,
    @RoleID INT
AS
BEGIN
    UPDATE dbo.RolesAssigned
    SET Until = GETDATE()
    WHERE UserID = @UserID AND RoleID = @RoleID;
END;
```

UpdateModulePassList MB

Aktualizuje listę zaliczonych modułów

```
ALTER PROCEDURE [dbo].[UpdateModulePassList]
    @ModuleID INT,
    @UserID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzenie, czy istnieje rekord w ModulePassList dla danego
ModuleID i UserID
    IF NOT EXISTS (SELECT 1 FROM dbo.ModulePassList WHERE ModuleID =
@ModuleID AND UserID = @UserID)
    BEGIN;
        THROW 50001, 'ModulePassList record does not exist for
specified ModuleID and UserID.', 1;
```

```
        RETURN;
    END

    -- Sprawdzenie, czy HasPassed jest ustawione na 0
    IF NOT EXISTS (SELECT 1 FROM dbo.ModulePassList WHERE ModuleID =
@ModuleID AND UserID = @UserID AND HasPassed = 0)
    BEGIN;
        THROW 50002, 'Cannot update ModulePassList record. HasPassed is
already set to 1.', 1;
        RETURN;
    END

    -- Aktualizacja HasPassed na 1
    UPDATE dbo.ModulePassList
    SET HasPassed = 1
    WHERE ModuleID = @ModuleID AND UserID = @UserID;

    PRINT 'ModulePass updated successfully.';
END;
```

7. Funkcje

GetModulesAndLecturesByRoomID ND

Wyświetla jakie zajęcia studyjne i moduły są przypisane do danej sali

```
CREATE FUNCTION [dbo].[GetModulesAndLecturesByRoomID](@RoomID INT)
RETURNS TABLE
AS
RETURN (
SELECT
        m.ModuleID as ID,
        m.Title AS Lecture,
        m.StartDate as StartDate,
        m.EndDate as EndDate,
        m.TranslatorID as TranslatorID
```

```
    FROM [u_ndziwak].[dbo].[Module] m
    JOIN [u_ndziwak].[dbo].[Rooms] r ON r.RoomID = m.RoomID
    WHERE m.RoomID = @RoomID

    UNION ALL

    SELECT
        l.LectureID as ID,
        s.SubjectName AS Lecture,
        l.StartDate as StartDate,
        l.EndDate as EndDate,
        l.TranslatorID as TranslatorID
    FROM [u_ndziwak].[dbo].[Lectures] l
    JOIN [u_ndziwak].[dbo].[Subjects] s on s.SubjectID = l.SubjectID
    JOIN [u_ndziwak].[dbo].[Rooms] r ON r.RoomID = l.RoomID
    WHERE l.RoomID = @RoomID
);
GO
```

GetModulesAndLecturesByTypeID ND

Wyszukiwane zajęć o podobnej tematyce, zwraca moduły i zajęcia studyjne dla danej kategorii

```
CREATE FUNCTION [dbo].[GetModulesAndLecturesByTypeID](@TypeID INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        m.ModuleID as ID,
        m.Title AS Lecture,
        m.StartDate as StartDate,
        m.EndDate as EndDate,
        m.RoomID as RomID,
        m.TranslatorID as TranslatorID
    FROM [u_ndziwak].[dbo].[Module] m
    JOIN [u_ndziwak].[dbo].[LecturesTypes] lt ON lt.TypeID = m.TypeID
    WHERE m.TypeID = @TypeID

    UNION ALL
```

```
    SELECT
        l.LectureID as ID,
        s.SubjectName AS Lecture,
        l.StartDate as StartDate,
        l.EndDate as EndDate,
        l.RoomID as RomID,
        l.TranslatorID as TranslatorID
    FROM [u_ndziwak].[dbo].[Lectures] l
    JOIN [u_ndziwak].[dbo].[Subjects] s on s.SubjectID = l.SubjectID
    JOIN [u_ndziwak].[dbo].[Syllabus] sy ON sy.SubjectID = s.SubjectID
    JOIN [u_ndziwak].[dbo].[LecturesTypes] lt ON lt.TypeID = sy.TypeID
    WHERE sy.TypeID = @TypeID
);
GO
```

GetStudentsOnCourse JK

Wyświetla użytkowników zapisanych na dany kurs

```
CREATE FUNCTION [dbo].[GetStudentsOnCourse]
(
    @CourseID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT u.UserID, u.FirstName, u.LastName
    FROM dbo.Users u
    JOIN dbo.Orders o ON u.UserID = o.UserID
    JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
    JOIN dbo.Products p ON od.ProductID = p.ProductID
    JOIN dbo.Courses c ON p.ProductID = c.ProductID
    WHERE c.CourseID = @CourseID
);
```

GetStudentsOnStudies JK

Wyświetla użytkowników zapisanych na dane studia

```
CREATE FUNCTION [dbo].[GetStudentsOnStudies]
(
    @StudiesID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT u.UserID, u.FirstName, u.LastName
    FROM dbo.Users u
    JOIN dbo.Orders o ON u.UserID = o.UserID
    JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
    JOIN dbo.Products p ON od.ProductID = p.ProductID
    JOIN dbo.Studies s ON p.ProductID = s.ProductID
    WHERE s.StudiesID = @StudiesID
);
GO
```

GetStudentsOnWebinar JK

Wyświetla użytkowników zapisanych na dany webinar

```
CREATE FUNCTION [dbo].[GetStudentsOnWebinar]
(
    @WebinarID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT u.UserID, u.FirstName, u.LastName
    FROM dbo.Users u
    JOIN dbo.Orders o ON u.UserID = o.UserID
    JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
    JOIN dbo.Products p ON od.ProductID = p.ProductID
    JOIN dbo.Webinars w ON p.ProductID = w.ProductID
    WHERE w.WebinarID = @WebinarID
);
GO
```

GetUserCoursesSchedule JK

Wyświetla harmonogram zajęć z kursów danego użytkownika

```
CREATE FUNCTION [dbo].[GetUserCoursesSchedule]
(
    @UserID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT c.Description, m.Title, m.StartDate, m.EndDate, lt.TypeName,
r.RoomNumber, l.LanguageName
    FROM dbo.Users u
    JOIN dbo.Orders o ON u.UserID = o.UserID
    JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
    JOIN dbo.Products p ON od.ProductID = p.ProductID
    JOIN dbo.Courses c ON p.ProductID = c.ProductID
    JOIN dbo.CourseDetails cd on c.CourseID = cd.CourseID
    JOIN dbo.Module m on m.ModuleID = cd.ModuleID
    JOIN dbo.Rooms r on r.RoomID = m.RoomID
    JOIN dbo.LecturesTypes lt on lt.TypeID = m.TypeID
```

```
    JOIN dbo.Languages l on l.LanguageID = c.LanguageID
    WHERE u.UserID = @UserID
);
GO
```

GetUserRoles JK

Wyświetla jaką rolę pełni dany użytkownik

```
CREATE FUNCTION [dbo].[GetUserRoles](@UserID INT)
RETURNS TABLE
AS
RETURN(
    SELECT R.RoleName
    FROM dbo.RolesAssigned RA
    JOIN dbo.Roles R ON RA.RoleID = R.RoleID
    WHERE RA.UserID = @UserID
);
GO
```

GetUsersEnrolledInModule MB

Wyświetla listę użytkowników zapisanych na dany moduł

```
CREATE FUNCTION [dbo].[GetUsersEnrolledInModule]
(
    @ModuleID INT
)
RETURNS TABLE
```

```
AS
RETURN
(
    SELECT DISTINCT
        o.UserID
    FROM
        Orders o
    JOIN
        OrderDetails od ON od.OrderID = o.OrderID
    JOIN
        Products p ON p.ProductID = od.ProductID
    JOIN
        Courses c ON c.ProductID = p.ProductID
    JOIN
        CourseDetails cd ON cd.CourseID = c.CourseID
    WHERE
        cd.ModuleID = @ModuleID
);
GO
```

GetUsersEnrolledOnLecture MB

Wyświetla listę użytkowników zapisanych na dane zajęcia studyjne

```
CREATE FUNCTION [dbo].[GetUsersEnrolledOnLecture]
(
    @LectureID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT DISTINCT o.UserID
    FROM Orders o
    JOIN OrderDetails od ON od.OrderID = o.OrderID
    JOIN Products p ON p.ProductID = od.ProductID
    JOIN Studies s on s.ProductID = p.ProductID
    JOIN Syllabus syl on syl.StudiesID = s.StudiesID
    join Subjects sub on sub.SubjectID = syl.SubjectID
    join Lectures l on l.SubjectID = sub.SubjectID
    where l.LectureID = @LectureID and (DeferredPayment>0 or IsPaid=1)
);
GO
```

GetUserStudies ND

Wyświetla na jakie studia zapisany jest dany użytkownik

```
CREATE FUNCTION [dbo].[GetUserStudies]
(
    @UserID INT
)
RETURNS TABLE
AS
RETURN
(
      select s.StudiesID from OrderDetails as od
      join products as p on p.ProductID = od.ProductID
      join Studies as s on s.ProductID = p.ProductID
      join orders as o on o.OrderID = od.OrderID
      WHERE od.ProductID IN
          (SELECT ProductID
           FROM Products p
           WHERE CategoryID = 2) AND (IsPaid = 1 OR DeferredPayment > 0)
AND o.userid = @UserID
);
GO
```

GetUserStudiesSchedule JK

Wyświetla harmonogram zajęć z studiów danego użytkownika

```
CREATE FUNCTION [dbo].[GetUserStudiesSchedule]
(
    @UserID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT l.StartDate, l.EndDate, sub.SubjectName, lt.TypeName AS
LectureType, r.RoomNumber
    FROM dbo.Users u
    JOIN dbo.Orders o ON u.UserID = o.UserID
    JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
    JOIN dbo.Products p ON od.ProductID = p.ProductID
    JOIN dbo.Studies stu ON p.ProductID = stu.ProductID
    JOIN dbo.Syllabus syl ON stu.StudiesID = syl.StudiesID
    JOIN dbo.Subjects sub ON syl.SubjectID = sub.SubjectID
    JOIN dbo.Lectures l ON sub.SubjectID = l.SubjectID
    JOIN dbo.LecturesTypes lt on syl.TypeID = lt.TypeID
```

```
    JOIN dbo.Rooms r on l.RoomID = r.RoomID
    WHERE u.UserID = @UserID
);
GO
```

GetUserWebinarsSchedule JK

Wyświetla harmonogram webinarów danego użytkownika

```
CREATE FUNCTION [dbo].[GetUserWebinarsSchedule]
(
    @UserID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT w.Date, w.Description, l.LanguageName AS LectureType
    FROM dbo.Users u
    JOIN dbo.Orders o ON u.UserID = o.UserID
    JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
    JOIN dbo.Products p ON od.ProductID = p.ProductID
    JOIN dbo.Webinars w ON p.ProductID = w.ProductID
    JOIN dbo.Languages l on l.LanguageID = w.LanguageID
    WHERE u.UserID = @UserID
);
GO
```

8. Triggery

tr_DeleteTranslatorOnRoleChange ND

Usuwa translatora, oraz informacje o tym jakie języki tłumaczył, gdy jego rola jako tłumacza się zakończy

```
CREATE TRIGGER tr_DeleteTranslatorOnRoleChange
ON [dbo].[RolesAssigned]
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    IF UPDATE(Until)
    BEGIN

        DELETE TL
        FROM [dbo].[TranslatedLanguages] TL
        INNER JOIN Translators T ON T.TranslatorID = TL.TranslatorID
        INNER JOIN Users U ON U.[UserID] = T.[UserID]
        INNER JOIN deleted D ON T.[UserID] = D.[UserID]
        WHERE NOT EXISTS (
            SELECT 1
            FROM [dbo].[RolesAssigned] RA
            WHERE RA.[UserID] = T.[UserID]
```

```
              AND RA.[RoleID] = 4
              AND RA.[Until] IS NULL
        );


        DELETE T
        FROM [dbo].[Translators] T
        INNER JOIN deleted D ON T.[UserID] = D.[UserID]
        WHERE NOT EXISTS (
            SELECT 1
            FROM [dbo].[RolesAssigned] RA
            WHERE RA.[UserID] = D.[UserID]
              AND RA.[RoleID] = 4
              AND RA.[Until] IS NULL
        );
    END
END;
```

9. Indeksy

Courses

Po LanguageID
CREATE NONCLUSTERED INDEX IX_Courses_LanguageID ON dbo.Courses(LanguageID);

Products

Po CategoryID
CREATE NONCLUSTERED INDEX IX_Products_CategoryID ON dbo.Products(CategoryID);

Users

Po Email
CREATE NONCLUSTERED INDEX IX_Users_Email ON dbo.Users(Email);

Po Login
CREATE NONCLUSTERED INDEX IX_Users_Login ON dbo.Users(Login);

CourseDetails

Po CourseID
CREATE NONCLUSTERED INDEX IX_CourseDetails_CourseID ON dbo.CourseDetails(CourseID);

Po ModuleID
CREATE NONCLUSTERED INDEX IX_CourseDetails_ModuleID ON dbo.CourseDetails(ModuleID);

Grades

Po UserID
CREATE NONCLUSTERED INDEX IX_Grades_UserID ON dbo.Grades(UserID);

Po StudiesID
CREATE NONCLUSTERED INDEX IX_Grades_StudiesID ON dbo.Grades(StudiesID);

Po SubjectID
CREATE NONCLUSTERED INDEX IX_Grades_SubjectID ON dbo.Grades(SubjectID);

ModulePassList

Po ModuleID
CREATE NONCLUSTERED INDEX IX_ModulePassList_ModuleID ON dbo.ModulePassList(ModuleID);

Po UserID
CREATE NONCLUSTERED INDEX IX_ModulePassList_UserID ON dbo.ModulePassList(UserID);

OrderDetails

Po OrderID
CREATE NONCLUSTERED INDEX IX_OrderDetails_OrderID ON dbo.OrderDetails(OrderID);

Po ProductID

CREATE NONCLUSTERED INDEX IX_OrderDetails_ProductID ON dbo.OrderDetails(ProductID);

PracticesAttendance

Po StudiesID
CREATE NONCLUSTERED INDEX IX_PracticesAttendance_StudiesID ON dbo.PracticesAttendance(StudiesID);

Po UserID
CREATE NONCLUSTERED INDEX IX_PracticesAttendance_UserID ON dbo.PracticesAttendance(UserID);

RolesAssigned

Po RoleID
CREATE NONCLUSTERED INDEX IX_RolesAssigned_RoleID ON dbo.RolesAssigned(RoleID);

Po UserID
CREATE NONCLUSTERED INDEX IX_RolesAssigned_UserID ON dbo.RolesAssigned(UserID);

Syllabus

Po StudiesID
CREATE NONCLUSTERED INDEX IX_Syllabus_StudiesID ON dbo.Syllabus(StudiesID);

Po SubjectID
CREATE NONCLUSTERED INDEX IX_Syllabus_SubjectID ON dbo.Syllabus(SubjectID);

TranslatedLanguages

Po TranslaotrID
CREATE NONCLUSTERED INDEX IX_TranslatedLanguages_TranslatorID ON dbo.TranslatedLanguages(TranslatorID);

Po LanguageID

CREATE NONCLUSTERED INDEX IX_TranslatedLanguages_LanguageID ON dbo.TranslatedLanguages(LanguageID);

StudiesPressenceList

Po LectureID
CREATE NONCLUSTERED INDEX IX_StudiesPressenceList_LectureID ON dbo.StudiesPressenceList(LectureID);

Po UserID
CREATE NONCLUSTERED INDEX IX_StudiesPressenceList_UserID ON dbo.StudiesPressenceList(UserID);

10. Uprawnienia

Księgowa

```
grant select on DebtorList to ksiegowa
grant select on CoursesFinanaceRaport to ksiegowa
grant select on StudiesFinanceRaport to ksiegowa
grant select on WebinarsFinanceRaport to ksiegowa
grant select on OplaconeZamowienia to ksiegowa
```

Dyrektor

```
grant select on Koszyk to dyrektor
grant exec on PostponePayment to dyrektor
```

Tłumacz

```
grant select on TlumaczWszystkieZajecia to tlumacz
```

Wykładowca

```
grant select on WykladowcaWszystkieZajecia to wykladowca
grant select on Grades to wykladowca
grant exec on AddGrade to wykladowca
grant exec on MarkAttendanceForLecture to wykladowca
grant exec on MarkAttendanceForModule to wykladowca
```

```
grant exec on UpdateModulePassList to wykladowca
grant select on LecturesPressenceList to wykladowca
grant select on [dbo].[ModulesPressenceList] to wykladowca
```

Student

```
grant select on [dbo].[DostepneOferty] to student
grant select on [dbo].[Koszyk] to student
grant select on [dbo].[LecturesPressenceList] to student
grant select on [dbo].[ModulesPressenceList] to student
grant select on [dbo].[OcenyStudentowZaPrzedmiot] to student
grant select on [dbo].[OplaconeZamowienia] to student
grant select on [dbo].[ProcentZaliczeniaModulow] to student
grant select on [dbo].[GetUserStudiesSchedule] to student
grant select on [dbo].[GetUserWebinarsSchedule] to student
grant exec on [dbo].[GetUserPracticeDaysCompleted] to student
grant exec on [dbo].[CreateOrder] to student
grant exec on [dbo].[PayForOrder] to student
```

Koordynator przedmiotów

```
grant select on [dbo].[ReportForFutureEventsRegistration] to
koordynator_przedmiotow
grant select on [dbo].[ReportOfBilocation] to koordynator_przedmiotow
grant select on [dbo].[ReportOfGeneralPressence] to koordynator_przedmiotow
grant exec on [dbo].[AddCourse] to koordynator_przedmiotow
grant exec on [dbo].[AddLanguageToTranslator] to koordynator_przedmiotow
grant exec on [dbo].[AddNewLecture] to koordynator_przedmiotow
grant exec on [dbo].[AddNewStudy] to koordynator_przedmiotow
grant exec on [dbo].[AddSyllabus] to koordynator_przedmiotow
grant exec on [dbo].[AddWebinar] to koordynator_przedmiotow
grant exec on [dbo].[CheckAndUpdateWebinarStatus] to koordynator_przedmiotow
grant select on [dbo].[GetModulesAndLecturesByRoomID] to
koordynator_przedmiotow
grant select on [dbo].[GetModulesAndLecturesByTypeID] to
koordynator_przedmiotow
grant select on [dbo].[GetStudentsOnCourse] to koordynator_przedmiotow
grant select on [dbo].[GetStudentsOnStudies] to koordynator_przedmiotow
grant select on [dbo].[GetStudentsOnWebinar] to koordynator_przedmiotow
grant select on [dbo].[GetUsersEnrolledInModule] to koordynator_przedmiotow
grant select on [dbo].[GetUsersEnrolledOnLecture] to koordynator_przedmiotow
grant select on [dbo].[GetUserStudies] to koordynator_przedmiotow
```

Niezalogowany

```
grant select on [dbo].[DostepneOferty] to niezalogowany
```