Baza danych: Kursy i szkolenia.
Projekt 2023/2024

Jan Mrowiec
Kacper Sobczyk

Spis treści:

Użytkownicy systemu oraz jego funkcje

Użytkownicy bazy danych:

1. Użytkownik bez założonego konta.
2. Kursant.
3. Wykładowca.
4. Dyrektor firmy.
5. Administrator systemu.

Funkcje realizowane przez system dla poszczególnych użytkowników:

Użytkownik bez założonego konta:

- Możliwość założenia konta.
- Dostęp do bezpłatnych usług:
  - przeglądanie harmonogramu szkoleń.
  - informacje o miejscu spotkań lub linkach do spotkań online.
- Dostęp do listy dostępnych usług z pełnymi opisami (sylabus, forma, cena, prowadzący, czas trwania, miejsce).
- Przeglądanie limitów miejsc dla kursów stacjonarnych i hybrydowych.

Kursant (rozszerza funkcjonalność użytkownika bez konta):

- Dołączenie do kursu
- Wpłata zaliczki za usługę.
- Dostęp do zapisanych usług:
  - Materiały szkoleniowe.
  - Harmonogram zajęć.
  - Linki do spotkań online lub informacje o miejscu spotkań.
  - Lista studentów zapisanych na kurs.
  - Zgłaszanie obecności.
  - Raporty o frekwencji, praktykach, i wynikach egzaminów.
  - Możliwość pełnej płatności za usługę.
  - Opcja rezygnacji z kursu.
  - Ustawienia konta (zmiana danych, hasła).
- Dostęp do nagrań szkoleń online (dla kursów i webinarów, które są nagrywane).

Wykładowca (rozszerza funkcjonalność użytkownika bez konta):

- Zarządzanie kursami: utworzenie, modyfikacja, usunięcie spotkań.
- Potwierdzanie obecności kursantów.
- Dostęp do danych kursantów (kontakt, imię, nazwisko).
- Generowanie raportów obecności.
- Zarządzanie egzaminami i praktykami (ustawianie dat, ocen).
- Edycja materiałów kursowych.
- Potwierdzanie zaliczenia kursu.
- Ustawienia konta.

Dyrektor (rozszerza funkcjonalność wykładowcy):

- Pełny dostęp do danych wszystkich kursantów i pracowników.
- Zarządzanie kursami: dodawanie, usuwanie, zmiana ceny, zmiana prowadzącego.
- Zarządzanie programem studiów (sylabus) i harmonogramem spotkań.
- Zarządzanie kontami (z wyjątkiem konta administratora).
- Tworzenie raportów: finansowe, dłużników, bilokacji.
- Ręczne dopisywanie kursantów do kursów.
- Decyzje o odroczeniu płatności dla stałych klientów.
- Zarządzanie limitami miejsc na kursach stacjonarnych i hybrydowych.

Administrator:

- Regularne tworzenie kopii zapasowych danych.
- Przywracanie danych z kopii zapasowej.
- Zarządzanie dostępem i uprawnieniami użytkowników.
- Edycja danych w poszczególnych tabelach.
- Tworzenie raportów dotyczących wydajności, dostępności i bezpieczeństwa bazy danych.
- Monitorowanie i optymalizacja wydajności systemu.
- Zarządzanie aktualizacjami i konserwacją systemu.

Funkcje systemowe:

- Automatyczne Generowanie Linków do Płatności:

Dla kursów płatnych, system automatycznie generuje linki do płatności po dodaniu ich do koszyka przez kursanta.

- Automatyczna Weryfikacja Płatności:

Integracja z zewnętrznym systemem płatności umożliwia automatyczne potwierdzanie statusu płatności i aktualizowanie statusu uczestnika kursu/studia.

- Automatyczne Przypomnienia o Zbliżających się Wydarzeniach:

System wysyła automatyczne powiadomienia e-mail lub SMS do zapisanych użytkowników przed rozpoczęciem webinarów, kursów czy spotkań studyjnych.

- Zarządzanie Dostępem do Materiałów Szkoleniowych:

Automatyczne udostępnianie materiałów szkoleniowych dla zarejestrowanych uczestników kursów/studiów.

- Monitoring Obecności i Automatyczne Raportowanie:

Śledzenie frekwencji na kursach stacjonarnych i online oraz generowanie automatycznych raportów obecności.

- Automatyczne Aktualizacje Harmonogramu:

W przypadku zmian w harmonogramie (np. z powodu przyczyn losowych), system automatycznie aktualizuje informacje i powiadamia zainteresowanych użytkowników.

- Automatyczne Zarządzanie Limitami Miejsc:

System monitoruje i zarządza limitami miejsc na kursach stacjonarnych i hybrydowych, automatycznie zamykając rejestrację, gdy limit zostanie osiągnięty.

- Automatyczne Tworzenie Kopii Zapasowych:

Regularne automatyczne tworzenie kopii zapasowych danych, aby zapobiec ich utracie.

- Automatyczna Weryfikacja Zaliczenia Modułów Kursu:

Dla kursów z modułami online, system może automatycznie weryfikować zaliczenie na podstawie aktywności użytkownika (np. oglądanie nagrań, uczestnictwo w quizach).

- Generowanie Automatycznych Raportów Finansowych i Statystycznych:

System regularnie generuje raporty dotyczące przychodów, frekwencji, oraz innych kluczowych wskaźników.

Opisy tabel

Tabela Users: Reprezentacja użytkowników

Klucz główny: UserID
Klucz obcy: AccountTypeID [**AccountTypes**]

ID użytkownika:	[UserID]	int
Login:	[Login]	nvarchar(50)
Hasło:	[Password]	nvarchar(50)
Imię:	[Name]	nvarchar(50)
Nazwisko:	[Surname]	nvarchar(50)
Adres e-mail:	[E-mail]	nvarchar(50)
Telefon:	[Phone]	nvarchar(50)

Warunki integralności:

- Unikalny login
- Phone składa się z samych liczb
- Adres e-mail posiada znak ‘@’
- Adres e-mail jest unikalny

```
CREATE TABLE [dbo].[Users](
    [UserID] [int] IDENTITY(1,1) NOT NULL,
    [Login] [nvarchar](50) NOT NULL,
    [Password] [nvarchar](50) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
    [Surname] [nvarchar](50) NOT NULL,
    [E-mail] [nvarchar](50) NOT NULL,
    [Phone] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_User] 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 [UniqueLogin] 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 [(isnumeric([Phone]])=(1))] CHECK
((isnumeric([Phone])=(1)))
GO

ALTER TABLE [dbo].[Users] CHECK CONSTRAINT [(isnumeric([Phone]])=(1))]
GO

ALTER TABLE [dbo].[Users]  WITH CHECK ADD  CONSTRAINT [CK_Users] CHECK  (([E-mail] like '%@%'))
GO

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

Tabela Students: Reprezentacja kursantów

Klucz główny: UserID
Klucz obcy: UserID [**Users**], AddressID [**Address**]

ID użytkownika:	[UserID]	int
Data urodzin:	[Birthdate]	date
Adres:	[AddressID]	int

Warunki integralności:

- Data urodzenia nie może być niższa niż 1900 rok
- Data urodzenia nie może być później niż aktualna data

```
CREATE TABLE [dbo].[Students](
    [UserID] [int] NOT NULL,
    [Birthdate] [date] NOT NULL,
    [AddressID] [int] NOT NULL,
 CONSTRAINT [PK_Students] 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]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Students]  WITH CHECK ADD  CONSTRAINT [FK_Students_Place] FOREIGN
KEY([AddressID])
REFERENCES [dbo].[Addresses] ([AddressID])
GO

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [FK_Students_Place]
GO

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

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [FK_Students_Users]
GO

ALTER TABLE [dbo].[Students]  WITH CHECK ADD  CONSTRAINT [BeforeToday] CHECK
(([Birthdate]<getdate()))
GO

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [BeforeToday]
GO

ALTER TABLE [dbo].[Students]  WITH CHECK ADD  CONSTRAINT [NotSoOld] CHECK  (([Birthdate]>'1900-01-
01'))
GO

ALTER TABLE [dbo].[Students] CHECK CONSTRAINT [NotSoOld]
GO
```

Tabela Interpreters: Reprezentacja tłumaczy

Klucz główny: UserID
Klucz obcy: UserID [**Users**], LanguageID [**Language**]

ID użytkownika:	[UserID]	int
Język:	[LanguageID]	int

```
CREATE TABLE [dbo].[Interpreters](
    [UserID] [int] NOT NULL,
    [LanguageID] [int] NOT NULL,
 CONSTRAINT [PK_Interpreters] 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]
) ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[Interpreters] CHECK CONSTRAINT [FK_Interpreters_Languages]
GO

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

ALTER TABLE [dbo].[Interpreters] CHECK CONSTRAINT [FK_Interpreters_Users]
GO
```

Tabela Teachers: Reprezentacja nauczycieli

Klucz główny: UserID
Klucz obcy: UserID [**Users**], RoomID [**ClassRooms**]

ID użytkownika:	[UserID]	int
Gabinet:	[RoomID]	int
Zainteresowania:	[Interests]	nvarchar(max)

Warunki integralności:

- Default zainteresowania = ‘Brak zainteresowań’

```
CREATE TABLE [dbo].[Teachers](
    [UserID] [int] NOT NULL,
    [RoomID] [int] NOT NULL,
    [Intrests] [nvarchar](50) NULL,
 CONSTRAINT [PK_Teachers] 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]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Teachers] ADD  CONSTRAINT [DF_Teachers_Intrests]  DEFAULT ('brak zainteresowan')
FOR [Intrests]
GO

ALTER TABLE [dbo].[Teachers]  WITH CHECK ADD  CONSTRAINT [FK_Teachers_Place] FOREIGN KEY([RoomID])
REFERENCES [dbo].[ClassRooms] ([PlaceID])
GO

ALTER TABLE [dbo].[Teachers] CHECK CONSTRAINT [FK_Teachers_Place]
GO

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

ALTER TABLE [dbo].[Teachers] CHECK CONSTRAINT [FK_Teachers_Users]
GO
```

Tabela UserAccountTypes: Tabela łącząca użytkowników z rolami

Klucz główny: UserAccountTypeID
Klucz obcy: AccountTypeID [**AccountType**]

ID typu konta użytkownika	[UserAccountTypeID]	int
ID użytkownika:	[UserID]	int
ID typu konta	[AccountTypeID]	int
Data wygaśnięcia	[ExpireDate]	date
Data rozpoczęcia	[StartDate]	date

Warunki integralności:

- Data rozpoczęcia nie może być później niż aktualna data
- Data wygaśnięcia nie może być wcześniejsza niż data rozpoczęcia

```
CREATE TABLE [dbo].[UserAccountTypes](
    [UserAccountTypeID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NOT NULL,
    [AccountTypeID] [int] NOT NULL,
    [StartDate] [date] NOT NULL,
    [ExpireDate] [date] NOT NULL,
 CONSTRAINT [PK_UserAccountTypes] PRIMARY KEY CLUSTERED
(
    [UserAccountTypeID] 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].[UserAccountTypes] ADD  CONSTRAINT [DF_UserAccountTypes_StartDate]  DEFAULT
(getdate()) FOR [StartDate]
GO

ALTER TABLE [dbo].[UserAccountTypes]  WITH CHECK ADD  CONSTRAINT [FK_UserAccountTypes_AccountTypes]
FOREIGN KEY([AccountTypeID])
REFERENCES [dbo].[AccountTypes] ([AccountTypeID])
GO

ALTER TABLE [dbo].[UserAccountTypes] CHECK CONSTRAINT [FK_UserAccountTypes_AccountTypes]
GO

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

ALTER TABLE [dbo].[UserAccountTypes] CHECK CONSTRAINT [FK_UserAccountTypes_Users]
GO

ALTER TABLE [dbo].[UserAccountTypes]  WITH CHECK ADD  CONSTRAINT [CK_UserAccountTypes] CHECK
(([ExpireDate]>=[StartDate]))
GO

ALTER TABLE [dbo].[UserAccountTypes] CHECK CONSTRAINT [CK_UserAccountTypes]
GO
```

Tabela AccountTypes: Reprezentacja typy użytkowników

Klucz główny: AccountID
Klucz obcy:

ID typu konta	[AccountTypeID]	int
Nazwa typu	[Name]	nvarchar(50)

```
CREATE TABLE [dbo].[AccountTypes](
    [AccountTypeID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_AccountType] PRIMARY KEY CLUSTERED
(
    [AccountTypeID] 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 Products: Reprezentacja wszystkie usługi w ramach systemu, które użytkownik może mieć dostęp
Klucz główny: ProductID
Klucz obcy: ProductTypeID [**ProductTypes**], Lecturer [**Teachers**]

ID produktu:	[ProductID]	int
Nazwa produktu:	[Name]	nvarchar(50)
Wykładowca:	[LecturerID]	int
Cena produktu:	[Price]	money
Typ produktu:	[ProductTypeID]	int
Czy jest dostępny	[IsAvailable]	bit

Warunki integralności:

- IsAvailable jest domyślnie ustawione jako Fałsz
- Cena produktu nie może być mniejsza od 0

```
CREATE TABLE [dbo].[Products](
    [ProductID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [varchar](50) NOT NULL,
    [LecturerID] [int] NULL,
    [Price] [money] NOT NULL,
    [ProductTypeID] [int] NOT NULL,
    [IsAvailable] [bit] 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] ADD  CONSTRAINT [DF_Products_IsAvaliable]  DEFAULT ((0)) FOR
[IsAvailable]
GO

ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT [FK_Products_ProductTypes] FOREIGN
KEY([ProductTypeID])
REFERENCES [dbo].[ProductTypes] ([ProductTypeID])
GO

ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_ProductTypes]
GO

ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT [FK_Products_User] FOREIGN
KEY([LecturerID])
REFERENCES [dbo].[Teachers] ([UserID])
GO

ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_User]
GO

ALTER TABLE [dbo].[Products]  WITH CHECK ADD  CONSTRAINT [Product_Price_Constrain] CHECK
(([Price]>=(0)))
GO

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

Tabela ProductTypes: Reprezentuje typy produktów

Klucz główny: ProductTypeID
Klucz obcy:

ID typu produktu:	[ProductTypeID]	int
Nazwa typu:	[Name]	nvarchar(50)

```
CREATE TABLE [dbo].[ProductTypes](
    [ProductTypeID] [int] NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_ProductTypes] PRIMARY KEY CLUSTERED
(
    [ProductTypeID] 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: Reprezentacja webinarów

Klucz główny: WebinarID
Klucz obcy: WebinarID [**Products**]

ID Webinaru:	[WebinarID]	int
Data i godzina wydarzenia:	[EventDate]	datetime
Link do spotkania:	[ConnectionLink]	nvarchar(50)
Link do materiału:	[ResourceLink]	nvarchar(50)

Warunki integralności:

- Data spotkania musi być później niż aktualna data

```
CREATE TABLE [dbo].[Webinars](
    [WebinarID] [int] NOT NULL,
    [EventDate] [datetime] NOT NULL,
    [ConnectionLink] [nvarchar](50) NULL,
    [ResourceLink] [nvarchar](50) 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_Products] FOREIGN
KEY([WebinarID])
REFERENCES [dbo].[Products] ([ProductID])
GO

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

Tabela Courses: Reprezentacja kursów

Klucz główny: CourseID
Klucz obcy: CourseID [**Products**]

ID kursu	[CourseID]	int
Kwota zaliczki	[EntryFee]	price
Data rozpoczęcia:	[StartDate]	datetime

Warunki integralności:

- Kwota zaliczki nie może być mniejsza od 0

```
CREATE TABLE [dbo].[Courses](
    [CourseID] [int] NOT NULL,
    [EntryFee] [money] NOT NULL,
    [StartDate] [datetime] NOT NULL,
 CONSTRAINT [PK_Course] 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]  WITH CHECK ADD  CONSTRAINT [FK_Course_Products] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Products] ([ProductID])
GO

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CK_Course_Fee] CHECK  (([EntryFee]>=(0)))
GO

ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [CK_Course_Fee]
GO
```

Tabela CourseModules: Reprezentacja modułów w kursie bazie danych

Klucz główny: CourseModuleID
Klucz obcy: CourseID [**Course**], Type [**ModuleMeetingType**]

ID modułu	[ModuleID]	int
Nazwa modułu	[Name]	nvarchar(MAX)
ID kursu	[CourseID]	int
Typ modułu	[Type]	int

```
CREATE TABLE [dbo].[CourseModules](
    [CourseModuleID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](max) NOT NULL,
    [CourseID] [int] NOT NULL,
    [Type] [int] NOT NULL,
 CONSTRAINT [PK_CourseModules] PRIMARY KEY CLUSTERED
(
    [CourseModuleID] 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] TEXTIMAGE_ON [PRIMARY]
GO

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

ALTER TABLE [dbo].[CourseModules] CHECK CONSTRAINT [FK_CourseModules_Course]
GO

ALTER TABLE [dbo].[CourseModules]  WITH CHECK ADD  CONSTRAINT [FK_CourseModules_ModuleMeetingType]
FOREIGN KEY([Type])
REFERENCES [dbo].[ModuleMeetingTypes] ([ModuleMeetingTypeID])
GO

ALTER TABLE [dbo].[CourseModules] CHECK CONSTRAINT [FK_CourseModules_ModuleMeetingType]
GO
```

Tabela ModuleMeetingTypes: Reprezentacja typów modułów

Klucz główny: ModuleMeetingTypeID

Klucz obcy:

ID typu modułu	[ModuleMeetingTypeID]	int
Nazwa typu:	[Name]	nvarchar(50)

```
CREATE TABLE [dbo].[ModuleMeetingTypes](
    [ModuleMeetingTypeID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_ModuleMeetingType] PRIMARY KEY CLUSTERED
(
    [ModuleMeetingTypeID] 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 StationaryModules: Reprezentacja stacjonarnych modułów

Klucz główny: CourseModuleID
Klucz obcy: CourseModuleID[**CourseModules**], PlaceID [**Place]**

ID modułu stacjonarnego:	[CourseModuleID]	int
ID miejsca:	[PlaceID]	int
Data i godzina spotkania:	[Date]	datetime
Limit miejsc:	[Limit]	int
Czas trwania:	[Duration]	int

Warunki integralności:

- Limit miejsc musi być większy od 0
- Czas trwania nie może być mniejszy niż 0
- Default czas trwania spotkania jest ustawiony na 90 minut
- Data i godzina spotkania musi być później niż aktualna data

```
CREATE TABLE [dbo].[StationaryModules](
    [CourseModuleID] [int] NOT NULL,
    [PlaceID] [int] NOT NULL,
    [Date] [datetime2](7) NOT NULL,
    [Limit] [int] NULL,
    [Duration] [int] NOT NULL,
 CONSTRAINT [PK_StationaryModules] PRIMARY KEY CLUSTERED
(
    [CourseModuleID] 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].[StationaryModules] ADD  CONSTRAINT [DF_StationaryModules_Duration]  DEFAULT
((90)) FOR [Duration]
GO

ALTER TABLE [dbo].[StationaryModules]  WITH CHECK ADD  CONSTRAINT
[FK_StationaryModules_CourseModules] FOREIGN KEY([CourseModuleID])
REFERENCES [dbo].[CourseModules] ([CourseModuleID])
GO

ALTER TABLE [dbo].[StationaryModules] CHECK CONSTRAINT [FK_StationaryModules_CourseModules]
GO

ALTER TABLE [dbo].[StationaryModules]  WITH CHECK ADD  CONSTRAINT [FK_StationaryModules_Place]
FOREIGN KEY([PlaceID])
REFERENCES [dbo].[ClassRooms] ([PlaceID])
GO

ALTER TABLE [dbo].[StationaryModules] CHECK CONSTRAINT [FK_StationaryModules_Place]
GO

ALTER TABLE [dbo].[StationaryModules]  WITH CHECK ADD  CONSTRAINT [CK_StationaryModules_Date] CHECK
(([Date]>getdate()))
GO

ALTER TABLE [dbo].[StationaryModules] CHECK CONSTRAINT [CK_StationaryModules_Date]
GO
```

```
ALTER TABLE [dbo].[StationaryModules]  WITH CHECK ADD  CONSTRAINT [CK_StationaryModules_Duration]
CHECK  (([Duration]>(0)))
GO

ALTER TABLE [dbo].[StationaryModules] CHECK CONSTRAINT [CK_StationaryModules_Duration]
GO

ALTER TABLE [dbo].[StationaryModules]  WITH CHECK ADD  CONSTRAINT [LimitGreaterThan0] CHECK
(([Limit] IS NULL OR [Limit]>(0)))
GO

ALTER TABLE [dbo].[StationaryModules] CHECK CONSTRAINT [LimitGreaterThan0]
GO
```

Tabela OnlineSyncModules: Reprezentacja modułów online synchronicznych

Klucz główny: CourseModuleID
Klucz obcy: CourseModuleID [**CourseModules**]

ID modułu stacjonarnego:	[CourseModuleID]	int
Link do spotkania:	[ConnectionLink]	nvarchar(50)
Link do zasobów:	[ResourceLink]	nvarchar(50)
Data i godzina spotkania:	[Date]	datetime
Czas trwania spotkania:	[Duration]	int

Warunki integralności:

- Czas trwania nie może być mniejszy niż 0
- Data musi być późniejsza niż data aktualna
- Default czas trwania spotkania jest ustawiony na 90 minut

```
CREATE TABLE [dbo].[OnlineSyncModules](
    [CourseModuleID] [int] NOT NULL,
    [ConnectionLink] [nvarchar](50) NULL,
    [Date] [datetime] NOT NULL,
    [ResourceLink] [nvarchar](50) NULL,
    [Duration] [int] NOT NULL,
 CONSTRAINT [PK_OnlineModules] PRIMARY KEY CLUSTERED
(
    [CourseModuleID] 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].[OnlineSyncModules] ADD  CONSTRAINT [DF_OnlineSyncModules_Duration]
DEFAULT ((90)) FOR [Duration]
GO

ALTER TABLE [dbo].[OnlineSyncModules]  WITH CHECK ADD  CONSTRAINT
[FK_OnlineModules_CourseModules] FOREIGN KEY([CourseModuleID])
REFERENCES [dbo].[CourseModules] ([CourseModuleID])
GO

ALTER TABLE [dbo].[OnlineSyncModules] CHECK CONSTRAINT [FK_OnlineModules_CourseModules]
GO

ALTER TABLE [dbo].[OnlineSyncModules]  WITH CHECK ADD  CONSTRAINT [(Date >= GETDATE())]
CHECK  (([Date]>=getdate()))
GO

ALTER TABLE [dbo].[OnlineSyncModules] CHECK CONSTRAINT [(Date >= GETDATE())]
GO

ALTER TABLE [dbo].[OnlineSyncModules]  WITH CHECK ADD  CONSTRAINT
[CK_OnlineSyncModules_Duration] CHECK  (([Duration]>(0)))
GO
```

```
ALTER TABLE [dbo].[OnlineSyncModules] CHECK CONSTRAINT [CK_OnlineSyncModules_Duration]
GO
```

Tabela OnlineAsyncModules: Reprezentacja modułów online synchronicznych
Klucz główny: CourseModuleID
Klucz obcy: CourseModuleID [**CourseModules**]

ID modułu stacjonarnego:	[CourseModuleID]	int
Link do zasobów:	[ResourceLink]	nvarchar(50)

```
CREATE TABLE [dbo].[OnlineAsyncModules](
    [CourseModuleID] [int] NOT NULL,
    [ResourceLink] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_OnlineAsyncModules] PRIMARY KEY CLUSTERED
(
    [CourseModuleID] 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].[OnlineAsyncModules]  WITH CHECK ADD  CONSTRAINT
[FK_OnlineAsyncModules_CourseModules] FOREIGN KEY([CourseModuleID])
REFERENCES [dbo].[CourseModules] ([CourseModuleID])
GO

ALTER TABLE [dbo].[OnlineAsyncModules] CHECK CONSTRAINT [FK_OnlineAsyncModules_CourseModules]
GO
```

Tabela Studies: Reprezentacja studiów

Klucz główny: StudyID
Klucz obcy: StudyID [**Products**]

ID studiów:	[StudyID]	int
Sylabus studiów:	[Sylabus]	nvarchar(MAX)
Limit miejsc:	[Limit]	int

Warunki integralności:

- Limit miejsc musi być większy od 0

```
CREATE TABLE [dbo].[Studies](
    [StudyID] [int] NOT NULL,
    [Sylabus] [nvarchar](max) NOT NULL,
    [Limit] [int] NOT NULL,
 CONSTRAINT [PK_Studies] PRIMARY KEY CLUSTERED
(
    [StudyID] 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] TEXTIMAGE_ON [PRIMARY]
GO

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

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

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [LimitHigherThat0] CHECK  (([Limit]>(0)))
GO

ALTER TABLE [dbo].[Studies] CHECK CONSTRAINT [LimitHigherThat0]
GO
```

Tabela Subjects: Reprezentacja przedmiotów

Klucz główny: SubjectID
Klucz obcy: StudyID [**Studies**]

ID przedmiotu:	[SubjectID]	int
Nazwa przedmiotu:	[Name]	nvarchar(50)
Sylabus studiów:	[Sylabus]	nvarchar(MAX)
ID studiów:	[StudiesID]	int

```
CREATE TABLE [dbo].[Subjects](
    [SubjectID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
    [Sylabus] [nvarchar](max) NOT NULL,
    [StudiesID] [int] 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] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[Subjects]  WITH CHECK ADD  CONSTRAINT [FK_Subjects_Studies] FOREIGN
KEY([StudiesID])
REFERENCES [dbo].[Studies] ([StudyID])
GO

ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [FK_Subjects_Studies]
GO
```

Tabela StudiesMeetings: Reprezentacja spotkań powiązanych ze studiami

Klucz główny: StudiesMeetingID
Klucz obcy: TypeID [**ModuleMeetingType**], StudyID [**Subject**]

ID spotkania studiów:	[StudyMeetingID]	int
ID przedmiotu:	[SubjectID]	int
Cena:	[Price]	money
Limit miejsc:	[Limit]	int
Data rozpoczęcia:	[StartDate]	datetime
ID Typ:	[TypeID]	int
Czas trwania:	[Duration]	int

Warunki integralności:

- Limit miejsc musi być większy od 0
- Cena musi być większa lub równa 0
- Czas trwania musi być większy od 0
- Default czas trwania spotkania jest ustawiony na 90 minut

```
CREATE TABLE [dbo].[StudiesMeetings](
    [StudyMeetingID] [int] IDENTITY(1,1) NOT NULL,
    [SubjectID] [int] NOT NULL,
    [Price] [money] NOT NULL,
    [Limit] [int] NOT NULL,
    [StartDate] [datetime] NOT NULL,
    [TypeID] [int] NOT NULL,
    [Duration] [int] NOT NULL,
 CONSTRAINT [PK_StudiesMeetings] PRIMARY KEY CLUSTERED
(
    [StudyMeetingID] 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].[StudiesMeetings] ADD  CONSTRAINT [DF_StudiesMeetings_Duration]  DEFAULT ((90))
FOR [Duration]
GO

ALTER TABLE [dbo].[StudiesMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_StudiesMeetings_ModuleMeetingType] FOREIGN KEY([TypeID])
REFERENCES [dbo].[ModuleMeetingTypes] ([ModuleMeetingTypeID])
GO

ALTER TABLE [dbo].[StudiesMeetings] CHECK CONSTRAINT [FK_StudiesMeetings_ModuleMeetingType]
GO

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

ALTER TABLE [dbo].[StudiesMeetings] CHECK CONSTRAINT [FK_StudiesMeetings_Subjects]
GO

ALTER TABLE [dbo].[StudiesMeetings]  WITH CHECK ADD  CONSTRAINT [CK_StudiesMeetings_Duration] CHECK
(([Duration]>=(0)))
GO
```

```
ALTER TABLE [dbo].[StudiesMeetings] CHECK CONSTRAINT [CK_StudiesMeetings_Duration]
GO

ALTER TABLE [dbo].[StudiesMeetings]  WITH CHECK ADD  CONSTRAINT [CK_StudiesMeetings_Limit] CHECK
(([Limit]>(0)))
GO

ALTER TABLE [dbo].[StudiesMeetings] CHECK CONSTRAINT [CK_StudiesMeetings_Limit]
GO

ALTER TABLE [dbo].[StudiesMeetings]  WITH CHECK ADD  CONSTRAINT [CK_StudiesMeetings_Price] CHECK
(([Price]>=(0)))
GO

ALTER TABLE [dbo].[StudiesMeetings] CHECK CONSTRAINT [CK_StudiesMeetings_Price]
GO
```

Tabela StationaryMeetings: Reprezentacja stacjonarnych spotkań w ramach studiów
Klucz główny: StudyMeetingID
Klucz obcy: PlaceID [**Addresses**], StudyMeetingID[**StudiesMeetings**]

ID stacjonarnego spotkania:	[StudyMeetingID]	int
ID miejsca:	[PlaceID]	int

```
CREATE TABLE [dbo].[StationaryMeetings](
    [StudyMeetingID] [int] NOT NULL,
    [PlaceID] [int] NOT NULL,
 CONSTRAINT [PK_StationaryMeetings] PRIMARY KEY CLUSTERED
(
    [StudyMeetingID] 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].[StationaryMeetings]  WITH CHECK ADD  CONSTRAINT [FK_StationaryMeetings_Place]
FOREIGN KEY([PlaceID])
REFERENCES [dbo].[ClassRooms] ([PlaceID])
GO

ALTER TABLE [dbo].[StationaryMeetings] CHECK CONSTRAINT [FK_StationaryMeetings_Place]
GO

ALTER TABLE [dbo].[StationaryMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_StationaryMeetings_StudiesMeetings] FOREIGN KEY([StudyMeetingID])
REFERENCES [dbo].[StudiesMeetings] ([StudyMeetingID])
GO

ALTER TABLE [dbo].[StationaryMeetings] CHECK CONSTRAINT [FK_StationaryMeetings_StudiesMeetings]
GO
```

Tabela OnlineMeetings: Reprezentacja online spotkań w ramach studiów

Klucz główny: StudyMeetingID
Klucz obcy: StudyMeetingID[**StudiesMeetings**]

ID spotkania online:	[StudyMeetingID]	int
Link do spotkania:	[ConnectionLink]	nvarchar(50)
Link do materiałów:	[ResourceLink]	nvarchar(50)

```
CREATE TABLE [dbo].[OnlineMeetings](
    [StudyMeetingID] [int] NOT NULL,
    [ConnectionLink] [nvarchar](50) NOT NULL,
    [ResourceLink] [nvarchar](50) NOT NULL,
 CONSTRAINT [PK_OnlineMeetings] PRIMARY KEY CLUSTERED
(
    [StudyMeetingID] 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].[OnlineMeetings]  WITH CHECK ADD  CONSTRAINT [FK_OnlineMeetings_StudiesMeetings]
FOREIGN KEY([StudyMeetingID])
REFERENCES [dbo].[StudiesMeetings] ([StudyMeetingID])
GO

ALTER TABLE [dbo].[OnlineMeetings] CHECK CONSTRAINT [FK_OnlineMeetings_StudiesMeetings]
GO
```

Tabela Internships: Reprezentacja praktyk studenta

Klucz główny: InternshipID
Klucz obcy: StudiesID [**Studies**], UserID [**Students**]

ID stażu:	[InternshipID]	int
ID studiów:	[StudiesID]	int
ID studenta	[UserID]	int
Data odbycia:	[Date]	datetime
Status:	[IsCompleted]	bit

Warunki integralności:

- Status jest domyślnie ustawiony jako 0

```
CREATE TABLE [dbo].[Internships](
    [InternshipID] [int] IDENTITY(1,1) NOT NULL,
    [StudiesID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [Date] [datetime] NOT NULL,
    [IsCompleted] [bit] NOT NULL,
 CONSTRAINT [PK_Intership] PRIMARY KEY CLUSTERED
(
    [InternshipID] 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].[Internships] ADD  CONSTRAINT [DF_Intership_IsCompleted]  DEFAULT ((0)) FOR
[IsCompleted]
GO

ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT [FK_Intership_Studies] FOREIGN
KEY([InternshipID])
REFERENCES [dbo].[Studies] ([StudyID])
GO

ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Intership_Studies]
GO

ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT [FK_Intership_Users] FOREIGN
KEY([UserID])
REFERENCES [dbo].[Students] ([UserID])
GO

ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Intership_Users]
GO
```

Tabela Exams: Reprezentacja egzaminów

Klucz główny: ExamID
Klucz obcy: PlaceID [**Place**], Studies [**Product**]

ID egzaminu:	[ExamID]	int
Nazwa:	[Name]	nvarchar(50)
ID produktu:	[StudyID]	int
Data i godzina:	[Date]	datetime
ID miejsca egzaminu:	[Place]	int
Link do egzaminu:	[ConnectionLInk]	nvarchar(50)
Opis:	[Description]	nvarchar(MAX)

Warunki integralności:

- default opis ustawiony jako ‘brak opisu’

```
CREATE TABLE [dbo].[Exams](
    [ExamID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) NOT NULL,
    [StudyID] [int] NOT NULL,
    [Date] [datetime] NOT NULL,
    [Place] [int] NULL,
    [ConnectionLInk] [nvarchar](50) NULL,
    [Description] [nvarchar](max) NULL,
 CONSTRAINT [PK_Exams] PRIMARY KEY CLUSTERED
(
    [ExamID] 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] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[Exams] ADD  CONSTRAINT [DF_Exams_Description]  DEFAULT ('brak opisu') FOR
[Description]
GO

ALTER TABLE [dbo].[Exams]  WITH CHECK ADD  CONSTRAINT [FK_Exams_Place] FOREIGN KEY([ExamID])
REFERENCES [dbo].[ClassRooms] ([PlaceID])
GO

ALTER TABLE [dbo].[Exams] CHECK CONSTRAINT [FK_Exams_Place]
GO

ALTER TABLE [dbo].[Exams]  WITH CHECK ADD  CONSTRAINT [FK_Exams_Products] FOREIGN KEY([StudyID])
REFERENCES [dbo].[Studies] ([StudyID])
GO

ALTER TABLE [dbo].[Exams] CHECK CONSTRAINT [FK_Exams_Products]
GO

ALTER TABLE [dbo].[Exams]  WITH CHECK ADD  CONSTRAINT [CK_Exams] CHECK  (([Place] IS NOT NULL OR
[ConnectionLInk] IS NOT NULL))
GO

ALTER TABLE [dbo].[Exams] CHECK CONSTRAINT [CK_Exams]
GO

ALTER TABLE [dbo].[Exams]  WITH CHECK ADD  CONSTRAINT [CK_Exams_Date] CHECK  (([Date]>=getdate()))
GO
```

```
ALTER TABLE [dbo].[Exams] CHECK CONSTRAINT [CK_Exams_Date]
GO
```

Tabela ExamResults: Reprezentacja wyników odbytych egzaminów

Klucz główny: (ExamID, StudentID, Attempt)
Klucz obcy: ExamID [**Exams**], StudentID [**Students**], ReviewerID [**Teachers**]

ID egzaminu:	[ExamID]		int
ID studenta:	[StudentID]		int
Próba podejścia:	[Attempt]		int
Ocena:	[Note]	int	
Recenzent	[RevieveID]		int
Data wpisania oceny	[SubmitDate]		date

Warunki integralności:

- Data wpisania oceny nie może być późniejsza niż aktualna data
- Wartość oceny musi zawierać się w przedziale od 1 do 5
- Próba podejścia nie może być mniejsza niż 0

```
CREATE TABLE [dbo].[ExamResults](
    [ExamID] [int] NOT NULL,
    [StudentID] [int] NOT NULL,
    [Attempt] [int] NOT NULL,
    [Note] [int] NOT NULL,
    [RevieverID] [int] NULL,
    [SubmitDate] [datetime] NOT NULL,
 CONSTRAINT [PK_ExamResults] PRIMARY KEY CLUSTERED
(
    [ExamID] ASC,
    [StudentID] ASC,
    [Attempt] 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].[ExamResults] ADD  CONSTRAINT [DF_ExamResults_SubmitDate]  DEFAULT (getdate()) FOR
[SubmitDate]
GO

ALTER TABLE [dbo].[ExamResults]  WITH CHECK ADD  CONSTRAINT [FK_ExamResults_Exams] FOREIGN
KEY([ExamID])
REFERENCES [dbo].[Exams] ([ExamID])
GO

ALTER TABLE [dbo].[ExamResults] CHECK CONSTRAINT [FK_ExamResults_Exams]
GO

ALTER TABLE [dbo].[ExamResults]  WITH CHECK ADD  CONSTRAINT [FK_ExamResults_Reviever] FOREIGN
KEY([RevieverID])
REFERENCES [dbo].[Teachers] ([UserID])
GO

ALTER TABLE [dbo].[ExamResults] CHECK CONSTRAINT [FK_ExamResults_Reviever]
GO

ALTER TABLE [dbo].[ExamResults]  WITH CHECK ADD  CONSTRAINT [FK_ExamResults_Student] FOREIGN
KEY([StudentID])
REFERENCES [dbo].[Students] ([UserID])
GO
```

```
ALTER TABLE [dbo].[ExamResults] CHECK CONSTRAINT [FK_ExamResults_Student]
GO

ALTER TABLE [dbo].[ExamResults]  WITH CHECK ADD  CONSTRAINT [(SubmitDate <= GETDATE())] CHECK
(([SubmitDate]<=getdate()))
GO

ALTER TABLE [dbo].[ExamResults] CHECK CONSTRAINT [(SubmitDate <= GETDATE())]
GO

ALTER TABLE [dbo].[ExamResults]  WITH CHECK ADD  CONSTRAINT [CK_ExamResults_Attempt] CHECK
(([Attempt]>(0)))
GO

ALTER TABLE [dbo].[ExamResults] CHECK CONSTRAINT [CK_ExamResults_Attempt]
GO

ALTER TABLE [dbo].[ExamResults]  WITH CHECK ADD  CONSTRAINT [CK_ExamResults_Note] CHECK
(([Note]>(0) AND [Note]<(5)))
GO

ALTER TABLE [dbo].[ExamResults] CHECK CONSTRAINT [CK_ExamResults_Note]
GO
```

Tabela Diplomes: Reprezentacja dyplomów uzyskanych z kursów i studiów

Klucz główny: DimplomaID
Klucz obcy: UserID [**Students**], ProductID [**Product**]

ID dyplomu	[DiplomaID]	int
ID użytkownika	[UserID]	int
ID produktu	[ProductID]	int
Data uzyskania	[Date]	int
Ocena	[Note]	int
Opis	[Description]	nvarchar(50)

Warunki integralności:

- Data uzyskania nie może być późniejsza niż aktualna data
- default opis ustawiony na ‘brak opisu’

```
CREATE TABLE [dbo].[Diplomes](
    [DiplomeID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NOT NULL,
    [ProductID] [int] NOT NULL,
    [Date] [int] NOT NULL,
    [Note] [int] NULL,
    [Description] [nvarchar](50) NULL,
 CONSTRAINT [PK_Diplomes] PRIMARY KEY CLUSTERED
(
    [DiplomeID] 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].[Diplomes] ADD  CONSTRAINT [DF_Diplomes_Description]  DEFAULT (N'brak
opisu') FOR [Description]
GO

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

ALTER TABLE [dbo].[Diplomes] CHECK CONSTRAINT [FK_Diplomes_Products]
GO

ALTER TABLE [dbo].[Diplomes]  WITH CHECK ADD  CONSTRAINT [FK_Diplomes_Users] FOREIGN
KEY([UserID])
REFERENCES [dbo].[Students] ([UserID])
GO

ALTER TABLE [dbo].[Diplomes] CHECK CONSTRAINT [FK_Diplomes_Users]
GO

ALTER TABLE [dbo].[Diplomes]  WITH CHECK ADD  CONSTRAINT [Date<=GetDate()] CHECK
(([Date]<=getdate()))
```

```
GO

ALTER TABLE [dbo].[Diplomes] CHECK CONSTRAINT [Date<=GetDate()]
GO
```

Tabela Translations: Reprezentacja tłumaczeń

Klucz główny: TranslationID
Klucz obcy: InterpreterID [**Users**], ProductID [**Product**]

ID tłumaczenia:	[TranslationID]	int
ID tłumacza:	[InterpreterID]	int
ID produktu:	[ProductID]	int

```
CREATE TABLE [dbo].[Translations](
    [TranslationID] [int] IDENTITY(1,1) NOT NULL,
    [InterpreterID] [int] NOT NULL,
    [ProductID] [int] NOT NULL,
 CONSTRAINT [PK_Translations] PRIMARY KEY CLUSTERED
(
    [TranslationID] 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].[Translations]  WITH CHECK ADD  CONSTRAINT [FK_Translations_Products] FOREIGN
KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[Translations] CHECK CONSTRAINT [FK_Translations_Products]
GO

ALTER TABLE [dbo].[Translations]  WITH CHECK ADD  CONSTRAINT [FK_Translations_Users] FOREIGN
KEY([InterpreterID])
REFERENCES [dbo].[Interpreters] ([UserID])
GO

ALTER TABLE [dbo].[Translations] CHECK CONSTRAINT [FK_Translations_Users]
GO
```

Tabela Languages: Reprezentacja dostępne języki tłumaczeń
Klucz główny: LanguageID
Klucz obcy:

ID języka:	[LanguageID]	int
Nazwa:	[Name]	nvarchar(50)

```
CREATE TABLE [dbo].[Languages](
    [LanguageID] [int] NOT NULL,
    [Name] [nvarchar](50) 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
```

Tabela ClassRooms: Reprezentacja miejsc stacjonarnych spotkań

Klucz główny: PlaceID
Klucz obcy: AddressID [**Addresses**]

ID miejsca:	[PlaceID]	int
Numer pokoju:	[Room]	nvarchar(50)
Adres miejsca:	[AddressID]	int

```
CREATE TABLE [dbo].[ClassRooms](
    [PlaceID] [int] IDENTITY(1,1) NOT NULL,
    [Room] [nvarchar](50) NOT NULL,
    [AddressID] [int] NOT NULL,
 CONSTRAINT [PK_Place] PRIMARY KEY CLUSTERED
(
    [PlaceID] 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].[ClassRooms]  WITH CHECK ADD  CONSTRAINT [FK_ClassRooms_Addresses] FOREIGN
KEY([AddressID])
REFERENCES [dbo].[Addresses] ([AddressID])
GO

ALTER TABLE [dbo].[ClassRooms] CHECK CONSTRAINT [FK_ClassRooms_Addresses]
GO
```

Tabela Addresses: Reprezentacja adresów

Klucz główny: AddressID
Klucz obcy: CityID [Cities]

ID miejsca:	[AddressID]	int
Adres miejsca:	[Address]	nvarchar(50)
Kod-pocztowy	[Zip-code]	nvarchar(50)
ID Miasta:	[CityID]	int

```
CREATE TABLE [dbo].[Addresses](
    [AddressID] [int] IDENTITY(1,1) NOT NULL,
    [Address] [nvarchar](50) NOT NULL,
    [Zipcode] [nvarchar](50) NOT NULL,
    [CityID] [int] 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: Reprezentacja miast w bazie danych
Klucz główny: CityID
Klucz obcy: CountryID [Countries]

Id miasta:	[CityID]	int
Nazwa miasta:	[Name]	nvarchar(50)
Id państwa	[CountryID]	int

```
CREATE TABLE [dbo].[Cities](
    [CityID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) 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
```

Tabela Countries: Reprezentacja państw w bazie danych

Klucz główny: CountryID
Klucz obcy: -

Id państwa:	[CountryID]	int
Nazwa państwa:	[Name]	nvarchar(50)

```
CREATE TABLE [dbo].[Countries](
    [CountryID] [int] IDENTITY(1,1) NOT NULL,
    [Name] [nvarchar](50) 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
```

Tabela AccessToProducts: Reprezentacja dostępu do produktu

Klucz główny: (ProductID, UserID)
Klucz obcy: ProductID [**Products**], UserID [**Users**]

ID produktu	[ProductID]	int
ID użytkownika	[UserID]	int
Data rozpoczęcia	[StartDate]	datetime
Data zakończenia	[EndDate]	datetime
ID dostępu do produktu	[AccessToProductID]	int

Warunki integralności:

- Data rozpoczęcia nie może być później niż data zakończenia
- Data zakończenia nie może być wcześniej niż data rozpoczęcia

```
CREATE TABLE [dbo].[AccessToProducts](
    [ProductID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [StartDate] [datetime] NOT NULL,
    [EndDate] [datetime] NULL,
    [AccessToProductID] [int] IDENTITY(1,1) NOT NULL,
 CONSTRAINT [PK_AccessToProduct] PRIMARY KEY CLUSTERED
(
    [AccessToProductID] 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].[AccessToProducts] ADD  CONSTRAINT [DF_AccessToProduct_Tstamp]  DEFAULT
(getdate()) FOR [StartDate]
GO

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

ALTER TABLE [dbo].[AccessToProducts] CHECK CONSTRAINT [FK_AccessToProduct_Products]
GO

ALTER TABLE [dbo].[AccessToProducts]  WITH CHECK ADD  CONSTRAINT [FK_AccessToProduct_Users1] FOREIGN
KEY([UserID])
REFERENCES [dbo].[Users] ([UserID])
GO

ALTER TABLE [dbo].[AccessToProducts] CHECK CONSTRAINT [FK_AccessToProduct_Users1]
GO

ALTER TABLE [dbo].[AccessToProducts]  WITH CHECK ADD  CONSTRAINT [CK_AccessToProduct_Date] CHECK
(([StartDate]<[EndDate]))
GO

ALTER TABLE [dbo].[AccessToProducts] CHECK CONSTRAINT [CK_AccessToProduct_Date]
GO
```

Tabela AccessToMeetings: Reprezentacja dostępu do spotkania

Klucz główny: AccessToMeetingID
Klucz obcy: MeetingID [**StudiesMeetings**], UserID [**Users**]

ID dostępu do spotkania:	[AccessToMeetingID]	int
ID spotkania	[MeetingID]	int
ID użytkownika	[UserID]	int
Czas rozpoczęcia:	[StartDate]	datetime
EndDate:	[EndDate]	datetime

Warunki integralności:

- StartDate nie może być później niż EndDate
- EndDate nie może zaczać się wcześniej niż StartDate

```
CREATE TABLE [dbo].[AccessToMeetings](
    [AccessToMeetingID] [int] IDENTITY(1,1) NOT NULL,
    [MeetingID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [StartDate] [datetime] NOT NULL,
    [EndDate] [datetime] NULL,
 CONSTRAINT [PK_AccessToMeeting] PRIMARY KEY CLUSTERED
(
    [AccessToMeetingID] 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].[AccessToMeetings] ADD  CONSTRAINT [DF_AccessToMeeting_TStamp]  DEFAULT
(getdate()) FOR [StartDate]
GO

ALTER TABLE [dbo].[AccessToMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_AccessToMeeting_StudiesMeetings] FOREIGN KEY([MeetingID])
REFERENCES [dbo].[StudiesMeetings] ([StudyMeetingID])
GO

ALTER TABLE [dbo].[AccessToMeetings] CHECK CONSTRAINT [FK_AccessToMeeting_StudiesMeetings]
GO

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

ALTER TABLE [dbo].[AccessToMeetings] CHECK CONSTRAINT [FK_AccessToMeeting_Users]
GO

ALTER TABLE [dbo].[AccessToMeetings]  WITH CHECK ADD  CONSTRAINT [CK_AccessToMeeting_Date] CHECK
(([StartDate]<=[EndDate]))
GO

ALTER TABLE [dbo].[AccessToMeetings] CHECK CONSTRAINT [CK_AccessToMeeting_Date]
GO
```

Tabela ModuleAttendances: Reprezentacja zaliczeń modułów kursów przez kursantów
Klucz główny: ModuleID, UserID
Klucz obcy: ModuleID [**CourseModules**], UserID [**Students**]

ID modułu	[ModuleID]	int
ID użytkownika	[UserID]	int
Czas	[Tstamp]	datetime

Warunki integralności:

- Tstamp domyślnie jest ustawiony na czas wprowadzenia rekordu

```
CREATE TABLE [dbo].[ModuleAttendances](
    [CourseModuleID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [TStamp] [datetime] NOT NULL,
 CONSTRAINT [PK_ModuleAttendances] PRIMARY KEY CLUSTERED
(
    [CourseModuleID] 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].[ModuleAttendances] ADD  CONSTRAINT [DF_ModuleAttendances_TStamp]  DEFAULT
(getdate()) FOR [TStamp]
GO

ALTER TABLE [dbo].[ModuleAttendances]  WITH CHECK ADD  CONSTRAINT
[FK_ModuleAttendances_CourseModules] FOREIGN KEY([CourseModuleID])
REFERENCES [dbo].[CourseModules] ([CourseModuleID])
GO

ALTER TABLE [dbo].[ModuleAttendances] CHECK CONSTRAINT [FK_ModuleAttendances_CourseModules]
GO

ALTER TABLE [dbo].[ModuleAttendances]  WITH CHECK ADD  CONSTRAINT [FK_ModuleAttendances_Users]
FOREIGN KEY([UserID])
REFERENCES [dbo].[Students] ([UserID])
GO

ALTER TABLE [dbo].[ModuleAttendances] CHECK CONSTRAINT [FK_ModuleAttendances_Users]
GO
```

Tabela MeetingAttendances: Reprezentacja zaliczeń zjazdów studiów przez studentów
Klucz główny: MeetingID, UserID
Klucz obcy: MeetingID [**StudiesMeetings**], UserID [**Student**]

ID spotkania	[MeetingID]	int
ID użytkownika	[UserID]	int
Czas	[Tstamp]	datetime
Czy zajęcia były odrabiane:	[IsCatchedUp]	bit

Warunki integralności:

- Tstamp domyślnie jest ustawiony na czas wprowadzenia rekordu
- IsCatchedUp domyślnie jest ustawiony jako 0

```
CREATE TABLE [dbo].[MeetingAttendances](
    [MeetingID] [int] NOT NULL,
    [UserID] [int] NOT NULL,
    [Tstamp] [datetime] NOT NULL,
    [IsCatchedUp] [bit] NOT NULL,
 CONSTRAINT [PK_MeetingAttendances] PRIMARY KEY CLUSTERED
(
    [MeetingID] 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].[MeetingAttendances] ADD  CONSTRAINT [DF_MeetingAttendances_Tstamp]  DEFAULT
(getdate()) FOR [Tstamp]
GO

ALTER TABLE [dbo].[MeetingAttendances] ADD  CONSTRAINT [DF_MeetingAttendances_IsCatchedUp]  DEFAULT
((0)) FOR [IsCatchedUp]
GO

ALTER TABLE [dbo].[MeetingAttendances]  WITH CHECK ADD  CONSTRAINT
[FK_MeetingAttendance_StudiesMeetings] FOREIGN KEY([MeetingID])
REFERENCES [dbo].[StudiesMeetings] ([StudyMeetingID])
GO

ALTER TABLE [dbo].[MeetingAttendances] CHECK CONSTRAINT [FK_MeetingAttendance_StudiesMeetings]
GO

ALTER TABLE [dbo].[MeetingAttendances]  WITH CHECK ADD  CONSTRAINT [FK_MeetingAttendance_Users1]
FOREIGN KEY([UserID])
REFERENCES [dbo].[Students] ([UserID])
GO

ALTER TABLE [dbo].[MeetingAttendances] CHECK CONSTRAINT [FK_MeetingAttendance_Users1]
GO
```

Tabela Orders: Reprezentacja zamówienia (koszyka)

Klucz główny: OrderID
Klucz obcy: UserID [**Users**]

ID zamówienia	[OrderID]	int
ID użytkownika:	[UserID]	int
Data zamówienia:	[OrderDate]	datetime
Data płatności:	[PayDate]	datetime
Link do płatności:	[PayLink]	nvarchar(50)
Status płatności:	[IsSucced]	bit

Warunki integralności:

- Status płatności jest domyślnie ustawiony na 0
- Data zamówienia nie może być późniejsza niż data płatności
- Data płatności nie może być wcześniejsza niż data zamówienia

```
CREATE TABLE [dbo].[Orders](
    [OrderID] [int] IDENTITY(1,1) NOT NULL,
    [UserID] [int] NOT NULL,
    [OrderDate] [datetime] NOT NULL,
    [PayDate] [datetime] NULL,
    [PayLink] [nvarchar](50) NULL,
    [IsSucced] [bit] NOT 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] ADD  CONSTRAINT [DF_Orders_IsSucced]  DEFAULT ((0)) FOR [IsSucced]
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_Orders_Date] CHECK
(([OrderDate]<=[PayDate]))
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [CK_Orders_Date]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CK_Orders_Succed] CHECK  (([IsSucced] IS
NULL OR [PayDate] IS NOT NULL))
GO

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

Tabela OrderAccessToMeetings: Reprezentacja powiązania zamówienia z dostępem do spotkania
Klucz główny: AccessToMeetingID
Klucz obcy: AccessToMeetingID[**AccessToMeeting**], OrderID [Orders]

ID dostępu do spotkania	[AccessToMeetingID]	int
ID zamówienia	[OrderID]	int

```
CREATE TABLE [dbo].[OrderAccessToMeetings](
    [AccessToMeetingID] [int] IDENTITY(1,1) NOT NULL,
    [OrderID] [int] NOT NULL,
 CONSTRAINT [PK_OrderAccessToMeeting] PRIMARY KEY CLUSTERED
(
    [AccessToMeetingID] 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].[OrderAccessToMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_OrderAccessToMeeting_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderAccessToMeetings] CHECK CONSTRAINT [FK_OrderAccessToMeeting_Orders]
GO

ALTER TABLE [dbo].[OrderAccessToMeetings]  WITH CHECK ADD  CONSTRAINT
[FK_OrderAccessToMeeting_StudiesMeetings] FOREIGN KEY([AccessToMeetingID])
REFERENCES [dbo].[AccessToMeetings] ([AccessToMeetingID])
GO

ALTER TABLE [dbo].[OrderAccessToMeetings] CHECK CONSTRAINT [FK_OrderAccessToMeeting_StudiesMeetings]
GO
```

Tabela OrderAccessToProduct: Reprezentacja powiązania zamówienia z dostępem do produktu
Klucz główny: AccessToProductID
Klucz obcy: AccessToProductID[**AccessToProduct**], OrderID [Orders]

ID dostępu do produktu:	[AccessToProductID]	int
ID zamówienia:	[OrderID]	int

```
CREATE TABLE [dbo].[OrderAccessToProducts](
    [AccessToProductID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
 CONSTRAINT [PK_OrderAccessToProduct] PRIMARY KEY CLUSTERED
(
    [AccessToProductID] 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].[OrderAccessToProducts]  WITH CHECK ADD  CONSTRAINT
[FK_OrderAccessToProduct_AccessToProduct] FOREIGN KEY([AccessToProductID])
REFERENCES [dbo].[AccessToProducts] ([AccessToProductID])
GO

ALTER TABLE [dbo].[OrderAccessToProducts] CHECK CONSTRAINT [FK_OrderAccessToProduct_AccessToProduct]
GO

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

ALTER TABLE [dbo].[OrderAccessToProducts] CHECK CONSTRAINT [FK_OrderAccessToProduct_Orders]
GO
```

Tabela OrderEntryFeeAccessToProduct: Reprezentacja powiązania zapłaty zaliczki z dostępem do produktu

Klucz główny: AccessToProductID
Klucz obcy: AccessToProductID[**AccessToProduct**], OrderID [Orders]

ID dostępu do produktu:	[AccessToProductID]	int
ID zamówienia:	[OrderID]	int

```
CREATE TABLE [dbo].[OrderEntryFeeAccessToProducts](
    [AccessToProductID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
 CONSTRAINT [PK_OrderEntryFeeAccessToProduct] PRIMARY KEY CLUSTERED
(
    [AccessToProductID] 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].[OrderEntryFeeAccessToProducts]  WITH CHECK ADD  CONSTRAINT
[FK_OrderEntryFeeAccessToProduct_AccessToProduct] FOREIGN KEY([AccessToProductID])
REFERENCES [dbo].[AccessToProducts] ([AccessToProductID])
GO

ALTER TABLE [dbo].[OrderEntryFeeAccessToProducts] CHECK CONSTRAINT
[FK_OrderEntryFeeAccessToProduct_AccessToProduct]
GO

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

ALTER TABLE [dbo].[OrderEntryFeeAccessToProducts] CHECK CONSTRAINT
[FK_OrderEntryFeeAccessToProduct_Orders]
GO
```

Tabela OrderEntireProductDetails: Reprezentacja szczegółów zamówienia pełnego produktu w koszyku

Klucz główny: OrderDetailID
Klucz obcy: OrderID [**Orders**], ProductID **[Products]**

ID	[OrderDetailID]	int
Cena:	[Price]	money
ID produktu:	[ProductID]	int
ID zamówienia:	[OrderID]	int

Warunki integralności:

- Cena musi być większa lub równa 0

```
CREATE TABLE [dbo].[OrderEntireProductDetails](
    [OrderDetailID] [int] IDENTITY(1,1) NOT NULL,
    [Price] [money] NOT NULL,
    [ProductID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
 CONSTRAINT [PK_OrderEntireProductDetails] PRIMARY KEY CLUSTERED
(
    [OrderDetailID] 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].[OrderEntireProductDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderEntireProductDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderEntireProductDetails] CHECK CONSTRAINT [FK_OrderEntireProductDetails_Orders]
GO

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

ALTER TABLE [dbo].[OrderEntireProductDetails] CHECK CONSTRAINT
[FK_OrderEntireProductDetails_Products]
GO

ALTER TABLE [dbo].[OrderEntireProductDetails]  WITH CHECK ADD  CONSTRAINT
[CK_OrderEntireProductDetails_Price] CHECK  (([Price]>=(0)))
GO

ALTER TABLE [dbo].[OrderEntireProductDetails] CHECK CONSTRAINT [CK_OrderEntireProductDetails_Price]
GO
```

Tabela OrderEntryFeeDetails: Reprezentacja szczegółów zapłaty zaliczki w koszyku
Klucz główny: OrderDetailID
Klucz obcy: OrderID [**Orders**], ProductID **[Products]**

ID zaliczki:	[OrderDetailID]	int
Pełna kwota produktu:	[Price]	money
ID produktu:	[CourseID]	int
ID zamówienia:	[OrderID]	int
Kwota zaliczki:	[EntryPrice]	money

Warunki integralności:

- Kwota zaliczki musi być większa lub równa 0
- Kwota zaliczki musi być mniejsza od pełnej kwoty produktu
- Pełna kwota produktu nie może być mniejsza od 0

```
CREATE TABLE [dbo].[OrderEntryFeeDetails](
    [OrderDetailID] [int] IDENTITY(1,1) NOT NULL,
    [Price] [money] NOT NULL,
    [ProductID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
    [EntryPrice] [money] NOT NULL,
 CONSTRAINT [PK_OrderEntryFeeDetails] PRIMARY KEY CLUSTERED
(
    [OrderDetailID] 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].[OrderEntryFeeDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderEntryFeeDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails] CHECK CONSTRAINT [FK_OrderEntryFeeDetails_Orders]
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderEntryFeeDetails_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Courses] ([CourseID])
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails] CHECK CONSTRAINT [FK_OrderEntryFeeDetails_Products]
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails]  WITH CHECK ADD  CONSTRAINT [[EntryPrice]] >=0] CHECK
(([EntryPrice]>=(0)))
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails] CHECK CONSTRAINT [[EntryPrice]] >=0]
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails]  WITH CHECK ADD  CONSTRAINT [CK_OrderEntryFeeDetails_Entry]
CHECK  (([Price]>[EntryPrice]))
GO
```

```
ALTER TABLE [dbo].[OrderEntryFeeDetails] CHECK CONSTRAINT [CK_OrderEntryFeeDetails_Entry]
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails]  WITH CHECK ADD  CONSTRAINT [CK_OrderEntryFeeDetails_Price]
CHECK  (([Price]>=(0)))
GO

ALTER TABLE [dbo].[OrderEntryFeeDetails] CHECK CONSTRAINT [CK_OrderEntryFeeDetails_Price]
GO
```

Tabela SingleMeetings: Reprezentacja spotkań dostępnych jako osobne produkty
Klucz główny: ProductID
Klucz obcy: StudyMeetingID [StudiesMeetings]

ID produktu:	[ProductID]	int
ID spotkania studiów:	[StudyMeetingID]	int

```
CREATE TABLE [dbo].[SingleMeetings](
    [ProductID] [int] NOT NULL,
    [StudyMeetingID] [int] NOT NULL,
 CONSTRAINT [PK_SingleMeetings] 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].[SingleMeetings]  WITH CHECK ADD  CONSTRAINT [FK_SingleMeetings_Products] FOREIGN
KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[SingleMeetings] CHECK CONSTRAINT [FK_SingleMeetings_Products]
GO

ALTER TABLE [dbo].[SingleMeetings]  WITH CHECK ADD  CONSTRAINT [FK_SingleMeetings_StudiesMeetings]
FOREIGN KEY([StudyMeetingID])
REFERENCES [dbo].[StudiesMeetings] ([StudyMeetingID])
GO

ALTER TABLE [dbo].[SingleMeetings] CHECK CONSTRAINT [FK_SingleMeetings_StudiesMeetings]
GO
```

Tabela OrderSingleMeetingDetails: Reprezentacja szczegółów zapłaty za jedno spotkanie w ramach studiów

Klucz główny: OrderDetailID
Klucz obcy: OrderID [**Orders**], StudiesMeetingID **[StudiesMeetings]**

ID	[OrderDetailID]	int
Cena spotkania:	[Price]	money
ID spotkania:	[StudiesMeetingID]	int
ID zamówienia:	[OrderID]	int

Warunki integralności:

- Cena spotkania musi być większa lub równa 0

```
CREATE TABLE [dbo].[OrderSingleMeetingDetails](
    [OrderDetailID] [int] IDENTITY(1,1) NOT NULL,
    [Price] [money] NOT NULL,
    [MeetingID] [int] NOT NULL,
    [OrderID] [int] NOT NULL,
 CONSTRAINT [PK_OrderSingleMeetingDetails] PRIMARY KEY CLUSTERED
(
    [OrderDetailID] 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].[OrderSingleMeetingDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderSingleMeetingDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO

ALTER TABLE [dbo].[OrderSingleMeetingDetails] CHECK CONSTRAINT [FK_OrderSingleMeetingDetails_Orders]
GO

ALTER TABLE [dbo].[OrderSingleMeetingDetails]  WITH CHECK ADD  CONSTRAINT
[FK_OrderSingleMeetingDetails_StudiesMeetings] FOREIGN KEY([MeetingID])
REFERENCES [dbo].[StudiesMeetings] ([StudyMeetingID])
GO

ALTER TABLE [dbo].[OrderSingleMeetingDetails] CHECK CONSTRAINT
[FK_OrderSingleMeetingDetails_StudiesMeetings]
GO

ALTER TABLE [dbo].[OrderSingleMeetingDetails]  WITH CHECK ADD  CONSTRAINT
[CK_OrderSingleMeetingDetails_Price] CHECK  (([Price]>=(0)))
GO

ALTER TABLE [dbo].[OrderSingleMeetingDetails] CHECK CONSTRAINT [CK_OrderSingleMeetingDetails_Price]
GO
```

Widoki

1. **Lista wszystkich obecności spotkań i modułów z zapisanymi na nie użytkownikami z ich obecnością** (Jan)

```
CREATE VIEW [dbo].[AllAtendenceReport]
AS
SELECT        EU.Type, EU.ID, EU.[Sub-Type], EU.StartDate, U.UserID, U.Login,
U.Name, U.Surname, IIF(ATT.ID IS NULL, 0, 1) AS [Was present]
FROM          [u_ksobczyk].[dbo].[AllMeetingModulesEnrolUser] EU INNER JOIN
                           Users U ON U.UserID = EU.UserID LEFT JOIN
                           (SELECT       'CourseModule' AS AType, MA.CourseModuleID
AS ID, MA.UserID
                           FROM          ModuleAttendances MA
                           UNION
                           SELECT        'StudyMeeting' AS AType, MEA.MeetingID AS
ID, MEA.UserID
                           FROM          MeetingAttendances MEA) ATT ON ATT.UserID =
EU.UserID AND ATT.AType = EU.Type AND ATT.ID = EU.ID
WHERE EU.Type <> 'Webinar'
GO
```

2. **Wszystkie spotkania nachodzące na siebie czasowo dla użytkowników** (Kacper)

```
CREATE VIEW [dbo].[AllBilocationMeetings]
AS
SELECT        MAIN.UserID, MAIN.Type, MAIN.ID, MAIN.StartDate, MAIN.EndDate,
JOINED.Type AS OtherType, JOINED.ID AS OtherID, JOINED.StartDate AS OtherStartDate,
JOINED.EndDate AS OtherEndDate
FROM          dbo.AllMeetingModulesEnrolUser AS MAIN INNER JOIN
                          dbo.AllMeetingModulesEnrolUser AS JOINED ON MAIN.UserID
= JOINED.UserID AND (MAIN.Type <> JOINED.Type OR
                          MAIN.ID <> JOINED.ID)
WHERE         (MAIN.EndDate BETWEEN JOINED.StartDate AND JOINED.EndDate) AND
(JOINED.EndDate >= GETDATE())
GO
```

3. **Podsumowanie finansowe wszystkich produktów** (Jan)

```
CREATE VIEW [dbo].[AllFinanseReport]
AS
SELECT        P.ProductID, P.Name, P.Price AS [Current single price], PT.Name AS
Type, P.IsAvailable, ISNULL(FullReport.[Sum of price], 0) AS [Full Product Sum
Purchase], ISNULL(FeeReport.[Sum of price], 0) AS [Fee Product Sum Purchase],
                         ISNULL(MeetingReport.[Sum of price], 0) AS [Meeting
Product Sum Purchase], ISNULL(FullReport.[Sum of price], 0) + ISNULL(FeeReport.[Sum
of price], 0) + ISNULL(MeetingReport.[Sum of price], 0) AS [Total Amount]
FROM          dbo.Products AS P LEFT OUTER JOIN
                         dbo.FullProductFinanseOrderReport AS FullReport ON
FullReport.ProductID = P.ProductID LEFT OUTER JOIN
                         dbo.FeeProductFinanseOrderReport AS FeeReport ON
FeeReport.ProductID = P.ProductID LEFT OUTER JOIN
                         dbo.MeetingsProductFinanseOrderReport AS MeetingReport
ON MeetingReport.ProductID = P.ProductID INNER JOIN
                         dbo.ProductTypes AS PT ON PT.ProductTypeID =
P.ProductTypeID
GO
```

4. Lista wszystkich spotkań i modułów z zapisanymi na nie użytkownikami (Jan)

```
CREATE VIEW [dbo].[AllMeetingModulesEnrolUser]
AS
SELECT        U.UserID, 'Webinar' AS [Type], 'Webinar' AS [Sub-Type], W.WebinarID AS
ID, w.EventDate AS StartDate, DATEADD(minute, W.Duration, W.EventDate) AS EndDate
FROM          Users U INNER JOIN
                          AccessToProducts ATP ON ATP.UserID = U.UserID RIGHT JOIN
                          Webinars W ON W.WebinarID = ATP.ProductID
UNION
SELECT        U.UserID, 'StudyMeeting' AS [Type], MMT.Name AS [Sub-Type],
SM.StudyMeetingID AS ID, SM.StartDate AS StartDate, DATEADD(minute, SM.Duration,
SM.StartDate) AS EndDate
FROM          Users U INNER JOIN
                          AccessToMeetings ATM ON ATM.UserID = U.UserID RIGHT JOIN
                          StudiesMeetings SM ON SM.StudyMeetingID = ATM.MeetingID
INNER JOIN
                          ModuleMeetingTypes MMT ON MMT.ModuleMeetingTypeID =
SM.TypeID
UNION
SELECT        U.UserID, 'CourseModule' AS [Type], MMT.Name AS [Sub-Type],
M.CourseModuleID AS ID, ISNULL(M.Date, C.StartDate) AS StartDate, DATEADD(minute,
M.Duration, M.Date) AS EndDate
FROM          Users U INNER JOIN
                          AccessToProducts ATP ON ATP.UserID = U.UserID RIGHT JOIN
                          Courses C ON ATP.ProductID = C.CourseID INNER JOIN
                          CourseModules CM ON CM.CourseID = C.CourseID INNER JOIN
                          (SELECT       OSM.CourseModuleID, OSM.Date, OSM.Duration
                          FROM          OnlineSyncModules OSM
                          UNION
                          SELECT        SSM.CourseModuleID, SSM.Date, SSM.Duration
                          FROM          StationaryModules SSM
                          UNION
                          SELECT        OAM.CourseModuleID, NULL AS [Date], 0 AS
Duration
                          FROM          OnlineAsyncModules OAM) M ON
M.CourseModuleID = CM.CourseModuleID INNER JOIN
                          ModuleMeetingTypes MMT ON MMT.ModuleMeetingTypeID =
CM.Type
GO
```

5. Raport bilokacji

```
CREATE VIEW [dbo].[BillocationReport]
AS
SELECT        A.UserID, MAX(U.Login) AS Login, MAX(U.Name) AS Name, MAX(U.Surname)
AS Surname, COUNT(*) AS [Bilocation Number]
FROM          dbo.AllBilocationMeetings AS A INNER JOIN
                          dbo.Users AS U ON U.UserID = A.UserID
GROUP BY A.UserID
GO
```

6. Lista w pełni opłaconych spotkań przez użytkowników (Jan)

```
CREATE VIEW [dbo].[CompletePaidMeetingAccess]
AS
SELECT        OAM.AccessToMeetingID
FROM          dbo.Orders AS O INNER JOIN
                          dbo.OrderAccessToMeetings AS OAM ON O.OrderID =
OAM.OrderID
WHERE         (O.IsSucced = 1)
GO
```

7. Lista w pełni opłaconych produktów przez użytkowników (Jan)

```
CREATE VIEW [dbo].[CompletePaidProductAccess]
AS
SELECT        OAP.AccessToProductID
FROM          dbo.Orders AS O INNER JOIN
                          dbo.OrderAccessToProducts AS OAP ON O.OrderID =
OAP.OrderID
WHERE         (O.IsSucced = 1)
GO
```

8. Harmonogram kursu (Kacper)

```
CREATE VIEW [dbo].[CoursePlanReport]
AS
SELECT        C.CourseID, P.Name AS Course, P.IsAvailable, CM.Name, MMT.Name AS
Type, ISNULL(SM.Date, ISNULL(OSM.Date, C.StartDate)) AS Date, ISNULL(SM.Duration,
OSM.Duration) AS Duration, CAST(CR.Room AS nvarchar)
                          + ' ' + AD.Address + ' ' + AD.Zipcode AS Address,
ISNULL(OSM.ConnectionLink, OAM.ResourceLink) AS Link, SM.Limit
FROM          dbo.Products AS P INNER JOIN
                          dbo.Courses AS C ON C.CourseID = P.ProductID INNER JOIN
                          dbo.CourseModules AS CM ON CM.CourseID = C.CourseID
INNER JOIN
                          dbo.ModuleMeetingTypes AS MMT ON MMT.ModuleMeetingTypeID
= CM.Type LEFT OUTER JOIN
                          dbo.StationaryModules AS SM ON CM.CourseModuleID =
SM.CourseModuleID LEFT OUTER JOIN
                          dbo.OnlineSyncModules AS OSM ON CM.CourseModuleID =
OSM.CourseModuleID LEFT OUTER JOIN
                          dbo.OnlineAsyncModules AS OAM ON OAM.CourseModuleID =
OAM.CourseModuleID LEFT OUTER JOIN
                          dbo.ClassRooms AS CR ON CR.PlaceID = SM.PlaceID LEFT
OUTER JOIN
                          dbo.Addresses AS AD ON AD.AddressID = CR.AddressID
GO
```

9. Podsumowanie wyników studentów z kursów na platformie (Jan)

```
CREATE VIEW [dbo].[CourseSummary]
AS
SELECT        CM.CourseID, AAR.UserID, SUM(AAR.[Was present]) AS present,
```

```
COUNT(AAR.[Was present]) - SUM(AAR.[Was present]) AS unpresent, CAST(SUM(AAR.[Was
present]) / CAST(COUNT(AAR.[Was present]) as DECIMAL(10,2)) AS DECIMAL(10,2)) AS
Ratio, IIF((SUM(AAR.[Was present])
                        / COUNT(AAR.[Was present])) > 0.8, 1, 0) AS [Is Passed]
FROM          dbo.AllAtendenceReport AS AAR INNER JOIN
                        dbo.CourseModules AS CM ON CM.CourseModuleID = AAR.ID
WHERE         (AAR.Type = 'CourseModule')
GROUP BY CM.CourseID, AAR.UserID
GO
```

10. Raport dłużników (Kacper)

```
CREATE VIEW [dbo].[DebtReport]
AS
SELECT        UserID, MAX(Login) AS Login, MAX(Name) AS Name, MAX(Surname) AS
Surname, SUM(Debt) AS Debt
FROM          (SELECT       UserID, Login, Name, Surname, Debt
                        FROM          dbo.UnpaidWebinarAccess
                        UNION
                        SELECT        UserID, Login, Name, Surname, Debt
                        FROM          dbo.UnpaidcCouseAccess
                        UNION
                        SELECT        UserID, Login, Name, Surname, Debt
                        FROM          dbo.UnpaidcStudiesAccess
                        UNION
                        SELECT        UserID, Login, Name, Surname, Debt
                        FROM          dbo.UnpaidMeetingAccess) AS MQ
GROUP BY UserID
GO
```

11. Raport sumy zapłaconych zaliczek za kurs (Kacper)

```
CREATE VIEW [dbo].[FeeProductFinanseOrderReport]
AS
SELECT        P.ProductID, P.Name, PT.Name AS Type, SUM(OEPD.EntryPrice) AS [Sum of
price], P.IsAvailable
FROM          dbo.Products AS P INNER JOIN
                          dbo.OrderEntryFeeDetails AS OEPD ON P.ProductID =
OEPD.ProductID INNER JOIN
                          dbo.Orders AS O ON O.OrderID = OEPD.OrderID INNER JOIN
                          dbo.ProductTypes AS PT ON PT.ProductTypeID =
P.ProductTypeID
WHERE         (O.IsSucced = 1)
GROUP BY P.ProductID, P.Name, PT.Name, P.IsAvailable
GO
```

12. Raport sumy zapłaconej za pełny produkt (Jan)

```
CREATE VIEW [dbo].[FullProductFinanseOrderReport]
AS
SELECT        P.ProductID, P.Name, PT.Name AS Type, SUM(OEPD.Price) AS [Sum of
price], P.IsAvailable
FROM          dbo.Products AS P INNER JOIN
                          dbo.OrderEntireProductDetails AS OEPD ON P.ProductID =
OEPD.ProductID INNER JOIN
                          dbo.Orders AS O ON O.OrderID = OEPD.OrderID INNER JOIN
                          dbo.ProductTypes AS PT ON PT.ProductTypeID =
P.ProductTypeID
WHERE         (O.IsSucced = 1)
GROUP BY P.ProductID, P.Name, PT.Name, P.IsAvailable
GO
```

13. Lista spotkań z liczbą osób zapisanych na nie (Kacper)

```
CREATE VIEW [dbo].[MeetingsEnrollUsersReport]
AS
SELECT        Type, ID, MAX([Sub-Type]) AS SubType, COUNT(UserID) AS [Users
Enrolled]
FROM          dbo.AllMeetingModulesEnrolUser
WHERE         (StartDate > GETDATE())
GROUP BY ID, Type
GO
```

14. Suma finansowa zapłacona za każde spotkanie w ramach studiów (Kacper)

```
CREATE VIEW [dbo].[MeetingsProductFinanseOrderReport]
AS
SELECT        P.ProductID, P.Name, PT.Name AS Type, SUM(OEPD.Price) AS [Sum of
price], P.IsAvailable
FROM          dbo.Products AS P INNER JOIN
                          dbo.Subjects AS S ON S.StudiesID = P.ProductID INNER
JOIN
                          dbo.StudiesMeetings AS SM ON SM.SubjectID = S.SubjectID
INNER JOIN
                          dbo.OrderSingleMeetingDetails AS OEPD ON
SM.StudyMeetingID = OEPD.MeetingID INNER JOIN
                          dbo.Orders AS O ON O.OrderID = OEPD.OrderID INNER JOIN
                          dbo.ProductTypes AS PT ON PT.ProductTypeID =
P.ProductTypeID
WHERE         (O.IsSucced = 1)
GROUP BY P.ProductID, P.Name, PT.Name, P.IsAvailable
GO
```

15. Procent obecności na każdym spotkaniu lub module (Jan)

```
CREATE VIEW [dbo].[ProcentAttendenceReport]
AS
SELECT        Type, ID, COUNT(*) AS [All students], SUM([Was present]) AS [Was
present], CAST(SUM([Was present]) * 100 / CAST(COUNT(*) AS float) AS decimal(10, 2))
AS Procent
FROM          dbo.AllAtendenceReport AS AAE
WHERE         (StartDate < GETDATE())
GROUP BY Type, ID
GO
```

16. Obecność każdego studenta w ramach studiów (Kacper)

```
CREATE VIEW [dbo].[StudiesAtendenceReport]
AS
SELECT        s.StudiesID, AAR.UserID, SUM(AAR.[Was present]) AS present,
COUNT(AAR.[Was present]) - SUM(AAR.[Was present]) AS unpresent, CAST(SUM(AAR.[Was
present]) / CAST(COUNT(AAR.[Was present]) AS float) AS DECIMAL(10, 2))
                          AS Ratio
FROM          dbo.AllAtendenceReport AS AAR INNER JOIN
                          dbo.StudiesMeetings AS sm ON sm.StudyMeetingID = AAR.ID
INNER JOIN
                          dbo.Subjects AS s ON s.SubjectID = sm.SubjectID
WHERE         (AAR.Type = 'StudyMeeting')
GROUP BY s.StudiesID, AAR.UserID
GO
```

17. **Harmonogram planu studiów** (Jan)

```
CREATE VIEW [dbo].[StudiesPlanReport]
AS
SELECT        S.StudyID, P.Name AS Study, P.IsAvailable, SB.Name AS Expr1,
SM.StudyMeetingID, MMT.Name AS Type, SM.StartDate, SM.Duration, ISNULL(SSI.Limit,
SM.Limit) AS Limit, CAST(CR.Room AS nvarchar)
                          + ' ' + Ad.Address + ' ' + Ad.Zipcode AS Address,
OM.ConnectionLink
FROM          dbo.Products AS P INNER JOIN
                          dbo.Studies AS S ON P.ProductID = S.StudyID INNER JOIN
                          dbo.Subjects AS SB ON S.StudyID = SB.StudiesID INNER
JOIN
                          dbo.StudiesMeetings AS SM ON SB.SubjectID = SM.SubjectID
INNER JOIN
                          dbo.ModuleMeetingTypes AS MMT ON MMT.ModuleMeetingTypeID
= SM.TypeID LEFT OUTER JOIN
                          dbo.StationaryMeetings AS STM ON STM.StudyMeetingID =
SM.StudyMeetingID LEFT OUTER JOIN
                          dbo.OnlineMeetings AS OM ON SM.StudyMeetingID =
OM.StudyMeetingID LEFT OUTER JOIN
                          dbo.ClassRooms AS CR ON CR.PlaceID = STM.PlaceID LEFT
OUTER JOIN
                          dbo.Addresses AS Ad ON CR.AddressID = Ad.AddressID LEFT
OUTER JOIN
                          dbo.SingleMeetings AS SI ON SI.StudyMeetingID =
SM.StudyMeetingID LEFT OUTER JOIN
                          dbo.Studies AS SSI ON SSI.StudyID = SI.ProductID
GO
```

18. Podsumowanie studiów (Kacper)

```
CREATE VIEW [dbo].[StudySummary]
AS
SELECT TOP (1000) [StudiesID]
       ,[UserID]
       ,[present]
       ,[unpresent]
       ,[Ratio]
      ,(select count(*) From GetPassedStudiesIntershipsForStudent([StudiesID],
[UserID])) [Completed Intership]
      ,(select max(Note) From GetExamsForStudent ([StudiesID], [UserID])) [Exam
Note]
      , IIF(Ratio >= 0.8 and dbo.IsExamPassed([StudiesID], [UserID]) = 1 and
dbo.IsIntershipCompleted([StudiesID], [UserID]) = 1, 1, 0) as [Is Study Pass]
  FROM [u_ksobczyk].[dbo].[StudiesAtendenceReport]
GO
```

19. **Nieopłacone kursy** (Jan)

```
CREATE VIEW [dbo].[UnpaidcCouseAccess]
AS
SELECT        U.UserID, U.Login, U.Name, U.Surname, P.Price -
dbo.GetFeePaidForAccessToCourse(A.AccessToProductID) AS Debt
FROM          dbo.Courses AS C INNER JOIN
                          dbo.AccessToProducts AS A ON A.ProductID = C.CourseID
INNER JOIN
                          dbo.Users AS U ON U.UserID = A.UserID INNER JOIN
                          dbo.Products AS P ON P.ProductID = A.ProductID LEFT
OUTER JOIN
                          (SELECT       AccessToProductID
                          FROM          dbo.CompletePaidProductAccess) AS AO ON
AO.AccessToProductID = A.AccessToProductID
WHERE         (AO.AccessToProductID IS NULL) AND (C.StartDate < GETDATE()) AND
(C.StartDate BETWEEN A.StartDate AND A.EndDate)
GO
```

20. **Nieopłacone Studia** (Kacper)

```
CREATE VIEW [dbo].[UnpaidcStudiesAccess]
AS
SELECT        U.UserID, U.Login, U.Name, U.Surname, P.Price AS Debt
FROM          dbo.Studies AS S INNER JOIN
                          dbo.AccessToProducts AS A ON A.ProductID = S.StudyID
INNER JOIN
                          dbo.Users AS U ON U.UserID = A.UserID INNER JOIN
                          dbo.Products AS P ON P.ProductID = A.ProductID LEFT
OUTER JOIN
                          (SELECT       AccessToProductID
                          FROM          dbo.CompletePaidProductAccess) AS AO ON
AO.AccessToProductID = A.AccessToProductID
WHERE         (dbo.GetStudiesStartDate(S.StudyID) < GETDATE()) AND
(AO.AccessToProductID IS NULL) AND (dbo.GetStudiesStartDate(S.StudyID) BETWEEN
A.StartDate AND A.EndDate)
GO
```

21. **Nieopłacone spotkania** (Kacper)

```
CREATE VIEW [dbo].[UnpaidMeetingAccess]
AS
SELECT        U.UserID, U.Login, U.Name, U.Surname,
dbo.GetSingleMeetingPriceForUser(U.UserID, Sm.StudyMeetingID) AS Debt
FROM          dbo.StudiesMeetings AS Sm INNER JOIN
                           dbo.AccessToMeetings AS A ON A.MeetingID =
Sm.StudyMeetingID INNER JOIN
                           dbo.Users AS U ON U.UserID = A.UserID LEFT OUTER JOIN
                           (SELECT       AccessToMeetingID
                           FROM          dbo.CompletePaidMeetingAccess) AS AO ON
AO.AccessToMeetingID = A.AccessToMeetingID
WHERE         (Sm.StartDate < GETDATE()) AND (Sm.StartDate BETWEEN A.StartDate AND
A.EndDate) AND (A.AccessToMeetingID IS NULL)
GO
```

22. **Nieopłacone Webinary** (Jan)

```
CREATE VIEW [dbo].[UnpaidWebinarAccess]
AS
SELECT        U.UserID, U.Login, U.Name, U.Surname, P.Price AS Debt
FROM          dbo.Webinars AS W INNER JOIN
                           dbo.AccessToProducts AS A ON A.ProductID = W.WebinarID
INNER JOIN
                           dbo.Users AS U ON U.UserID = A.UserID INNER JOIN
                           dbo.Products AS P ON P.ProductID = A.ProductID LEFT
OUTER JOIN
                           (SELECT       AccessToProductID
                           FROM          dbo.CompletePaidProductAccess) AS AO ON
AO.AccessToProductID = A.AccessToProductID
WHERE         (W.EventDate < GETDATE()) AND (W.EventDate BETWEEN A.StartDate AND
A.EndDate) AND (AO.AccessToProductID IS NULL)
GO
```

Procedury

4. **Procedura aktualizacji danych użytkownika.** (Kacper)

```
CREATE PROCEDURE [dbo].[UpdateUserDetails]
       @UserID INT,
       @Login VARCHAR(50),
       @Password VARCHAR(50),
       @Name VARCHAR(50),
       @Surname VARCHAR(50),
       @Email VARCHAR(50),
       @Phone VARCHAR(50
)
AS
BEGIN
-- Checking if a user with the given UserID exists
IF EXISTS(SELECT 1 FROM Users WHERE UserID = @UserID)
BEGIN
-- Updating user details
UPDATE Users
SET Login = @Login,
Password = @Password,
Name = @Name,
Surname = @Surname,
[E-mail] = @Email,
Phone = @Phone
WHERE UserID = @UserID;
END
ELSE
BEGIN
-- Handle the case where the user does not exist
RAISERROR('User with the specified UserID does not exist.', 16, 1);
END
END
```

2. **Procedura dodania produktu.** (Kacper)

```
CREATE PROCEDURE [dbo].[AddProduct]
	@Name VARCHAR(50),
	@LecturerID INT,
	@Price MONEY,
	@ProductTypeID INT,
	@IsAvailable BIT
AS
BEGIN
	INSERT INTO Products (Name, LecturerID, Price, ProductTypeID, IsAvailable)
	VALUES (@Name, @LecturerID, @Price, @ProductTypeID, @IsAvailable)
END
```

3. **Procedura nadania użytkownikowi uprawnień do produktu.** (Kacper)

```
CREATE PROCEDURE GrantAccessToProduct
	@UserID INT,
	@ProductID INT
AS
BEGIN
	INSERT INTO AccessToProducts (UserID, ProductID, StartDate, EndDate)
	VALUES (@UserID, @ProductID, GETDATE(), DATEADD(DAY, 30, GETDATE()))
END
```

4. **Procedura zmiany przypisanej sali.** (Kacper)

```
CREATE PROCEDURE [dbo].[UpdateRoomForAddress]
	@Room NVARCHAR(50),
	@AddressID INT = NULL,
	@AddressName NVARCHAR(50) = NULL
AS
BEGIN
	IF @AddressID IS NOT NULL
	BEGIN
	UPDATE ClassRooms
	SET Room = @Room
	WHERE AddressID = @AddressID;
	END
	ELSE IF @AddressName IS NOT NULL
	BEGIN
	UPDATE ClassRooms
	SET Room = @Room
	FROM ClassRooms
	INNER JOIN Addresses ON ClassRooms.AddressID = Addresses.AddressID
	WHERE Addresses.Address = @AddressName;
	END
	ELSE
	BEGIN
	RAISERROR('You must provide either an AddressID or an AddressName.', 16, 1);
	END
END
```

6. Dodanie zaliczki do koszyka (Jan)

```
Create PROCEDURE [dbo].[AddCourseFeeToOrder]
    @OrderID int,
    @CourseID int
AS
BEGIN
    DECLARE @FeePrice money
    DECLARE @Price money
    SELECT @FeePrice = C.EntryFee, @Price = p.Price From Courses C INNER JOIN
Products P on c.CourseID = p.ProductID Where C.CourseID = @CourseID
    INSERT INTO OrderEntryFeeDetails (Price, ProductID, OrderID, EntryPrice) VALUES
(@Price, @CourseID, @OrderID, @FeePrice)
END;
GO
```

7. Dodanie kursu do koszyka (Jan)

```
CREATE PROCEDURE [dbo].[AddCourseToOrder]
    @OrderID int,
    @CourseID int
AS
BEGIN
    DECLARE @UserID int
    SELECT @UserID = O.UserID FROM Orders O Where O.OrderID = @OrderID
    DECLARE @AccessToFee int
    SELECT @AccessToFee = a.AccessToProductID FROM AccessToProducts A Where A.UserID
= @UserID AND A.ProductID = @CourseID and GETDATE() Between A.StartDate and
a.EndDate
    DECLARE @ToPay money

    SELECT @ToPay = OEFD.Price - OEFD.EntryPrice FROM AccessToProducts ATP
    INNER JOIN OrderEntryFeeAccessToProducts OEF ON OEF.AccessToProductID =
ATP.AccessToProductID
    Inner JOIN Orders O ON O.OrderID = OEF.OrderID
    INNER JOIN OrderEntryFeeDetails OEFD ON OEFD.OrderID = O.OrderID
    Where OEFD.ProductID = @CourseID AND O.IsSucced = 1 AND ATP.AccessToProductID =
@AccessToFee

    IF @ToPay IS NULL
        SELECT @ToPay = p.Price FROM Products P where P.ProductID = @CourseID

    INSERT INTO OrderEntireProductDetails (Price, ProductID, OrderID) VALUES
(@ToPay, @CourseID, @OrderID)
END;
GO
```

8. Dodanie spotkania do koszyka (Jan)

```
CREATE PROCEDURE [dbo].[AddMeetingToOrder]
	@OrderID int,
	@MeetingID int
AS
BEGIN
	DECLARE @StudiesID int
	SELECT @StudiesID = S.StudiesID FROM StudiesMeetings SM INNER JOIN Subjects S
ON S.SubjectID = SM.SubjectID Where @MeetingID = SM.StudyMeetingID
	DECLARE @UserID int
	SELECT @UserID = O.UserID FROM Orders O WHERE O.OrderID = @OrderID

	DECLARE @Price int
	IF EXISTS (SELECT * FROM AccessToProducts ATP WHERE ATP.UserID = @UserID AND
ATP.ProductID = @StudiesID AND GETDATE() BETWEEN ATP.StartDate AND ATP.EndDate)
	BEGIN
		SELECT @Price = SM.Price FROM StudiesMeetings SM WHERE
SM.StudyMeetingID = @MeetingID
		INSERT INTO OrderSingleMeetingDetails(Price, MeetingID, OrderID)
VALUES (@Price, @MeetingID, @OrderID)
	END
	ELSE
	BEGIN
		DECLARE @ProductID int
		SELECT @Price = P.Price, @ProductID = P.ProductID FROM SingleMeetings
SM INNER JOIN Products P ON SM.ProductID = P.ProductID WHERE SM.StudyMeetingID =
@MeetingID and p.IsAvailable = 1
		IF @ProductID is NULL
			THROW 51000, N'You cant bay this single meeting', 1;
		INSERT INTO OrderEntireProductDetails (Price, ProductID, OrderID)
VALUES (@Price, @ProductID, @OrderID)
	END




END;
GO
```

9. Dodanie ogólnego produktu do koszyka (Kacper)

```
Create PROCEDURE [dbo].[AddProductToOrder]
    @OrderID int,
    @ProductID int
AS
BEGIN
```

```
    DECLARE @TypeName nvarchar(50)
    SELECT @TypeName = Pt.Name FROM Products P INNER JOIN ProductTypes PT ON
PT.ProductTypeID = P.ProductTypeID Where P.ProductID = @ProductID;

    IF @TypeName = 'Webinar'
        Exec [dbo].[AddWebinarToOrder] @OrderID = @OrderID, @WebinarID = @ProductID
    ELSE IF @TypeName = 'Course'
        Exec [dbo].[AddCourseToOrder] @OrderID = @OrderID, @CourseID = @ProductID
    ELSE IF @TypeName = 'Studies'
        Exec [dbo].[AddStudiesToOrder] @OrderID = @OrderID, @StudiesID = @ProductID
    ELSE IF @TypeName = 'Single Meeting'
    BEGIN
        DECLARE @MeetingID int
        SELECT @MeetingID = SM.StudyMeetingID FROM SingleMeetings sm WHERE
SM.ProductID = @ProductID
        IF @MeetingID IS NULL
               THROW 51000, N'You cant buy this meeting', 1;
        Exec [dbo].[AddMeetingToOrder] @OrderID = @OrderID, @MeetingID = @MeetingID
    END
    ELSE
        THROW 51000, N'Type cant be founded', 1;
END;
GO
```

11. Dodanie studiów do koszyka (Jan)

```
Create PROCEDURE [dbo].[AddStudiesToOrder]
    @OrderID int,
    @StudiesID int
AS
BEGIN
    DECLARE @Price money
    SELECT @Price = p.Price FROM Products P where P.ProductID = @StudiesID

    INSERT INTO OrderEntireProductDetails (Price, ProductID, OrderID) VALUES
(@Price, @StudiesID, @OrderID)
END;
GO
```

12. Dodanie webinaru do koszyka (Kacper)

```
Create PROCEDURE [dbo].[AddWebinarToOrder]
    @OrderID int,
    @WebinarID int
AS
BEGIN
    DECLARE @Price money
    SELECT @Price = p.Price FROM Products P where P.ProductID = @WebinarID

    INSERT INTO OrderEntireProductDetails (Price, ProductID, OrderID) VALUES
(@Price, @WebinarID, @OrderID)
END;
GO
```

13. Zatwierdzenie zamówienia (Kacper)

```
CREATE PROCEDURE [dbo].[CompleteOrder]
    @OrderID int
AS
BEGIN
    IF EXISTS (SELECT * FROM Orders O WHERE O.OrderID = @OrderID AND O.IsSucced = 1)
        THROW 51000, N'Order was already booked', 1;

    Update Orders SET IsSucced = 1, PayDate = GETDATE() Where OrderID = @OrderID

    DECLARE @UserID int
    SELECT @UserID = UserID FROM Orders Where OrderID = @OrderID

    --Give access to meetings
    DECLARE @ATMInserted table([ID] int)
    INSERT INTO AccessToMeetings (MeetingID, UserID, StartDate, EndDate) OUTPUT
inserted.AccessToMeetingID into @ATMInserted SELECT MeetingID, @UserID, GETDATE(),
DATEADD(YEAR, 1, GETDATE()) FROM OrderSingleMeetingDetails OSMD Where OSMD.OrderID =
@OrderID
    INSERT INTO OrderAccessToMeetings (AccessToMeetingID, OrderID) SELECT ID,
@OrderID FROM @ATMInserted

    --acknowledge fee
    DECLARE @ATMInsertedFee table([ID] int)
    INSERT INTO AccessToProducts (ProductID, UserID, StartDate, EndDate) OUTPUT
inserted.AccessToProductID into @ATMInsertedFee SELECT OEFD.ProductID, @UserID,
GETDATE(), DATEADD(day, -3, C.StartDate) FROM OrderEntryFeeDetails OEFD INNER JOIN
Courses C On OEFD.ProductID = c.CourseID Where OEFD.OrderID = @OrderID
    INSERT INTO OrderEntryFeeAccessToProducts (AccessToProductID, OrderID) SELECT
ID, @OrderID FROM @ATMInsertedFee

    --Give access to Webinars
    DECLARE @ATPInsertedWebinar table([ID] int)
    INSERT INTO AccessToProducts (ProductID, UserID, StartDate, EndDate) OUTPUT
inserted.AccessToProductID into @ATPInsertedWebinar SELECT OEFD.ProductID, @UserID,
GETDATE(), DATEADD(MONTH, 1, GETDATE()) FROM OrderEntireProductDetails OEFD INNER
JOIN Webinars W On OEFD.ProductID = W.WebinarID Where OEFD.OrderID = @OrderID
    INSERT INTO OrderAccessToProducts (AccessToProductID, OrderID) SELECT ID,
@OrderID FROM @ATPInsertedWebinar

    --Give access to other Products
    DECLARE @ATPInsertedOtherProducts table([ID] int)
    INSERT INTO AccessToProducts (ProductID, UserID, StartDate, EndDate) OUTPUT
inserted.AccessToProductID into @ATPInsertedOtherProducts SELECT OEFD.ProductID,
@UserID, GETDATE(), DATEADD(YEAR, 1, GETDATE()) FROM OrderEntireProductDetails OEFD
LEFT JOIN Webinars W On OEFD.ProductID = W.WebinarID Where OEFD.OrderID = @OrderID
AND WebinarID is null
    INSERT INTO OrderAccessToProducts (AccessToProductID, OrderID) SELECT ID,
@OrderID FROM @ATPInsertedOtherProducts

    --Give access to meeting for single meeting purchase
    DECLARE @ATPInsertedSingleMeeting table([ID] int)
```

```
    INSERT INTO AccessToMeetings (MeetingID, UserID, StartDate, EndDate) OUTPUT
inserted.AccessToMeetingID into @ATPInsertedSingleMeeting SELECT SM.StudyMeetingID,
@UserID, GETDATE(), DATEADD(YEAR, 1, GETDATE()) FROM OrderEntireProductDetails OEFD
INNER JOIN SingleMeetings sm on SM.ProductID = OEFD.ProductID WHERE OEFD.OrderID =
@OrderID
    INSERT INTO OrderAccessToMeetings (AccessToMeetingID, OrderID) SELECT ID,
@OrderID FROM @ATPInsertedSingleMeeting
END;
GO
```

14. Stworzenie zamówienia (Jan)

```
Create PROCEDURE [dbo].[CreateOrder]
    @UserID int
AS
BEGIN
    INSERT INTO Orders (UserID, OrderDate, PayLink) VALUES (@UserID, GETDATE(),
'LINK')
    RETURN SCOPE_IDENTITY()
END;
GO
```

15. Dostań nowy adres dla użytkownika (Kacper)

```
CREATE PROCEDURE [dbo].[GetAddress]
(
    @Address nvarchar(50),
    @Zipcode nvarchar(50),
    @City nvarchar(50),
    @Country nvarchar(50)
)
AS
BEGIN
    DECLARE @CountryID INT;
    SET @CountryID = (SELECT [CountryID] FROM Countries Where Name = @Country)
    if @CountryID is null
    BEGIN;
        THROW 51000, N'Country cant be founded', 1;
    END

    DECLARE @CityID INT;
    SET @CityID = (SELECT CityID FROM Cities Where Name = @City and CountryID =
@CountryID)
    if @CityID is null
    BEGIN;
        INSERT INTO [dbo].[Cities]
              ([Name]
              ,[CountryID])
        VALUES(
              @City
              ,@CountryID)
        SET @CityID = SCOPE_IDENTITY()
    END

    DECLARE @AddressID INT;
    SET @AddressID = (SELECT CityID FROM Addresses A Where A.Address  = @Address and
A.Zipcode = @Zipcode and A.CityID = @CityID)
    if @AddressID is null
    BEGIN;
        INSERT INTO [dbo].[Addresses]
               ([Address]
              ,[Zipcode]
              ,[CityID])
        VALUES
              (@Address
              ,@Zipcode
              ,@CityID)
        SET @AddressID = SCOPE_IDENTITY()
    END
    RETURN @AddressID
END

GO
```

16. Dodaj nowego dyrektora (Jan)

```
CREATE PROCEDURE [dbo].[InsertHeadmaster]
    @Login varchar(50),
    @Password varchar(50),
    @Name varchar(50),
    @Surname varchar(50),
    @mail varchar(50),
    @phone varchar(50)
AS
BEGIN
    DECLARE @UserID TABLE([UserID] int);
    DECLARE @HeadmasterTypeID INT;
    SET @HeadmasterTypeID = (SELECT [AccountTypeID] FROM AccountTypes Where Name =
'Headmaster')

    IF @HeadmasterTypeID is null
    BEGIN;
        THROW 51000, N'Type cant be founded', 1;
    END

    INSERT INTO [dbo].[Users]
                    ([Login]
                    ,[Password]
                    ,[Name]
                    ,[Surname]
                    ,[E-mail]
                    ,[Phone])
        OUTPUT inserted.UserID into @UserID
         VALUES
                    (@Login
                    ,@Password
                    ,@Name
                    ,@Surname
                    ,@mail
                    ,@phone)
    INSERT INTO [dbo].[UserAccountTypes]
              ([UserID]
              ,[AccountTypeID]
              ,[StartDate]
              ,[ExpireDate])
       VALUES
              ((SELECT TOP 1 UserID FROM @UserID)
              ,@HeadmasterTypeID
              ,GETDATE()
              ,datefromparts(3000, 1, 1))
END;
GO
```

17. Dodaj nowego tłumacza (Kacper)

```
Create PROCEDURE [dbo].[InsertInterpreter]
```

```
    @Login varchar(50),
    @Password varchar(50),
    @Name varchar(50),
    @Surname varchar(50),
    @mail varchar(50),
    @phone varchar(50),
    @Language nvarchar(50)
AS
BEGIN
    DECLARE @UserID int;
    DECLARE @InterpreterTypeID INT;
    DECLARE @Languageid int;

    SET @InterpreterTypeID = (SELECT [AccountTypeID] FROM AccountTypes Where Name =
'Interpreter')

    IF @InterpreterTypeID is null
    BEGIN;
        THROW 51000, N'Type cant be founded', 1;
    END

    SET @LanguageID = (SELECT LanguageID FROM Languages Where Name = @Language)

    IF @LanguageID is null
    BEGIN;
        THROW 51000, N'Language cant be found', 1;
    END

    INSERT INTO [dbo].[Users]
                    ([Login]
                    ,[Password]
                    ,[Name]
                    ,[Surname]
                    ,[E-mail]
                    ,[Phone])
         VALUES
                    (@Login
                    ,@Password
                    ,@Name
                    ,@Surname
                    ,@mail
                    ,@phone)

    SET @UserID = SCOPE_IDENTITY()
    INSERT INTO [dbo].[UserAccountTypes]
              ([UserID]
              ,[AccountTypeID]
              ,[StartDate]
              ,[ExpireDate])
       VALUES
              (@UserID
              ,@InterpreterTypeID
              ,GETDATE()
```

```
              ,datefromparts(3000, 1, 1))


    INSERT INTO [dbo].[Interpreters]
              ([UserID]
              ,[LanguageID])
       VALUES
              (@UserID
              ,@LanguageID)
END;
GO
```

18. Dodaj nowego wykładowcę (Jan)

```
CREATE PROCEDURE [dbo].[InsertLecturer]
    @Login varchar(50),
    @Password varchar(50),
    @Name varchar(50),
    @Surname varchar(50),
    @mail varchar(50),
    @phone varchar(50),
    @RoomID int
AS
BEGIN
    DECLARE @UserID int;
    DECLARE @LecturerTypeID INT;
    SET @LecturerTypeID = (SELECT [AccountTypeID] FROM AccountTypes Where Name =
'Lecturer')

    IF @LecturerTypeID is null
    BEGIN;
        THROW 51000, N'Type cant be founded', 1;
    END

    IF (SELECT cr.PlaceID  FROM ClassRooms cr WHERE cr.PlaceID = @RoomID) is null
    BEGIN;
        THROW 51000, N'Room not exist in database', 1;
    END

    INSERT INTO [dbo].[Users]
                    ([Login]
                    ,[Password]
                    ,[Name]
                    ,[Surname]
                    ,[E-mail]
                    ,[Phone])
         VALUES
                    (@Login
                    ,@Password
                    ,@Name
                    ,@Surname
                    ,@mail
                    ,@phone)

    SET @UserID = SCOPE_IDENTITY()
    INSERT INTO [dbo].[UserAccountTypes]
              ([UserID]
              ,[AccountTypeID]
              ,[StartDate]
              ,[ExpireDate])
       VALUES
              (@UserID
              ,@LecturerTypeID
              ,GETDATE()
              ,datefromparts(3000, 1, 1))
```

```
    INSERT INTO [dbo].[Teachers]
               ([UserID]
               ,[RoomID])
          VALUES
               (@UserID
               ,@RoomID)
END;
GO
```

19. Dodaj nowego studenta (Kacper)

```
Create PROCEDURE [dbo].[InsertStudent]
    @Login varchar(50),
    @Password varchar(50),
    @Name varchar(50),
    @Surname varchar(50),
    @mail varchar(50),
    @phone varchar(50),
    @AddressID int,
    @Birthdate Date
AS
BEGIN
    DECLARE @UserID int;
    DECLARE @StudentTypeID INT;
    SET @StudentTypeID = (SELECT [AccountTypeID] FROM AccountTypes Where Name =
'Student')

    IF @StudentTypeID is null
    BEGIN;
        THROW 51000, N'Type cant be founded', 1;
    END

    IF (SELECT a.AddressID  FROM Addresses a WHERE a.AddressID = @AddressID) is null
    BEGIN;
        THROW 51000, N'Room not exist in database', 1;
    END

    INSERT INTO [dbo].[Users]
                    ([Login]
                    ,[Password]
                    ,[Name]
                    ,[Surname]
                    ,[E-mail]
                    ,[Phone])
         VALUES
                    (@Login
                    ,@Password
                    ,@Name
                    ,@Surname
                    ,@mail
                    ,@phone)

    SET @UserID = SCOPE_IDENTITY()
    INSERT INTO [dbo].[UserAccountTypes]
              ([UserID]
              ,[AccountTypeID]
              ,[StartDate]
              ,[ExpireDate])
       VALUES
              (@UserID
              ,@StudentTypeID
              ,GETDATE()
```

```
            ,datefromparts(3000, 1, 1))

    INSERT INTO [dbo].[Students]
            ([UserID]
            ,[Birthdate]
            ,[AddressID])
       VALUES
            (@UserID,
            @Birthdate
            ,@AddressID)
END;
GO
```

20. Ustaw wszystkich studentów jako obecnych na spotkaniu (Jan)

```
CREATE PROCEDURE [dbo].[SetAllUserPresentOnMeeting]
    @MeetingID int
AS
BEGIN
    INSERT INTO MeetingAttendances(MeetingID, UserID) SELECT AM.ID, AM.UserID FROM
AllMeetingModulesEnrolUser AM Where AM.ID = @MeetingID and AM.Type = 'StudyMeeting'
END;
GO
```

21. Ustaw wszystkie studentów jako obecnych w module (Kacper)

```
CREATE PROCEDURE [dbo].[SetAllUserPresentOnModule]
    @ModuleID int
AS
BEGIN
    INSERT INTO ModuleAttendances (CourseModuleID, UserID) SELECT AM.ID, AM.UserID
FROM AllMeetingModulesEnrolUser AM Where AM.ID = @ModuleID and AM.Type =
'CourseModule'
END;
GO
```

22. Ustaw użytkownika jako obecnego na module (Kacper)

```
Create PROCEDURE [dbo].[SetUserPresentOnModule]
    @ModuleID int,
    @UserID int
AS
BEGIN
    IF EXISTS (SELECT * FROM ModuleAttendances MA WHERE MA.UserID = @UserID AND
MA.CourseModuleID = @ModuleID)
        RETURN;
    INSERT INTO ModuleAttendances (CourseModuleID, UserID) VALUES (@ModuleID,
@UserID)
END;
GO
```

23. Ustaw użytkownika jako nieobecnego na module (Kacper)

```
Create PROCEDURE [dbo].[SetUserUnpresentOnModule]
    @ModuleID int,
    @UserID int
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM ModuleAttendances MA WHERE MA.UserID = @UserID AND
MA.CourseModuleID = @ModuleID)
        RETURN;
    DELETE FROM ModuleAttendances WHERE CourseModuleID = @ModuleID and UserID =
@UserID
END;
```

24. Ustaw użytkownika jako obecnego na spotkaniu (Jan)

```
Create PROCEDURE [dbo].[SetUserPresentOnMeeting]
    @MeetingID int,
    @UserID int
AS
BEGIN
    IF EXISTS (SELECT * FROM MeetingAttendances MA WHERE MA.UserID = @UserID AND
MA.MeetingID = @MeetingID)
        RETURN;
    INSERT INTO MeetingAttendances (MeetingID, UserID) VALUES (@MeetingID, @UserID)
END;
GO
```

25. Ustaw użytkownika jako nieobecnego na spotkaniu (Jan)

```
Create PROCEDURE [dbo].[SetUserUnpresentOnMeeting]
    @MeetingID int,
    @UserID int
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM MeetingAttendances MA WHERE MA.UserID =
@UserID AND MA.MeetingID = @MeetingID)
        RETURN;
    DELETE FROM MeetingAttendances WHERE MeetingID = @MeetingID and
UserID = @UserID
END;
GO
```

25. Dodaje egzamin (Jan)

```
Create PROCEDURE [dbo].[SetUserUnpresentOnMeeting]
    @MeetingID int,
    @UserID int
```

```
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM MeetingAttendances MA WHERE MA.UserID =
@UserID AND MA.MeetingID = @MeetingID)
       RETURN;
    DELETE FROM MeetingAttendances WHERE MeetingID = @MeetingID and
UserID = @UserID
END;
GO
```

25. Dodaje wynik egzaminy (Jan)

```
Create PROCEDURE [dbo].[SetUserUnpresentOnMeeting]
    @MeetingID int,
    @UserID int
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM MeetingAttendances MA WHERE MA.UserID =
@UserID AND MA.MeetingID = @MeetingID)
       RETURN;
    DELETE FROM MeetingAttendances WHERE MeetingID = @MeetingID and
UserID = @UserID
END;
GO
```

25. Dodaje staż (Jan)

```
Create PROCEDURE [dbo].[SetUserUnpresentOnMeeting]
    @MeetingID int,
    @UserID int
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM MeetingAttendances MA WHERE MA.UserID =
@UserID AND MA.MeetingID = @MeetingID)
       RETURN;
    DELETE FROM MeetingAttendances WHERE MeetingID = @MeetingID and
UserID = @UserID
END;
GO
```

26. Dodaje tłumaczenie (Jan)

```
Create PROCEDURE [dbo].[SetUserUnpresentOnMeeting]
    @MeetingID int,
    @UserID int
```

```
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM MeetingAttendances MA WHERE MA.UserID =
@UserID AND MA.MeetingID = @MeetingID)
       RETURN;
    DELETE FROM MeetingAttendances WHERE MeetingID = @MeetingID and
UserID = @UserID
END;
GO
```

27. Dodaje przedmiot (Jan)

```
Create PROCEDURE [dbo].[SetUserUnpresentOnMeeting]
    @MeetingID int,
    @UserID int
AS
BEGIN
    IF NOT EXISTS (SELECT * FROM MeetingAttendances MA WHERE MA.UserID =
@UserID AND MA.MeetingID = @MeetingID)
       RETURN;
    DELETE FROM MeetingAttendances WHERE MeetingID = @MeetingID and
UserID = @UserID
END;
GO
```

25. Dodaje Webinar (Jan)

```
ALTER PROCEDURE [dbo].[AddWebinar]
    @Name VARCHAR(50),
    @LecturerID INT,
    @Price MONEY,
    @IsAvailable BIT,
      @EventDate datetime,
      @ConnectionLink nvarchar(50),
      @ResourceLink nvarchar(50),
      @Duration int
AS
BEGIN
      DECLARE     @ProductTypeID int;
      (SELECT @ProductTypeID =  PT.ProductTypeID FROM ProductTypes PT
Where PT.Name = 'Webinar')
      DECLARE @ProductID int;
      EXEC @ProductID = [dbo].[AddProduct] @Name = @Name, @LecturerID =
@LecturerID, @Price = @Price, @ProductTypeID = @ProductTypeID,
@IsAvailable = @IsAvailable
```

```
      INSERT INTO Webinars VALUES (@ProductID, @EventDate,
@ConnectionLink, @ResourceLink, @Duration)
END
```

25. Dodaje kurs (Jan)

```
ALTER PROCEDURE [dbo].[AddCourse]
    @Name VARCHAR(50),
    @LecturerID INT,
    @Price MONEY,
      @StartDate datetime,
      @EntryFee int = NULL
AS
BEGIN
      DECLARE     @ProductTypeID int;
      (SELECT @ProductTypeID =  PT.ProductTypeID FROM ProductTypes PT
Where PT.Name = 'Course')
      DECLARE @ProductID int;
      EXEC @ProductID = [dbo].[AddProduct] @Name = @Name, @LecturerID =
@LecturerID, @Price = @Price, @ProductTypeID = @ProductTypeID,
@IsAvailable = 0
      IF @EntryFee is null
            SET @EntryFee = @Price
      INSERT INTO Courses VALUES (@ProductID, @EntryFee, @StartDate)
END
```

25. Dodaje studia (Jan)

```
ALTER PROCEDURE [dbo].[AddStudies]
    @Name VARCHAR(50),
    @LecturerID INT,
    @Price MONEY,
      @Sylabus nvarchar(MAX),
      @Limit int,
      @StartDate datetime,
      @EndDate datetime
AS
BEGIN
      DECLARE     @ProductTypeID int;
      (SELECT @ProductTypeID =  PT.ProductTypeID FROM ProductTypes PT
Where PT.Name = 'Studies')
      DECLARE @ProductID int;
      EXEC @ProductID = [dbo].[AddProduct] @Name = @Name, @LecturerID =
```

```
@LecturerID, @Price = @Price, @ProductTypeID = @ProductTypeID,
@IsAvailable = 0
      INSERT INTO Studies VALUES (@ProductID, @Sylabus, @Limit,
@StartDate, @EndDate)
END
```

Funkcje:

1. **Funkcja zwraca ID, nazwę kursu i startDate kursów zaczynających się w bieżącym tygodniu.** (Kacper)

```
CREATE FUNCTION [dbo].[GetCourseIDandNameThisWeek]()
RETURNS TABLE
AS
RETURN
       SELECT c.CourseID, p.Name, c.StartDate
       FROM Courses c
       INNER JOIN Products p ON c.CourseID = p.ProductID
       WHERE DATEPART(week, c.StartDate) = DATEPART(week, GETDATE())
       AND DATEPART(year, c.StartDate) = DATEPART(year, GETDATE());
```

5. **Funkcja zwraca ID, nazwę kursu i startDate webinarów zaczynających się w bieżącym tygodniu.** (Kacper)

```
CREATE FUNCTION [dbo].[GetCourseIDandNameThisWeek]()
RETURNS TABLE
AS
RETURN
       SELECT c.CourseID, p.Name, c.StartDate
       FROM Courses c
       INNER JOIN Products p ON c.CourseID = p.ProductID
       WHERE DATEPART(week, c.StartDate) = DATEPART(week, GETDATE())
       AND DATEPART(year, c.StartDate) = DATEPART(year, GETDATE());
```

6. **Funkcja zwraca historię zamówień konkretnego klienta (podanie ID klienta)** (Kacper)

```
CREATE FUNCTION [dbo].[GetCustomerOrderHistory](@CustomerID INT)
RETURNS TABLE
AS
RETURN
       SELECT
       o.OrderID,
       p.Name AS ProductName,
       od.Price
       FROM Orders o
       INNER JOIN OrderEntireProductDetails od ON o.OrderID = od.OrderID
       INNER JOIN Products p ON od.ProductID = p.ProductID
       WHERE o.CustomerID = @CustomerID;
```

7. **Funkcja zwraca szczegóły zamówienia (nazwa produktu, cena) o podanym ID.** (Kacper)

```
CREATE FUNCTION [dbo].[GetOrderDetailsWithoutCount](@OrderID INT)
RETURNS TABLE
AS
RETURN
    SELECT
       p.Name AS ProductName,
       od.Price
    FROM OrderEntireProductDetails od
    INNER JOIN Products p ON od.ProductID = p.ProductID
    WHERE od.OrderID = @OrderID;
```

8. **Funkcja zwraca wartość danego zamówienia (zwraca łączną wartość całego zamówienia)** (Kacper)

```
CREATE FUNCTION [dbo].[GetTotalOrderValue](@OrderID INT)
RETURNS DECIMAL(10,2)
AS
BEGIN
       DECLARE @TotalValue DECIMAL(10,2)
       SELECT @TotalValue = SUM(Price)
       FROM OrderEntireProductDetails
       WHERE OrderID = @OrderID
       RETURN @TotalValue
```

9. **Funkcja zwraca wartość wszystkich zamówień w danym tygodniu/ miesiącu/ roku** (Kacper)

```
CREATE FUNCTION [dbo].[GetTotalValueByPeriod](@Date DATE, @PeriodType NVARCHAR(10))
RETURNS DECIMAL(10, 2)
AS
BEGIN
	DECLARE @TotalValue DECIMAL(10, 2)
	IF @PeriodType = 'Week'
	SELECT @TotalValue = SUM(od.Price)
	FROM Orders o
	INNER JOIN OrderEntireProductDetails od ON o.OrderID = od.OrderID
	WHERE DATEPART(week, o.OrderDate) = DATEPART(week, @Date)
	AND DATEPART(year, o.OrderDate) = DATEPART(year, @Date)
	ELSE IF @PeriodType = 'Month'
	SELECT @TotalValue = SUM(od.Price)
	FROM Orders o
	INNER JOIN OrderEntireProductDetails od ON o.OrderID = od.OrderID
	WHERE DATEPART(month, o.OrderDate) = DATEPART(month, @Date)
	AND DATEPART(year, o.OrderDate) = DATEPART(year, @Date)
	ELSE IF @PeriodType = 'Year'
	SELECT @TotalValue = SUM(od.Price)
	FROM Orders o
	INNER JOIN OrderEntireProductDetails od ON o.OrderID = od.OrderID
	WHERE DATEPART(year, o.OrderDate) = DATEPART(year, @Date)
	RETURN ISNULL(@TotalValue, 0)
END
```

10. **Funkcja zwraca imiona i nazwiska wszystkich pracowników.** (Kacper)

```
CREATE FUNCTION [dbo].[GetEmployeesByCompany](@CompanyID INT)
RETURNS TABLE
AS
RETURN
	SELECT
	u.Name,
	u.Surname
	FROM Users u
	INNER JOIN UserAccountTypes uat ON u.UserID = uat.UserID
	INNER JOIN AccountType at ON uat.AccountTypeID = at.AccountTypeID
	WHERE at.AccountTypeID IN (2, 3, 4)
	AND u.CompanyID = @CompanyID;
```

11. Dostań aktualną ocenę z egzaminu dla studenta (Jan)

```
CREATE FUNCTION [dbo].[GetExamsForStudent]
(
    @StudiesID int,
    @UserID int
)
RETURNS table
AS
RETURN SELECT E.ExamID, MAX(E.NAME) as Name, Max(Er.Attempt) as Attempt,
Max(er.Note) as Note, Max(er.SubmitDate) as SubmitDate FROM Exams E INNER JOIN
ExamResults ER ON ER.StudentID = @UserID Where E.StudyID = @StudiesID and er.Note >
2 Group by E.ExamID

GO
```

12. Dostań zaliczone staże przez studenta (Jan)

```
CREATE FUNCTION [dbo].[GetPassedStudiesIntershipsForStudent]
(
    @StudiesID int,
    @UserID int
)
RETURNS table
AS
RETURN SELECT * FROM Internships I Where I.StudiesID = @StudiesID AND I.UserID =
@UserID and I.IsCompleted = 1
GO
```

13. Dostań kwotę wpłaconej zaliczki dla kursu (Jan)

```
CREATE FUNCTION [dbo].[GetFeePaidForAccessToCourse]
(
    @AccessToCourseID int
)
RETURNS int
AS
BEGIN
    DECLARE @Result INT;
    SELECT @Result = ISNULL((Select TOP 1 OEFD.EntryPrice From OrderEntryFeeDetails
OEFD INNER JOIN Orders O ON O.OrderID = OEFD.OrderID INNER JOIN
OrderEntryFeeAccessToProducts OEFATP ON OEFATP.OrderID = O.OrderID WHERE O.IsSucced
= 1 AND @AccessToCourseID = OEFATP.AccessToProductID), 0)
RETURN @Result
END
GO
```

14. Dostań cenę za spotkanie dla danego użytkownika (Jan)

```
CREATE FUNCTION [dbo].[GetSingleMeetingPriceForUser]
(
    @UserID int,
    @SingleMeetingID int
)
RETURNS int
AS
BEGIN
    DECLARE @StudyID INT;
    SELECT @StudyID = S.StudyID FROM Studies S INNER JOIN Subjects SU ON S.StudyID =
SU.SubjectID INNER JOIN StudiesMeetings SM ON SU.SubjectID = SM.SubjectID WHERE
SM.StudyMeetingID = @SingleMeetingID;
    DECLARE @MeetingDate Datetime;
    SELECT @MeetingDate = SM.StartDate FROM StudiesMeetings SM WHERE
SM.StudyMeetingID = @SingleMeetingID
    DECLARE @Result INT;
    SELECT @Result =     (
                            CASE WHEN EXISTS (Select AP.AccessToProductID From
AccessToProducts AP INNER JOIN Studies S ON S.StudyID = AP.ProductID INNER JOIN
Subjects SU ON SU.StudiesID = S.StudyID WHERE AP.UserID = @UserID AND @MeetingDate
BETWEEN AP.StartDate AND AP.EndDate)
                            THEN (SELECT TOP 1 SM.Price FROM StudiesMeetings SM
WHERE SM.StudyMeetingID = @SingleMeetingID)
                            ELSE ISNULL((Select TOP 1 P.Price FROM SingleMeetings
SSM INNER JOIN Products P ON P.ProductID = SSM.ProductID Where @SingleMeetingID =
SSM.StudyMeetingID), (SELECT TOP 1 SM.Price FROM StudiesMeetings SM WHERE
SM.StudyMeetingID = @SingleMeetingID))
                            END
                            )
RETURN @Result
END
GO
```

15. Dostań datę rozpoczęcia studiów (Jan)

```
CREATE FUNCTION [dbo].[GetStudiesStartDate]
(
    @StudiesID int
)
RETURNS datetime
AS
BEGIN
    DECLARE @Result datetime;
    SELECT @Result = MIN(SM.StartDate) FROM StudiesMeetings SM INNER JOIN Subjects
SB ON SB.SubjectID = SM.SubjectID Where SB.StudiesID = @StudiesID
RETURN @Result
END
GO
```

16. Sprawdza czy dany test został zdany (Jan)

```
CREATE FUNCTION [dbo].[IsExamPassed]
(
    @StudiesID int,
    @UserID int
)
RETURNS bit
AS
BEGIN
    DECLARE @Result int;
    SELECT @Result = Max(e.Note) FROM GetExamsForStudent(@StudiesID, @UserID) e
RETURN IIF(@Result > 2, 1, 0)
END
GO
```

17. Sprawdza czy staż został zaliczony (Jan)

```
CREATE FUNCTION [dbo].[IsIntershipCompleted]
(
    @StudiesID int,
    @UserID int
)
RETURNS bit
AS
BEGIN
    DECLARE @Result int;
    SELECT @Result = count(*) FROM GetPassedStudiesIntershipsForStudent(@StudiesID,
@UserID)
RETURN IIF(@Result >= 2, 1, 0)
END
```

Triggery:

1. **Uniemożliwia przypisanie niestacjonarnym modułom do tablicy stacjonarnych modułów** (Jan)

```
CREATE TRIGGER [dbo].[keep_sync_integral_stationary_modules] ON
[dbo].[StationaryModules]
FOR INSERT
AS
       -- Specify your condition here
       IF not EXISTS (SELECT * FROM inserted i
    INNER JOIN CourseModules CM ON cm.CourseModuleID = i.CourseModuleID
    INNER JOIN ModuleMeetingTypes MT ON MT.ModuleMeetingTypeID = cm.Type
    Where MT.Name = 'Stacjonarne' or MT.Name = 'hybrydowe')
        THROW 51000, N'Inserted blocked: You cant add module which is not stationary
to this table', 1;
GO

ALTER TABLE [dbo].[StationaryModules] ENABLE TRIGGER
[keep_sync_integral_stationary_modules]
GO
```

2. **Uniemożliwia przypisanie niestacjonarnych spotkań do tablicy stacjonarnych spotkań** (Kacper)

```
CREATE TRIGGER [dbo].[keep_sync_integral_stationary_meetings] ON
[dbo].[StationaryMeetings]
FOR INSERT
AS
       -- Specify your condition here
       IF not EXISTS (SELECT * FROM inserted i
    INNER JOIN StudiesMeetings sm ON sm.StudyMeetingID = i.StudyMeetingID
    INNER JOIN ModuleMeetingTypes MT ON MT.ModuleMeetingTypeID = sm.TypeID
    Where MT.Name = 'Stacjonarne')
        THROW 51000, N'Inserted blocked: You cant add meeting which is not
stationary to this table', 1;
GO

ALTER TABLE [dbo].[StationaryMeetings] ENABLE TRIGGER
[keep_sync_integral_stationary_meetings]
GO
```

3. **Uniemożliwia wystawienie studiów, które nie posiadają harmonogramu** (Jan)

```
CREATE TRIGGER [dbo].[prevent_studies_no__plan] ON [dbo].[Products]
FOR UPDATE
AS
       -- Specify your condition here
       IF (UPDATE(IsAvailable) and
    EXISTS (SELECT * FROM inserted i
    INNER JOIN ProductTypes pt on pt.ProductTypeID = i.ProductTypeID
    INNER JOIN Subjects S ON S.StudiesID = I.ProductID
    LEFT JOIN StudiesMeetings SM ON S.SubjectID = SM.SubjectID
    WHERE sm.StartDate is null and pt.Name = 'Studies' and i.IsAvailable = 1))
        THROW 51000, N'Update blocked: for studies has to exist plan', 1;
GO

ALTER TABLE [dbo].[Products] ENABLE TRIGGER [prevent_studies_no__plan]
GO
```

4. **Uniemożliwia wystawienie studiów, bez sylabusa** (Kacper)

```
CREATE TRIGGER [dbo].[prevent_studies_no_sylabus] ON [dbo].[Products]
FOR UPDATE
AS
       -- Specify your condition here
       IF (UPDATE(IsAvailable) and
    EXISTS (SELECT * FROM inserted i
    INNER JOIN ProductTypes pt on pt.ProductTypeID = i.ProductTypeID
    LEFT JOIN Subjects S ON S.StudiesID = I.ProductID
    WHERE S.Sylabus is NULl and pt.Name = 'Studies' and i.IsAvailable = 1))
        THROW 51000, N'Update blocked: for studies has to exist sylabus', 1;
GO

ALTER TABLE [dbo].[Products] ENABLE TRIGGER [prevent_studies_no_sylabus]
GO
```

5. **Uniemożliwia przypisanie stacjonarnych modułów do tablicy niestacjonarnych modułów-synchronicznych** (Jan)

```
CREATE TRIGGER [dbo].[keep_sync_integral_OnlineSync_modules] ON
[dbo].[OnlineSyncModules]
FOR INSERT
AS
       -- Specify your condition here
       IF not EXISTS (SELECT * FROM inserted i
    INNER JOIN CourseModules CM ON cm.CourseModuleID = i.CourseModuleID
    INNER JOIN ModuleMeetingTypes MT ON MT.ModuleMeetingTypeID = cm.Type
    Where MT.Name = 'on-line synchroniczne' or MT.Name = 'hybrydowe')
        THROW 51000, N'Inserted blocked: You cant add module which is not on-line
sync to this table', 1;
GO

ALTER TABLE [dbo].[OnlineSyncModules] ENABLE TRIGGER
[keep_sync_integral_OnlineSync_modules]
GO
```

6. **Uniemożliwia przypisanie stacjonarnych spotkań do tablicy niestacjonarnych spotkań** (Kacper)

```
CREATE TRIGGER [dbo].[keep_sync_integral_online_meetings] ON [dbo].[OnlineMeetings]
FOR INSERT
AS
       -- Specify your condition here
       IF not EXISTS (SELECT * FROM inserted i
    INNER JOIN StudiesMeetings sm ON sm.StudyMeetingID = i.StudyMeetingID
    INNER JOIN ModuleMeetingTypes MT ON MT.ModuleMeetingTypeID = sm.TypeID
    Where MT.Name = 'on-line synchroniczne' or MT.Name = 'on-line asynchronicznie'
or MT.Name = 'hybrydowe')
        THROW 51000, N'Inserted blocked: You cant add meeting which is not online to
this table', 1;
GO

ALTER TABLE [dbo].[OnlineMeetings] ENABLE TRIGGER
[keep_sync_integral_online_meetings]
GO
```

7. **Uniemożliwia przypisanie stacjonarnych modułów do tablicy niestacjonarnych modułów-asynchronicznych** (Jan)

```
CREATE TRIGGER [dbo].[keep_sync_integral_OnlineASync_modules] ON
[dbo].[OnlineAsyncModules]
FOR INSERT
AS
       -- Specify your condition here
       IF not EXISTS (SELECT * FROM inserted i
    INNER JOIN CourseModules CM ON cm.CourseModuleID = i.CourseModuleID
    INNER JOIN ModuleMeetingTypes MT ON MT.ModuleMeetingTypeID = cm.Type
    Where MT.Name = 'on-line asynchronicznie' or MT.Name = 'hybrydowe')
        THROW 51000, N'Inserted blocked: You cant add module which is not on-line
async to this table', 1;
GO

ALTER TABLE [dbo].[OnlineAsyncModules] ENABLE TRIGGER
[keep_sync_integral_OnlineASync_modules]
GO
```

8. **Uniemożliwia wydanie dyplomu osobie, która nie zdała kursu lub studiów** (Kacper)

```
CREATE TRIGGER [dbo].[give_diplome_only_who_passed] ON [dbo].[Diplomes]
FOR INSERT
AS
    DECLARE @Pass bit;
    SET @Pass = 0
       IF EXISTS (SELECT * FROM inserted i
    INNER JOIN StudySummary ss ON ss.StudiesID = i.ProductID and ss.UserID =
i.UserID
    Where ss.[Is Study Pass] = 1)
        SET @Pass = 1
    IF EXISTS (SELECT * FROM inserted i
    INNER JOIN CourseSummary cs ON cs.CourseID = i.ProductID and cs.UserID =
i.UserID
    Where cs.[Is Passed] = 1)
        SET @Pass = 1

    if @Pass <> 1
        THROW 51000, N'Inserted blocked: User didint pass this course or studies',
1;
GO

ALTER TABLE [dbo].[Diplomes] ENABLE TRIGGER [give_diplome_only_who_passed]
GO
```

9. **Uniemożliwia ustawienia limitu spotkania poniżej limitu studiów** (Jan)

```
CREATE TRIGGER keep_limit_greater_studies_limit
   ON  StudiesMeetings
   For INSERT, UPDATE
AS
BEGIN
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join
Subjects S on i.SubjectID = s.SubjectID inner join Studies st on
st.StudyID = s.StudiesID Where st.Limit > i.Limit)
       THROW 51000, N'You cant set meeting limit below studies limit', 1;

END
GO
```

10. **Uniemożliwia ustawienia limitu spotkania powyżej jakiekolwiek spotkania** (Jan)

```
CREATE TRIGGER keep_limit_below_meetings_limit
   ON  Studies
   For INSERT, UPDATE
AS
BEGIN
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join Subjects S on
i.StudyID = s.StudiesID inner join StudiesMeetings sm on sm.SubjectID = s.SubjectID
WHERE i.Limit > sm.Limit)
       THROW 51000, N'You cant set studies limit greater than any studies meeting',
1;
END
GO
```

11. **Uniemożliwia ustawienia limitu spotkania powyżej jakiekolwiek spotkania** (Jan)

```
CREATE TRIGGER keep_limit_below_meetings_limit
   ON  Studies
   For INSERT, UPDATE
AS
BEGIN
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join Subjects S on
i.StudyID = s.StudiesID inner join StudiesMeetings sm on sm.SubjectID = s.SubjectID
WHERE i.Limit > sm.Limit)
       THROW 51000, N'You cant set studies limit greater than any studies meeting',
1;
END
GO
```

12. **Uniemożliwia ustawienia planu nauczyciela nachodzącego na siebie** (Jan)

```
CREATE TRIGGER keep_limit_below_meetings_limit
   ON  Studies
   For INSERT, UPDATE
AS
BEGIN
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join Subjects S on
i.StudyID = s.StudiesID inner join StudiesMeetings sm on sm.SubjectID = s.SubjectID
WHERE i.Limit > sm.Limit)
        THROW 51000, N'You cant set studies limit greater than any studies meeting',
1;
END
GO
```

13. **Uniemożliwia ustawienie planu modułów stacjonarnych nachodzącego na siebie**((Jan)

```
CREATE TRIGGER keep_limit_below_meetings_limit
   ON  Studies
   For INSERT, UPDATE
AS
BEGIN
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join Subjects S on
i.StudyID = s.StudiesID inner join StudiesMeetings sm on sm.SubjectID = s.SubjectID
WHERE i.Limit > sm.Limit)
        THROW 51000, N'You cant set studies limit greater than any studies meeting',
1;
END
GO
```

14. **Uniemożliwia ustawienie planu studiów nachodzącego na siebie**(Jan)

```
CREATE TRIGGER keep_limit_below_meetings_limit
   ON  Studies
   For INSERT, UPDATE
AS
BEGIN
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join Subjects S on
i.StudyID = s.StudiesID inner join StudiesMeetings sm on sm.SubjectID = s.SubjectID
WHERE i.Limit > sm.Limit)
        THROW 51000, N'You cant set studies limit greater than any studies meeting',
1;
END
GO
```

15. **Uniemożliwia tłumaczowi mieć nachodzący plan** (Jan)

```
CREATE TRIGGER keep_limit_below_meetings_limit
   ON  Studies
   For INSERT, UPDATE
AS
BEGIN
```

```
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join Subjects S on
i.StudyID = s.StudiesID inner join StudiesMeetings sm on sm.SubjectID = s.SubjectID
WHERE i.Limit > sm.Limit)
        THROW 51000, N'You cant set studies limit greater than any studies meeting',
1;
END
GO
```

16. Uniemożliwia zrobienia nachodzącego planu modułów online (Jan)

```
CREATE TRIGGER keep_limit_below_meetings_limit
   ON  Studies
   For INSERT, UPDATE
AS
BEGIN
    IF UPDATE(Limit) and EXISTS (SELECT * FROM inserted i inner join Subjects S on
i.StudyID = s.StudiesID inner join StudiesMeetings sm on sm.SubjectID = s.SubjectID
WHERE i.Limit > sm.Limit)
        THROW 51000, N'You cant set studies limit greater than any studies meeting',
1;
END
GO
```

Uprawnienia

Użytkownik indywidualny z założonym kontem (individual_client):

```
CREATE ROLE individual_client;

-- Przydzielenie uprawnień do widoków
GRANT SELECT ON dbo.AllAttendenceReport TO individual_client;
GRANT SELECT ON dbo.AllAvailableProducts TO individual_client;
GRANT SELECT ON dbo.CourseSummary TO individual_client;
GRANT SELECT ON dbo.OrderSummary TO individual_client;

-- Przydzielenie uprawnień do procedur
GRANT EXECUTE ON [dbo].[AddWebinarToOrder] TO individual_client;
GRANT EXECUTE ON [dbo].[CompleteOrder] TO individual_client;
GRANT EXECUTE ON [dbo].[UpdateUserDetails] TO individual_client;
GRANT EXECUTE ON [dbo].[AddStudiesToOrder] TO individual_client;
GRANT EXECUTE ON [dbo].[AddMeetingToOrder] TO individual_client;
GRANT EXECUTE ON [dbo].[AddProductToOrder] TO individual_client;
GRANT EXECUTE ON [dbo].[AddCourseToOrder] TO individual_client;
GRANT EXECUTE ON [dbo].[CreateOrder] TO individual_client;
```

```
-- Przydzielenie uprawnień do funkcji
GRANT EXECUTE ON [dbo].[GetExamsForStudent] TO individual_client;
GRANT EXECUTE ON [dbo].[GetOrderDetails] TO individual_client;
GRANT EXECUTE ON [dbo].[GetCustomerOrderHistory] TO individual_client;
GRANT EXECUTE ON [dbo].[IsExamPassed] TO individual_client;
GRANT EXECUTE ON [dbo].[IsInternshipCompleted] TO individual_client;
GRANT EXECUTE ON [dbo].[GetSingleMeetingPriceForUser] TO individual_client;
```

Wykładowca (lecturer):

```
-- Tworzenie roli 'lecturer'
CREATE ROLE lecturer;

-- Przydzielenie uprawnień do widoków
GRANT SELECT ON dbo.CoursePlanReport TO lecturer;
GRANT SELECT ON dbo.CourseSummary TO lecturer;
GRANT SELECT ON dbo.TeacherCoursePlan TO lecturer;
GRANT SELECT ON dbo.TeacherStudiesPlan TO lecturer;
GRANT SELECT ON dbo.TeacherSummaryPlan TO lecturer;
GRANT SELECT ON dbo.TeacherWebinarPlan TO lecturer;

-- Przydzielenie uprawnień do procedur
GRANT EXECUTE ON [dbo].[InsertStudent] TO lecturer;
GRANT EXECUTE ON [dbo].[SetAllUserPresentOnMeeting] TO lecturer;
GRANT EXECUTE ON [dbo].[SetAllUserPresentOnModule] TO lecturer;
GRANT EXECUTE ON [dbo].[SetUserPresentOnMeeting] TO lecturer;
GRANT EXECUTE ON [dbo].[SetUserPresentOnModule] TO lecturer;
GRANT EXECUTE ON [dbo].[SetUserUnpresentOnMeeting] TO lecturer;
GRANT EXECUTE ON [dbo].[SetUserUnpresentOnModule] TO lecturer;
GRANT EXECUTE ON [dbo].[UpdateMeetingDate] TO lecturer;
GRANT EXECUTE ON [dbo].[UpdateSylabus] TO lecturer;
GRANT EXECUTE ON [dbo].[GetAddress] TO lecturer;

 -- Przydzielenie uprawnień do funkcji
GRANT EXECUTE ON [dbo].[GetCourseIDandNameThisWeek] TO lecturer;
GRANT EXECUTE ON [dbo].[GetExamsForStudent] TO lecturer;
GRANT EXECUTE ON [dbo].[GetPassedStudiesIntershipsForStudent] TO lecturer;
GRANT EXECUTE ON [dbo].[GetWebinarIDandNameThisWeek] TO lecturer;
GRANT EXECUTE ON [dbo].[IsExamPassed] TO lecturer;
GRANT EXECUTE ON [dbo].[IsInternshipCompleted] TO lecturer;
GRANT EXECUTE ON [dbo].[GetStudiesStartDate] TO lecturer;
GRANT EXECUTE ON [dbo].[GetStudiesEndDate] TO lecturer;
```

Tłumacz (interpreter):

```
-- Tworzenie roli 'interpreter'
CREATE ROLE interpreter;

-- Przydzielenie uprawnień do widoków
GRANT SELECT ON dbo.InterpreterCoursePlan TO interpreter;
```

```
GRANT SELECT ON dbo.InterpreterStudiesPlan TO interpreter;
GRANT SELECT ON dbo.InterpreterSummaryPlan TO interpreter;
GRANT SELECT ON dbo.InterpreterWebinarPlan TO interpreter;

-- Przydzielenie uprawnień do procedur
GRANT EXECUTE ON [dbo].[AddTranslation] TO interpreter;

-- Przydzielenie uprawnień do funkcji
GRANT EXECUTE ON [dbo].[GetCourseIDandNameThisWeek] TO interpreter;
GRANT EXECUTE ON [dbo].[GetWebinarIDandNameThisWeek] TO interpreter;
```

Headmaster:

```
-- Tworzenie roli 'headmaster' i przydzielenie pełnych uprawnień na poziomie bazy danych
CREATE ROLE headmaster;
grant all privileges TO headmaster

-- Tworzenie roli 'main_administrator' i przydzielenie pełnych uprawnień na poziomie bazy
danych
CREATE ROLE main_administrator;
grant all privileges TO main_administrator
```

Indeksy

```
-- Index for AccessToMeeting
CREATE INDEX idx_AccessToMeeting_StartDate ON AccessToMeetings (StartDate);
CREATE INDEX idx_AccessToMeeting_EndDate ON AccessToMeetings (EndDate);
CREATE INDEX idx_AccessToMeeting_UserID ON AccessToMeetings (UserID);

-- Index for AccessToProduct
CREATE INDEX idx_AccessToProduct_StartDate ON AccessToProducts (StartDate);

-- Index for CourseModule (assuming CourseModuleID is a typo and it's CourseModules)
CREATE INDEX idx_CourseModule_CourseID ON CourseModules (CourseID)

-- Index for Courses
CREATE INDEX idx_Courses_StartDate ON Courses (StartDate);

-- Index for Orders
CREATE INDEX idx_Orders_PayDate ON Orders (PayDate);
CREATE INDEX idx_Orders_OrderDate ON Orders (OrderDate);
CREATE INDEX idx_Orders_UserID ON Orders (UserID);

-- Index for OrderSingleMeetingDetails
CREATE INDEX idx_OrderSingleMeetingDetails_MeetingID ON OrderSingleMeetingDetails
(MeetingID);
```

```sql
-- Index for Products
CREATE INDEX idx_Products_Name ON Products (Name);

-- Index for StudiesMeetings
CREATE INDEX idx_StudiesMeetings_StartDate ON StudiesMeetings (StartDate);

-- Index for Studies
CREATE INDEX idx_Studies_StartDate ON Studies (StartDate);
CREATE INDEX idx_Studies_EndDate ON Studies (EndDate);

-- Index for Users
CREATE INDEX idx_Users_Email ON Users (Email);
-- Index for Webinars
CREATE INDEX idx_Webinars_EventDate ON Webinars (EventDate);
```

Schemat logiczny