Projekt bazy danych

Szymon Miękina & Mateusz Bartnicki & Dominik Marek

Spis treści

Użytkownicy

- Administrator systemu
- System
- Dyrektor
- Pracownicy sekretariatu
- Koordynator
- Prowadzący
- Tłumacz
- Uczestnicy
- Gość (niezarejestrowany)

Administrator systemu ma uprawnienia do każdej funkcji

Funkcje

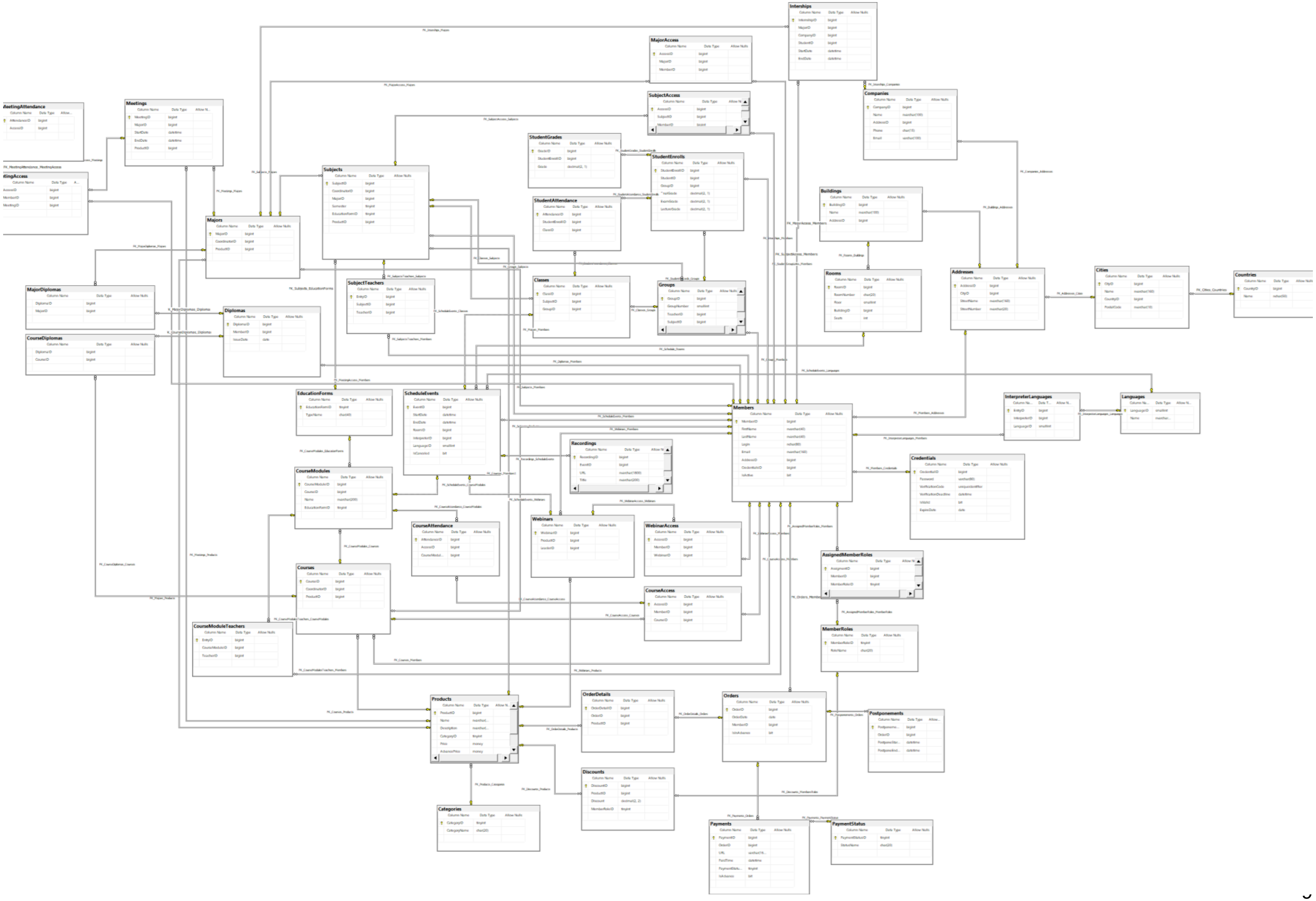
- Ogólne:
  1. założenie konta (login, hasło, e-mail, adres korespondencyjny) - gość
  2. weryfikacja konta - system
  3. zmiana roli użytkownika - administrator
  4. dezaktywacja konta - każdy posiadający konto
  5. koszyk - możliwość zakupu różnych usług (opisane w punkcie Integracja z systemem płatności) - zarejestrowani uczestnicy
  6. wybór języka, w którym prowadzone są zajęcia - dyrektor
  7. wysłanie dyplomu za ukończenie wybranej formy zajęć - pracownicy sekretariatu
  8. umożliwienie uczestnictwa w zajęciach, pomimo braku uiszczenia płatności - dyrektor
  9. tworzenie backupów bazy danych - system, administrator
  10. zobaczenie oferty - każdy
- Integracja z systemem płatności - koszyk
  1. dodanie produktu do koszyka - każdy posiadający konto
  2. modyfikacja koszyka - jak powyżej
  3. wyliczenie wartości koszyka - system,
  4. informacja o wpisaniu zapłaty do systemu - system
  5. informacja o statusie płatności -system
  6. rezygnacja z zapłaty pełnej kwoty - użytkownik z kontem
  7. przegląd poprzednich płatności - jak powyżej
  8. uiszczenie opłaty - każdy posiadający konto:
     - opłata za webinar(możliwa do momentu rozpoczęcia)
     - zapłata pełnej kwoty bądź zaliczki/wpisowego(+ dopłata maksymalnie do 3 dni przed startem kursu/zjazdu)
     - przypomnienie o czasie pozostałym do uzupełnienia kwoty

- Webinary
  1. dodanie, modyfikacja, usunięcie - prowadzący
  2. dołączenie do webinaru (po wykupieniu dostępu) - uczestnicy

3. dodanie nagrania - prowadzący
4. obejrzenie nagrania - wszyscy użytkownicy (w przypadku płatnego - wszyscy, którzy wykupili)
5. dodanie tłumaczenia - prowadzący

- Kursy
    1. założenie kursu (+ wybór jego formy), usunięcie - prowadzący
    2. udzielenie zaliczenia - prowadzący
    3. zapisy - uczestnicy
    4. dodanie nagrania - prowadzący,
    5. Obejrzenie nagrania (w przypadku kursu online lub hybrydowego) - uczestnicy
    6. dodanie tłumaczenia -prowadzący
    7. ustalenie limitu miejsc (dla hybrydowych i stacjonarnych) - pracownicy sekretariatu
- Studia
    1. utworzenie sylabusa (niemodyfikowalny po rozpoczęciu studiów) - dyrektor, prowadzący, koordynator
    2. utworzenie harmonogramu zajęć - pracownicy sekretariatu
    3. modyfikacja harmonogramu - pracownicy sekretariatu
    4. wybór formy studiów - dyrektor
    5. zapisanie się na poszczególne zajęcia (w przypadku, osób spoza studium) - uczestnicy
    6. ustalenie terminu praktyk - pracownicy sekretariatu
    7. udzielenie zaliczenia praktyk - prowadzący
    8. udzielenie zaliczenia zajęć - prowadzący, koordynator
    9. ustalenie limitu miejsc (z uwzględnieniem maksymalnego limitu miejsc najmniejszego obiektu) - pracownicy sekretariatu
    10. sprawdzanie obecności - prowadzący
    11. dodanie tłumaczenia -prowadzący

- Raporty
    1. raport finansowy - pracownicy sekretariatu
    2. lista dłużników - pracownicy sekretariatu, dyrektor
    3. statystyki wydarzeń, frekwencja na wydarzeniach - dyrektor, pracownicy sekretariatu
    4. lista obecności - prowadzący, dyrektor
    5. raport bilokacji - sekretariat


Interships
InternshipID bigint
MajorID bigint
CompanyID bigint
StudentID bigint
StartDate datetime
EndDate datetime
MajorAccess
AccessID bigint
MajorID bigint
MemberID bigint
SubjectAccess
AccessID bigint
SubjectID bigint
MemberID bigint
Companies
CompanyID bigint
Name nvarchar(100)
AddressID bigint
Phone char(15)
Email varchar(100)
leetingAttendance
AttendanceID bigint
AccessID bigint
Meetings
MeetingID bigint
MajorID bigint
StartDate datetime
EndDate datetime
ProductID bigint
tingAccess
AccessID bigint
MemberID bigint
MeetingID bigint
StudentGrades
GradeID bigint
StudentEnrollID bigint
Grade decimal(2, 1)
StudentEnrolls
StudentEnrollID bigint
StudentID bigint
GroupID bigint
FinalGrade decimal(2, 1)
ExamGrade decimal(2, 1)
LectureGrade decimal(2, 1)
Subjects
SubjectID bigint
CoordinatorID bigint
MajorID bigint
Semester tinyint
EducationFormID tinyint
ProductID bigint
StudentAttendance
AttendanceID bigint
StudentEnrollID bigint
ClassID bigint
Buildings
BuildingID bigint
Name nvarchar(100)
AddressID bigint
Majors
MajorID bigint
CoordinatorID bigint
ProductID bigint
SubjectTeachers
EntryID bigint
SubjectID bigint
TeacherID bigint
Classes
ClassID bigint
SubjectID bigint
GroupID bigint
Groups
GroupID bigint
GroupNumber smallint
TeacherID bigint
SubjectID bigint
Rooms
RoomID bigint
RoomNumber char(20)
Floor smallint
BuildingID bigint
Seats int
Addresses
AddressID bigint
CityID bigint
StreetName nvarchar(160)
StreetNumber nvarchar(20)
Cities
CityID bigint
Name nvarchar(160)
CountryID bigint
PostalCode nvarchar(10)
Countries
CountryID bigint
Name nchar(50)
MajorDiplomas
DiplomaID bigint
MajorID bigint
Diplomas
DiplomaID bigint
MemberID bigint
IssueDate date
CourseDiplomas
DiplomaID bigint
CourseID bigint
EducationForms
EducationFormID tinyint
TypeName char(40)
ScheduleEvents
EventID bigint
StartDate datetime
EndDate datetime
RoomID bigint
InterpreterID bigint
LanguageID smallint
IsCanceled bit
Members
MemberID bigint
FirstName nvarchar(40)
LastName nvarchar(40)
Login nchar(80)
Email nvarchar(160)
AddressID bigint
CredentialsID bigint
IsActive bit
InterpreterLanguages
EntryID bigint
InterpreterID bigint
LanguageID smallint
Languages
LanguageID smallint
Name nvarchar...
Recordings
RecordingID bigint
EventID bigint
URL nvarchar(1800)
Title nvarchar(200)
Credentials
CredentialsID bigint
Password varchar(80)
VerificationCode uniqueidentifier
VerificationDeadline datetime
IsValid bit
ExpireDate date
CourseModules
CourseModuleID bigint
CourseID bigint
Name nvarchar(200)
EducationFormID tinyint
CourseAttendance
AttendanceID bigint
AccessID bigint
CourseModul... bigint
Webinars
WebinarID bigint
ProductID bigint
LeaderID bigint
WebinarAccess
AccessID bigint
MemberID bigint
WebinarID bigint
AssignedMemberRoles
AssignmentID bigint
MemberID bigint
MemberRoleID tinyint
Courses
CourseID bigint
CoordinatorID bigint
ProductID bigint
CourseAccess
AccessID bigint
MemberID bigint
CourseID bigint
CourseModuleTeachers
EntryID bigint
CourseModuleID bigint
TeacherID bigint
MemberRoles
MemberRoleID tinyint
RoleName char(20)
Products
ProductID bigint
Name nvarchar(...
Description nvarchar(...
CategoryID tinyint
Price money
AdvancePrice money
OrderDetails
OrderDetailID bigint
OrderID bigint
ProductID bigint
Orders
OrderID bigint
OrderDate date
MemberID bigint
IsInAdvance bit
Postponements
Postponeme... bigint
OrderID bigint
PostponeStar... datetime
PostponeEnd... datetime
Discounts
DiscountID bigint
ProductID bigint
Discount decimal(2, 2)
MemberRoleID tinyint
Categories
CategoryID tinyint
CategoryName char(20)
Payments
PaymentID bigint
OrderID bigint
URL varchar(16...
PaidTime datetime
PaymentStatu... tinyint
IsAdvance bit
PaymentStatus
PaymentStatusID tinyint
StatusName char(20)
FK_MeetingAttendance_MeetingAccess
FK_MajorDiplomas_Majors
K_MajorDiplomas_Diplomas
K_CourseDiplomas_Diplomas
FK_Subjects_EducationForms
FK_MajorAccess_Members
FK_SubjectAccess_Members
FK_Orders_Members
FK_Cities_Countries
FK_Addresses_Cities
FK_Companies_Addresses
FK_Buildings_Addresses
FK_Rooms_Buildings
FK_Members_Addresses
FK_Members_Credentials
FK_InterpreterLanguages_Members
FK_AssignedMemberRoles_MemberRoles
FK_Discounts_MemberRoles
FK_Payments_Orders
FK_Payments_PaymentStatus
FK_OrderDetails_Orders
FK_OrderDetails_Products
FK_Products_Categories
FK_Discounts_Products
FK_Postponements_Orders
FK_Courses_Products
FK_Courses_Members
FK_Webinars_Products
FK_CourseModuleTeachers_Members
FK_CourseModules_Courses
FK_CourseAccess_Courses
FK_CourseAttendance_CourseAccess
FK_CourseAttendance_CourseModules
FK_ScheduleEvents_CourseModules
FK_ScheduleEvents_Webinars
FK_WebinarAccess_Webinars
FK_Recordings_ScheduleEvents
FK_SubjectTeachers_Subjects
FK_SubjectTeachers_Members
FK_ScheduleEvents_Classes
FK_Classes_Groups
FK_Diplomas_Members
FK_ScheduleEvents_Languages
FK_ScheduleEvents_Members
FK_Webinars_Members
FK_Interships_Majors
FK_MajorAccess_Majors
FK_SubjectAccess_Subjects
FK_Meetings_Majors
FK_Interships_Companies

Tabele

Addresses

Przechowuje adresy użytkowników oraz budynków, w których odbywają się zajęcia

1. *AddressID* (PK)
2. *CityID* (FK: Cities)
3. *StreetName* - nazwa ulicy
4. *StreetNumber* - numer budynku na ulicy

Warunki integralności:

- numer budynku musi zaczynać się od cyfry

```
CREATE TABLE [dbo].[Addresses](
      [AddressID] [bigint] IDENTITY(1,1) NOT NULL,
      [CityID] [bigint] NOT NULL,
      [StreetName] [nvarchar](160) NOT NULL,
      [StreetNumber] [nvarchar](20) 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 NOCHECK ADD  CONSTRAINT
[FK_Addresses_Cities] FOREIGN KEY([CityID])
REFERENCES [dbo].[Cities] ([CityID])
GO
ALTER TABLE [dbo].[Addresses] CHECK CONSTRAINT [FK_Addresses_Cities]
GO
ALTER TABLE [dbo].[Addresses]  WITH NOCHECK ADD  CONSTRAINT [CK_Addresses]
CHECK  (([StreetNumber] like '[1-9]%'))
GO
ALTER TABLE [dbo].[Addresses] CHECK CONSTRAINT [CK_Addresses]
GO
```

AssignedMemberRoles

Zawiera przypisy użytkowników do ról

1. *AssignmentID* (PK)
2. *MemberID* (FK: Members)
3. *MemberRoleID* (FK: Addresses)

```sql
CREATE TABLE [dbo].[AssignedMemberRoles](
	[AssignmentID] [bigint] IDENTITY(1,1) NOT NULL,
	[MemberID] [bigint] NOT NULL,
	[MemberRoleID] [tinyint] NOT NULL,
 CONSTRAINT [PK_AssignedMemberRoles] PRIMARY KEY CLUSTERED
(
	[AssignmentID] 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].[AssignedMemberRoles]  WITH NOCHECK ADD  CONSTRAINT
[FK_AssignedMemberRoles_MemberRoles] FOREIGN KEY([MemberRoleID])
REFERENCES [dbo].[MemberRoles] ([MemberRoleID])
GO
ALTER TABLE [dbo].[AssignedMemberRoles] CHECK CONSTRAINT
[FK_AssignedMemberRoles_MemberRoles]
GO
ALTER TABLE [dbo].[AssignedMemberRoles]  WITH NOCHECK ADD  CONSTRAINT
[FK_AssignedMemberRoles_Members] FOREIGN KEY([MemberID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[AssignedMemberRoles] CHECK CONSTRAINT
[FK_AssignedMemberRoles_Members]
GO
```

- Buildings

Zawiera opisy budynków, w których odbywają się zajęcia

1. *BuildingID* (PK)
2. *Name* - nazwa budynku
3. *AdresssID* (FK: Addresses)

```
CREATE TABLE [dbo].[Buildings](
	[BuildingID] [bigint] IDENTITY(1,1) NOT NULL,
	[Name] [nvarchar](100) NOT NULL,
	[AddressID] [bigint] NOT NULL,
 CONSTRAINT [PK_Buildings] PRIMARY KEY CLUSTERED
(
	[BuildingID] 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].[Buildings]  WITH NOCHECK ADD  CONSTRAINT
[FK_Buildings_Addresses] FOREIGN KEY([AddressID])
REFERENCES [dbo].[Addresses] ([AddressID])
GO
ALTER TABLE [dbo].[Buildings] CHECK CONSTRAINT [FK_Buildings_Addresses]
GO
```

- Categories

Przechowuje kategorie oferowanych usług: darmowy webinar, płatny webinar, kursy, studia, zjazdy itp.

1. *CategoryID* (PK)
2. *CategoryName* - nazwa kategorii

```
CREATE TABLE [dbo].[Categories](
	[CategoryID] [tinyint] IDENTITY(1,1) NOT NULL,
	[CategoryName] [char](20) NOT NULL,
 CONSTRAINT [PK_Categories] PRIMARY KEY CLUSTERED
(
	[CategoryID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UK_Categories] UNIQUE NONCLUSTERED
(
	[CategoryName] 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
```

- ## Cities

Dla każdego miasta przechowuje informację w jakim kraju się znajduje oraz jaki jest jego kod pocztowy

1. *CityID* (PK)
2. *Name* - nazwa miasta
3. *CountryID* (FK: Countries)
4. *PostalCode* - kod pocztowy

```
CREATE TABLE [dbo].[Cities](
      [CityID] [bigint] IDENTITY(1,1) NOT NULL,
      [Name] [nvarchar](160) NOT NULL,
      [CountryID] [bigint] NOT NULL,
      [PostalCode] [nvarchar](10) 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 NOCHECK ADD  CONSTRAINT
[FK_Cities_Countries] FOREIGN KEY([CountryID])
REFERENCES [dbo].[Countries] ([CountryID])
GO
ALTER TABLE [dbo].[Cities] CHECK CONSTRAINT [FK_Cities_Countries]
GO
```

- ## Classes

Reprezentuje zajęcia na studiach

1. *ClassID* (PK)
2. *SubjectID* (FK: Subjects) - realizowany przedmiot
3. *GroupID* (FK: Groups) - grupa zajęciowa

```
CREATE TABLE [dbo].[Classes](
	[ClassID] [bigint] NOT NULL,
	[SubjectID] [bigint] NOT NULL,
	[GroupID] [bigint] NOT NULL,
 CONSTRAINT [PK_Classes] PRIMARY KEY CLUSTERED
(
	[ClassID] 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].[Classes]  WITH NOCHECK ADD  CONSTRAINT
[FK_Classes_Groups] FOREIGN KEY([GroupID])
REFERENCES [dbo].[Groups] ([GroupID])
GO
ALTER TABLE [dbo].[Classes] CHECK CONSTRAINT [FK_Classes_Groups]
GO
ALTER TABLE [dbo].[Classes]  WITH NOCHECK ADD  CONSTRAINT
[FK_Classes_Subjects] FOREIGN KEY([SubjectID])
REFERENCES [dbo].[Subjects] ([SubjectID])
GO
ALTER TABLE [dbo].[Classes] CHECK CONSTRAINT [FK_Classes_Subjects]
GO
```

- ## Companies

Nazwy i dane kontaktowe firm, w których odbywają się praktyki podczas studiów

1. *CompanyID* (PK)
2. *Name* - nazwa firmy
3. *AdressID* (FK: Addresses) - adres firmy
4. *Phone* - telefon kontaktowy firmy
5. *Email* - adres e-mail firmy

Warunki integralności:

- *Email* musi zawierać ‘@’, a po niej znak ‘.’ , unikalny
- *Phone* unikalny

```
CREATE TABLE [dbo].[Companies](
	[CompanyID] [bigint] IDENTITY(1,1) NOT NULL,
	[Name] [nvarchar](100) NOT NULL,
	[AddressID] [bigint] NOT NULL,
	[Phone] [char](15) NULL,
	[Email] [varchar](100) NULL,
 CONSTRAINT [PK_Companies] PRIMARY KEY CLUSTERED
(
	[CompanyID] 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 [IX_Companies] UNIQUE NONCLUSTERED
(
	[Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [IX_Companies_1] UNIQUE NONCLUSTERED
(
	[Phone] 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].[Companies]  WITH NOCHECK ADD  CONSTRAINT
[FK_Companies_Addresses] FOREIGN KEY([AddressID])
REFERENCES [dbo].[Addresses] ([AddressID])
GO
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [FK_Companies_Addresses]
GO
ALTER TABLE [dbo].[Companies]  WITH NOCHECK ADD  CONSTRAINT [CK_Companies]
CHECK  (([Email] like '%@%.%'))
GO
ALTER TABLE [dbo].[Companies] CHECK CONSTRAINT [CK_Companies]
GO
```

- ## Countries

Słownik przechowujący kraje występujące w bazie

1. *CountryID* (PK)
2. *Name* - nazwa kraju

```
CREATE TABLE [dbo].[Countries](
      [CountryID] [bigint] IDENTITY(1,1) NOT NULL,
      [Name] [nchar](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
```

- CourseAccess

Dostęp użytkownika do kursu

1. *AccessID* (PK)
2. *MemberID* (FK: Members)
3. *CourseID* (FK: Courses)

```
CREATE TABLE [dbo].[CourseAccess](
      [AccessID] [bigint] IDENTITY(1,1) NOT NULL,
      [MemberID] [bigint] NOT NULL,
      [CourseID] [bigint] NOT NULL,
 CONSTRAINT [PK_CourseAccess] PRIMARY KEY CLUSTERED
(
      [AccessID] 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].[CourseAccess]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseAccess_Courses] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Courses] ([CourseID])
GO
ALTER TABLE [dbo].[CourseAccess] CHECK CONSTRAINT
[FK_CourseAccess_Courses]
GO
ALTER TABLE [dbo].[CourseAccess]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseAccess_Members] FOREIGN KEY([MemberID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[CourseAccess] CHECK CONSTRAINT
[FK_CourseAccess_Members]
GO
```

- ## CourseAttendance

Przechowuje informacje o obecności na poszczególnych modułach kursu, dzięki czemu możliwe jest sprawdzenie, czy uczestnik zaliczył dany kurs

1. *AttendanceID* (PK)
2. *AccessID* (FK: CourseAccess)
3. *CourseModuleID* (FK: CourseModules)

```
CREATE TABLE [dbo].[CourseAttendance](
	[AttendanceID] [bigint] IDENTITY(1,1) NOT NULL,
	[AccessID] [bigint] NOT NULL,
	[CourseModuleID] [bigint] NOT NULL,
 CONSTRAINT [PK_CourseAttendance] PRIMARY KEY CLUSTERED
(
	[AttendanceID] 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].[CourseAttendance]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseAttendance_CourseAccess] FOREIGN KEY([AccessID])
REFERENCES [dbo].[CourseAccess] ([AccessID])
GO

ALTER TABLE [dbo].[CourseAttendance] CHECK CONSTRAINT
[FK_CourseAttendance_CourseAccess]
GO

ALTER TABLE [dbo].[CourseAttendance]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseAttendance_CourseModules] FOREIGN KEY([CourseModuleID])
REFERENCES [dbo].[CourseModules] ([CourseModuleID])
GO

ALTER TABLE [dbo].[CourseAttendance] CHECK CONSTRAINT
[FK_CourseAttendance_CourseModules]
GO
```

- ## CourseDiplomas

Tabela przechowująca wszystkie dyplomy otrzymane za ukończenie kursów

1. *DiplomaID* (FK: Diplomas)
2. *CourseID* (FK: Courses) - ID ukończonego kursu

```
CREATE TABLE [dbo].[CourseDiplomas](
	[DiplomaID] [bigint] NOT NULL,
	[CourseID] [bigint] NOT NULL,
 CONSTRAINT [PK_CourseDiplomas] PRIMARY KEY CLUSTERED
(
	[DiplomaID] 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].[CourseDiplomas]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseDiplomas_Courses] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Courses] ([CourseID])
GO
ALTER TABLE [dbo].[CourseDiplomas] CHECK CONSTRAINT
[FK_CourseDiplomas_Courses]
GO
```

CourseModules

Zawiera nazwy poszczególnych modułów z kursu. Po połączeniu się z tabelami Courses, EducationForms oraz CourseModuleTeachers można uzyskać informację kolejno, do którego kursu należy dany moduł, w jakiej formie odbywają się zajęcia oraz którzy nauczyciele je prowadzą

1. *CourseModuleID* (PK)
2. *CourseID* (FK: Courses) - kurs, do którego odnosi się moduł
3. *Name* - nazwa modułu
4. *EducationFormID* (FK: EducationForms) - ID formy modułu

```
CREATE TABLE [dbo].[CourseModules](
	[CourseModuleID] [bigint] NOT NULL,
	[CourseID] [bigint] NOT NULL,
	[Name] [nvarchar](200) NOT NULL,
	[EducationFormID] [tinyint] 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]
GO
ALTER TABLE [dbo].[CourseModules]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseModules_Courses] FOREIGN KEY([CourseID])
REFERENCES [dbo].[Courses] ([CourseID])
```

```
GO
ALTER TABLE [dbo].[CourseModules] CHECK CONSTRAINT
[FK_CourseModules_Courses]
GO
ALTER TABLE [dbo].[CourseModules]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseModules_EducationForms] FOREIGN KEY([EducationFormID])
REFERENCES [dbo].[EducationForms] ([EducationFormID])
GO
ALTER TABLE [dbo].[CourseModules] CHECK CONSTRAINT
[FK_CourseModules_EducationForms]
GO
```

- ## CourseModuleTeachers

Pozwala na sprawdzenie, którzy nauczyciele prowadzą dany kurs

1. *EntryID* (PK)
2. *CourseModuleID* (FK: CourseModules)
3. *TeacherID* (FK: Members)

```
CREATE TABLE [dbo].[CourseModuleTeachers](
      [EntryID] [bigint] IDENTITY(1,1) NOT NULL,
      [CourseModuleID] [bigint] NOT NULL,
      [TeacherID] [bigint] NOT NULL,
 CONSTRAINT [PK_CourseModuleTeachers] PRIMARY KEY CLUSTERED
(
      [EntryID] 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].[CourseModuleTeachers]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseModulesTeachers_CourseModules] FOREIGN KEY([CourseModuleID])
REFERENCES [dbo].[CourseModules] ([CourseModuleID])
GO
ALTER TABLE [dbo].[CourseModuleTeachers] CHECK CONSTRAINT
[FK_CourseModulesTeachers_CourseModules]
GO
ALTER TABLE [dbo].[CourseModuleTeachers]  WITH NOCHECK ADD  CONSTRAINT
[FK_CourseModulesTeachers_Members] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[CourseModuleTeachers] CHECK CONSTRAINT
[FK_CourseModulesTeachers_Members]
GO
```

- Courses

Wszystkie kursy oferowane przez firmę, wraz z informacją o koordynatorze i odniesieniem do produktu w sklepie

1. *CourseID* (PK)
2. *CoordinatorID* (FK: Members) - koordynator kursu
3. *ProductID* (FK: Product) - produkt w sklepie

```
CREATE TABLE [dbo].[Courses](
	[CourseID] [bigint] IDENTITY(1,1) NOT NULL,
	[CoordinatorID] [bigint] NOT NULL,
	[ProductID] [bigint] NOT NULL,
 CONSTRAINT [PK_Courses] PRIMARY KEY CLUSTERED
(
	[CourseID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Courses]  WITH NOCHECK ADD  CONSTRAINT
[FK_Courses_Members] FOREIGN KEY([CoordinatorID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [FK_Courses_Members]
GO
ALTER TABLE [dbo].[Courses]  WITH NOCHECK ADD  CONSTRAINT
[FK_Courses_Members1] FOREIGN KEY([CoordinatorID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [FK_Courses_Members1]
GO
ALTER TABLE [dbo].[Courses]  WITH NOCHECK ADD  CONSTRAINT
[FK_Courses_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO
ALTER TABLE [dbo].[Courses] CHECK CONSTRAINT [FK_Courses_Products]
GO
```

- Credentials

Poświadczenia użytkowników systemu. Każde poświadczenie musi być aktywowane kodem, w postaci UUID wysyłanym do użytkownika mailem, w odpowiednich ramach czasowych. Wtedy też użytkownik podaje swoje hasło dostępu. Poświadczenie wygasa po ustalonej dacie, może być też wygaszone ręcznie, przez administratora, lub poprzez zmianę hasła

1. *CredentialID* (PK)
2. *Password* - hasło użytkownika, gdy NULL to nie ustawione
3. *VerificationCode* - kod weryfikacyjny widoczny dla użytkownika zakodowany jako URL do ustawienia hasła
4. *VerficationDeadline* - czas wygaśnięcia zmiany hasła/weryfikacji
5. *IsValid* - czy dane poświadczenia są ważne
6. *ExpireDate* - czas wygaśnięcia poświadczenia

```
CREATE TABLE [dbo].[Credentials](
      [CredentialID] [bigint] IDENTITY(1,1) NOT NULL,
      [Password] [varchar](80) NULL,
      [VerificationCode] [uniqueidentifier] NOT NULL,
      [VerificationDeadline] [datetime] NOT NULL,
      [IsValid] [bit] NOT NULL,
      [ExpireDate] [date] NULL,
 CONSTRAINT [PK_Credentials] PRIMARY KEY CLUSTERED
(
      [CredentialID] 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].[Credentials] ADD  CONSTRAINT [DF_Credentials_Password]
DEFAULT (NULL) FOR [Password]
GO
ALTER TABLE [dbo].[Credentials] ADD  CONSTRAINT
[DF_Credentials_VerificationCode]  DEFAULT (newid()) FOR
[VerificationCode]
GO
ALTER TABLE [dbo].[Credentials] ADD  CONSTRAINT [DF_Credentials_IsValid]
DEFAULT ((1)) FOR [IsValid]
GO
```

- ## Diplomas

Dyplomy wydane przez sekretariat, mogą dotyczyć kursu bądź kierunku studiów. Przechowują informację dotyczącą odbiorcy i daty wydania

1. *DiplomaID* (PK)
2. *MemberID* (FK: Members) - właściciel dyplomu
3. *IssueDate* - data wydania dyplomu

```
CREATE TABLE [dbo].[Diplomas](
	[DiplomaID] [bigint] IDENTITY(1,1) NOT NULL,
	[MemberID] [bigint] NOT NULL,
	[IssueDate] [date] NOT NULL,
 CONSTRAINT [PK_Diplomas] PRIMARY KEY CLUSTERED
(
	[DiplomaID] 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].[Diplomas]  WITH NOCHECK ADD  CONSTRAINT
[FK_Diplomas_Members] FOREIGN KEY([MemberID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Diplomas] CHECK CONSTRAINT [FK_Diplomas_Members]
GO
```

- ## Discounts

Zniżki dla poszczególnych grup użytkowników i produktów

1. *DiscountID* (PK)
2. *ProductID* (FK: Products) - produkt do którego odnosi się zniżka
3. *Discount* - oferowana zniżka
4. *MemberRoleID* (FK: MemberRoles) - role użytkowników do których zniżka ma zastosowanie

Warunki integralności:

- *Discount* z zakresu [0, 1]

```
CREATE TABLE [dbo].[Discounts](
      [DiscountID] [bigint] IDENTITY(1,1) NOT NULL,
      [ProductID] [bigint] NOT NULL,
      [Discount] [decimal](2, 2) NOT NULL,
      [MemberRoleID] [tinyint] NOT NULL,
 CONSTRAINT [PK_Discounts] PRIMARY KEY CLUSTERED
(
      [DiscountID] 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].[Discounts]  WITH NOCHECK ADD  CONSTRAINT
[FK_Discounts_MembersRoles] FOREIGN KEY([MemberRoleID])
REFERENCES [dbo].[MemberRoles] ([MemberRoleID])
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [FK_Discounts_MembersRoles]
GO

ALTER TABLE [dbo].[Discounts]  WITH NOCHECK ADD  CONSTRAINT
[FK_Discounts_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [FK_Discounts_Products]
GO

ALTER TABLE [dbo].[Discounts]  WITH NOCHECK ADD  CONSTRAINT
[CK_Discounts_Discount] CHECK  (([Discount]>=(0) AND [Discount]<=(1)))
GO

ALTER TABLE [dbo].[Discounts] CHECK CONSTRAINT [CK_Discounts_Discount]
GO
```

- ## EducationForms

Słownik zawierający formy kursów/studiów: Stationary, Hybrid, Online Synchronous, Online Asynchronous. Używana przez Subjects i CourseModules do określenia ich formy

1. *EducationFormID* (PK)
2. *TypeName* - typ nauczania

```
CREATE TABLE [dbo].[EducationForms](
      [EducationFormID] [tinyint] IDENTITY(1,1) NOT NULL,
      [TypeName] [char](40) NOT NULL,
 CONSTRAINT [PK_CourseTypes] PRIMARY KEY CLUSTERED
(
      [EducationFormID] 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
```

Groups

Poszczególne grupy zajęciowe. Zawierają informacje na temat przedmiotu i prowadzącego

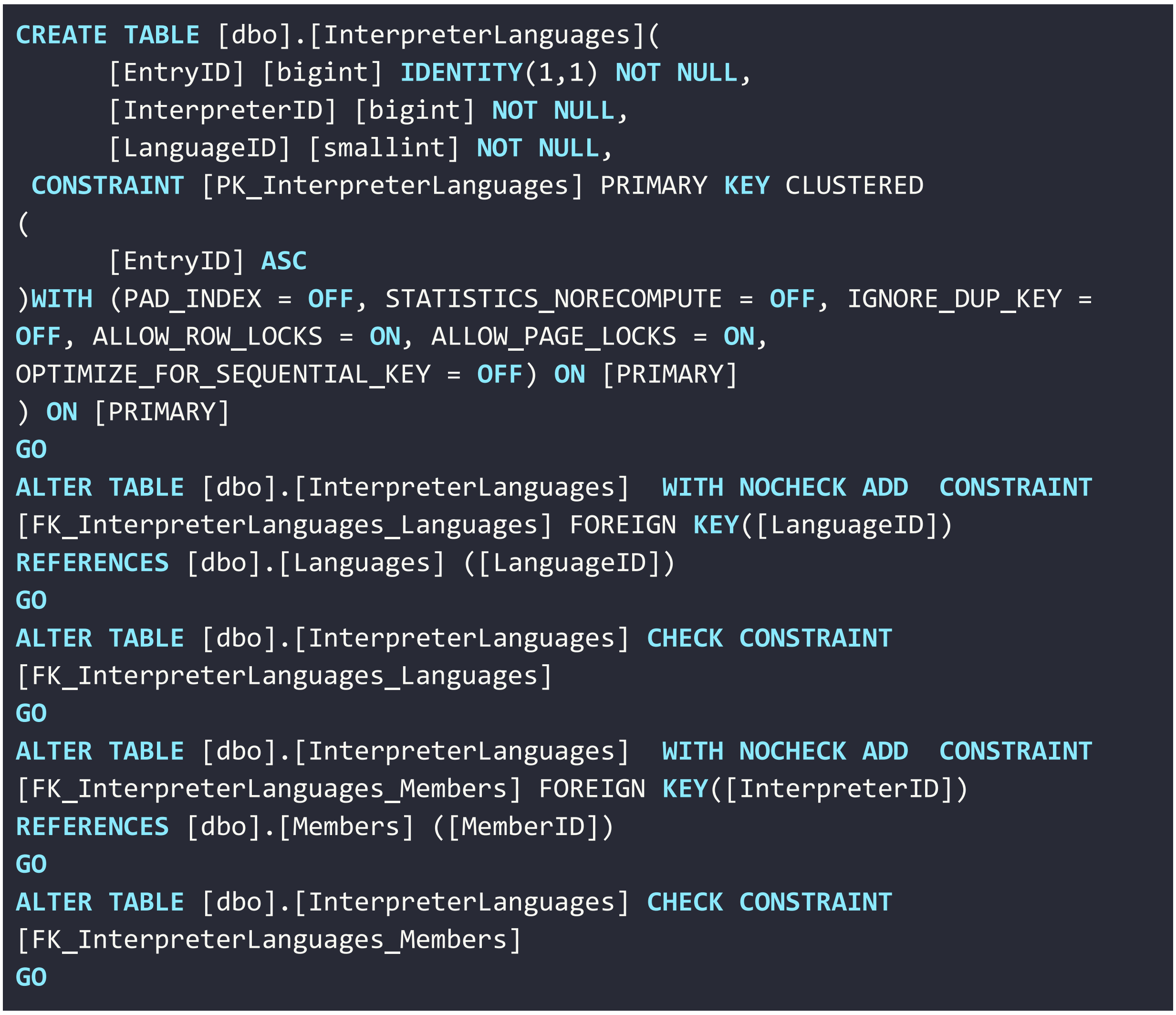
1. *GroupID* (PK)
2. *GroupNumber* - numer grupy zajęciowej
3. *TeacherID* (FK: Members) - prowadzący grupę

```
CREATE TABLE [dbo].[Groups](
      [GroupID] [bigint] IDENTITY(1,1) NOT NULL,
      [GroupNumber] [smallint] NOT NULL,
      [TeacherID] [bigint] NOT NULL,
 CONSTRAINT [PK_Groups] PRIMARY KEY CLUSTERED
(
      [GroupID] 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].[Groups]  WITH NOCHECK ADD  CONSTRAINT
[FK_Groups_Members] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Groups] CHECK CONSTRAINT [FK_Groups_Members]
GO
```

InterpreterLanguages

Zapisuje języki znane przez tłumaczy

1. *EntryID* (PK)
2. *InterpreterID* (FK: Members) - tłumacz
3. *LanguageID* (FK: Languages) - język



```
CREATE TABLE [dbo].[InterpreterLanguages](
      [EntryID] [bigint] IDENTITY(1,1) NOT NULL,
      [InterpreterID] [bigint] NOT NULL,
      [LanguageID] [smallint] NOT NULL,
 CONSTRAINT [PK_InterpreterLanguages] PRIMARY KEY CLUSTERED
(
      [EntryID] 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].[InterpreterLanguages]  WITH NOCHECK ADD  CONSTRAINT
[FK_InterpreterLanguages_Languages] FOREIGN KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
GO
ALTER TABLE [dbo].[InterpreterLanguages] CHECK CONSTRAINT
[FK_InterpreterLanguages_Languages]
GO
ALTER TABLE [dbo].[InterpreterLanguages]  WITH NOCHECK ADD  CONSTRAINT
[FK_InterpreterLanguages_Members] FOREIGN KEY([InterpreterID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[InterpreterLanguages] CHECK CONSTRAINT
[FK_InterpreterLanguages_Members]
GO
```

- ## Internships

Informacje na temat praktyk odbywanych przez studentów, firmy udzielającej praktyk, daty rozpoczęcia i zakończenia

1. *InternshipID* (PK)
2. *MajorID* (FK: Majors) - kierunek studiów
3. *CompanyID* (FK: Companies) - firma, w której odbywają się praktyki
4. *StudentID* (FK: Members) - student odbywający praktykę
5. *StartDate* - data rozpoczęcia praktyki
6. *EndDate* - data zakończenia praktyki

Warunki integralności:

- *EndDate > StartDate*

```
CREATE TABLE [dbo].[Internships](
      [InternshipID] [bigint] NOT NULL,
      [MajorID] [bigint] NOT NULL,
      [CompanyID] [bigint] NOT NULL,
      [StudentID] [bigint] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NOT NULL,
 CONSTRAINT [PK_Interships] 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]  WITH CHECK ADD  CONSTRAINT
[FK_Interships_Companies] FOREIGN KEY([CompanyID])
REFERENCES [dbo].[Companies] ([CompanyID])
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Interships_Companies]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT
[FK_Interships_Majors] FOREIGN KEY([MajorID])
REFERENCES [dbo].[Majors] ([MajorID])
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Interships_Majors]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT
[FK_Interships_Members] FOREIGN KEY([StudentID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [FK_Interships_Members]
GO
ALTER TABLE [dbo].[Internships]  WITH CHECK ADD  CONSTRAINT
[CK_Internships_EndDate] CHECK  (([EndDate]>[StartDate]))
GO
ALTER TABLE [dbo].[Internships] CHECK CONSTRAINT [CK_Internships_EndDate]
GO
```

- Languages

Słownik przechowujący języki

1. *LanguageID* (PK)
2. *Name* - nazwa języka

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

- MajorDiplomas

Tabela przechowująca wszystkie dyplomy otrzymane za ukończenie studiów

1. *DiplomaID* (FK: Diplomas)
2. *MajorID* (FK: Majors) - ID ukończonego kierunku

```
CREATE TABLE [dbo].[MajorDiplomas](
      [DiplomaID] [bigint] NOT NULL,
      [MajorID] [bigint] NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[MajorDiplomas]  WITH CHECK ADD  CONSTRAINT
[FK_MajorDiplomas_Diplomas] FOREIGN KEY([DiplomaID])
REFERENCES [dbo].[Diplomas] ([DiplomaID])
GO
ALTER TABLE [dbo].[MajorDiplomas] CHECK CONSTRAINT
[FK_MajorDiplomas_Diplomas]
GO
ALTER TABLE [dbo].[MajorDiplomas]  WITH NOCHECK ADD  CONSTRAINT
[FK_MajorDiplomas_Majors] FOREIGN KEY([MajorID])
REFERENCES [dbo].[Majors] ([MajorID])
GO
ALTER TABLE [dbo].[MajorDiplomas] CHECK CONSTRAINT
[FK_MajorDiplomas_Majors]
GO
```

- ## Majors

Kierunki oferowane w ramach studiów

1. *MajorID* (PK)
2. *CoordinatorID* (FK: Members) - koordynator kierunku
3. *ProductID* (FK: Products) - produkt oferowany w sklepie

```
CREATE TABLE [dbo].[Majors](
      [MajorID] [bigint] IDENTITY(1,1) NOT NULL,
      [CoordinatorID] [bigint] NOT NULL,
      [ProductID] [bigint] NOT NULL,
 CONSTRAINT [PK_Majors] PRIMARY KEY CLUSTERED
(
      [MajorID] 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].[Majors]  WITH NOCHECK ADD  CONSTRAINT
[FK_Majors_Members] FOREIGN KEY([CoordinatorID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Majors] CHECK CONSTRAINT [FK_Majors_Members]
GO
ALTER TABLE [dbo].[Majors]  WITH NOCHECK ADD  CONSTRAINT
[FK_Majors_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO
ALTER TABLE [dbo].[Majors] CHECK CONSTRAINT [FK_Majors_Products]
GO
```

MeetingAccess

Uczniowie zapisani na dany zjazd

1. *AccessID* (PK)
2. *MemberID* (FK: Members) - ID ucznia zapisanego na zjazd
3. *MeetingID* (FK: Products) - ID zjazdu

```
CREATE TABLE [dbo].[MeetingAccess](
      [AccessID] [bigint] IDENTITY(1,1) NOT NULL,
      [MemberID] [bigint] NOT NULL,
      [MeetingID] [bigint] NOT NULL,
 CONSTRAINT [PK_MeetingAccess] PRIMARY KEY CLUSTERED
(
      [AccessID] 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].[MeetingAccess]  WITH CHECK ADD  CONSTRAINT
[FK_MeetingAccess_Meetings] FOREIGN KEY([MeetingID])
REFERENCES [dbo].[Meetings] ([MeetingID])
GO
ALTER TABLE [dbo].[MeetingAccess] CHECK CONSTRAINT
[FK_MeetingAccess_Meetings]
GO
ALTER TABLE [dbo].[MeetingAccess]  WITH CHECK ADD  CONSTRAINT
[FK_MeetingAccess_Members] FOREIGN KEY([MemberID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[MeetingAccess] CHECK CONSTRAINT
[FK_MeetingAccess_Members]
GO
```

MeetingAttendance

Obecność uczniów na zjeździe

1. *AttendanceID* (PK)
2. *AccessID* (FK: Members) - ID zapisu ucznia na zjazd
3. *MemberID* (FK: Products) - ID ucznia obecnego na zjeździe

```
CREATE TABLE [dbo].[MeetingAttendance](
      [AttendanceID] [bigint] IDENTITY(1,1) NOT NULL,
      [AccessID] [bigint] NOT NULL,
      [MemberID] [bigint] NOT NULL,
 CONSTRAINT [PK_MeetingAttendance] PRIMARY KEY CLUSTERED
(
      [AttendanceID] 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].[MeetingAttendance]  WITH CHECK ADD  CONSTRAINT
[FK_MeetingAttendance_MeetingAccess] FOREIGN KEY([AccessID])
REFERENCES [dbo].[MeetingAccess] ([AccessID])
GO
ALTER TABLE [dbo].[MeetingAttendance] CHECK CONSTRAINT
[FK_MeetingAttendance_MeetingAccess]
GO
ALTER TABLE [dbo].[MeetingAttendance]  WITH CHECK ADD  CONSTRAINT
[FK_MeetingAttendance_Members] FOREIGN KEY([MemberID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[MeetingAttendance] CHECK CONSTRAINT
[FK_MeetingAttendance_Members]
GO
```

- ## Meetings

Tabela zawierająca informacje o zjazdach organizowanych na studiach

1. *MeetingID* (PK)
2. *MajorID* (FK: Majors) - kierunek studiów, na którym organizowany jest zjazd
3. *StartDate* - data rozpoczęcia zjazdu
4. *EndDate* - data zakończenia zjazdu
5. *ProductID* (FK: Products) - produkt oferowany w sklepie

```
CREATE TABLE [dbo].[Meetings](
      [MeetingID] [bigint] IDENTITY(1,1) NOT NULL,
      [MajorID] [bigint] NOT NULL,
      [StartDate] [datetime] NOT NULL,
      [EndDate] [datetime] NOT NULL,
      [ProductID] [bigint] NOT NULL,
 CONSTRAINT [PK_Meetings] PRIMARY KEY CLUSTERED
(
      [MeetingID] 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].[Meetings]  WITH CHECK ADD  CONSTRAINT
[FK_Meetings_Majors] FOREIGN KEY([MajorID])
REFERENCES [dbo].[Majors] ([MajorID])
GO
ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [FK_Meetings_Majors]
GO
ALTER TABLE [dbo].[Meetings]  WITH CHECK ADD  CONSTRAINT
[FK_Meetings_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO
ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [FK_Meetings_Products]
GO
ALTER TABLE [dbo].[Meetings]  WITH CHECK ADD  CONSTRAINT [CK_Meetings]
CHECK  (([StartDate]<[EndDate]))
GO
ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [CK_Meetings]
GO
```

- MemberRoles

Role użytkowników, określające dostęp do różnych funkcji systemu

1. *MemberRoleID* (PK)
2. *RoleName* - nazwa roli

```
CREATE TABLE [dbo].[MemberRoles](
      [MemberRoleID] [tinyint] IDENTITY(1,1) NOT NULL,
      [RoleName] [char](20) NOT NULL,
 CONSTRAINT [PK_MemberRole] PRIMARY KEY CLUSTERED
(
      [MemberRoleID] 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
```

- Members

Użytkownicy systemu wraz z danymi personalnymi

1. *MemberID* (PK)
2. *FirstName* - imię użytkownika
3. *LastName* - nazwisko użytkownika
4. *Login* - login użytkownika
5. *Email* - email użytkownika
6. *AddressID* (FK: Addresses) - adres zamieszkania użytkownika
7. *CredentialsID* (FK: Credentials) - aktywne poświadczenie użytkownika
8. *IsActive* - czy konto jest aktywne

Warunki integralności:

- *Email* musi zawierać ‘@’, a po niej znak ‘.’
- *Login* unikalny w tabeli

```
CREATE TABLE [dbo].[Members](
      [MemberID] [bigint] IDENTITY(1,1) NOT NULL,
      [FirstName] [nvarchar](40) NOT NULL,
      [LastName] [nvarchar](40) NOT NULL,
      [Login] [nchar](80) NOT NULL,
      [Email] [nvarchar](160) NOT NULL,
      [AddressID] [bigint] NOT NULL,
      [CredentialsID] [bigint] NOT NULL,
      [IsActive] [bit] NOT NULL,
 CONSTRAINT [PK_PersonalInfo] PRIMARY KEY CLUSTERED
(
      [MemberID] 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 [UK_Members_Email] UNIQUE NONCLUSTERED
(
      [Email] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [UK_Members_Login] UNIQUE NONCLUSTERED
(
      [Login] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Members]  WITH NOCHECK ADD  CONSTRAINT
[FK_Members_Addresses] FOREIGN KEY([AddressID])
REFERENCES [dbo].[Addresses] ([AddressID])
GO
ALTER TABLE [dbo].[Members] CHECK CONSTRAINT [FK_Members_Addresses]
GO
ALTER TABLE [dbo].[Members]  WITH NOCHECK ADD  CONSTRAINT
[FK_Members_Credentials] FOREIGN KEY([CredentialsID])
REFERENCES [dbo].[Credentials] ([CredentialID])
GO
ALTER TABLE [dbo].[Members] CHECK CONSTRAINT [FK_Members_Credentials]
GO
ALTER TABLE [dbo].[Members]  WITH NOCHECK ADD  CONSTRAINT
[CK_Members_Email] CHECK  (([Email] like '%@%.%'))
GO
ALTER TABLE [dbo].[Members] CHECK CONSTRAINT [CK_Members_Email]
GO
ALTER TABLE [dbo].[Members]  WITH NOCHECK ADD  CONSTRAINT
[CK_Members_Name] CHECK  ((patindex('%[^a-zA-Z ]%',[FirstName])=(0) AND
[FirstName] IS NOT NULL))
GO
ALTER TABLE [dbo].[Members] CHECK CONSTRAINT [CK_Members_Name]
GO
```

OrderDetails

Pozycje zamówienia

1. *OrderDetailID* (PK)
2. *OrderID* (FK: Orders) - zamówienie, na którym znajduje się pozycja
3. *ProductID* (FK: Products) - zamówiony produkt

```
CREATE TABLE [dbo].[OrderDetails](
      [OrderDetailID] [bigint] IDENTITY(1,1) NOT NULL,
      [OrderID] [bigint] NOT NULL,
      [ProductID] [bigint] NOT NULL,
 CONSTRAINT [PK_OrderDetails] 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].[OrderDetails]  WITH NOCHECK ADD  CONSTRAINT
[FK_OrderDetails_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT [FK_OrderDetails_Orders]
GO
ALTER TABLE [dbo].[OrderDetails]  WITH NOCHECK ADD  CONSTRAINT
[FK_OrderDetails_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO
ALTER TABLE [dbo].[OrderDetails] CHECK CONSTRAINT
[FK_OrderDetails_Products]
GO
```

Orders

Tabela reprezentująca zamówienia. Zawiera informację o odroczeniu płatności, której może dokonać dyrektor

1. *OrderID* (PK)
2. *OrderDate* - data złożenia zamówienia
3. *MemberID* (FK: Members) - osoba składająca zamówienie

```
CREATE TABLE [dbo].[Orders](
      [OrderID] [bigint] IDENTITY(1,1) NOT NULL,
      [OrderDate] [date] NOT NULL,
      [MemberID] [bigint] NOT NULL,
      [IsInAdvance] [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]  WITH NOCHECK ADD  CONSTRAINT
[FK_Orders_Members] FOREIGN KEY([MemberID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Members]
GO
```

- Payments

Szczegóły dotyczące płatności

1. *PaymentID* (PK)
2. *OrderID* (FK: Orders) - numer zamówienia, którego dotyczy płatność
3. *URL* - link wygenerowany w celu dokonania płatności
4. *PaidTime* - czas, kiedy została dokonana płatność
5. *PaymentStatusID (FK: PaymentStatus)* - informuje o obecnym stanie płatności
6. *IsAdvance* - informacja, czy jest to cała kwota czy tylko zaliczka

```
CREATE TABLE [dbo].[Payments](
      [PaymentID] [bigint] IDENTITY(1,1) NOT NULL,
      [OrderID] [bigint] NOT NULL,
      [URL] [varchar](1600) NOT NULL,
      [PaidTime] [datetime] NOT NULL,
      [PaymentStatusID] [tinyint] NOT NULL,
      [IsAdvance] [bit] NOT NULL,
 CONSTRAINT [PK_Payments] PRIMARY KEY CLUSTERED
(
      [PaymentID] 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].[Payments]  WITH NOCHECK ADD  CONSTRAINT
[FK_Payments_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[Payments] CHECK CONSTRAINT [FK_Payments_Orders]
GO
ALTER TABLE [dbo].[Payments]  WITH NOCHECK ADD  CONSTRAINT
[FK_Payments_PaymentStatus] FOREIGN KEY([PaymentStatusID])
REFERENCES [dbo].[PaymentStatus] ([PaymentStatusID])
GO
ALTER TABLE [dbo].[Payments] CHECK CONSTRAINT [FK_Payments_PaymentStatus]
GO
```

- PaymentStatus

Słownik stanów płatności

1. *PaymentStatusID* (PK)
2. *StatusName* - informuje o stanie płatności

```
CREATE TABLE [dbo].[PaymentStatus](
      [PaymentStatusID] [tinyint] IDENTITY(1,1) NOT NULL,
      [StatusName] [char](20) NOT NULL,
 CONSTRAINT [PK_PaymentStatus] PRIMARY KEY CLUSTERED
(
      [PaymentStatusID] 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
```

Postponements

Przechowuje szczegóły odroczenia płatności

1. *PostponementID* (PK)
2. *OrderID* (FK: Orders) - odroczona płatność dla danego zamówienia
3. *PostponeStartDate* - data początkowa odroczenia
4. *PostponeEndDate* - data końcowa odroczenia

```
CREATE TABLE [dbo].[Postponements](
	[PostponementID] [bigint] IDENTITY(1,1) NOT NULL,
	[OrderID] [bigint] NOT NULL,
	[PostponeStartDate] [datetime] NOT NULL,
	[PostponeEndDate] [datetime] NOT NULL,
 CONSTRAINT [PK_Postponements] PRIMARY KEY CLUSTERED
(
	[PostponementID] 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].[Postponements]  WITH NOCHECK ADD  CONSTRAINT
[FK_Postponements_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
GO
ALTER TABLE [dbo].[Postponements] CHECK CONSTRAINT
[FK_Postponements_Orders]
GO
ALTER TABLE [dbo].[Postponements]  WITH NOCHECK ADD  CONSTRAINT
[CK_Postponements] CHECK  (([PostponeEndDate]>[PostponeStartDate]))
GO
ALTER TABLE [dbo].[Postponements] CHECK CONSTRAINT [CK_Postponements]
GO
```

Products

Produkty dostępne do kupienia, takie jak dostęp do webinaru, zapis na kurs, studia, płatności za zjazdy na studia itp.

1. *ProductID* (PK)
2. *Name* - nazwa produktu
3. *Description* - opis produktu
4. *CategoryID* (FK: Categories) - kategoria do jakiej należy produkt
5. *Price* - cena produktu
6. *AdvancePrice* - jeśli nie jest nullem, to oznacza, że dostępna jest opcja zapłacenia zaliczki oraz informuje o jej wysokości

Warunki integralności:

- *AdvancePrice* default = NULL

```
CREATE TABLE [dbo].[Products](
      [ProductID] [bigint] IDENTITY(1,1) NOT NULL,
      [Name] [nvarchar](200) NOT NULL,
      [Description] [nvarchar](max) NOT NULL,
      [CategoryID] [tinyint] NOT NULL,
      [Price] [money] NOT NULL,
      [AdvancePrice] [money] 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] TEXTIMAGE_ON [PRIMARY]
GO
ALTER TABLE [dbo].[Products] ADD  CONSTRAINT [DF_Products_AdvancePrice]
DEFAULT (NULL) FOR [AdvancePrice]
GO
ALTER TABLE [dbo].[Products]  WITH NOCHECK ADD  CONSTRAINT
[FK_Products_Categories] FOREIGN KEY([CategoryID])
REFERENCES [dbo].[Categories] ([CategoryID])
GO
ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [FK_Products_Categories]
GO
ALTER TABLE [dbo].[Products]  WITH NOCHECK ADD  CONSTRAINT
[CK_Products_AdvancePrice] CHECK  (([AdvancePrice]>(0) OR [AdvancePrice]
IS NULL))
GO
ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [CK_Products_AdvancePrice]
GO
ALTER TABLE [dbo].[Products]  WITH NOCHECK ADD  CONSTRAINT
[CK_Products_Price] CHECK  (([Price]>=(0)))
GO
ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [CK_Products_Price]
GO
ALTER TABLE [dbo].[Products]  WITH NOCHECK ADD  CONSTRAINT
[CK_Products_Prices] CHECK  (([AdvancePrice]<[Price]))
GO
ALTER TABLE [dbo].[Products] CHECK CONSTRAINT [CK_Products_Prices]
GO
```

Recordings

Linki do nagrań przechowywanych poza bazą

1. *RecordingID* (PK)
2. *EventID* (FK: ScheduleEvents) - wydarzenie, z którego pochodzi nagranie
3. *URL* - link do nagrania
4. *Title* - tytuł nagrania

```
CREATE TABLE [dbo].[Recordings](
      [RecordingID] [bigint] IDENTITY(1,1) NOT NULL,
      [EventID] [bigint] NOT NULL,
      [URL] [nvarchar](1800) NOT NULL,
      [Title] [nvarchar](200) NOT NULL,
 CONSTRAINT [PK_Recordings] PRIMARY KEY CLUSTERED
(
      [RecordingID] 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].[Recordings]  WITH NOCHECK ADD  CONSTRAINT
[FK_Recordings_ScheduleEvents] FOREIGN KEY([EventID])
REFERENCES [dbo].[ScheduleEvents] ([EventID])
GO
ALTER TABLE [dbo].[Recordings] CHECK CONSTRAINT
[FK_Recordings_ScheduleEvents]
GO
```

Rooms

Opisy sal znajdujących się w budynkach

1. *RoomID* (PK)
2. *RoomNumber* - numer pokoju
3. *Floor* - piętro, na którym znajduje się pokój
4. *BuildingID* (FK: Buildings) - budynek, w którym jest sala
5. *Seats* - liczba dostępnych miejsc na sali

Warunki integralności:

- Numer pokoju zaczyna się od cyfry
- Liczba miejsc jest nieujemna

```
CREATE TABLE [dbo].[Rooms](
	[RoomID] [bigint] IDENTITY(1,1) NOT NULL,
	[RoomNumber] [char](20) NOT NULL,
	[Floor] [smallint] NOT NULL,
	[BuildingID] [bigint] NOT NULL,
	[Seats] [int] NOT NULL,
 CONSTRAINT [PK_Rooms] PRIMARY KEY CLUSTERED
(
	[RoomID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Rooms]  WITH NOCHECK ADD  CONSTRAINT
[FK_Rooms_Buildings] FOREIGN KEY([BuildingID])
REFERENCES [dbo].[Buildings] ([BuildingID])
GO
ALTER TABLE [dbo].[Rooms] CHECK CONSTRAINT [FK_Rooms_Buildings]
GO
ALTER TABLE [dbo].[Rooms]  WITH NOCHECK ADD  CONSTRAINT [CK_Rooms] CHECK
(([RoomNumber] like '[1-9]%'))
GO
ALTER TABLE [dbo].[Rooms] CHECK CONSTRAINT [CK_Rooms]
GO
ALTER TABLE [dbo].[Rooms]  WITH CHECK ADD  CONSTRAINT [CK_Seats] CHECK
(([Seats]>(0)))
GO
ALTER TABLE [dbo].[Rooms] CHECK CONSTRAINT [CK_Seats]
GO
```

- ## ScheduleEvents

Harmonogram wydarzeń, takich jak webinary czy kursy

1. *EventID* (PK)
2. *StartDate* - data rozpoczęcia wydarzenia
3. *EndDate* - data zakończenia wydarzenia
4. *RoomID* (FK: Rooms) - pokój, w którym odbywa się wydarzenie
5. *InterpreterID* (FK: Members) - tłumacz przypisany do wydarzenia
6. *LanguageID* (FK: Languages) - język, w jakim prowadzone jest wydarzenie
7. *IsCanceled* - wartość prawda/fałsz, czy dane wydarzenie jest aktualne czy odwołane

Warunki integralności:

- data zakończenia nie jest wcześniejsza niż data rozpoczęcia wydarzenia
- *RecordingID* default = NULL
- *InterpreterID* default = NULL

```
CREATE TABLE [dbo].[ScheduleEvents](
	[EventID] [bigint] IDENTITY(1,1) NOT NULL,
	[StartDate] [datetime] NOT NULL,
	[EndDate] [datetime] NOT NULL,
	[RoomID] [bigint] NULL,
	[InterpreterID] [bigint] NULL,
	[LanguageID] [smallint] NOT NULL,
	[IsCanceled] [bit] NOT NULL,
 CONSTRAINT [PK_Schedule] PRIMARY KEY CLUSTERED
(
	[EventID] 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].[ScheduleEvents] ADD  CONSTRAINT
[DF_ScheduleEvents_InterpeterID]  DEFAULT (NULL) FOR [InterpreterID]
GO
ALTER TABLE [dbo].[ScheduleEvents] ADD  CONSTRAINT
[DF_ScheduleEvents_IsCancelled]  DEFAULT ((0)) FOR [IsCanceled]
GO
ALTER TABLE [dbo].[ScheduleEvents]  WITH NOCHECK ADD  CONSTRAINT
[FK_Schedule_Rooms] FOREIGN KEY([RoomID])
REFERENCES [dbo].[Rooms] ([RoomID])
GO
ALTER TABLE [dbo].[ScheduleEvents] CHECK CONSTRAINT [FK_Schedule_Rooms]
GO
ALTER TABLE [dbo].[ScheduleEvents]  WITH NOCHECK ADD  CONSTRAINT
[FK_ScheduleEvents_Classes] FOREIGN KEY([EventID])
REFERENCES [dbo].[Classes] ([ClassID])
GO
ALTER TABLE [dbo].[ScheduleEvents] NOCHECK CONSTRAINT
[FK_ScheduleEvents_Classes]
GO
ALTER TABLE [dbo].[ScheduleEvents]  WITH NOCHECK ADD  CONSTRAINT
[FK_ScheduleEvents_CourseModules] FOREIGN KEY([EventID])
REFERENCES [dbo].[CourseModules] ([CourseModuleID])
GO
ALTER TABLE [dbo].[ScheduleEvents] NOCHECK CONSTRAINT
[FK_ScheduleEvents_CourseModules]
GO
ALTER TABLE [dbo].[ScheduleEvents]  WITH CHECK ADD  CONSTRAINT
[FK_ScheduleEvents_Languages] FOREIGN KEY([LanguageID])
REFERENCES [dbo].[Languages] ([LanguageID])
GO
```

```
ALTER TABLE [dbo].[ScheduleEvents] CHECK CONSTRAINT
[FK_ScheduleEvents_Languages]
GO
ALTER TABLE [dbo].[ScheduleEvents]  WITH NOCHECK ADD  CONSTRAINT
[FK_ScheduleEvents_Members] FOREIGN KEY([InterpreterID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[ScheduleEvents] CHECK CONSTRAINT
[FK_ScheduleEvents_Members]
GO
ALTER TABLE [dbo].[ScheduleEvents]  WITH NOCHECK ADD  CONSTRAINT
[FK_ScheduleEvents_Webinars] FOREIGN KEY([EventID])
REFERENCES [dbo].[Webinars] ([WebinarID])
GO
ALTER TABLE [dbo].[ScheduleEvents] NOCHECK CONSTRAINT
[FK_ScheduleEvents_Webinars]
GO
ALTER TABLE [dbo].[ScheduleEvents]  WITH NOCHECK ADD  CONSTRAINT
[CK_ScheduleEvents_EndDate] CHECK  (([EndDate]>[StartDate]))
GO
ALTER TABLE [dbo].[ScheduleEvents] CHECK CONSTRAINT
[CK_ScheduleEvents_EndDate]
GO
```

- StudentAttendance

Dziennik obecności uczniów na zajęciach na studiach

1. *AttendanceID* (PK)
2. *StudentEnrollID* (FK: StudentEnrolls) - grupa zajęciowa
3. *ClassID* (FK: Classes) - klasa, której dotyczy obecność

```
CREATE TABLE [dbo].[StudentAttendance](
	[AttendanceID] [bigint] IDENTITY(1,1) NOT NULL,
	[StudentEnrollID] [bigint] NOT NULL,
	[ClassID] [bigint] NOT NULL,
 CONSTRAINT [PK_StudentAttendance] PRIMARY KEY CLUSTERED
(
	[AttendanceID] 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].[StudentAttendance]  WITH NOCHECK ADD  CONSTRAINT
[FK_StudentAttendance_Classes] FOREIGN KEY([ClassID])
REFERENCES [dbo].[Classes] ([ClassID])
GO
ALTER TABLE [dbo].[StudentAttendance] CHECK CONSTRAINT
[FK_StudentAttendance_Classes]
GO
ALTER TABLE [dbo].[StudentAttendance]  WITH NOCHECK ADD  CONSTRAINT
[FK_StudentAttendance_StudentEnrolls] FOREIGN KEY([StudentEnrollID])
REFERENCES [dbo].[StudentEnrolls] ([StudentEnrollID])
GO
ALTER TABLE [dbo].[StudentAttendance] CHECK CONSTRAINT
[FK_StudentAttendance_StudentEnrolls]
GO
```

StudentEnrolls

Połączenie ucznia z konkretnymi zajęciami, zawiera jego oceny końcowe

1. *StudentEnrollID* (PK)
2. *StudentID* (FK: Members) - uczeń
3. *GroupID* (FK: Groups) - grupa, do której przypisany jest uczeń
4. *FinalGrade* - ocena końcowa ucznia
5. *ExamGrade* - ocena z egzaminu
6. *LectureGrade* - ocena z zajęć

Warunki integralności:

- *FinalGrade* default = NULL
- *ExamGrade* default = NULL
- *LectureGrade* default = NULL
- *FinalGrade*, *ExamGrade* oraz *LectureGrade* mogą być liczbą ze zbioru {2, 2.5, 3, 3.5, 4, 4.5, 5}

```
CREATE TABLE [dbo].[StudentEnrolls](
      [StudentEnrollID] [bigint] IDENTITY(1,1) NOT NULL,
      [StudentID] [bigint] NOT NULL,
      [GroupID] [bigint] NOT NULL,
      [FinalGrade] [decimal](2, 1) NULL,
      [ExamGrade] [decimal](2, 1) NULL,
      [LectureGrade] [decimal](2, 1) NULL,
 CONSTRAINT [PK_StudentRecords] PRIMARY KEY CLUSTERED
(
      [StudentEnrollID] 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].[StudentEnrolls] ADD  CONSTRAINT
[DF_StudentGroupLinks_FinalGrade]  DEFAULT (NULL) FOR [FinalGrade]
GO
ALTER TABLE [dbo].[StudentEnrolls] ADD  CONSTRAINT
[DF_StudentGroupLinks_ExamGrade]  DEFAULT (NULL) FOR [ExamGrade]
GO
ALTER TABLE [dbo].[StudentEnrolls] ADD  CONSTRAINT
[DF_StudentGroupLinks_LectureGrade]  DEFAULT (NULL) FOR [LectureGrade]
GO
ALTER TABLE [dbo].[StudentEnrolls]  WITH NOCHECK ADD  CONSTRAINT
[FK_StudentGroupLinks_Members] FOREIGN KEY([StudentID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[StudentEnrolls] CHECK CONSTRAINT
[FK_StudentGroupLinks_Members]
GO
ALTER TABLE [dbo].[StudentEnrolls]  WITH NOCHECK ADD  CONSTRAINT
```

```
[FK_StudentRecords_Groups] FOREIGN KEY([GroupID])
REFERENCES [dbo].[Groups] ([GroupID])
GO
ALTER TABLE [dbo].[StudentEnrolls] CHECK CONSTRAINT
[FK_StudentRecords_Groups]
GO
ALTER TABLE [dbo].[StudentEnrolls]  WITH NOCHECK ADD  CONSTRAINT
[CK_StudentGroupLinks_ExamGrade] CHECK  (([ExamGrade]=(5) OR
[ExamGrade]=(4.5) OR [ExamGrade]=(4) OR [ExamGrade]=(3.5) OR
[ExamGrade]=(3) OR [ExamGrade]=(2)))
GO
ALTER TABLE [dbo].[StudentEnrolls] CHECK CONSTRAINT
[CK_StudentGroupLinks_ExamGrade]
GO
ALTER TABLE [dbo].[StudentEnrolls]  WITH NOCHECK ADD  CONSTRAINT
[CK_StudentGroupLinks_FinalGrade] CHECK  (([FinalGrade]=(5) OR
[FinalGrade]=(4.5) OR [FinalGrade]=(4) OR [FinalGrade]=(3.5) OR
[FinalGrade]=(3) OR [FinalGrade]=(2)))
GO
ALTER TABLE [dbo].[StudentEnrolls] CHECK CONSTRAINT
[CK_StudentGroupLinks_FinalGrade]
GO
ALTER TABLE [dbo].[StudentEnrolls]  WITH NOCHECK ADD  CONSTRAINT
[CK_StudentGroupLinks_LectureGrade] CHECK  (([LectureGrade]=(5) OR
[LectureGrade]=(4.5) OR [LectureGrade]=(4) OR [LectureGrade]=(3.5) OR
[LectureGrade]=(3) OR [LectureGrade]=(2)))
GO
ALTER TABLE [dbo].[StudentEnrolls] CHECK CONSTRAINT
[CK_StudentGroupLinks_LectureGrade]
GO
```

StudentGrades

Oceny cząstkowe ucznia na konkretnych zajęciach

1. *GradeID* (PK)
2. *StudenEnrollID* (FK: StudentEnrolls) - zajęcia, na których zdobyta została ocena
3. *Grade* - ocena cząstkowa

Warunki integralności:

- *Grade* może być liczbą ze zbioru {2, 2.5, 3, 3.5, 4, 4.5, 5}

```
CREATE TABLE [dbo].[StudentGrades](
      [GradeID] [bigint] IDENTITY(1,1) NOT NULL,
      [StudentEnrollID] [bigint] NOT NULL,
      [Grade] [decimal](2, 1) NOT NULL,
 CONSTRAINT [PK_StudentGrades] PRIMARY KEY CLUSTERED
(
      [GradeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[StudentGrades]  WITH NOCHECK ADD  CONSTRAINT
[FK_StudentGrades_StudentEnrolls] FOREIGN KEY([StudentEnrollID])
REFERENCES [dbo].[StudentEnrolls] ([StudentEnrollID])
GO
ALTER TABLE [dbo].[StudentGrades] CHECK CONSTRAINT
[FK_StudentGrades_StudentEnrolls]
GO
ALTER TABLE [dbo].[StudentGrades]  WITH NOCHECK ADD  CONSTRAINT
[CK_StudentGrades_Grade] CHECK  (([Grade]=(5) OR [Grade]=(4.5) OR
[Grade]=(4) OR [Grade]=(3.5) OR [Grade]=(3) OR [Grade]=(2)))
GO
ALTER TABLE [dbo].[StudentGrades] CHECK CONSTRAINT
[CK_StudentGrades_Grade]
GO
```

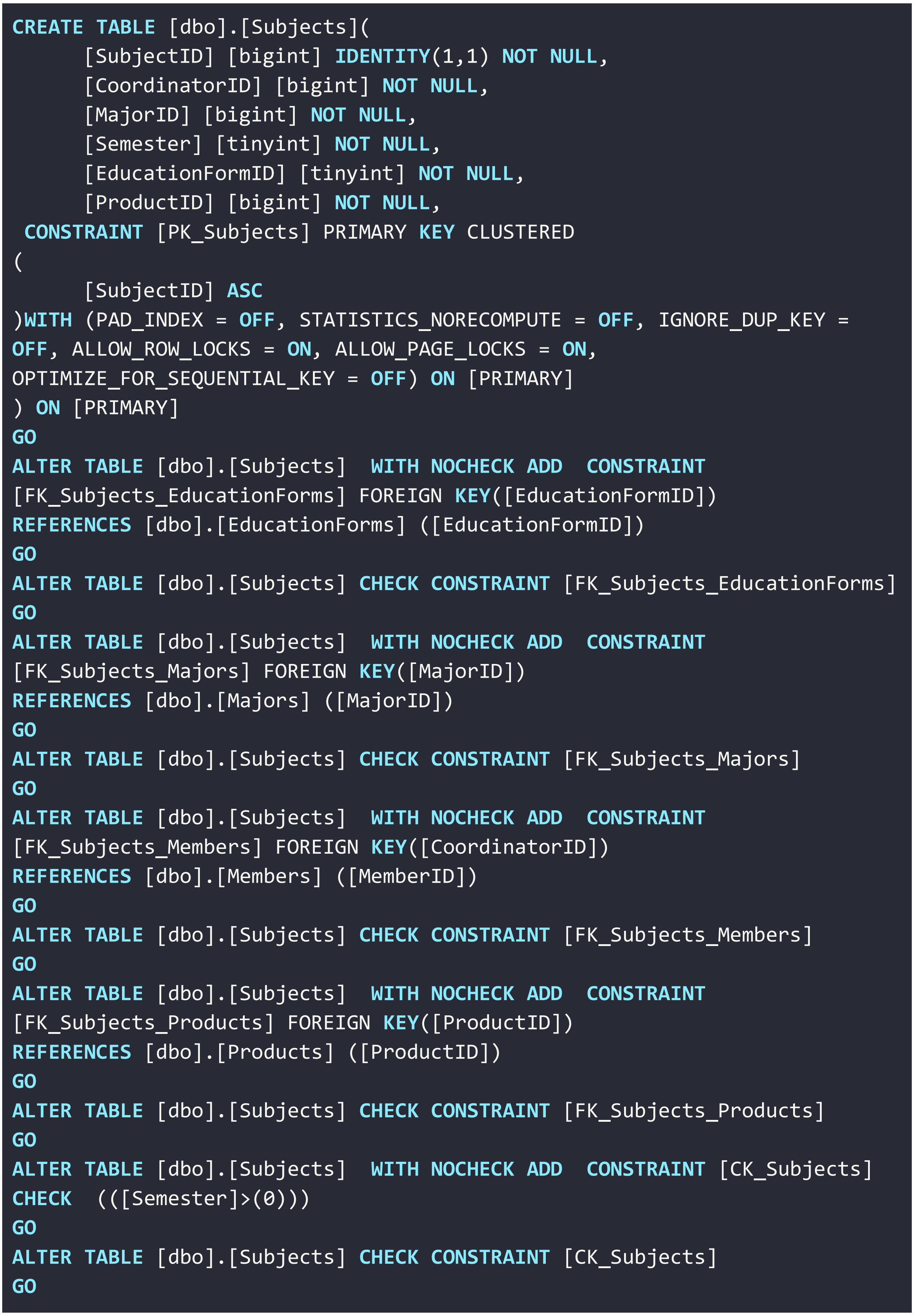
- Subjects

Przedmioty nauczane na studiach

1. *SubjectID* (PK)
2. *CoordinatorID* (FK: Members) - koordynator przedmiotu
3. *MajorID* (FK: Majors) - kierunek, na którym odbywa się przedmiot
4. *Semester* - semestr, na którym odbywa się dany przedmiot
5. *EducationFormID* (FK: EducationForms) - forma odbywania się przedmiotu
6. *ProductID* (FK: Products) - odniesienie do produktu w sklepie

Warunki integralności:

- *InterpreterID* default = NULL
- *Semester* > 0



```
CREATE TABLE [dbo].[Subjects](
      [SubjectID] [bigint] IDENTITY(1,1) NOT NULL,
      [CoordinatorID] [bigint] NOT NULL,
      [MajorID] [bigint] NOT NULL,
      [Semester] [tinyint] NOT NULL,
      [EducationFormID] [tinyint] NOT NULL,
      [ProductID] [bigint] NOT NULL,
 CONSTRAINT [PK_Subjects] PRIMARY KEY CLUSTERED
(
      [SubjectID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Subjects]  WITH NOCHECK ADD  CONSTRAINT
[FK_Subjects_EducationForms] FOREIGN KEY([EducationFormID])
REFERENCES [dbo].[EducationForms] ([EducationFormID])
GO
ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [FK_Subjects_EducationForms]
GO
ALTER TABLE [dbo].[Subjects]  WITH NOCHECK ADD  CONSTRAINT
[FK_Subjects_Majors] FOREIGN KEY([MajorID])
REFERENCES [dbo].[Majors] ([MajorID])
GO
ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [FK_Subjects_Majors]
GO
ALTER TABLE [dbo].[Subjects]  WITH NOCHECK ADD  CONSTRAINT
[FK_Subjects_Members] FOREIGN KEY([CoordinatorID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [FK_Subjects_Members]
GO
ALTER TABLE [dbo].[Subjects]  WITH NOCHECK ADD  CONSTRAINT
[FK_Subjects_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO
ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [FK_Subjects_Products]
GO
ALTER TABLE [dbo].[Subjects]  WITH NOCHECK ADD  CONSTRAINT [CK_Subjects]
CHECK  (([Semester]>(0)))
GO
ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [CK_Subjects]
GO
```

- SubjectTeachers

Nauczyciele przypisani do nauczania przedmiotu

1. *EntryID* (PK)
2. *SubjectID* (FK: Subjects) - nauczany przedmiot
3. *TeacherID* (FK: Members) - prowadzący podpięty do przedmiotu

```
CREATE TABLE [dbo].[SubjectTeachers](
      [EntryID] [bigint] IDENTITY(1,1) NOT NULL,
      [SubjectID] [bigint] NOT NULL,
      [TeacherID] [bigint] NOT NULL,
 CONSTRAINT [PK_SubjectsTeachers] PRIMARY KEY CLUSTERED
(
      [EntryID] 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].[SubjectTeachers]  WITH NOCHECK ADD  CONSTRAINT
[FK_SubjectsTeachers_Members] FOREIGN KEY([TeacherID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[SubjectTeachers] CHECK CONSTRAINT
[FK_SubjectsTeachers_Members]
GO
ALTER TABLE [dbo].[SubjectTeachers]  WITH NOCHECK ADD  CONSTRAINT
[FK_SubjectsTeachers_Subjects] FOREIGN KEY([SubjectID])
REFERENCES [dbo].[Subjects] ([SubjectID])
GO
ALTER TABLE [dbo].[SubjectTeachers] CHECK CONSTRAINT
[FK_SubjectsTeachers_Subjects]
GO
```

- WebinarAccess

Reprezentuje dostępy do webinarów przez użytkowników

1. *AccessID* (PK)
2. *MemberID* (FK: Members) - użytkownik z dostępem
3. *WebinarID* (FK: Webinars) - webinar, do którego przyznano dostęp

```sql
CREATE TABLE [dbo].[WebinarAccess](
	[AccessID] [bigint] IDENTITY(1,1) NOT NULL,
	[MemberID] [bigint] NOT NULL,
	[WebinarID] [bigint] NOT NULL,
 CONSTRAINT [PK_WebinarAccess] PRIMARY KEY CLUSTERED
(
	[AccessID] 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].[WebinarAccess]  WITH NOCHECK ADD  CONSTRAINT
[FK_WebinarAccess_Members] FOREIGN KEY([MemberID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[WebinarAccess] CHECK CONSTRAINT
[FK_WebinarAccess_Members]
GO
ALTER TABLE [dbo].[WebinarAccess]  WITH NOCHECK ADD  CONSTRAINT
[FK_WebinarAccess_Webinars] FOREIGN KEY([WebinarID])
REFERENCES [dbo].[Webinars] ([WebinarID])
GO
ALTER TABLE [dbo].[WebinarAccess] CHECK CONSTRAINT
[FK_WebinarAccess_Webinars]
GO
```

- Webinars

Dane o odbywających się webinarach

1. *WebinarID* (PK)
2. *Name* - nazwa webinaru
3. *Description* - opis webinaru
4. *ProductID* (FK: Products) - odpowiedni produkt w sklepie

```
CREATE TABLE [dbo].[Webinars](
      [WebinarID] [bigint] NOT NULL,
      [ProductID] [bigint] NOT NULL,
      [LeaderID] [bigint] NOT NULL,
 CONSTRAINT [PK_Webinars] PRIMARY KEY CLUSTERED
(
      [WebinarID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY =
OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT
[FK_Webinars_Members] FOREIGN KEY([LeaderID])
REFERENCES [dbo].[Members] ([MemberID])
GO
ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [FK_Webinars_Members]
GO
ALTER TABLE [dbo].[Webinars]  WITH NOCHECK ADD  CONSTRAINT
[FK_Webinars_Products] FOREIGN KEY([ProductID])
REFERENCES [dbo].[Products] ([ProductID])
GO
ALTER TABLE [dbo].[Webinars] CHECK CONSTRAINT [FK_Webinars_Products]
GO
```

Widoki

- ## GetAllEventsSignedUpCount – SM, MB, DM

Wyświetla liczbę osób zapisanych na poszczególne wydarzenia, z rozróżnieniem czy zajęcia odbywają się zdalnie czy stacjonarnie

```
CREATE VIEW [dbo].[GetAllEventsSignedUpCount]
AS
SELECT  COALESCE (P1.Name, P2.Name) AS Name,
COUNT(COALESCE (CA.AccessID, WA.AccessID)) / (CASE WHEN
COUNT(CM.CourseModuleID) > 0 THEN COUNT(CM.CourseModuleID) ELSE 1 END) AS
[Signed up count],
CASE WHEN SE.RoomID IS NOT NULL THEN 'stationary' ELSE 'online' END AS
Type
FROM dbo.ScheduleEvents AS SE
LEFT OUTER JOIN dbo.Webinars AS W ON W.WebinarID = SE.EventID
LEFT OUTER JOIN dbo.CourseModules AS CM ON CM.CourseModuleID = SE.EventID
LEFT OUTER JOIN dbo.Courses AS C ON C.CourseID = CM.CourseID
LEFT OUTER JOIN dbo.CourseAccess AS CA ON CA.CourseID = C.CourseID
LEFT OUTER JOIN dbo.WebinarAccess AS WA ON WA.WebinarID = W.WebinarID
LEFT OUTER JOIN dbo.Products AS P1 ON P1.ProductID = C.ProductID
LEFT OUTER JOIN dbo.Products AS P2 ON P2.ProductID = W.ProductID
GROUP BY COALESCE (P1.ProductID, P2.ProductID), COALESCE (P1.Name,
P2.Name), CASE WHEN SE.RoomID IS NOT NULL THEN 'stationary' ELSE 'online'
END, P1.Name, P2.Name
```

- GetBilocation – SM, MB, DM

Wyświetla listę osób zapisanych na zajęcia , których terminy kolidują ze sobą

```
CREATE VIEW [dbo].[GetBilocation]
AS
SELECT        dbo.Members.FirstName, dbo.Members.LastName,
ScheduleEvents_1.StartDate AS StartFirstEvent,
ScheduleEvents_1.EndDate AS EndFirstEvent,
dbo.ScheduleEvents.StartDate AS StartSecondEvent,
dbo.ScheduleEvents.EndDate AS EndSecondEvent
FROM dbo.Members
INNER JOIN dbo.CourseAccess ON Members.MemberID = CourseAccess.MemberID
INNER JOIN Courses ON dbo.CourseAccess.CourseID = dbo.Courses.CourseID
INNER JOIN CourseModules ON dbo.Courses.CourseID = CourseModules.CourseID
INNER JOIN ScheduleEvents
ON CourseModules.CourseModuleID = ScheduleEvents.EventID
INNER JOIN CourseAccess AS CourseAccess_1
ON dbo.Members.MemberID = CourseAccess_1.MemberID
AND dbo.CourseAccess.AccessID > CourseAccess_1.AccessID
INNER JOIN dbo.Courses AS Courses_1
ON Courses_1.CourseID = CourseAccess_1.CourseID
INNER JOIN dbo.ScheduleEvents AS ScheduleEvents_1
INNER JOIN dbo.CourseModules AS CourseModules_1
ON ScheduleEvents_1.EventID = CourseModules_1.CourseModuleID
ON CourseModules_1.CourseID = Courses_1.CourseID
AND dbo.ScheduleEvents.EventID <> ScheduleEvents_1.EventID
WHERE (ScheduleEvents_1.StartDate BETWEEN dbo.ScheduleEvents.StartDate
AND dbo.ScheduleEvents.EndDate)
OR (dbo.ScheduleEvents.StartDate BETWEEN ScheduleEvents_1.StartDate AND
ScheduleEvents_1.EndDate)
GROUP BY dbo.Members.FirstName, dbo.Members.LastName,
ScheduleEvents_1.StartDate, dbo.ScheduleEvents.StartDate,
dbo.ScheduleEvents.EndDate, ScheduleEvents_1.EndDate
```

- GetDebts – SM, MB, DM

Wyświetla listę dłużników

```
CREATE VIEW [dbo].[GetDebts]
AS
SELECT DISTINCT
MB.FirstName, MB.LastName, MB.Login, MB.Email, M.MajorID, W.WebinarID,
S.SubjectID, C.CourseID, COALESCE (dbo.GetBeginCourseDate(C.CourseID),
dbo.GetBeginMajorDate(M.MajorID), dbo.GetBeginSubjectDate(S.SubjectID),
dbo.GetBeginWebinarDate(W.WebinarID)) AS [Begin date]
FROMdbo.Members AS MB
INNER JOIN dbo.Orders AS O ON O.MemberID = MB.MemberID
INNER JOIN dbo.OrderDetails AS OD ON OD.OrderID = O.OrderID
INNER JOIN
dbo.Products AS P ON P.ProductID = OD.ProductID
LEFT OUTER JOIN dbo.Subjects AS S ON P.ProductID = S.ProductID
LEFT OUTER JOIN dbo.Majors AS M ON P.ProductID = M.ProductID
LEFT OUTER JOIN dbo.Courses AS C ON C.ProductID = P.ProductID
LEFT OUTER JOIN dbo.Webinars AS W ON W.ProductID = P.ProductID
WHERE (COALESCE (dbo.GetBeginCourseDate(C.CourseID),
dbo.GetBeginMajorDate(M.MajorID), dbo.GetBeginSubjectDate(S.SubjectID),
dbo.GetBeginWebinarDate(W.WebinarID)) < GETDATE())
AND (NOT EXISTS (SELECT  P.OrderID FROM dbo.Payments AS P INNER JOIN
dbo.PaymentStatus AS PS ON PS.PaymentStatusID = P.PaymentStatusID
WHERE (P.OrderID = O.OrderID) AND (P.IsAdvance = 0) AND (PS.StatusName =
'success')))
```

- GetCoursesProductsNext30Days – SM, MB, DM

Wyświetla ofertę kursów na najbliższe 30 dni

```
CREATE VIEW [dbo].[GetCoursesProductsNext30Days]
AS
SELECT  dbo.Products.Name, dbo.Products.Description,
MIN(dbo.ScheduleEvents.StartDate) AS [Start time],
COUNT(dbo.CourseModules.CourseModuleID) AS [Modules count]
FROM dbo.Courses
INNER JOIN dbo.CourseModules
ON dbo.Courses.CourseID = dbo.CourseModules.CourseID
INNER JOIN dbo.ScheduleEvents
ON dbo.CourseModules.CourseModuleID = dbo.ScheduleEvents.EventID
INNER JOIN dbo.Products ON dbo.Courses.ProductID = dbo.Products.ProductID
GROUP BY dbo.Products.Name, dbo.Products.Description
HAVING  (MIN(dbo.ScheduleEvents.StartDate) BETWEEN GETDATE() AND
DATEADD(DAY, 30, GETDATE()))
```

- GetFailedStuents – SM, MB, DM

Wyświetla listę studentów którzy, nie zdali co najmniej jednego przedmiotu

```
CREATE VIEW [dbo].[GetFailedStudents]
AS
SELECT  dbo.Members.FirstName, dbo.Members.LastName, dbo.Members.Login,
dbo.Members.Email, dbo.StudentEnrolls.FinalGrade
FROM dbo.StudentEnrolls
INNER JOIN dbo.Members
ON dbo.StudentEnrolls.StudentID = dbo.Members.MemberID
WHERE (dbo.StudentEnrolls.FinalGrade = 2)
```

- GetGraduated – SM, MB, DM

Wyświetla użytkowników, którzy ukończyli studia albo kurs

```sql
CREATE VIEW [dbo].[GetGraduated]
AS
SELECT  M.FirstName, M.LastName, P.Name AS Graduated
FROM      dbo.Members AS M
INNER JOIN dbo.Diplomas AS D ON M.MemberID = D.MemberID
INNER JOIN dbo.CourseDiplomas
ON D.DiplomaID = dbo.CourseDiplomas.DiplomaID
INNER JOIN dbo.Courses
ON dbo.CourseDiplomas.CourseID = dbo.Courses.CourseID
INNER JOIN dbo.MajorDiplomas
ON D.DiplomaID = dbo.MajorDiplomas.DiplomaID
INNER JOIN dbo.Majors ON dbo.MajorDiplomas.MajorID = dbo.Majors.MajorID
INNER JOIN dbo.Products AS P ON COALESCE (dbo.Courses.ProductID,
dbo.Majors.ProductID) = P.ProductID
GROUP BY M.FirstName, M.LastName, P.Name
```

- GetInternships – SM, MB, DM

Wyświetla studentów oraz odbyte przez nich praktyki

```sql
CREATE VIEW [dbo].[GetInternships]
AS
SELECT  I.StartDate, I.EndDate, I.StudentID, M.FirstName, M.LastName,
C.Name AS [Company Name], P.Name AS [Major Name]
FROM dbo.Interships AS I
INNER JOIN dbo.Members AS M ON M.MemberID = I.StudentID
INNER JOIN dbo.Companies AS C ON C.CompanyID = I.CompanyID
INNER JOIN dbo.Majors AS MA ON MA.MajorID = I.MajorID
INNER JOIN dbo.Products AS P ON P.ProductID = MA.ProductID
```

- ## GetMeetingDebts – SM, MB, DM

Wyświetla osoby, które nie opłaciły zjazdów

```sql
CREATE VIEW [dbo].[GetMeetingsDebts]
AS
SELECT DISTINCT MB.FirstName, MB.LastName, MB.Login, MB.Email
FROM dbo.Members AS MB
INNER JOIN dbo.Orders AS O ON O.MemberID = MB.MemberID
INNER JOIN dbo.OrderDetails AS OD ON OD.OrderID = O.OrderID
INNER JOIN dbo.Products AS P ON P.ProductID = OD.ProductID
LEFT OUTER JOIN dbo.Meetings AS M ON M.ProductID = P.ProductID
WHERE (dbo.GetBeginMeetingDate(M.MeetingID) < DATEDIFF(DAY, 3, GETDATE()))
AND (NOT EXISTS (SELECT  P.OrderID FROM dbo.Payments AS P INNER JOIN
dbo.PaymentStatus AS PS ON PS.PaymentStatusID = P.PaymentStatusID
WHERE (P.OrderID = O.OrderID) AND (P.isAdvance = 0) AND (PS.StatusName =
'success')))
```

- ## GetMembersWithCurrentPostponments – SM, MB, DM

Wyświetla użytkowników, którzy mają odroczone płatności.

```sql
CREATE VIEW [dbo].[GetMembersWithCurrentPostponments]
AS
SELECT  dbo.Members.FirstName, dbo.Members.LastName
FROM dbo.Postponements
INNER JOIN dbo.Orders ON dbo.Postponements.OrderID = dbo.Orders.OrderID
INNER JOIN dbo.Members ON dbo.Orders.MemberID = dbo.Members.MemberID
WHERE (GETDATE() BETWEEN dbo.Postponements.PostponeStartDate AND
dbo.Postponements.PostponeEndDate)
```

- ## GetMembersWithUnpaidProducts – SM, MB, DM

Zwraca wszystkich użytkowników, którzy mają nieopłacone produkty

```sql
CREATE VIEW [dbo].[GetMembersWithUnpaidProducts]
AS
SELECT DISTINCT M.FirstName, M.LastName, M.Login, M.Email
FROM dbo.Members AS M
INNER JOIN dbo.Orders AS O ON O.MemberID = M.MemberID
WHERE (NOT EXISTS (SELECT  P.OrderID FROM dbo.Payments AS P
INNER JOIN dbo.PaymentStatus AS PS
ON PS.PaymentStatusID = P.PaymentStatusID
WHERE (P.OrderID = O.OrderID) AND (P.isAdvance = 0) AND (PS.StatusName =
'SUCCESS')))
```

- GetMonthlyIncome – SM, MB, DM

Wyświetla miesięczny dochód z wszystkich sprzedaży

```
CREATE VIEW [dbo].[GetMonthlyIncome]
AS
SELECT  YEAR(dbo.Orders.OrderDate) AS Year,
        MONTH(dbo.Orders.OrderDate) AS Month,
        ROUND(SUM(dbo.Products.Price *
        (1 - ISNULL(dbo.GetDiscount(dbo.Products.ProductID,
        dbo.Members.MemberID), 0))), 2) AS [Total income]
FROM  dbo.Products
INNER JOIN dbo.OrderDetails ON dbo.Products.ProductID = dbo.OrderDetails.ProductID
INNER JOIN dbo.Orders ON dbo.OrderDetails.OrderID = dbo.Orders.OrderID
INNER JOIN dbo.Members ON dbo.Orders.MemberID = dbo.Members.MemberID
GROUP BY YEAR(dbo.Orders.OrderDate), MONTH(dbo.Orders.OrderDate)
```

- GetStudentsPerCountry – SM, MB, DM

Wyświetla studentów z podziałem na kraj, z którego pochodzą.

```
CREATE VIEW [dbo].[GetStudentsPerCountry]
AS
SELECT  dbo.Countries.Name AS [Country name], COUNT(dbo.Members.MemberID)
AS [Students count]
FROM  dbo.Cities
INNER JOIN dbo.Countries ON dbo.Cities.CountryID = dbo.Countries.CountryID
INNER JOIN dbo.Addresses ON dbo.Cities.CityID = dbo.Addresses.CityID
INNER JOIN dbo.Members ON dbo.Addresses.AddressID = dbo.Members.AddressID
WHERE   (dbo.IsMemberOfRole(dbo.Members.MemberID, 'Student') = 1)
GROUP BY dbo.Countries.Name
```

- GetStudentsWithFinishedInternships – SM, MB, DM

Wyświetla studentów, którzy ukończyli praktyki.

```
CREATE VIEW [dbo].[GetStudentsWithFinishedInternships]
AS
SELECT  M.FirstName, M.LastName, P.Name AS [Major name]
FROM    dbo.Interships AS I
INNER JOIN dbo.Members AS M ON I.StudentID = M.MemberID
INNER JOIN dbo.Majors ON I.MajorID = dbo.Majors.MajorID
INNER JOIN  dbo.Products AS P ON dbo.Majors.ProductID = P.ProductID
WHERE   (I.EndDate < GETDATE())
GROUP BY M.FirstName, M.LastName, P.Name
HAVING  (COUNT(*) >= 2)
```

- GetTodayEvents – SM, MB, DM

Wyświetla zajęcia ,które odbywają się dzisiaj.

```
CREATE VIEW [dbo].[GetTodayEvents]
AS
SELECT  FORMAT(dbo.ScheduleEvents.StartDate, N'hh:mm') AS [Start time],
COALESCE (dbo.Products.Name, dbo.CourseModules.Name) AS [Event Name]
FROM  dbo.ScheduleEvents
INNER JOIN dbo.Classes ON dbo.ScheduleEvents.EventID = dbo.Classes.ClassID
INNER JOIN dbo.CourseModules
ON dbo.ScheduleEvents.EventID = dbo.CourseModules.CourseModuleID
INNER JOIN dbo.Webinars
ON dbo.ScheduleEvents.EventID = dbo.Webinars.WebinarID
INNER JOIN dbo.Subjects ON dbo.Classes.SubjectID = dbo.Subjects.SubjectID
INNER JOIN dbo.Products ON dbo.Products.ProductID = COALESCE
(dbo.Subjects.SubjectID, dbo.Webinars.WebinarID)
WHERE (DATEDIFF(day, dbo.ScheduleEvents.StartDate, GETDATE()) = 0)
```

- GetTop100Students – SM, MB, DM

Wyświetla 100 studentów z najwyższą średnią.

```
CREATE VIEW [dbo].[GetTop100Students]
AS
SELECT TOP (100) M.FirstName, M.LastName,
ROUND(AVG(SE.FinalGrade), 2) AS Grade
FROM  dbo.StudentEnrolls AS SE
INNER JOIN dbo.Members AS M ON SE.StudentID = M.MemberID
GROUP BY M.FirstName, M.LastName ORDER BY Grade DESC
```

- ListCourseModules - SM, MB, DM

Wyświetla wszystkie moduły kursów

```
CREATE VIEW [dbo].[ListCourseModules]
AS
SELECT  dbo.CourseModules.CourseModuleID, dbo.CourseModules.Name AS
CourseModuleName, dbo.Courses.CourseID, dbo.Products.Name AS CourseName,
dbo.ScheduleEvents.StartDate, dbo.ScheduleEvents.EndDate,
dbo.Products.ProductID
FROM      dbo.CourseModules INNER JOIN
                   dbo.Courses ON dbo.CourseModules.CourseID =
dbo.Courses.CourseID INNER JOIN
                   dbo.Products ON dbo.Courses.ProductID =
dbo.Products.ProductID INNER JOIN
                   dbo.ScheduleEvents ON dbo.CourseModules.CourseModuleID
= dbo.ScheduleEvents.EventID
```

- ListCourses - SM, MB, DM

Wyświetla listę wszystkich kursów

```
CREATE VIEW [dbo].[ListCourses]
AS
SELECT  dbo.Courses.CourseID, dbo.Products.Name, dbo.Members.FirstName AS
CoordinatorFirstName, dbo.Members.LastName AS CoordinatorLastName,
dbo.Products.ProductID
FROM      dbo.Courses INNER JOIN
                   dbo.Products ON dbo.Courses.ProductID =
dbo.Products.ProductID INNER JOIN
                   dbo.Members ON dbo.Courses.CoordinatorID =
dbo.Members.MemberID AND dbo.Courses.CoordinatorID = dbo.Members.MemberID
```

- ListMajors - SM, MB, DM

Wyświetla listę wszystkich kierunków

```
CREATE VIEW [dbo].[ListCourses]
AS
SELECT  dbo.Courses.CourseID, dbo.Products.Name, dbo.Members.FirstName AS
CoordinatorFirstName, dbo.Members.LastName AS CoordinatorLastName,
dbo.Products.ProductID
FROM      dbo.Courses INNER JOIN
                   dbo.Products ON dbo.Courses.ProductID =
dbo.Products.ProductID INNER JOIN
                   dbo.Members ON dbo.Courses.CoordinatorID =
dbo.Members.MemberID AND dbo.Courses.CoordinatorID = dbo.Members.MemberID
```

- ListMeetings - SM, MB, DM

Wyświetla listę wszystkich zjazdów

```
CREATE VIEW [dbo].[ListMeetings]
AS
SELECT  dbo.Meetings.MeetingID, dbo.Products.Name AS MeetingName,
dbo.Meetings.MajorID, Products_1.Name AS MajorName,
dbo.Meetings.StartDate, dbo.Meetings.EndDate
FROM      dbo.Meetings INNER JOIN
                   dbo.Products ON dbo.Meetings.ProductID =
dbo.Products.ProductID INNER JOIN
                   dbo.Majors ON dbo.Meetings.MajorID = dbo.Majors.MajorID
INNER JOIN
                   dbo.Products AS Products_1 ON dbo.Majors.ProductID =
Products_1.ProductID
```

- ListSubjects - SM, MB, DM

Wyświetla listę wszystkich przedmiotów

```sql
CREATE VIEW [dbo].[ListSubjects]
AS
SELECT  dbo.Subjects.SubjectID, Products_1.Name AS SubjectName,
dbo.Subjects.Semester, dbo.Subjects.EducationFormID, dbo.Majors.MajorID,
dbo.Products.Name AS MajorName, dbo.Members.FirstName AS
CoordinatorFirstName, dbo.Members.LastName AS CoordinatorLastName,
                   dbo.Products.ProductID
FROM      dbo.Subjects INNER JOIN
                   dbo.Majors ON dbo.Subjects.MajorID = dbo.Majors.MajorID
INNER JOIN
                   dbo.Products ON dbo.Subjects.ProductID =
dbo.Products.ProductID INNER JOIN
                   dbo.Products AS Products_1 ON dbo.Majors.ProductID =
Products_1.ProductID INNER JOIN
                   dbo.Members ON dbo.Subjects.CoordinatorID =
dbo.Members.MemberID
```

- ListWebinars - SM, MB, DM

Wyświetla listę wszystkich webinarów

```sql
CREATE VIEW [dbo].[ListWebinars]
AS
SELECT  dbo.Products.Name, dbo.Webinars.WebinarID, dbo.Members.FirstName
AS LeaderFirstName, dbo.Members.LastName AS LeaderLastName,
dbo.Products.ProductID
FROM      dbo.Webinars INNER JOIN
                   dbo.Products ON dbo.Webinars.ProductID =
dbo.Products.ProductID INNER JOIN
                   dbo.Members ON dbo.Webinars.LeaderID =
dbo.Members.MemberID
```

- ViewCoordinators - SM, MB, DM

Wyświetla listę wszystkich koordynatorów

```
CREATE VIEW [dbo].[ViewCoordinators]
AS
SELECT  MemberID, FirstName, LastName, Email, StreetName, StreetNumber,
CityName, PostalCode, CountryName
FROM      dbo.ListMembersByRole('coordinator') AS ListMembersByRole_1
```

- ViewCourseProducts - SM, MB, DM

Wyświetla listę wszystkich kursów w sklepie

```
CREATE VIEW [dbo].[ViewCourseProducts]
AS
SELECT  ProductID, Name, Description, Price
FROM      dbo.GetProductsByCategory('course') AS GetProductsByCategory_1
```

- ViewMajorProducts - SM, MB, DM

Wyświetla listę wszystkich kierunków w sklepie

```
CREATE VIEW [dbo].[ViewMajorProducts]
AS
SELECT  ProductID, Name, Description, Price
FROM      dbo.GetProductsByCategory('major') AS GetProductsByCategory_1
```

- ViewMeetingProducts - SM, MB, DM

Wyświetla listę wszystkich zjazdów w sklepie

```
CREATE VIEW [dbo].[ViewMeetingProducts]
AS
SELECT  ProductID, Name, Description, Price
FROM      dbo.GetProductsByCategory('meeting') AS GetProductsByCategory_1
```

- ViewSubjectProducts - SM, MB, DM

Wyświetla listę wszystkich przedmiotów (na studiach) w sklepie

```
CREATE VIEW [dbo].[ViewSubjectProducts]
AS
SELECT  ProductID, Name, Description, Price
FROM      dbo.GetProductsByCategory('subject') AS GetProductsByCategory_1
GO
```

- ViewTeachers - SM, MB, DM

Wyświetla listę wszystkich nauczycieli

```
CREATE VIEW [dbo].[ViewTeachers]
AS
SELECT  MemberID, FirstName, LastName, Email, StreetName, StreetNumber,
CityName, PostalCode, CountryName
FROM      dbo.ListMembersByRole('teacher') AS ListMembersByRole_1
```

- ViewWebinarProducts - SM, MB, DM

Wyświetl produkty webinarów.

```
CREATE VIEW [dbo].[ViewWebinarProducts]
AS
SELECT  ProductID, Name, Description, Price
FROM      dbo.GetProductsByCategory('webinar') AS GetProductsByCategory_1
```

- ViewMembers - SM, MB, DM

Wyświetl wszystkich użytkowników

```
CREATE VIEW [dbo].[ViewMembers]
AS
SELECT  dbo.Members.MemberID, dbo.Members.FirstName, dbo.Members.LastName,
dbo.Members.Login, dbo.Members.Email, dbo.Addresses.StreetName,
dbo.Addresses.StreetNumber, dbo.Cities.Name AS CityName,
dbo.Cities.PostalCode, dbo.Countries.Name AS CountryName
FROM      dbo.Members INNER JOIN
                   dbo.Addresses ON dbo.Members.AddressID =
dbo.Addresses.AddressID INNER JOIN
                   dbo.Cities ON dbo.Addresses.CityID = dbo.Cities.CityID
INNER JOIN
                   dbo.Countries ON dbo.Cities.CountryID =
dbo.Countries.CountryID
```

GetWebinarsProductsNext30Days – SM, MB, DM

Wyświetla ofertę webinarów na najbliższe 30 dni.

```sql
CREATE VIEW [dbo].[GetWebinarsProductsNext30Days]
AS
SELECT dbo.Products.Name, dbo.Products.Description,
dbo.ScheduleEvents.StartDate, DATEDIFF(minute,
dbo.ScheduleEvents.StartDate, dbo.ScheduleEvents.EndDate) AS Expr1
FROM dbo.Products
INNER JOIN dbo.Webinars
ON dbo.Products.ProductID = dbo.Webinars.ProductID
INNER JOIN dbo.ScheduleEvents
ON dbo.Webinars.WebinarID = dbo.ScheduleEvents.EventID
WHERE (dbo.ScheduleEvents.StartDate BETWEEN GETDATE() AND DATEADD(day, 30,
GETDATE()))
```

Funkcje

- ## GetClassAttendance – SM, MB, DM

Zwraca listę obecności na zajęciach o podanym ID

```sql
CREATE FUNCTION [dbo].[GetClassAttendance] (@ClassID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT SE.StudentID FROM dbo.StudentAttendance AS SA
      INNER JOIN dbo.StudentEnrolls AS SE ON SE.StudentEnrollID =
SA.StudentEnrollID
      WHERE SA.ClassID = @ClassID
)
```

- ## GetClassMembers – SM, MB, DM

Zwraca listę osób w zadanej klasie

```sql
CREATE FUNCTION [dbo].[GetClassMembers] (@ClassID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT SE.StudentID FROM dbo.Groups AS G
      INNER JOIN dbo.Classes AS C ON C.GroupID = G.GroupID
      INNER JOIN dbo.StudentEnrolls AS SE ON SE.GroupID = G.GroupID
      WHERE C.ClassID = @ClassID
)
```

- ## GetCourseMembers – SM, MB, DM

Wyświetla tabelę imion i nazwisk użytkowników zapisanych na kurs o podanym ID

```sql
CREATE FUNCTION [dbo].[GetCourseMembers](@CourseID BIGINT)
RETURNS TABLE
AS RETURN (
      SELECT FirstName, LastName FROM dbo.Members
      INNER JOIN dbo.CourseAccess ON CourseAccess.MemberID =
Members.MemberID
      WHERE CourseID = @CourseID
);
```

- GetCourseModuleAttendance – SM, MB, DM

Zwraca listę obecności dla danego modułu z kursu

```sql
CREATE FUNCTION [dbo].[GetCourseModuleAttendance] (@CourseModulesID
BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT CA.MemberID FROM dbo.CourseAttendance AS CT
      INNER JOIN dbo.CourseAccess AS CA ON CA.AccessID = CT.AccessID
      WHERE CT.CourseModuleID = @CourseModulesID
)
```

- GetCourseModulesOfCourse – SM, MB, DM

Zwraca listę modułów zapisanych do danego kursu

```sql
CREATE FUNCTION [dbo].[GetCourseModulesOfCourse] (@CourseID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT CM.CourseModuleID, CM.[Name] FROM CourseModules AS CM
      WHERE CM.CourseID = @CourseID
)
```

- GetMajorMembers – SM, MB, DM

Wyświetla tabelę imion i nazwisk użytkowników zapisanych na kierunek o podanym ID

```sql
CREATE FUNCTION [dbo].[GetMajorMembers](@MajorID BIGINT)
RETURNS TABLE
AS RETURN (
      SELECT DISTINCT FirstName, LastName FROM dbo.Members
      INNER JOIN dbo.StudentEnrolls ON StudentEnrolls.StudentID =
Members.MemberID
      INNER JOIN dbo.Groups ON Groups.GroupID = StudentEnrolls.GroupID
      INNER JOIN dbo.Classes ON Classes.GroupID = Groups.GroupID
      INNER JOIN dbo.Subjects ON Subjects.SubjectID = Classes.SubjectID
      WHERE Subjects.MajorID = @MajorID
);
```

- GetMajorSyllabus – SM, MB, DM

Zwraca syllabus dla podanej nazwy kierunku

```
CREATE FUNCTION [dbo].[GetMajorSyllabus] (@MajorName VARCHAR(200))
RETURNS TABLE
AS
RETURN
(
      SELECT P.[Name],
           P.[Description],
           CONCAT(MS.FirstName, ' ', MS.LastName) AS [Coordinator],
           S.Semester,
           EF.TypeName AS [Education form]
 FROM dbo.Majors AS M
      INNER JOIN dbo.Subjects AS S ON S.MajorID = M.MajorID
      INNER JOIN dbo.Products AS P ON P.ProductID = S.ProductID
      INNER JOIN dbo.EducationForms AS EF ON EF.EducationFormID =
S.EducationFormID
      INNER JOIN dbo.Members AS MS ON MS.MemberID = S.CoordinatorID
      WHERE M.MajorID = dbo.GetMajorID(@MajorName)
)
```

- GetMajorSyllabusBySemester – SM, MB, DM

Zwraca syllabus na poszczególnych semestr dla podanej nazwy kierunku

```
CREATE FUNCTION [dbo].[GetMajorSyllabusBySemester]
(
      @MajorName VARCHAR(200),
      @Semester TINYINT
)
RETURNS TABLE
AS
RETURN
(
      SELECT [Name],
           [Description],
           Coordinator
           Semester,
               [Education form]
      FROM dbo.GetMajorSyllabus(@MajorName) WHERE Semester = @Semester
)
```

- GetMemberAttendances – SM, MB, DM

Zwraca listę obecności zadanego studenta dla modułów kursów i zajęć na studiach, na które jest zapisany

```
CREATE FUNCTION [dbo].[GetMemberAttendance] (@MemberID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT C.ClassID AS ID, P.[Name], SS.StartDate, 'present' AS
Attendance FROM dbo.Subjects AS S
      LEFT JOIN Classes AS C ON C.SubjectID = S.SubjectID
      LEFT JOIN StudentEnrolls AS SE ON SE.GroupID = C.GroupID
      LEFT JOIN StudentAttendance AS SA ON SA.StudentEnrollID =
SE.StudentEnrollID
      LEFT JOIN Products AS P ON P.ProductID = S.ProductID
      LEFT JOIN ScheduleEvents AS SS ON SS.EventID = C.ClassID
      WHERE SE.StudentID = @MemberID AND SA.AttendanceID IS NOT NULL
      GROUP BY C.ClassID, P.[Name], SA.AttendanceID, SS.StartDateUNION
      SELECT C.ClassID AS ID, P.[Name], SS.StartDate, 'absent' AS
Attendance FROM dbo.Subjects AS S
      LEFT JOIN Classes AS C ON C.SubjectID = S.SubjectID
      LEFT JOIN StudentEnrolls AS SE ON SE.GroupID = C.GroupID
      LEFT JOIN StudentAttendance AS SA ON SA.StudentEnrollID =
SE.StudentEnrollID
      LEFT JOIN Products AS P ON P.ProductID = S.ProductID
      LEFT JOIN ScheduleEvents AS SS ON SS.EventID = C.ClassID
      WHERE SE.StudentID = @MemberID AND SA.AttendanceID IS NULL
      GROUP BY C.ClassID, P.[Name], SA.AttendanceID, SS.StartDate
      UNION
      SELECT CM.CourseModuleID AS ID, CM.[Name], SS.StartDate, 'present'
AS Attendance FROM dbo.Courses AS C
      LEFT JOIN CourseModules AS CM ON CM.CourseID = C.CourseID
      LEFT JOIN CourseAccess AS CA ON CA.CourseID = C.CourseID
      LEFT JOIN CourseAttendance AS CT ON CT.CourseModuleID =
CM.CourseModuleID
      LEFT JOIN Products AS P ON P.ProductID = C.ProductID
      LEFT JOIN ScheduleEvents AS SS ON SS.EventID = CM.CourseModuleID
      WHERE CA.MemberID = @MemberID AND CT.AttendanceID IS NOT NULL
      GROUP BY CM.CourseModuleID, CM.[Name], CT.AttendanceID, SS.StartDate
      UNION
      SELECT CM.CourseModuleID AS ID, CM.[Name], SS.StartDate, 'absent' AS
Attendance FROM dbo.Courses AS C
      LEFT JOIN CourseModules AS CM ON CM.CourseID = C.CourseID
      LEFT JOIN CourseAccess AS CA ON CA.CourseID = C.CourseID
      LEFT JOIN CourseAttendance AS CT ON CT.CourseModuleID =
CM.CourseModuleID
      LEFT JOIN Products AS P ON P.ProductID = C.ProductID
      LEFT JOIN ScheduleEvents AS SS ON SS.EventID = CM.CourseModuleID
      WHERE CA.MemberID = @MemberID AND CT.AttendanceID IS NULL
      GROUP BY CM.CourseModuleID, CM.[Name], CT.AttendanceID, SS.StartDate
)
```

- GetMemberClasses – SM, MB, DM

Dla danego użytkownika oraz podanego przedziału czasu wyświetla nazwy przedmiotów ze studiów , które są zaplanowane w podanym przedziale czasu

```
CREATE FUNCTION [dbo].[GetMemberClasses]
(
      @MemberID BIGINT,
      @StartDate DATE,
      @EndDate DATE
)
RETURNS TABLE
AS
RETURN
(
      SELECT SE.StartDate,
           SE.EndDate,
               P.[Name]
    FROM dbo.ScheduleEvents AS SE
      INNER JOIN dbo.Classes AS C ON C.ClassID = SE.EventID
      INNER JOIN dbo.Subjects AS S ON S.SubjectID = C.SubjectID
      INNER JOIN dbo.StudentEnrolls AS E ON E.GroupID = C.GroupID
      INNER JOIN dbo.Products AS P ON P.ProductID = S.ProductID
      WHERE dbo.IsDayInRange(SE.StartDate, @StartDate, @EndDate) = 1 AND
E.StudentID = @MemberID
)
```

- ## GetMemberCourseModules – SM, MB, DM

Dla danego użytkownika oraz podanego przedziału czasu wyświetla nazwy modułów z kursów, które są zaplanowane w podanym przedziale czas

```
CREATE FUNCTION [dbo].[GetMemberCourseModules]
(
      @MemberID BIGINT,
      @StartDate DATE,
      @EndDate DATE
)
RETURNS TABLE
AS
RETURN
(
      SELECT SE.StartDate, SE.EndDate, P.[Name]
    FROM dbo.ScheduleEvents AS SE
      INNER JOIN dbo.CourseModules AS CM ON CM.CourseModuleID = SE.EventID
      INNER JOIN dbo.Courses AS C ON C.CourseID = CM.CourseID
      INNER JOIN dbo.CourseAccess AS CA ON CA.CourseID = C.CourseID
      INNER JOIN dbo.Products AS P ON P.ProductID = C.ProductID
      WHERE dbo.IsDayInRange(SE.StartDate, @StartDate, @EndDate) = 1 AND
CA.MemberID = @MemberID
)
```

- ## GetMemberOrderDetails – SM, MB, DM

Dla konkretnego użytkownika wyświetla szczegóły jego zamówień

```
CREATE FUNCTION [dbo].[GetMemberOrderDetails]
(
      @OrderID BIGINT
)
RETURNS TABLE
AS
RETURN
(
      SELECT P.[Name], P.[Description], P.Price,
      ROUND((1 - dbo.GetDiscount(P.ProductID, O.MemberID)) * P.Price, 2)
AS [Price w/ discount],
           C.CategoryName AS [Category] FROM dbo.Orders AS O
      INNER JOIN dbo.OrderDetails AS OD ON OD.OrderID = O.OrderID
      INNER JOIN dbo.Products AS P ON P.ProductID = OD.ProductID
      INNER JOIN dbo.Categories AS C ON C.CategoryID = P.CategoryID
      WHERE O.OrderID = @OrderID
)
```

- GetMemberOrders – SM, MB, DM

Dla konkretnego użytkownika zwraca jego zamówienia

```
CREATE FUNCTION [dbo].[GetMemberOrders] (@MemberID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT O.OrderDate,
           O.IsInAdvance,
               dbo.GetLastPaymentStatus(O.OrderID) AS [Payment status],
               (SELECT SUM([Price w/ discount]) FROM
dbo.GetMemberOrderDetails(O.OrderID)) AS [Total price]
               FROM dbo.Orders AS O
      WHERE O.MemberID = @MemberID
)
```

- GetMemberRoles – SM, MB, DM

Dla danego użytkownika zwraca wszystkie jego role

```
CREATE FUNCTION [dbo].[GetMemberRoles] (@MemberID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT DISTINCT MR.RoleName FROM dbo.AssignedMemberRoles AS AMR
      INNER JOIN dbo.MemberRoles AS MR ON MR.MemberRoleID =
AMR.MemberRoleID
      WHERE AMR.MemberID = @MemberID
)
```

GetMemberScheduleEvents – SM, MB, DM

Dla zadanego studenta wyświetla listę wszystkich klas, modułów kursów oraz webinarów jako wydarzenia, na które jest zapisany od podanej daty

```
CREATE FUNCTION [dbo].[GetMemberScheduleEvents]
(
      @MemberID BIGINT,
      @StartDate DATE
)
RETURNS TABLE
AS
RETURN
(
      SELECT * FROM dbo.GetMemberClasses(@MemberID, @StartDate,
DATEADD(YEAR, 1, @StartDate)) UNION
      SELECT * FROM dbo.GetMemberCourseModules(@MemberID, @StartDate,
DATEADD(YEAR, 1, @StartDate)) UNION
      SELECT * FROM dbo.GetMemberWebinars(@MemberID, @StartDate,
DATEADD(YEAR, 1, @StartDate))
);
```

GetMemberWebinars – SM, MB, DM

Dla danego użytkownika oraz podanego przedziału czasu wyświetla nazwy webinarów na które zapisał się dany użytkownik

```
CREATE FUNCTION [dbo].[GetMemberWebinars]
(
      @MemberID BIGINT,
      @StartDate DATE,
      @EndDate DATE
)
RETURNS TABLE
AS
RETURN
(
      SELECT SE.StartDate,
           SE.EndDate,
           P.[Name]
      FROM dbo.ScheduleEvents AS SE
      INNER JOIN dbo.Webinars AS W ON W.WebinarID = SE.EventID
      INNER JOIN dbo.WebinarAccess AS WA ON WA.WebinarID = W.WebinarID
      INNER JOIN dbo.Products AS P ON P.ProductID = W.ProductID
      WHERE dbo.IsDayInRange(SE.StartDate, @StartDate, @EndDate) = 1 AND
WA.MemberID = @MemberID
)
```

- ## GetMemberWeekScheduleEvents - SM, MB, DM

Wyświetla harmonogram zajęć na przyszły tydzień dla danego użytkownika

```
CREATE FUNCTION [dbo].[GetMemberWeekScheduleEvents]
(
      @MemberID BIGINT
)
RETURNS TABLE
AS
RETURN
(
      SELECT * FROM dbo.GetMemberClasses(@MemberID, GETDATE(),
DATEADD(DAY, 7, GETDATE()))
      UNION
      SELECT * FROM dbo.GetMemberCourseModules(@MemberID, GETDATE(),
DATEADD(DAY, 7, GETDATE()))
      UNION
      SELECT * FROM dbo.GetMemberWebinars(@MemberID, GETDATE(),
DATEADD(DAY, 7, GETDATE()))
)
```

- ## GetProductsByCategory – SM, MB, DM

Wyświetla wszystkie produkty z danej kategorii

```
CREATE FUNCTION [dbo].[GetProductsByCategory]
(
      @CategoryName VARCHAR(80)
)
RETURNS TABLE
AS
RETURN
(
      SELECT [Name],
           [Description],
           Price
               FROM dbo.Products
      WHERE CategoryID = dbo.GetCategoryID(@CategoryName)
)
```

- GetStudentGrades – SM, MB, DM

Wyświetla cząstkowe oceny studenta

```
CREATE FUNCTION [dbo].[GetStudentGrades] (@StudentID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT P.[Name], SG.Grade FROM StudentEnrolls AS SE
      INNER JOIN StudentGrades AS SG ON SG.StudentEnrollID =
SE.StudentEnrollID
      INNER JOIN Classes AS C ON C.GroupID = SE.GroupID
      INNER JOIN Subjects AS S ON S.SubjectID = C.SubjectID
      INNER JOIN Products AS P ON P.ProductID = S.ProductID
      WHERE SE.StudentID = @StudentID
)
```

- GetSubjectTeachers – SM, MB, DM

Zwraca wszystkich prowadzących dany przedmiot

```
CREATE FUNCTION [dbo].[GetSubjectTeachers] (@SubjectID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT M.FirstName, M.LastName FROM dbo.SubjectTeachers AS ST
      INNER JOIN dbo.Members AS M ON M.MemberID = ST.TeacherID
      WHERE ST.SubjectID = @SubjectID
)
```

- ## GetTeacherEvents – SM, MB, DM

Zwraca wszystkie wydarzenia, które prowadzi dany nauczyciel

```
CREATE FUNCTION [dbo].[GetTeacherEvents] (@TeacherID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT C.ClassID AS [ID], SE.StartDate, SE.EndDate, 'class' AS
[Type], P.[Name]
      FROM Groups AS G
      INNER JOIN Classes AS C ON C.GroupID = G.GroupID
      INNER JOIN ScheduleEvents AS SE ON SE.EventID = C.ClassID
      INNER JOIN Subjects AS S ON S.SubjectID = C.SubjectID
      INNER JOIN Products AS P ON P.ProductID = S.SubjectID
      WHERE G.TeacherID = @TeacherID
      UNION
      SELECT CM.CourseModuleID AS [ID], SE.StartDate, SE.EndDate, 'course
module' AS [Type], P.[Name]
      FROM CourseModules AS CM
      INNER JOIN CourseModuleTeachers AS CMT ON CMT.CourseModuleID =
CM.CourseModuleID
      INNER JOIN ScheduleEvents AS SE ON SE.EventID = CM.CourseModuleID
      INNER JOIN Courses AS C ON C.CourseID = CM.CourseID
      INNER JOIN Products AS P ON P.ProductID = C.ProductID
      WHERE CMT.TeacherID = @TeacherID
      UNION
      SELECT W.WebinarID AS [ID], SE.StartDate, SE.EndDate, 'webinar' AS
[Type], P.[Name]
      FROM Webinars AS W
      INNER JOIN ScheduleEvents AS SE ON SE.EventID = W.WebinarID
      INNER JOIN Products AS P ON P.ProductID = W.ProductID
      WHERE W.LeaderID = @TeacherID
)
```

- ## GetWebinarMembers – SM, MB, DM

Wyświetla tabelę imion i nazwisk użytkowników zapisanych na webinar o podanym ID

```
CREATE FUNCTION [dbo].[GetWebinarMembers](@WebinarID BIGINT)
RETURNS TABLE
AS RETURN (
      SELECT FirstName, LastName FROM dbo.Members
      INNER JOIN dbo.WebinarAccess ON WebinarAccess.MemberID =
Members.MemberID
      WHERE WebinarID = @WebinarID
);
```

- ## ListClassAttendance – SM, MB, DM

Wypisuję liczbę osób obecnych na danych zajęciach

```
CREATE FUNCTION [dbo].[ListClassAttendance] (@ClassID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT P.[Name],
           SE.StartDate,
               (SELECT COUNT(1) FROM dbo.GetClassMembers(C.ClassID)) AS
[Signed up],
           (SELECT COUNT(1) FROM dbo.GetClassAttendance(C.ClassID)) AS
[Attendance] FROM dbo.ScheduleEvents AS SE
      INNER JOIN dbo.Classes AS C ON C.ClassID = SE.EventID
      INNER JOIN dbo.Subjects AS S ON S.SubjectID = C.SubjectID
      INNER JOIN dbo.Products AS P ON P.ProductID = S.ProductID
      WHERE C.ClassID = @ClassID
)
```

- ListClassesAttendance – SM, MB, DM

Wypisuje liczbę osób obecnych na zajęciach w danym roku (do statystyki)

```
CREATE FUNCTION [dbo].[ListClassesAttendance] (@ClassID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT P.[Name],
           SE.StartDate,
               (SELECT COUNT(1) FROM dbo.GetClassMembers(C.ClassID)) AS
[Signed up],
           (SELECT COUNT(1) FROM dbo.GetClassAttendance(C.ClassID)) AS
[Attendance] FROM dbo.ScheduleEvents AS SE
      INNER JOIN dbo.Classes AS C ON C.ClassID = SE.EventID
      INNER JOIN dbo.Subjects AS S ON S.SubjectID = C.SubjectID
      INNER JOIN dbo.Products AS P ON P.ProductID = S.ProductID
      WHERE C.ClassID = @ClassID
)
```

- ListCourseMembers – SM, MB, DM

Wypisuje informacje o użytkownikach zapisanych na kurs o podanej nazwie

```
CREATE FUNCTION [dbo].[ListCourseMembers] (@CourseName VARCHAR(200))
RETURNS TABLE
AS
RETURN
(
      SELECT * FROM dbo.GetCourseMembers(dbo.getCourseID(@CourseName))
)
```

● ListCourseModuleAttendance – SM, MB, DM

Zwraca listę osób obecnych na poszczególnym module kursu

```
CREATE FUNCTION [dbo].[ListCourseModuleAttendance] (@CourseModuleID
BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT P.[Name],
           SE.StartDate,
               (SELECT COUNT(1) FROM dbo.GetCourseMembers(C.CourseID)) AS
[Signed up],
           (SELECT COUNT(1) FROM
dbo.GetCourseModuleAttendance(CM.CourseModuleID)) AS [Attendance] FROM
dbo.ScheduleEvents AS SE
      INNER JOIN dbo.CourseModules AS CM ON CM.CourseModuleID = SE.EventID
      INNER JOIN dbo.Courses AS C ON C.CourseID = CM.CourseID
      INNER JOIN dbo.Products AS P ON P.ProductID = C.ProductID
      WHERE @CourseModuleID = CM.CourseModuleID
)
```

● ListFreeRoomsAtDatetimeRange – SM, MB, DM

Wypisuje listę pokojów z wolnymi miejscami w zadanym przedziale czasowym

```
CREATE FUNCTION [dbo].[ListFreeRoomsAtDatetimeRange]
(
      @StartDate DATETIME,
      @EndDate DATETIME
)
RETURNS TABLE
AS
RETURN
(
      SELECT R1.RoomID, R1.RoomNumber, R1.[Floor], R1.Seats FROM Rooms AS
R1
      EXCEPT
      SELECT SE.RoomID, R2.RoomNumber, R2.[Floor], R2.Seats FROM
ScheduleEvents AS SE
      INNER JOIN Rooms AS R2 ON R2.RoomID = SE.RoomID
      WHERE dbo.IsDatetimeRangeInEvent(SE.EventID, @StartDate, @EndDate) =
1
)
```

- ## ListMajorMembers – SM, MB, DM

Wypisuje infromacje o użytkownikach zapisanych na kierunek studiów o podanej nazwie

```sql
CREATE FUNCTION [dbo].[ListMajorMembers] (@MajorName VARCHAR(200))
RETURNS TABLE
AS
RETURN
(
      SELECT * FROM dbo.GetMajorMembers(dbo.getMajorID(@MajorName))
)
```

- ## ListMembersByRole – SM, MB, DM

Wypisuje listę wszystkich użytkowników którzy mają przypisaną zadaną rolę

```sql
CREATE FUNCTION [dbo].[ListMembersByRole] (@RoleName VARCHAR(40))
RETURNS TABLE
AS
RETURN
(
      SELECT M.MemberID,M.FirstName, M.LastName, M.Email, A.StreetName,
A.StreetNumber, CT.[Name] AS CityName, CT.PostalCode, CO.[Name] AS
CountryName FROM Members AS M
      INNER JOIN Addresses AS A ON A.AddressID = M.AddressID
      INNER JOIN Cities AS CT ON CT.CityID = A.CityID
      INNER JOIN Countries AS CO ON CO.CountryID = CT.CountryID
      WHERE dbo.IsMemberOfRole(M.MemberID, @RoleName) = 1
)
```

- ## ListWebinarMembers – SM, MB, DM

Wypisuje informacje o użytkownikach zapisanych na webinar o podanej nazwie

```sql
CREATE FUNCTION [dbo].[ListWebinarMembers] (@WebinarName VARCHAR(200))
RETURNS TABLE
AS
RETURN
(
      SELECT * FROM dbo.GetWebinarMembers(dbo.getWebinarID(@WebinarName))
)
```

- ListWebinarsStats – SM, MB, DM

Wypisuje informacje o użytkownikach zapisanych na zadany webinar

```
CREATE FUNCTION [dbo].[ListWebinarsStats] (@WebinarID BIGINT)
RETURNS TABLE
AS
RETURN
(
      SELECT P.[Name],
           SE.StartDate,
               (SELECT COUNT(1) FROM dbo.GetWebinarMembers(W.WebinarID))
AS [Signed up] FROM dbo.ScheduleEvents AS SE
      INNER JOIN dbo.Webinars AS W ON W.WebinarID = SE.EventID
      INNER JOIN dbo.Products AS P ON P.ProductID = W.ProductID
      WHERE W.WebinarID = @WebinarID
)
```

- GetBeginCourseDate – SM, MB, DM

Zwraca datę rozpoczęcia kursu o podanym ID

```
CREATE FUNCTION [dbo].[GetBeginCourseDate] (@CourseID BIGINT)
RETURNS DATETIME
AS
BEGIN
      RETURN (SELECT MIN(SE.StartDate) FROM dbo.CourseModules AS CM
      INNER JOIN dbo.ScheduleEvents AS SE ON SE.EventID =
CM.CourseModuleID
      WHERE CM.CourseID = @CourseID)
END
```

- GetBeginMajorDate – SM, MB, DM

Zwraca datę rozpoczęcia kierunku o podanym ID

```
CREATE FUNCTION [dbo].[GetBeginMajorDate] (@MajorID BIGINT)
RETURNS DATETIME
AS
BEGIN
      RETURN (SELECT MIN(SE.StartDate) FROM dbo.Majors AS M
      INNER JOIN dbo.Subjects AS S ON S.MajorID = M.MajorID
      INNER JOIN dbo.Classes AS C ON C.SubjectID = S.SubjectID
      INNER JOIN dbo.ScheduleEvents AS SE ON SE.EventID = C.ClassID
      WHERE M.MajorID = @MajorID)
END
```

- GetBeginMeetingDate – SM, MB, DM

Zwraca datę rozpoczęcia danego zjazdu

```sql
CREATE FUNCTION [dbo].[GetBeginMeetingDate] (@MeetingID BIGINT)
RETURNS DATETIME
AS
BEGIN
      RETURN (SELECT MIN(M.StartDate) FROM dbo.Meetings AS M
                  WHERE M.MeetingID = @MeetingID)
END
```

- GetBeginSubjectDate – SM, MB, DM

Zwraca datę rozpoczęcia przedmiotu o podanym ID

```sql
CREATE FUNCTION [dbo].[GetBeginSubjectDate] (@SubjectID BIGINT)
RETURNS DATETIME
AS
BEGIN
      RETURN (SELECT MIN(SE.StartDate) FROM dbo.Subjects AS S
      INNER JOIN dbo.Classes AS C ON C.SubjectID = S.SubjectID
      INNER JOIN dbo.ScheduleEvents AS SE ON SE.EventID = C.ClassID
      WHERE S.SubjectID = @SubjectID)
END
```

- GetBeginWebinarDate – SM, MB, DM

Zwraca datę rozpoczęcia webinaru o podanym ID

```sql
CREATE FUNCTION [dbo].[GetBeginWebinarDate] (@WebinarID BIGINT)
RETURNS DATETIME
AS
BEGIN
      RETURN (SELECT MIN(SE.StartDate) FROM dbo.Webinars AS W
      INNER JOIN dbo.ScheduleEvents AS SE ON SE.EventID = W.WebinarID
      WHERE W.WebinarID = @WebinarID)
END
```

- GetCategoryID – SM, MB, DM

Dla podanej nazwy kategorii zwraca jej ID z tabeli Categories

```sql
CREATE FUNCTION [dbo].[GetCategoryID] (@CategoryName VARCHAR(100))
RETURNS TINYINT
AS
BEGIN
      DECLARE @CategoryID TINYINT;
      SELECT @CategoryID = CategoryID FROM dbo.Categories WHERE
@CategoryName = CategoryName;
      RETURN @CategoryID;

END
```

- GetClassFreeSeats – SM, MB, DM

Zwraca liczbę wolnych miejsc na danym wydarzeniu

```sql
CREATE FUNCTION [dbo].[GetClassFreeSeats] (@EventID BIGINT)
RETURNS INT
AS
BEGIN
      DECLARE @RoomID BIGINT;
      DECLARE @RoomSeats INT;
      DECLARE @OccupiedSeats INT;
      SELECT @RoomID = SE.RoomID FROM dbo.ScheduleEvents AS SE;
      IF @RoomID IS NULL
            RETURN NULL
      SELECT @RoomSeats = R.Seats FROM dbo.Rooms AS R
      WHERE R.RoomID = @RoomID;
      SELECT @OccupiedSeats = COUNT(1) FROM Classes AS C
      WHERE C.ClassID = @EventID;
      RETURN @RoomSeats - @OccupiedSeats;

END
```

GetCourseAccessID – SM, MB, DM

Zwraca ID dostępu do kursu dla danego ID użytkownika i ID modułu

```sql
CREATE FUNCTION [dbo].[GetCourseAccessID]
(
      @MemberID BIGINT,
      @CourseModuleID BIGINT
)
RETURNS BIGINT
AS
BEGIN
      DECLARE @CourseAccessID BIGINT;

      SELECT @CourseAccessID = CA.AccessID FROM CourseAccess AS CA
      INNER JOIN CourseModules AS CM ON CM.CourseID = CA.CourseID
      WHERE CM.CourseModuleID = @CourseModuleID AND CA.MemberID =
@MemberID;

      RETURN @CourseAccessID;
END
```

GetCourseID – SM, MB, DM

Dla podanej nazwy kursu zwraca jego ID z tabeli Courses

```sql
CREATE FUNCTION [dbo].[GetCourseID] (@CourseName VARCHAR(200))
RETURNS BIGINT
AS
BEGIN
      DECLARE @CourseID BIGINT;
      SELECT @CourseID = C.CourseID FROM dbo.Courses AS C
      INNER JOIN dbo.Products AS P ON P.ProductID = C.ProductID
      WHERE P.[Name] = @CourseName;
      RETURN @CourseID;

END
```

- ## GetCourseModuleFreeSeats – SM, MB, DM

Zwraca liczbę wolnych miejsc na poszczególnych modułach kursu

```
CREATE FUNCTION [dbo].[GetCourseModuleFreeSeats] (@EventID BIGINT)
RETURNS INT
AS
BEGIN
      DECLARE @RoomID BIGINT;
      DECLARE @RoomSeats INT;
      DECLARE @OccupiedSeats INT;
      SELECT @RoomID = SE.RoomID FROM dbo.ScheduleEvents AS SE;
      IF @RoomID IS NULL
            RETURN NULL
      SELECT @RoomSeats = R.Seats FROM dbo.Rooms AS R
      WHERE R.RoomID = @RoomID;
      SELECT @OccupiedSeats = COUNT(1) FROM CourseModules AS CM
      INNER JOIN Courses AS C ON C.CourseID = CM.CourseID
      INNER JOIN CourseAccess AS CA ON CA.CourseID = C.CourseID
      WHERE CM.CourseModuleID = @EventID;
      RETURN @RoomSeats - @OccupiedSeats;
END
```

- ## GetDiscount – SM, MB, DM

Zwraca zniżkę na dany produkt zakupiony przez danego użytkownika

```
CREATE FUNCTION [dbo].[GetDiscount]
(
      @ProductID BIGINT,
      @MemberID BIGINT
)
RETURNS DECIMAL(2,2)
AS
BEGIN
      DECLARE @Discount DECIMAL(2,2);
      DECLARE @MemberRoleID SMALLINT;
      SELECT @MemberRoleID = @MemberRoleID FROM dbo.Members
      WHERE @MemberID = MemberID;
      SELECT @Discount = MAX(Discount) FROM dbo.Discounts
      WHERE @ProductID = ProductID AND @MemberRoleID = MemberRoleID;

      RETURN ISNULL(@Discount, 0)
END
```

- GetEducationFormID – SM, MB, DM

Dla podanej nazwy formy realizacji zajęć zwraca jej ID

```
CREATE FUNCTION [dbo].[GetEducationFormID] (@TypeName CHAR(40))
RETURNS TINYINT
AS
BEGIN
      DECLARE @EducationFormID TINYINT;

      SELECT @EducationFormID = EducationFormID FROM dbo.EducationForms
WHERE @TypeName = TypeName;

      RETURN @EducationFormID
END
```

- GetEventRecording – SM, MB, DM

Zwraca link do nagrania dla danego eventu

```
CREATE FUNCTION [dbo].[GetEventRecording] (@EventID BIGINT)
RETURNS VARCHAR(1600)
AS
BEGIN
      DECLARE @RecordingURL VARCHAR(1600);

      SELECT @RecordingURL = [URL] FROM dbo.Recordings
      WHERE EventID = @EventID;

      RETURN @RecordingURL;
END
```

- GetLanguageID – SM, MB, DM

Dla podanej nazwy języka tłumaczenia zwraca jego ID z tabeli Languages

```
CREATE FUNCTION [dbo].[GetLanguageID] (@LanguageName NVARCHAR(40))
RETURNS SMALLINT
AS
BEGIN
      DECLARE @LanguageID SMALLINT;
      SELECT @LanguageID = LanguageID FROM dbo.Languages
      WHERE [Name] = @LanguageName;
      RETURN @LanguageID;
END
```

- GetLastPaymentStatus – SM, MB, DM

Dla podanego ID zamówienia zwraca jego status płatności

```
CREATE FUNCTION [dbo].[GetLastPaymentStatus] (@OrderID BIGINT)
RETURNS CHAR(20)
AS
BEGIN
      DECLARE @PaymentStatus CHAR(20);

      SELECT @PaymentStatus = PS.StatusName FROM dbo.Orders AS O
      INNER JOIN dbo.Payments AS P ON P.OrderID = O.OrderID
      INNER JOIN dbo.PaymentStatus AS PS ON PS.PaymentStatusID =
P.PaymentStatusID
      WHERE O.OrderID = @OrderID
      ORDER BY P.PaidTime DESC;

      RETURN @PaymentStatus
END
```

- GetMajorID – SM, MB, DM

Dla podanej nazwy kierunku studiów zwraca jego ID z tabeli Majors

```
CREATE FUNCTION [dbo].[GetMajorID] (@MajorName VARCHAR(200))
RETURNS BIGINT
AS
BEGIN
      DECLARE @MajorID BIGINT;

      SELECT @MajorID = M.MajorID FROM dbo.Majors AS M
      INNER JOIN dbo.Products AS P ON P.ProductID = M.ProductID
      WHERE P.[Name] = @MajorName;

      RETURN @MajorID;

END
```

- GetMemberRoleID – SM, MB, DM

Dla podanej nazwy roli zwraca jej ID z tabeli MemberRoles

```
CREATE FUNCTION [dbo].[GetMemberRoleID]
(
      @RoleName VARCHAR(20)
)
RETURNS TINYINT
AS
BEGIN
      DECLARE @MemberRoleID TINYINT;
      SELECT @MemberRoleID = MemberRoleID FROM dbo.MemberRoles WHERE
@RoleName = RoleName;
      RETURN @MemberRoleID
END
```

- GetOrderDetailCount – SM, MB, DM

Dla podanego ID zamówienia zwraca liczbę poszczególnych produktów wchodzących w jego skład

```
CREATE FUNCTION [dbo].[GetOrderDetailCount] (@OrderID BIGINT)
RETURNS INT
AS
BEGIN
      DECLARE @OrderDetailCount INT = 0;
      SELECT @OrderDetailCount = COUNT(1) FROM dbo.OrderDetails
      WHERE OrderID = @OrderID;
      RETURN @OrderDetailCount;
END
```

- GetPaymentStatusID – SM, MB, DM

Dla podanej nazwy statusu zwraca jej ID z tabeli PaymentStatus

```
CREATE FUNCTION [dbo].[GetPaymentStatusID] (@StatusName NCHAR(10))
RETURNS TINYINT
AS
BEGIN
      DECLARE @PaymentStatusID TINYINT;
      SELECT @PaymentStatusID = PaymentStatusID FROM dbo.PaymentStatus
WHERE @StatusName = StatusName;

      RETURN @PaymentStatusID
END
```

- GetStudentEnrollID – SM, MB, DM

Dla podanych ID studenta i zajęć zwraca ID przypisania studenta do grupy z tabeli StdudentEnrolls

```
CREATE FUNCTION [dbo].[GetStudentEnrollID] (@StudentID BIGINT, @ClassID
BIGINT)
RETURNS BIGINT
AS
BEGIN
      DECLARE @StudentEnrollID BIGINT;

      SELECT @StudentEnrollID = SE.StudentEnrollID FROM dbo.StudentEnrolls
AS SE
      INNER JOIN dbo.Groups AS G ON G.GroupID = SE.GroupID
      INNER JOIN dbo.Classes AS C ON C.GroupID = G.GroupID
      WHERE SE.StudentID = @StudentID AND C.ClassID = @ClassID

      RETURN @StudentEnrollID
END
```

- GetSubjectFreeSeats – SM, MB, DM

Zwraca liczbę wolnych miejsc dla poszczególnego przedmiotu

```
CREATE FUNCTION [dbo].[GetSubjectFreeSeats] (@SubjectID BIGINT)
RETURNS INT
AS
BEGIN
      DECLARE @RoomSeats INT;
      DECLARE @OccupiedSeats INT;

      SELECT @RoomSeats = SUM(R.Seats) FROM dbo.ScheduleEvents AS SE
      INNER JOIN Classes AS C ON C.ClassID = SE.EventID
      INNER JOIN Subjects AS S ON S.SubjectID = C.SubjectID
      INNER JOIN Rooms AS R ON R.RoomID = SE.RoomID
      WHERE S.SubjectID = @SubjectID;

      SELECT @OccupiedSeats = SUM(1) FROM Classes AS C
      INNER JOIN Subjects AS S ON S.SubjectID = C.SubjectID
      INNER JOIN Groups AS G ON G.GroupID = C.GroupID
      INNER JOIN StudentEnrolls AS SE ON SE.GroupID = G.GroupID
      WHERE S.SubjectID = @SubjectID;

      RETURN @RoomSeats - @OccupiedSeats;

END
```

- GetSubjectID – SM, MB, DM

Dla podanej nazwy przedmiotu zwraca jego ID z tabeli Subjects

```
CREATE FUNCTION [dbo].[GetSubjectID] (@SubjectName VARCHAR(200))
RETURNS BIGINT
AS
BEGIN
      DECLARE @SubjectID BIGINT;

      SELECT @SubjectID = S.SubjectID FROM dbo.Subjects AS S
      INNER JOIN dbo.Products AS P ON P.ProductID = S.ProductID
      WHERE P.[Name] = @SubjectName;

      RETURN @SubjectID;

END
```

- GetWebinarAccessID – SM, MB, DM

Dla danego użytkownika i webinaru zwraca ID dostępu do tego webinaru.

```
CREATE FUNCTION [dbo].[GetWebinarAccessID]
(
      @MemberID BIGINT,
      @WebinarID BIGINT
)
RETURNS BIGINT
AS
BEGIN
      DECLARE @WebinarAccessID BIGINT;

      SELECT @WebinarAccessID = WA.AccessID FROM WebinarAccess AS WA
      INNER JOIN Webinars AS W ON W.WebinarID = WA.WebinarID
      WHERE W.WebinarID = @WebinarID AND WA.MemberID = @MemberID;

      RETURN @WebinarAccessID;
END
```

- ## GetWebinarID – SM, MB, DM

Dla podanej nazwy Webinaru zwraca jego ID z tabeli Webinars.

```sql
CREATE FUNCTION [dbo].[GetWebinarID] (@WebinarName VARCHAR(200))
RETURNS BIGINT
AS
BEGIN
      DECLARE @WebinarID BIGINT;

      SELECT @WebinarID = W.WebinarID FROM dbo.Webinars AS W
      INNER JOIN dbo.Products AS P ON P.ProductID = W.ProductID
      WHERE P.[Name] = @WebinarName;

      RETURN @WebinarID;

END
```

- ## HasNewEventCollision – SM, MB, DM

Sprawdza, czy nowe wydarzenie nie ma kolizji czasowej

```sql
CREATE FUNCTION [dbo].[HasNewEventCollision] (
      @EventID BIGINT,
      @StartDate DATETIME,
      @EndDate DATETIME
)
RETURNS BIT
AS
BEGIN
      DECLARE @HasCollision BIT = 0;

      SELECT @HasCollision = 1 FROM ScheduleEvents AS SE
      WHERE ((SE.StartDate BETWEEN @StartDate AND @EndDate) OR
                  (@StartDate BETWEEN SE.StartDate AND SE.EndDate)) AND
                  @EventID = SE.EventID;

      RETURN @HasCollision;
END
```

- HasRoomCollision – SM, MB, DM

Sprawdza, czy dany pokój nie został już wykorzystany do organizacji innego wydarzenia w tym samym czasie

```
CREATE FUNCTION [dbo].[HasRoomCollision]
(
      @RoomID BIGINT,
      @StartDate DATETIME,
      @EndDate DATETIME
)
RETURNS BIT
AS
BEGIN
      DECLARE @HasCollision BIT = 0;

      IF @RoomID IS NULL
            RETURN 0;

      SELECT @HasCollision = 1
      WHERE 1 = ANY (
            SELECT dbo.IsDatetimeRangeInEvent(SE.EventID, @StartDate,
@EndDate) FROM ScheduleEvents AS SE
            WHERE SE.RoomID = @RoomID);

      RETURN @HasCollision;
END
```

- HasTeacherCollision – SM, MB, DM

Sprawdza, czy dany nie prowadzi innego wydarzenia w tym samym czasie

```
CREATE FUNCTION [dbo].[HasTeacherCollision]
(
      @TeacherID BIGINT,
      @StartDate DATETIME,
      @EndDate DATETIME
)
RETURNS BIT
AS
BEGIN
      DECLARE @CourseModulesCollision BIT = 0;
      DECLARE @SubjectCollision BIT = 0;
      DECLARE @WebinarCollision BIT = 0;

      SELECT @CourseModulesCollision = 1
      WHERE 1 = ANY (
            SELECT dbo.HasNewEventCollision(CMT.CourseModuleID,
@StartDate, @EndDate)
            FROM CourseModuleTeachers AS CMT
            WHERE CMT.TeacherID = @TeacherID);
      IF @CourseModulesCollision = 1
            RETURN 1;
      SELECT @SubjectCollision = 1
      WHERE 1 = ANY (
            SELECT dbo.HasNewEventCollision(C.ClassID, @StartDate,
@EndDate)
            FROM SubjectTeachers AS ST
            INNER JOIN Classes AS C ON C.SubjectID = ST.SubjectID
            WHERE ST.TeacherID = @TeacherID);
      IF @SubjectCollision = 1
            RETURN 1;
      SELECT @WebinarCollision = 1
      WHERE 1 = ANY (
            SELECT dbo.HasNewEventCollision(W.WebinarID, @StartDate,
@EndDate)
            FROM Webinars AS W
            WHERE W.LeaderID = @TeacherID);

      IF @WebinarCollision = 1
            RETURN 1;

      RETURN 0;
END
```

- IsDatetimeEqualsDay – SM, MB, DM

Dla dwóch podanych data zwraca czy są one równe co do dnia

```
CREATE FUNCTION [dbo].[IsDatetimeEqualsDay]
(
      @Datetime DATETIME,
      @Date DATE
)
RETURNS BIT
AS
BEGIN
      DECLARE @IsEqauls BIT = 0;

      SELECT @IsEqauls = 1
      WHERE YEAR(@Datetime) = YEAR(@Date) AND
              MONTH(@Datetime) = MONTH(@Date) AND
              DAY(@Datetime) = DAY(@Date);

      RETURN @IsEqauls;
END
```

- IsDatetimeRangeInEvent – SM, MB, DM

Sprawdza, czy zadany przedział czasowy nie koliduje z podanym wydarzeniem

```
CREATE FUNCTION [dbo].[IsDatetimeRangeInEvent]
(
      @EventID BIGINT,
      @StartDate DATETIME,
      @EndDate DATETIME
)
RETURNS BIT
AS
BEGIN
DECLARE @IsOverlap BIT = 0;

      SELECT @IsOverlap = 1 FROM dbo.ScheduleEvents AS SE
      WHERE
            ((@StartDate BETWEEN SE.StartDate AND SE.EndDate) OR
            (SE.StartDate BETWEEN @StartDate AND @EndDate)) AND SE.EventID
= @EventID;

      RETURN @IsOverlap;

END
```

- ## IsDayInRange – SM, MB, DM

Dla podanej dokładnej daty oraz pewnego przedziału czasu zwraca czy podana data zawiera się w podanych ramach czasowych

```
CREATE FUNCTION [dbo].[IsDayInRange]
(
      @Datetime DATETIME,
      @StartDate DATE,
      @EndDate DATE
)
RETURNS BIT
AS
BEGIN
      DECLARE @IsInRange BIT = 0;

      SELECT @IsInRange = 1
      WHERE YEAR(@Datetime) >= YEAR(@StartDate) AND YEAR(@Datetime) <= YEAR(@EndDate) AND
              MONTH(@Datetime) >= MONTH(@StartDate) AND MONTH(@Datetime) <= MONTH(@EndDate) AND
              DAY(@Datetime) >= DAY(@StartDate) AND DAY(@Datetime) <= DAY(@EndDate);

      RETURN @IsInRange;
END
```

- ## IsEventOverlap – SM, MB, DM

Zwraca wartość prawda/fałsz, czy podane dwa wydarzenia kolidują ze sobą czasowo

```
CREATE FUNCTION [dbo].[IsEventOverlap]
(
      @Event1ID BIGINT,
      @Event2ID BIGINT
)
RETURNS BIT
AS
BEGIN
      DECLARE @IsOverlap BIT = 0;
      SELECT @IsOverlap = 1 FROM dbo.ScheduleEvents AS SE1
      INNER JOIN dbo.ScheduleEvents AS SE2 ON SE2.EventID = @Event2ID
      WHERE SE1.EventID = @Event1ID AND (
            (SE1.StartDate BETWEEN SE2.StartDate AND SE2.EndDate) OR
            (SE2.StartDate BETWEEN SE1.StartDate AND SE1.EndDate));
      RETURN @IsOverlap;
END
```

- IsMemberOfRole – SM, MB, DM

Zwraca wartość prawda/fałsz (1/0), czy użytkownik o danym ID ma nadaną rolę o podanej nazwie

```
CREATE FUNCTION [dbo].[IsMemberOfRole]
(
      @MemberID BIGINT,
      @RoleName VARCHAR(20)
)
RETURNS BIT
AS
BEGIN
      DECLARE @MemberRoleID TINYINT;

      IF @MemberID IS NULL
            RETURN 1;

      SELECT @MemberRoleID = MemberRoleID FROM dbo.MemberRoles WHERE
@RoleName = RoleName;

      IF EXISTS (SELECT 1 FROM dbo.AssignedMemberRoles WHERE @MemberRoleID
= MemberRoleID AND @MemberID = MemberID)
            RETURN 1;
      RETURN 0;
END
```

- IsOrderPaid – SM, MB, DM

Zwraca informację o tym, czy dane zamówienie zostało opłacone

```sql
CREATE FUNCTION [dbo].[IsOrderPaid] (@OrderID BIGINT)
RETURNS BIT
AS
BEGIN
      DECLARE @IsPaid BIT = 0;
      DECLARE @IsAdvancePaid BIT = 0;
      DECLARE @HasAdvance BIT = 0;
      DECLARE @PostponementEndDate DATETIME;

      SELECT @PostponementEndDate = PP.PostponeEndDate FROM dbo.Postponements AS PP
      WHERE PP.OrderID = @OrderID;

      IF GETDATE() < ISNULL(@PostponementEndDate, '1900-01-01 00:00:00')
            RETURN 1;

      SELECT @HasAdvance = O.IsInAdvance FROM dbo.Orders AS O
      WHERE O.OrderID = @OrderID;

      IF @HasAdvance = 1
            SELECT @IsAdvancePaid = 1 FROM dbo.Payments AS P
            INNER JOIN dbo.PaymentStatus AS PS ON PS.PaymentStatusID = P.PaymentStatusID
            WHERE P.OrderID = @OrderID AND P.IsAdvance = 1 AND PS.StatusName = 'success';

      SELECT @IsPaid = 1 FROM dbo.Payments AS P
      INNER JOIN dbo.PaymentStatus AS PS ON PS.PaymentStatusID = P.PaymentStatusID
      WHERE P.OrderID = @OrderID AND P.IsAdvance = 0 AND PS.StatusName = 'success';

      DECLARE @IsAllPaid BIT = 0;

      SELECT @IsAllPaid = 1 WHERE @IsPaid = 1 AND (@HasAdvance = 0 OR @IsAdvancePaid = 1);

      RETURN @IsAllPaid;
END
```

- IsProductInCategory – SM, MB, DM

Dla podanej nazwy kategorii i ID produktu, sprawdza czy taki produkt należy do podanej kategorii

```
CREATE FUNCTION [dbo].[IsProductInCategory] (@ProductID BIGINT,
@CategoryName CHAR(20))
RETURNS BIT
AS
BEGIN
      DECLARE @CategoryID TINYINT;

      IF @ProductID IS NULL
            RETURN 1;

      SELECT @CategoryID = CategoryID FROM dbo.Categories WHERE
CategoryName = @CategoryName;

      IF EXISTS (SELECT 1 FROM dbo.Products WHERE @CategoryID = CategoryID
AND @ProductID = ProductID)
            RETURN 1;
      RETURN 0;
END
```

Procedury

AddAccesToCourse – SM, MB, DM

Udzielenie użytkownikowi dostępu do kursu

```sql
CREATE PROCEDURE [dbo].[AddAccessToCourse]
    @MemberID BIGINT,
    @CourseID BIGINT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        INSERT INTO dbo.CourseAccess (MemberID, CourseID)
        VALUES (@MemberID, @CourseID);
        COMMIT;
        PRINT 'Access (Course) added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

AddAccessToWebinar – SM, MB, DM

Udzielenie użytkownikowi dostępu do danego webinaru

```sql
CREATE PROCEDURE [dbo].[AddAccessToCourse]
    @MemberID BIGINT,
    @CourseID BIGINT
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
        INSERT INTO dbo.CourseAccess (MemberID, CourseID)
        VALUES (@MemberID, @CourseID);
        COMMIT;
        PRINT 'Access (Course) added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddAddress – SM, MB, DM

Dodanie nowego adresu

```
CREATE PROCEDURE [dbo].[AddAddress]
    @CityName NVARCHAR(160),
      @PostalCode NVARCHAR(10),
      @CountryName NVARCHAR(100),
    @StreetName NVARCHAR(160),
    @StreetNumber NVARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        DECLARE @CityID BIGINT;
            SELECT @CityID = CityID FROM dbo.Cities WHERE [Name] =
@CityName;

            IF @CityID IS NULL
                  EXEC dbo.AddCity @Name = @CityName,
                                   @CountryName = @CountryName,
                                   @PostalCode = @PostalCode;

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

        INSERT INTO [dbo].[Addresses] ([CityID], [StreetName],
[StreetNumber])
        VALUES (@CityID, @StreetName, @StreetNumber);

        COMMIT;
        PRINT 'Address added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
        PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- AddBuilding – SM, MB, DM

Dodanie nowego budynku

```
CREATE PROCEDURE [dbo].[AddBuilding]
    @Name nvarchar(100),
    @AddressID bigint
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Addresses] WHERE [AddressID] =
@AddressID)
            THROW 50000, 'Invalid AddressID. AddressID does not exist.',
1;

        IF NOT EXISTS (SELECT 1 FROM [dbo].[Buildings] WHERE [Name] =
@Name)
        BEGIN
            INSERT INTO [dbo].[Buildings] ([Name], [AddressID])
            VALUES (@Name, @AddressID);

            COMMIT;
            PRINT 'Building added successfully.';
        END
        ELSE
            THROW 50000, 'Building with the same name already exists.', 1;
    END TRY
    BEGIN CATCH
        ROLLBACK;
        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- AddCategory – SM, MB, DM

Dodanie nowej kategorii

```sql
CREATE PROCEDURE [dbo].[AddCategory]
    @CategoryName nvarchar(100)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[Categories] ([CategoryName])
        VALUES (@CategoryName);

        COMMIT;
        PRINT 'Category added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
        PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- AddCity – SM, MB, DM

Dodawanie nowego miasta

```
CREATE PROCEDURE [dbo].[AddCity]
    @Name NVARCHAR(160),
    @CountryName VARCHAR(100),
    @PostalCode NVARCHAR(10)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

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

            IF @CountryID IS NULL
                  EXEC dbo.AddCountry @Name = @CountryName;

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

        INSERT INTO Cities ([Name], CountryID, PostalCode)
        VALUES (@Name, @CountryID, @PostalCode);

        PRINT 'City added successfully.';
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddClass – SM, MB, DM

Utworzenie nowyej klasy

```
CREATE PROCEDURE [dbo].[AddClass]
      @SubjectID BIGINT,
      @GroupID BIGINT,
    @StartDate DATETIME,
    @EndDate DATETIME,
    @RoomID BIGINT = NULL,
    @InterpreterID BIGINT = NULL,
      @Language NVARCHAR(40)
AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @TeacherID BIGINT;
            DECLARE @EventID BIGINT;
            DECLARE @HasClassCollision BIT = 0;
            IF dbo.IsMemberOfRole(@InterpreterID, 'interpreter') = 0
                  THROW 50000, 'Specified non-interpreter user.', 1;
            SELECT @TeacherID = G.TeacherID FROM Groups AS G
            WHERE @GroupID = G.GroupID;
            SELECT @HasClassCollision = 1
            WHERE 1 = ANY (
                  SELECT dbo.HasNewEventCollision(C.ClassID, @StartDate,
@EndDate) FROM Groups AS G
                  INNER JOIN Classes AS C ON C.GroupID = G.GroupID
                  WHERE @GroupID = G.GroupID);
            IF @HasClassCollision = 1
                  THROW 50000, 'Class collision.', 1;
            IF dbo.HasTeacherCollision(@TeacherID, @StartDate, @EndDate) =
1
                  THROW 50000, 'Teacher collision.', 1;
            IF dbo.HasRoomCollision(@RoomID, @StartDate, @EndDate) = 1
                  THROW 50000, 'Room collision.', 1;
            EXEC dbo.AddScheduledEvent @StartDate = @StartDate,
                                                   @EndDate = @EndDate,
                                                   @RoomID = @RoomID,
                                                   @InterpreterID =
@InterpreterID,
                                                   @Language = @Language,
                                                   @NewEventID = @EventID
OUTPUT;
```

```
            INSERT INTO dbo.Classes (ClassID, SubjectID, GroupID)
            VALUES (@EventID, @SubjectID, @GroupID);
            PRINT 'Class added sucessfully.';
            COMMIT;
      END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddCompany – SM, MB, DM

Dodawanie nowej firmy

```
CREATE PROCEDURE [dbo].[AddCompany]
    @Name NVARCHAR(100),
    @AddressID BIGINT,
    @Phone NCHAR(15),
    @Email NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        IF NOT EXISTS (SELECT 1 FROM [dbo].[Addresses] WHERE [AddressID] =
@AddressID)
            THROW 50000, 'Invalid AddressID. AddressID does not exist.',
1;

            INSERT INTO [dbo].[Companies] ([Name], [AddressID], [Phone],
[Email])
        VALUES (@Name, @AddressID, @Phone, @Email);

            COMMIT;
            PRINT('Company added successfully.');
    END TRY
    BEGIN CATCH
        ROLLBACK;
        PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- AddCountry – SM, MB, DM

Dodanie nowego kraju

```
CREATE PROCEDURE [dbo].[AddCountry]
    @Name NCHAR(50)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRY
        BEGIN TRANSACTION;

        INSERT INTO dbo.Countries ([Name])
        VALUES (@Name);

        PRINT 'Country added successfully.';
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
        PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddCourse – SM, MB, DM

Dodanie nowego kursu

```
CREATE PROCEDURE [dbo].[AddCourse]
      @Name NVARCHAR(200),
    @Description NVARCHAR(MAX),
    @Price MONEY,
    @AdvancePrice MONEY = NULL,
    @CoordinatorID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @CategoryID TINYINT;
            DECLARE @ProductID BIGINT;

            IF dbo.IsMemberOfRole(@CoordinatorID, 'Coordinator') = 0
                  THROW 50000, 'Specified a user without a coordinator
role.', 1

```

```
            SELECT @CategoryID = dbo.GetCategoryID('course');

            EXEC dbo.AddProduct @ProductName = @Name,
                                          @ProductDescription =
@Description,
                                          @CategoryID = @CategoryID,
                                          @Price = @Price,
                                          @AdvancePrice = @AdvancePrice,
                                          @NewProductID = @ProductID
OUTPUT;

        INSERT INTO [dbo].[Courses] ([CoordinatorID], [ProductID])
        VALUES (@CoordinatorID, @ProductID);

        COMMIT;
        PRINT 'Course added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- ## AddCourseDiploma – SM, MB, DM

Dodawanie nowego dyplomu za ukończenie kursu

```
CREATE PROCEDURE [dbo].[AddCourseDiploma]
    @MemberID BIGINT,
      @IssueDate DATE,
    @CourseID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @DiplomaID BIGINT;

            EXEC dbo.AddDiploma @MemberID = @MemberID,
                                          @IssueDate = @IssueDate,
                                          @NewDiplomaID = @DiplomaID
OUTPUT;

        INSERT INTO [dbo].[CourseDiplomas] ([DiplomaID], [CourseID])
        VALUES (@DiplomaID, @CourseID);

        COMMIT;
        PRINT 'Diploma (Course) added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddCourseModule – SM, MB, DM

Dodanie nowego modułu

```
CREATE PROCEDURE [dbo].[AddCourseModule]
      @CourseID BIGINT,
      @ModuleName NVARCHAR(200),
      @EducationForm CHAR(40),
    @StartDate DATETIME,
    @EndDate DATETIME,
    @RoomID BIGINT = NULL,
    @InterpreterID BIGINT = NULL,
      @Language NVARCHAR(40)
AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @EventID BIGINT;
            DECLARE @EducationFormID TINYINT =
dbo.GetEducationFormID(@EducationForm);
            DECLARE @HasCourseModuleCollision BIT = 0;

            IF dbo.IsMemberOfRole(@InterpreterID, 'interpreter') = 0
                  THROW 50000, 'Specified non-interpreter user.', 1;

            SELECT @HasCourseModuleCollision = 1
            WHERE 1 = ANY (
                  SELECT dbo.HasNewEventCollision(CM.CourseModuleID,
@StartDate, @EndDate)
                  FROM CourseModules AS CM
                  WHERE CM.CourseID = @CourseID);

            IF @HasCourseModuleCollision = 1
                  THROW 50000, 'Course module collision.', 1;

            IF dbo.HasRoomCollision(@RoomID, @StartDate, @EndDate) = 1
                  THROW 50000, 'Room collision.', 1;

            EXEC dbo.AddScheduledEvent @StartDate = @StartDate,
                                                   @EndDate = @EndDate,
                                                   @RoomID = @RoomID,
                                                   @InterpreterID =
@InterpreterID,
                                                   @Language = @Language,
                                                   @NewEventID = @EventID
OUTPUT;

            INSERT INTO dbo.CourseModules (CourseModuleID, CourseID,
[Name], EducationFormID)
            VALUES (@EventID, @CourseID, @ModuleName, @EducationFormID);

            PRINT 'Course module added sucessfully.';
            COMMIT;
      END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
GO
```

- AddCourseModuleAttendance – SM, MB, DM

Dodanie obecności dla danego użytkownika w module z kursu.

```
CREATE PROCEDURE [dbo].[AddCourseModuleAttendance]
    @MemberID BIGINT,
    @CourseModuleID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @AccessID BIGINT = dbo.GetCourseAccessID(@MemberID,
@CourseModuleID);

        INSERT INTO dbo.CourseAttendance (AccessID, CourseModuleID)
        VALUES (@AccessID, @CourseModuleID);

        COMMIT TRANSACTION;
        PRINT 'Course module attendance added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddCredential – SM, MB, DM

Dodawanie nowego klucza uwierzytelniania.

```
CREATE PROCEDURE [dbo].[AddCredential]
      @ExpireDate DATE,
      @NewCredentialID BIGINT OUTPUT
AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @NewCredential TABLE (CredentialID BIGINT NOT NULL);

        INSERT INTO [dbo].[Credentials] ([VerificationDeadline],
[ExpireDate])
            OUTPUT Inserted.CredentialID INTO @NewCredential
        VALUES (DATEADD(MINUTE, 10, GETDATE()), @ExpireDate);

            SELECT @NewCredentialID = CredentialID FROM @NewCredential;

        COMMIT;
        PRINT 'Credential added successfully.';
            RETURN;
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddDiploma – SM, MB, DM

Dodanie nowego dyplomu

```
CREATE PROCEDURE [dbo].[AddDiploma]
    @MemberID BIGINT,
    @IssueDate DATE,
      @NewDiplomaID BIGINT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @NewDiploma TABLE (DiplomaID BIGINT NOT NULL);

        INSERT INTO [dbo].[Diplomas] ([MemberID], [IssueDate])
            OUTPUT Inserted.DiplomaID INTO @NewDiploma
        VALUES (@MemberID, @IssueDate);
        SELECT @NewDiplomaID = DiplomaID FROM @NewDiploma;

        COMMIT;
            RETURN;
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## AddDiscount – SM, MB, DM

Dodawanie nowej zniżki

```
CREATE PROCEDURE [dbo].[AddDiscount]
    @ProductID BIGINT,
    @Discount DECIMAL(2, 2),
    @RoleName VARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @MemberRoleID TINYINT =
dbo.GetMemberRoleID(@RoleName);

        INSERT INTO [dbo].[Discounts] ([ProductID], [Discount],
[MemberRoleID])
        VALUES (@ProductID, @Discount, @MemberRoleID);

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## AddEducationForm – SM, MB, DM

Dodanie nowej formy edukacji do oferty strony

```
CREATE PROCEDURE [dbo].[AddEducationForm]
    @TypeName NVARCHAR(40)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[EducationForms] ([TypeName])
        VALUES (@TypeName);

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddGroup – SM, MB, DM

Dodawanie nowej grupy zajęciowej

```
CREATE PROCEDURE [dbo].[AddGroup]
    @GroupNumber SMALLINT,
    @TeacherID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[Groups] ([GroupNumber], [TeacherID])
        VALUES (@GroupNumber, @TeacherID);

        COMMIT;
            PRINT('Group added successfully');
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddIntership – SM, MB, DM

Dodawanie nowej praktyki

```
CREATE PROCEDURE [dbo].[AddInternship]
    @MajorID BIGINT,
    @CompanyID BIGINT,
    @StudentID BIGINT,
    @StartDate DATETIME,
    @EndDate DATETIME
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;
    BEGIN TRY
        INSERT INTO [dbo].[Interships] ([MajorID], [CompanyID],
[StudentID], [StartDate], [EndDate])
        VALUES (@MajorID, @CompanyID, @StudentID, @StartDate, @EndDate);

        COMMIT;
            PRINT('Internship added successfully.');
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## AddInterpreterLanguage – SM, MB, DM

Dodanie nowego tłumacza

```
CREATE PROCEDURE [dbo].[AddInterpreterLanguage]
    @InterpreterID BIGINT,
    @LanguageID SMALLINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            IF dbo.IsMemberOfRole(@InterpreterID, 'interpreter') = 0
                  THROW 50000, 'Specified a member without an interpreter
role', 1;

        INSERT INTO [dbo].[InterpreterLanguages] ([InterpreterID],
[LanguageID])
        VALUES (@InterpreterID, @LanguageID);

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## AddLanguage – SM, MB, DM

Dodawanie nowego języka tłumaczenia

```
CREATE PROCEDURE [dbo].[AddLanguage]
    @Name NVARCHAR(40)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[Languages] ([Name])
        VALUES (@Name);

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddMajor – SM, MB, DM

Dodawanie nowego kierunku studiów

```
CREATE PROCEDURE [dbo].[AddMajor]
      @Name NVARCHAR(200),
    @Description NVARCHAR(MAX),
    @Price MONEY,
    @AdvancePrice MONEY = NULL,
    @CoordinatorID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @CategoryID TINYINT;
            DECLARE @ProductID BIGINT;

            IF dbo.IsMemberOfRole(@CoordinatorID, 'coordinator') = 0
                  THROW 50000, 'Specified a user without a coordinator
role.', 1

            SELECT @CategoryID = dbo.GetCategoryID('major');

            EXEC dbo.AddProduct @ProductName = @Name,
                                          @ProductDescription =
@Description,
                                          @CategoryID = @CategoryID,
                                          @Price = @Price,
                                          @AdvancePrice = @AdvancePrice,
                                          @NewProductID = @ProductID
OUTPUT;

            INSERT INTO dbo.Majors (CoordinatorID, ProductID)
            VALUES (@CoordinatorID, @ProductID);

        COMMIT;
        PRINT 'Major added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddMajorDiploma – SM, MB, DM

Dodawanie nowego dyplomu za ukończenie danego kierunku studiów

```
CREATE PROCEDURE [dbo].[AddMajorDiploma]
    @MemberID BIGINT,
      @IssueDate DATE,
    @MajorID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @DiplomaID BIGINT;

            EXEC dbo.AddDiploma @MemberID = @MemberID,
                                          @IssueDate = @IssueDate,
                                          @NewDiplomaID = @DiplomaID
OUTPUT;

        INSERT INTO [dbo].[MajorDiplomas] ([DiplomaID], [MajorID])
        VALUES (@DiplomaID, @MajorID);

        COMMIT;
        PRINT 'Diploma (Major) added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddMeeting – SM, MB, DM

Dodawanie nowego zjazdu na studiach

```
CREATE PROCEDURE [dbo].[AddMeeting]
      @MajorID BIGINT,
      @Name NVARCHAR(200),
    @Description NVARCHAR(MAX),
      @StartDate DATETIME,
      @EndDate DATETIME,
    @Price MONEY,
    @AdvancePrice MONEY = NULL
AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @CategoryID TINYINT;
            DECLARE @ProductID BIGINT;

            SELECT @CategoryID = dbo.GetCategoryID('meeting');

            EXEC dbo.AddProduct @ProductName = @Name,
                                          @ProductDescription =
@Description,
                                          @CategoryID = @CategoryID,
                                          @Price = @Price,
                                          @AdvancePrice = @AdvancePrice,
                                          @NewProductID = @ProductID
OUTPUT;

            INSERT INTO dbo.Meetings (MajorID, ProductID, StartDate,
EndDate)
            VALUES (@MajorID, @ProductID, @StartDate, @EndDate);
```

```
        COMMIT;
        PRINT 'Meeting added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddMeetingAccess – SM, MB, DM

Przyznanie użytkownikowi dostępu do zjazdu.

```
CREATE PROCEDURE [dbo].[AddMeetingAccess]
      @MeetingID BIGINT,
      @MemberID BIGINT
AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            INSERT INTO MeetingAccess (MeetingID, MemberID)
            VALUES (@MeetingID, @MemberID);

        COMMIT;
        PRINT 'Meeting access added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## AddMeetingAttendance – SM, MB, DM

Dodanie obecności dla studenta na danym zjeździe

```
CREATE PROCEDURE [dbo].[AddMeetingAttendance]
      @AccessID BIGINT
AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            INSERT INTO dbo.MeetingAttendance (AccessID, MemberID)
            VALUES (@AccessID, @MemberID);

        COMMIT;
        PRINT 'Meeting attendance added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## AddMember – SM, MB, DM

Zarejestrowanie nowego użytkownika

```
CREATE PROCEDURE [dbo].[AddMember]
    @FirstName NVARCHAR(40),
    @LastName NVARCHAR(40),
    @Login NCHAR(80),
    @Email NVARCHAR(160),
    @AddressID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @CredentialsID BIGINT;
            DECLARE @NewMember TABLE (MemberID BIGINT NOT NULL);

            EXEC dbo.AddCredential @ExpireDate = NULL,
                                             @NewCredentialID =
@CredentialsID OUTPUT;
```

```
        INSERT INTO [dbo].[Members] ([FirstName], [LastName], [Login],
[Email], [AddressID], [CredentialsID], [IsActive])
        OUTPUT Inserted.MemberID INTO @NewMember
            VALUES (@FirstName, @LastName, @Login, @Email, @AddressID,
@CredentialsID, 1);

        COMMIT;
        PRINT 'Member added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- AddMemberRole – SM, MB, DM

Dodawanie nowej roli użytkownika

```
CREATE PROCEDURE [dbo].[AddMemberRole]
    @RoleName nchar(20)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[MemberRoles] ([RoleName])
        VALUES (@RoleName);

        COMMIT;
        PRINT 'Member Role added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- ## AddOrder – SM, MB, DM

Dodawanie nowego zamówienia

```
CREATE PROCEDURE [dbo].[AddOrder]
    @OrderDate DATE,
    @MemberID BIGINT,
    @IsInAdvance BIT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[Orders] ([OrderDate], [MemberID],
[IsInAdvance])
        VALUES (@OrderDate, @MemberID, @IsInAdvance);

        COMMIT;
        PRINT 'Order added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- ## AddOrderDetail – SM, MB, DM

Dodanie szczegółów nowego zamówienia

```
CREATE PROCEDURE [dbo].[AddOrderDetail]
    @OrderID BIGINT,
    @ProductID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[OrderDetails] ([OrderID], [ProductID])
        VALUES (@OrderID, @ProductID);

        COMMIT;
        PRINT 'OrderDetail added successfully.';
    END TRY
    BEGIN CATCH
    ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- ## AddPayment – SM, MB, DM

Dodawanie nowej płatności

```
CREATE PROCEDURE [dbo].[AddPayment]
    @OrderID BIGINT,
    @URL NCHAR(1600),
    @isAdvance BIT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @PaymentStatusID TINYINT;

            IF dbo.GetOrderDetailCount(@OrderID) = 0
                  THROW 50000, 'Payment cannot be added to empty order',
1;

            SELECT @PaymentStatusID =
dbo.GetPaymentStatusID('processing');

        INSERT INTO [dbo].[Payments] ([OrderID], [URL], [PaidTime],
[PaymentStatusID], [isAdvance])
        VALUES (@OrderID, @URL, GETDATE(), @PaymentStatusID, @isAdvance);

        COMMIT;
        PRINT 'Payment added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- ## AddPaymentStatus – SM, MB, DM

Dodawanie nowego statusu płatności

```
CREATE PROCEDURE [dbo].[AddPaymentStatus]
    @StatusName nchar(10)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO [dbo].[PaymentStatus] ([StatusName])
        VALUES (@StatusName);

        COMMIT;
        PRINT 'PaymentStatus added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- AddPostponement – SM, MB, DM

Dodanie nowego odroczenia płatności dla danego zamówienia

```
CREATE PROCEDURE [dbo].[AddPostponement]
      @OrderID BIGINT,
      @PostponeDate DATE
AS
BEGIN
      SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO dbo.Postponements (OrderID, PostponeStartDate, 
PostponeEndDate)
        VALUES (@OrderID, GETDATE(), @PostponeDate);

        COMMIT;
        PRINT 'Postponement added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddProduct – SM, MB, DM

Dodawanie nowego produktu

```
CREATE PROCEDURE [dbo].[AddProduct]
    @ProductName NVARCHAR(200),
    @ProductDescription NVARCHAR(MAX),
    @CategoryID TINYINT,
    @Price MONEY,
    @AdvancePrice MONEY = NULL,
      @NewProductID BIGINT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @NewProduct TABLE (ProductID BIGINT NOT NULL);

        INSERT INTO [dbo].[Products] ([Name], [Description], [CategoryID],
[Price], [AdvancePrice])
            OUTPUT Inserted.ProductID INTO @NewProduct
        VALUES (@ProductName, @ProductDescription, @CategoryID, @Price,
@AdvancePrice);

            SELECT @NewProductID = ProductID FROM @NewProduct;
        COMMIT;
            RETURN;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- AddRecording – SM, MB, DM

Dodanie nowego nagrania

```
CREATE PROCEDURE [dbo].[AddRecording]
    @EventID BIGINT,
    @URL VARCHAR(1600),
    @Title VARCHAR(200)
AS
BEGIN
    SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO dbo.Recordings (EventID, URL, Title)
        VALUES (@EventID, @URL, @Title);

        COMMIT;
        PRINT 'Recording added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddRoom – SM, MB, DM

Dodawanie nowego pokoju

```
CREATE PROCEDURE [dbo].[AddRoom]
    @RoomNumber NCHAR(20),
    @Floor SMALLINT,
    @BuildingID BIGINT,
      @Seats INT
AS
BEGIN
    SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO dbo.Rooms (RoomNumber, [Floor], BuildingID, Seats)
        VALUES (@RoomNumber, @Floor, @BuildingID, @Seats);

        COMMIT;
        PRINT 'Room added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddScheduledEvent – SM, MB, DM

Dodanie nowego wydarzenie w harmonogramie

```
CREATE PROCEDURE [dbo].[AddScheduledEvent]
    @StartDate DATETIME,
    @EndDate DATETIME,
    @RoomID BIGINT = NULL,
    @InterpreterID BIGINT = NULL,
      @Language NVARCHAR(40),
      @NewEventID BIGINT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @NewEvent TABLE (EventID BIGINT NOT NULL);

            DECLARE @LanguageID SMALLINT = dbo.GetLanguageID(@Language);

        INSERT INTO dbo.ScheduleEvents (StartDate, EndDate, RoomID,
InterpreterID, LanguageID)
            OUTPUT Inserted.EventID INTO @NewEvent
        VALUES (@StartDate, @EndDate, @RoomID, @InterpreterID,
@LanguageID);
            SELECT @NewEventID = EventID FROM @NewEvent;

        COMMIT;
        PRINT 'Scheduled event added successfully.';
            RETURN;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- ## AddStudentAttendance – SM, MB, DM

Dodawanie obecności studentowi na danych zajęciach

```
CREATE PROCEDURE [dbo].[AddStudentAttendance]
    @StudentID BIGINT,
    @ClassID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @StudentEnrollID BIGINT =
dbo.GetStudentEnrollID(@StudentID, @ClassID);

        INSERT INTO dbo.StudentAttendance (StudentEnrollID, ClassID)
        VALUES (@StudentEnrollID, @ClassID);

        PRINT 'Student attendance added successfully.';
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- ## AddStudentGrade – SM, MB, DM

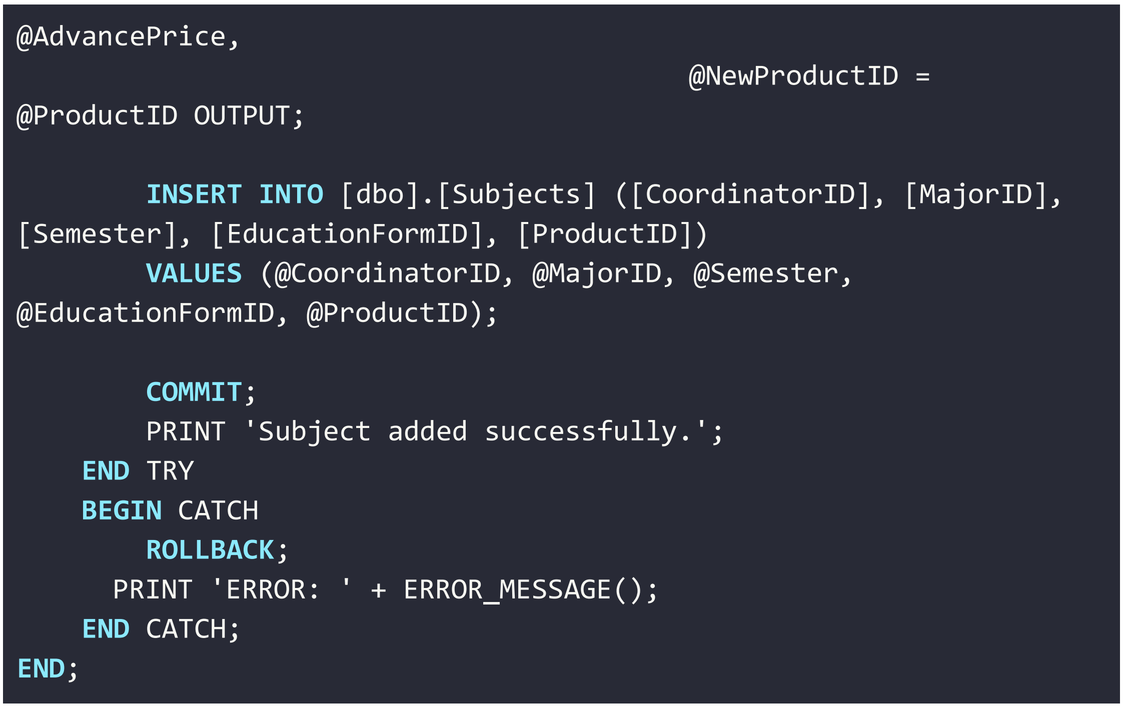
Dodanie nowej oceny

```
CREATE PROCEDURE [dbo].[AddStudentGrade]
    @StudentEnrollID BIGINT,
    @Grade DECIMAL(2, 1)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        INSERT INTO StudentGrades (StudentEnrollID, Grade)
        VALUES (@StudentEnrollID, @Grade);

            COMMIT;
        PRINT 'Grade added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- AddSubject – SM, MB, DM

Dodanie nowego przedmiotu na studiach

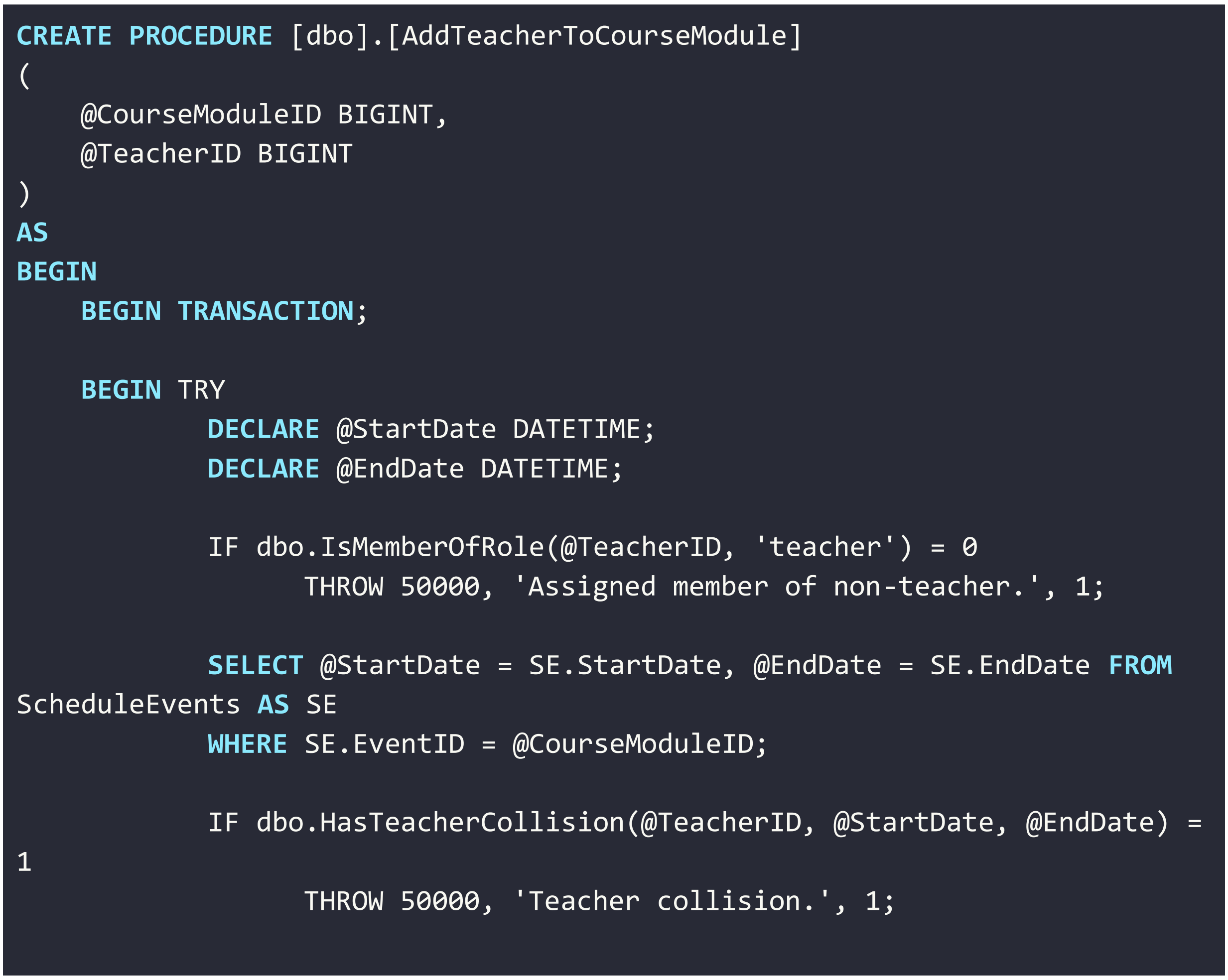
```
CREATE PROCEDURE [dbo].[AddSubject]
      @Name NVARCHAR(200),
    @Description NVARCHAR(MAX),
      @EducationForm CHAR(40),
    @Price MONEY,
    @AdvancePrice MONEY = NULL,
      @MajorID BIGINT,
    @CoordinatorID BIGINT,
      @Semester TINYINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @CategoryID TINYINT;
            DECLARE @EducationFormID TINYINT =
dbo.GetEducationFormID(@EducationForm);
            DECLARE @ProductID BIGINT;

            IF dbo.IsMemberOfRole(@CoordinatorID, 'coordinator') =
0
                  THROW 50000, 'Specified a user without a
coordinator role.', 1

            SELECT @CategoryID = dbo.GetCategoryID('subject');

            EXEC dbo.AddProduct @ProductName = @Name,
                                          @ProductDescription =
@Description,
                                          @CategoryID =
@CategoryID,
                                          @Price = @Price,
                                          @AdvancePrice =
```



```
@AdvancePrice,
                                          @NewProductID =
@ProductID OUTPUT;

        INSERT INTO [dbo].[Subjects] ([CoordinatorID], [MajorID],
[Semester], [EducationFormID], [ProductID])
        VALUES (@CoordinatorID, @MajorID, @Semester,
@EducationFormID, @ProductID);

        COMMIT;
        PRINT 'Subject added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- ## AddTeacherToCourseModule – SM, MB, DM

Dodawanie prowadzącego do danego modułu z kursu



```
CREATE PROCEDURE [dbo].[AddTeacherToCourseModule]
(
    @CourseModuleID BIGINT,
    @TeacherID BIGINT
)
AS
BEGIN
    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @StartDate DATETIME;
            DECLARE @EndDate DATETIME;

            IF dbo.IsMemberOfRole(@TeacherID, 'teacher') = 0
                  THROW 50000, 'Assigned member of non-teacher.', 1;

            SELECT @StartDate = SE.StartDate, @EndDate = SE.EndDate FROM
ScheduleEvents AS SE
            WHERE SE.EventID = @CourseModuleID;

            IF dbo.HasTeacherCollision(@TeacherID, @StartDate, @EndDate) =
1
                  THROW 50000, 'Teacher collision.', 1;
```

```sql
        INSERT INTO dbo.CourseModuleTeachers (CourseModuleID, TeacherID)
        VALUES (@CourseModuleID, @TeacherID);

        COMMIT;
        PRINT 'Teacher assigned to course module successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END;
```

- AddTeacherToSubject – SM, MB, DM

Dodawanie prowadzącego do przedmiotu na studiach

```sql
CREATE PROCEDURE [dbo].[AddTeacherToSubject]
(
    @SubjectID BIGINT,
    @TeacherID BIGINT
)
AS
BEGIN
    BEGIN TRANSACTION;

    BEGIN TRY
            IF dbo.IsMemberOfRole(@TeacherID, 'teacher') = 0
                  THROW 50000, 'Assigned member of non-teacher.', 1;

        INSERT INTO dbo.SubjectTeachers (SubjectID, TeacherID)
        VALUES (@SubjectID, @TeacherID);

        COMMIT;
        PRINT 'Teacher assigned to subject successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END;
```

- AddWebinar – SM, MB, DM

Dodanie nowego webinaru

```
CREATE PROCEDURE [dbo].[AddWebinar]
      @Name NVARCHAR(200),
    @Description NVARCHAR(MAX),
    @Price MONEY,
    @AdvancePrice MONEY = NULL,
      @StartDate DATETIME,
    @EndDate DATETIME,
      @InterpreterID BIGINT = NULL,
      @LeaderID BIGINT,
      @Language NVARCHAR(40)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @CategoryID TINYINT;
            DECLARE @ProductID BIGINT;
            DECLARE @EventID BIGINT;

            IF dbo.IsMemberOfRole(@InterpreterID, 'interpreter') = 0
                  THROW 50000, 'Specified non-interpreter user.', 1;

            IF dbo.IsMemberOfRole(@LeaderID, 'teacher') = 0
                  THROW 50000, 'Specified non-teacher user.', 1;

            IF dbo.HasTeacherCollision(@LeaderID, @StartDate, @EndDate) =
1
                  THROW 50000, 'Teacher collision.', 1;

            SELECT @CategoryID = dbo.GetCategoryID('webinar');

            EXEC dbo.AddProduct @ProductName = @Name,
                                          @ProductDescription =
@Description,
                                          @CategoryID = @CategoryID,
                                          @Price = @Price,
                                          @AdvancePrice = @AdvancePrice,
                                          @NewProductID = @ProductID
OUTPUT;

            EXEC dbo.AddScheduledEvent @StartDate = @StartDate,
                                                  @EndDate = @EndDate,
                                                  @RoomID = NULL,
                                                  @InterpreterID =
@InterpreterID,
                                                  @Language = @Language,
                                                  @NewEventID = @EventID
OUTPUT;

        INSERT INTO [dbo].[Webinars] (WebinarID, ProductID, LeaderID)
        VALUES (@EventID, @ProductID, @LeaderID);

        COMMIT;
        PRINT 'Webinar added successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END;
```

- AssignMemberRole – SM, MB, DM

Przypisanie roli użytkownikowi

```
CREATE PROCEDURE [dbo].[AssignMemberRole]
      @MemberID BIGINT,
      @RoleName VARCHAR(20)
AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @MemberRoleID TINYINT;

            SELECT @MemberRoleID = dbo.GetMemberRoleID(@RoleName);

        INSERT INTO [dbo].[AssignedMemberRoles] ([MemberID],
[MemberRoleID])
        VALUES (@MemberID, @MemberRoleID);

        COMMIT;
        PRINT 'Member role assigned successfully.';
    END TRY
    BEGIN CATCH
```

```
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- CancelEvent – SM, MB, DM

Odwoływanie wydarzenia.

```
CREATE PROCEDURE [dbo].[CancelEvent]
    @EventID BIGINT
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        UPDATE dbo.ScheduleEvents SET IsCanceled = 1
            WHERE EventID = @EventID;

        COMMIT;
        PRINT 'Event canceled successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- CreateBackup – SM, MB, DM

Tworzenie backupu baz danych i zapisywanie go na dysku

```
CREATE PROCEDURE [dbo].[CreateBackup]

AS
BEGIN
      SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @BackupFilename CHAR(240) = CONCAT('backup_',
FORMAT(GETDATE(), 'dd_MM_yyyy_HH_mm_ss'));
            BACKUP DATABASE u_bartnick TO DISK = @BackupFilename;

```

```
        COMMIT;
        PRINT 'Credential added successfully.';
            RETURN;
    END TRY
    BEGIN CATCH
        ROLLBACK;
      PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- EnrollStudent – SM, MB, DM

Przypisywanie studenta do danej grupy zajęciowej

```
CREATE PROCEDURE [dbo].[EnrollStudent]
    @StudentID bigint,
    @GroupID bigint,
    @FinalGrade decimal(2, 1) = NULL,
    @ExamGrade decimal(2, 1) = NULL,
    @LectureGrade decimal(2, 1) = NULL
AS
BEGIN
    SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            IF dbo.IsMemberOfRole(@StudentID, 'student') = 0
                  THROW 50000, 'Specified non-student user.', 1;

        INSERT INTO StudentEnrolls (StudentID, GroupID, FinalGrade,
ExamGrade, LectureGrade)
        VALUES (@StudentID, @GroupID, @FinalGrade, @ExamGrade,
@LectureGrade);

            COMMIT;
        PRINT 'Student enrolled successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- MoveEvent – SM, MB, DM

Przesuwanie wydarzenia na nowy termin

```
CREATE PROCEDURE [dbo].[MoveEvent]
    @EventID BIGINT,
      @StartDate DATETIME
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @PreviousStartDate DATETIME;
            DECLARE @PreviousEndDate DATETIME;

            SELECT @PreviousStartDate = StartDate, @PreviousEndDate =
EndDate FROM dbo.ScheduleEvents
            WHERE EventID = @EventID;

            IF @StartDate <= GETDATE()
                  THROW 50000, 'Cannot move event into the past.', 1;

            DECLARE @EventDuration INT = DATEDIFF(SECOND,
@PreviousStartDate, @PreviousEndDate);
            PRINT @EventDuration

            UPDATE dbo.ScheduleEvents SET StartDate = @StartDate, EndDate
= DATEADD(SECOND, @EventDuration, @StartDate)
            WHERE EventID = @EventID;

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- RemoveAssignedRole – SM, MB, DM

Usuwanie danej roli użytkownikowi

```
CREATE PROCEDURE [dbo].[RemoveAssignedRole]
    @MemberID BIGINT,
    @RoleName VARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @MemberRoleID TINYINT =
dbo.GetMemberRoleID(@RoleName);

        DELETE FROM dbo.AssignedMemberRoles
            WHERE MemberID = @MemberID AND MemberRoleID = @MemberRoleID;

            COMMIT TRANSACTION;
        PRINT 'Assigned role removed successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- SetEventInterpreter – SM, MB, DM

Przypisanie tłumacza do wydarzenia

```
CREATE PROCEDURE [dbo].[SetEventInterpreter]
    @EventID bigint,
    @InterpreterID bigint
AS
BEGIN
    SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
        UPDATE dbo.ScheduleEvents
        SET InterpreterID = @InterpreterID
        WHERE EventID = @EventID;

        COMMIT;
        PRINT 'Interpreter set successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- SetEventLanguage – SM, MB, DM

Ustawianie języka, w którym prowadzone będzie dane wydarzenie

```
CREATE PROCEDURE [dbo].[SetEventLanguage]
    @EventID BIGINT,
    @LanguageName VARCHAR(40)
AS
BEGIN
    SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
            DECLARE @LanguageID SMALLINT =
dbo.GetLanguageID(@LanguageName);

        UPDATE dbo.ScheduleEvents
        SET LanguageID = @LanguageID
        WHERE EventID = @EventID;

        COMMIT;
        PRINT 'Language set successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- SetEventRoom – SM, MB, DM

Przypisanie sali do danych zajęć

```
CREATE PROCEDURE [dbo].[SetEventRoom]
    @EventID bigint,
    @RoomID bigint
AS
BEGIN
    SET NOCOUNT ON;

      BEGIN TRANSACTION;

    BEGIN TRY
        UPDATE dbo.ScheduleEvents
        SET RoomID = @RoomID
        WHERE EventID = @EventID;

        COMMIT;
        PRINT 'Room set successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- SetExamGrade – SM, MB, DM

Wpisanie danemu użytkownikowi oceny za egzamin

```
CREATE PROCEDURE [dbo].[SetExamGrade]
    @StudentEnrollID BIGINT,
    @Grade DECIMAL(2, 1)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        UPDATE dbo.StudentEnrolls
        SET ExamGrade = @Grade
        WHERE StudentEnrollID = @StudentEnrollID;

        COMMIT;
        PRINT 'Exam grade set successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- SetFinalGrade – SM, MB, DM

Wystawienie danemu studentowi oceny końcowej za dany przedmiot

```
CREATE PROCEDURE [dbo].[SetFinalGrade]
    @StudentEnrollID BIGINT,
    @Grade DECIMAL(2, 1)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
        UPDATE dbo.StudentEnrolls
        SET FinalGrade = @Grade
        WHERE StudentEnrollID = @StudentEnrollID;

        COMMIT;
        PRINT 'Final grade set successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- SetLectureGrade – SM, MB, DM

Wpisanie studentowi oceny z wykładu

```
CREATE PROCEDURE [dbo].[SetLectureGrade]
    @StudentEnrollID BIGINT,
    @Grade DECIMAL(2, 1)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;
    BEGIN TRY
        UPDATE dbo.StudentEnrolls
        SET LectureGrade = @Grade
        WHERE StudentEnrollID = @StudentEnrollID;

        COMMIT;
        PRINT 'Lecture grade set successfully.';
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH
END
```

- SetProductAdvancePrice – SM, MB, DM

Ustawianie kwoty zaliczki na podaną kwotę

```
CREATE PROCEDURE [dbo].[SetProductAdvancePrice]
    @ProductID BIGINT,
      @ProductAdvancePrice MONEY
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            UPDATE dbo.Products SET AdvancePrice = @ProductAdvancePrice
            WHERE ProductID = @ProductID;

            PRINT 'Product advance price updated successfully'
        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## SetProductDescription – SM, MB, DM

Ustawianie opisu produktu z oferty

```sql
CREATE PROCEDURE [dbo].[SetProductDescription]
    @ProductID BIGINT,
      @ProductDescription NVARCHAR(MAX)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            UPDATE dbo.Products SET [Description] = @ProductDescription
            WHERE ProductID = @ProductID;

            PRINT 'Product description updated successfully'
        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- ## SetProductName – SM, MB, DM

Ustawianie nazwy produktu z oferty

```sql
CREATE PROCEDURE [dbo].[SetProductName]
    @ProductID BIGINT,
      @ProductName NVARCHAR(200)
AS
BEGIN
    SET NOCOUNT ON;
    BEGIN TRANSACTION;
    BEGIN TRY
            UPDATE dbo.Products SET [Name] = @ProductName
            WHERE ProductID = @ProductID;
            PRINT 'Product name updated successfully'
        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

- SetProductPrice – SM, MB, DM

Ustawianie ceny produktu z oferty

```
CREATE PROCEDURE [dbo].[SetProductPrice]
    @ProductID BIGINT,
      @ProductPrice MONEY
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRANSACTION;

    BEGIN TRY
            UPDATE dbo.Products SET Price = @ProductPrice
            WHERE ProductID = @ProductID;

            PRINT 'Product price updated successfully'
        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
            PRINT 'ERROR: ' + ERROR_MESSAGE();
    END CATCH;
END
```

Triggery

- ## GrantAccessToCourses – SM, MB, DM

Przyznanie użytkownikowi dostępu do wykupionych kursów

```
CREATE TRIGGER [dbo].[GrantAccessToCourses] ON [dbo].[Payments]
      FOR UPDATE
AS
BEGIN
      DECLARE @OrderID BIGINT;
      DECLARE @MemberID BIGINT;
      DECLARE @CourseID BIGINT;

      SELECT @OrderID = I.OrderID FROM Inserted AS I
      INNER JOIN Deleted AS D ON D.PaymentID = I.PaymentID

      IF dbo.IsOrderPaid(@OrderID) = 0
            RETURN;

      SELECT @MemberID = MemberID FROM dbo.Orders
      WHERE OrderID = @OrderID

      DECLARE CourseCursor CURSOR LOCAL FORWARD_ONLY READ_ONLY FOR
            SELECT C.CourseID FROM dbo.Courses AS C
            INNER JOIN dbo.OrderDetails AS OD ON OD.ProductID = C.ProductID
            WHERE OD.OrderID = @OrderID;
      OPEN CourseCursor;

      FETCH NEXT FROM CourseCursor INTO @CourseID;

      WHILE (@@FETCH_STATUS = 0)
      BEGIN
            EXEC dbo.AddAccessToCourse @MemberID = @MemberID,
                                       @CourseID = @CourseID;
            FETCH NEXT FROM CourseCursor INTO @CourseID;
      END

      CLOSE CourseCursor;
    DEALLOCATE CourseCursor;
END
```

- ## GrantAccessToWebinars – SM, MB, DM

Przyznanie użytkownikowi dostępu do wykupionych webinarów.

```
CREATE TRIGGER [dbo].[GrantAccessToWebinars] ON [dbo].[Payments]
      FOR UPDATE
AS
BEGIN
      DECLARE @OrderID BIGINT;
      DECLARE @MemberID BIGINT;
      DECLARE @WebinarID BIGINT;

      SELECT @OrderID = I.OrderID FROM Inserted AS I
      INNER JOIN Deleted AS D ON D.PaymentID = I.PaymentID

      IF dbo.IsOrderPaid(@OrderID) = 0
            RETURN;

      SELECT @MemberID = MemberID FROM dbo.Orders
      WHERE OrderID = @OrderID

      DECLARE WebinarCursor CURSOR LOCAL FORWARD_ONLY READ_ONLY FOR
            SELECT W.WebinarID FROM dbo.Webinars AS W
            INNER JOIN dbo.OrderDetails AS OD ON OD.ProductID = W.ProductID
            WHERE OD.OrderID = @OrderID;
      OPEN WebinarCursor;

      FETCH NEXT FROM WebinarCursor INTO @WebinarID;

      WHILE (@@FETCH_STATUS = 0)
      BEGIN
            EXEC dbo.AddAccessToWebinar @MemberID = @MemberID,
                                        @WebinarID = @WebinarID;
            FETCH NEXT FROM WebinarCursor INTO @WebinarID;
      END

      CLOSE WebinarCursor;
    DEALLOCATE WebinarCursor;
END
```

- ## GrantAccessToMeetings

Przyznanie dostępu do zjazdu po opłaceniu zjazdu.

```
CREATE TRIGGER [dbo].[GrantAccessToMeetings] ON [dbo].[Payments]
   FOR UPDATE
AS
BEGIN
      SET NOCOUNT ON;

    DECLARE @OrderID BIGINT;
      DECLARE @MemberID BIGINT;
      DECLARE @MeetingID BIGINT;

      SELECT @OrderID = I.OrderID FROM Inserted AS I
      INNER JOIN Deleted AS D ON D.PaymentID = I.PaymentID

      IF dbo.IsOrderPaid(@OrderID) = 0
            RETURN;

      SELECT @MemberID = MemberID FROM dbo.Orders
      WHERE OrderID = @OrderID

      DECLARE MeetingCursor CURSOR LOCAL FORWARD_ONLY READ_ONLY FOR
            SELECT M.MeetingID FROM dbo.Meetings AS M
            INNER JOIN dbo.OrderDetails AS OD ON OD.ProductID =
M.ProductID
            WHERE OD.OrderID = @OrderID;
      OPEN MeetingCursor;

      FETCH NEXT FROM MeetingCursor INTO @MeetingID;

      WHILE (@@FETCH_STATUS = 0)
      BEGIN
            EXEC dbo.AddAccessToMeeting @MemberID = @MemberID,
                                        @MeetingID = @MeetingID;
            FETCH NEXT FROM MeetingCursor INTO @MeetingID;
      END

      CLOSE MeetingCursor;
    DEALLOCATE MeetingCursor;

END
```

- GrantStudentRoles – SM, MB, DM

Przyznanie użytkownikowi roli studenta po dokonaniu zakupów produktów o odpowiednich kategoriach.

```
CREATE TRIGGER [dbo].[GrantStudentRoles] ON [dbo].[Payments]
      FOR UPDATE
AS
BEGIN
      DECLARE @OrderID BIGINT;
      DECLARE @MemberID BIGINT;

      SELECT @OrderID = I.OrderID FROM Inserted AS I
      INNER JOIN Deleted AS D ON D.PaymentID = I.PaymentID

      IF dbo.IsOrderPaid(@OrderID) = 0
            RETURN;

      SELECT @MemberID = MemberID FROM dbo.Orders
      WHERE OrderID = @OrderID

      IF  @MemberID IS NULL
            RETURN;

      DECLARE @BoughtSubjectMajor BIT = 0;

      SELECT @BoughtSubjectMajor = 1 FROM dbo.OrderDetails AS OD
      LEFT JOIN dbo.Majors AS M ON M.ProductID = OD.ProductID
      LEFT JOIN dbo.Subjects AS S ON S.ProductID = OD.ProductID
      WHERE OD.OrderID = @OrderID
      HAVING COUNT(OD.ProductID) > 0;

    IF @BoughtSubjectMajor = 0
            RETURN;

      EXEC AssignMemberRole @MemberID = @MemberID,
                                      @RoleName = 'student';
END
```

- GrantAcessToSubjects

Przyznanie dostępu do przedmiotu.

```
CREATE TRIGGER [dbo].[GrantAccessToSubjects] ON [dbo].[Payments]
      FOR UPDATE
AS
BEGIN
      DECLARE @OrderID BIGINT;
      DECLARE @MemberID BIGINT;
      DECLARE @SubjectID BIGINT;

      SELECT @OrderID = I.OrderID FROM Inserted AS I
      INNER JOIN Deleted AS D ON D.PaymentID = I.PaymentID

      IF dbo.IsOrderPaid(@OrderID) = 0
            RETURN;

      SELECT @MemberID = MemberID FROM dbo.Orders
      WHERE OrderID = @OrderID

      DECLARE SubjectCursor CURSOR LOCAL FORWARD_ONLY READ_ONLY FOR
            SELECT S.SubjectID FROM dbo.Subjects AS S
            INNER JOIN dbo.OrderDetails AS OD ON OD.ProductID =
S.ProductID
            WHERE OD.OrderID = @OrderID;
      OPEN SubjectCursor;

      FETCH NEXT FROM SubjectCursor INTO @SubjectID;

      WHILE (@@FETCH_STATUS = 0)
      BEGIN
            EXEC dbo.AddAccessToSubject @MemberID = @MemberID,
                                        @SubjectID = @SubjectID;
            FETCH NEXT FROM SubjectCursor INTO @SubjectID;
      END

      CLOSE SubjectCursor;
    DEALLOCATE SubjectCursor;
END
```

Indeksy

- Indeks na *CategoryName, CategoryID* w tabeli Categories

```
CREATE NONCLUSTERED INDEX [IX_Categories] ON [dbo].[Categories]
(
      [CategoryID] ASC,
      [CategoryName] ASC
)
```

- Indeks na *Name, Price, CategoryID, ProductID* w tabeli Products

```
CREATE NONCLUSTERED INDEX [IX_Products] ON [dbo].[Products]
(
      [ProductID] ASC,
      [Name] ASC,
      [CategoryID] ASC,
      [Price] ASC
)
```

- Indeks na *ProductID, OrderID* w tabeli OrderDetails

```
CREATE NONCLUSTERED INDEX [IX_Orders] ON [dbo].[Orders]
(
      [OrderID] ASC,
      [MemberID] ASC
)
```

- Indeks na *ProductID, Discount, MemberRoleID* w tabeli Discounts

```sql
CREATE NONCLUSTERED INDEX [IX_Discounts] ON [dbo].[Discounts]
(
      [ProductID] ASC,
      [Discount] ASC,
      [MemberRoleID] ASC
)
```

- Indeks na *PaymentStatusID, StatusName* w tabeli PaymentStatus

```sql
CREATE NONCLUSTERED INDEX [IX_PaymentStatus] ON [dbo].[PaymentStatus]
(
      [PaymentStatusID] ASC,
      [StatusName] ASC
)
```

- Indeks na *MemberID, OrderID* w tabeli Orders

```sql
CREATE NONCLUSTERED INDEX [IX_Orders] ON [dbo].[Orders]
(
      [OrderID] ASC,
      [MemberID] ASC
)
```

- Indeks na *RoleName* w MemberRoles

```sql
CREATE UNIQUE NONCLUSTERED INDEX [IX_RoleName] ON [dbo].[MemberRoles]
(
      [RoleName] ASC
)
```

- Indeks na *MemberID, FirstName, LastName, Login* w tabeli Members

```sql
CREATE NONCLUSTERED INDEX [IX_Members] ON [dbo].[Members]
(
      [MemberID] ASC,
      [FirstName] ASC,
      [LastName] ASC,
      [Login] ASC
)
```

- Indeks na *RoomID, Seats* w tabeli Rooms

```
CREATE NONCLUSTERED INDEX [IX_Rooms] ON [dbo].[Rooms]
(
      [RoomID] ASC,
      [Seats] ASC
)
```

- Indeks na *MajorID, StudentID, StartDate, EndDate* w tabeli Internships

```
CREATE NONCLUSTERED INDEX [IX_Internships] ON [dbo].[Interships]
(
      [MajorID] ASC,
      [StudentID] ASC,
      [StartDate] ASC,
      [EndDate] ASC
)
```

- Indeks na *RoleName* w tabeli MemberRoles

```
CREATE UNIQUE NONCLUSTERED INDEX [IX_RoleName] ON [dbo].[MemberRoles]
(
      [RoleName] ASC
)
```

- Indeks na *MajorID, ProductID, MeetingID* w tabeli Meetings

```
CREATE NONCLUSTERED INDEX [IX_Meetings] ON [dbo].[Meetings]
(
      [MeetingID] ASC,
      [MajorID] ASC,
      [ProductID] ASC
)
```

- Indeks na *CourseModuleID, CourseID, Name* w tabeli CourseModules

```
CREATE NONCLUSTERED INDEX [IX_CourseModules] ON [dbo].[CourseModules]
(
      [CourseID] ASC,
      [CourseModuleID] ASC,
      [Name] ASC
)
```

- Indeks na *AttendanceID* w tabeli MeetingAttendance

```
CREATE NONCLUSTERED INDEX [IX_MeetingAttendance] ON
[dbo].[MeetingAttendance]
(
      [AttendanceID] ASC
)
```

- Indeks na *CourseID, ProductID* w tabeli Courses

```
CREATE NONCLUSTERED INDEX [IX_Courses] ON [dbo].[Courses]
(
      [CourseID] ASC,
      [ProductID] ASC
)
```

- Indeks na *DiplomaID, MemberID* w tabeli Diplomas

```
CREATE NONCLUSTERED INDEX [IX_Diplomas] ON [dbo].[Diplomas]
(
      [DiplomaID] ASC,
      [MemberID] ASC
)
```

- Indeks na *EventID* w tabeli Recordings

```
CREATE NONCLUSTERED INDEX [IX_Recordings] ON [dbo].[Recordings]
(
      [EventID] ASC
)
```

- Indeks na *MajorID, ProductID* w tabeli Majors

```
CREATE NONCLUSTERED INDEX [IX_Majors] ON [dbo].[Majors]
(
      [MajorID] ASC,
      [ProductID] ASC
)
```

- Indeks na *EventID, StartDate, EndDate, InterpreterID, LanguageID* w tabeli ScheduleEvents

```
CREATE NONCLUSTERED INDEX [IX_ScheduleEvents] ON [dbo].[ScheduleEvents]
(
      [EventID] ASC,
      [InterpreterID] ASC,
      [StartDate] ASC,
      [EndDate] ASC,
      [LanguageID] ASC
)
```

- Indeks na *ClassID, SubjectID, GroupID* w tabeli Classes

```
CREATE NONCLUSTERED INDEX [IX_Classes] ON [dbo].[Classes]
(
      [ClassID] ASC,
      [SubjectID] ASC,
      [GroupID] ASC
)
```

- Indeks na *SubjectID, MajorID, ProductID* w tabeli Subjects

```
CREATE NONCLUSTERED INDEX [IX_Subjects] ON [dbo].[Subjects]
(
      [ProductID] ASC,
      [SubjectID] ASC,
      [MajorID] ASC
)
```

- Indeks na *GroupID, SubjectID* w tabeli Groups

```
CREATE NONCLUSTERED INDEX [IX_Groups] ON [dbo].[Groups]
(
      [GroupID] ASC,
      [SubjectID] ASC
)
```

- Indeks na *WebinarID* w tabeli Webinars

```
CREATE NONCLUSTERED INDEX [IX_Webinars] ON [dbo].[Webinars]
(
      [WebinarID] ASC
)
```

- Indeks na *LanguageID, Name* w tabeli Languages

```
CREATE NONCLUSTERED INDEX [IX_Languages] ON [dbo].[Languages]
(
      [LanguageID] ASC,
      [Name] ASC
)
```

- Indeks na *MajorID, MemberID* w tabeli MajorAccess

```
CREATE NONCLUSTERED INDEX [IX_MajorAccess] ON [dbo].[MajorAccess]
(
      [MajorID] ASC,
      [MemberID] ASC
)
```

- Indeks na *SubjectID, MemberID* w tabeli SubjectAccess

```
CREATE NONCLUSTERED INDEX [IX_SubjectAccess] ON [dbo].[SubjectAccess]
(
      [MemberID] ASC,
      [SubjectID] ASC
)
```

- Indeks na *AttendanceID, CourseModuleID* w tabeli CourseAttendance

```
CREATE NONCLUSTERED INDEX [IX_CourseAttendance] ON
[dbo].[CourseAttendance]
(
      [AttendanceID] ASC,
      [CourseModuleID] ASC
)
```

- Indeks na *AttendanceID, StudentEnrollID* w tabeli StudentAttendance

```
CREATE NONCLUSTERED INDEX [IX_StudentAttendance] ON
[dbo].[StudentAttendance]
(
      [AttendanceID] ASC,
      [StudentEnrollID] ASC
)
```

- Indeks na *CourseModuleID, TeacherID* w tabeli CourseModuleTeachers

```
CREATE NONCLUSTERED INDEX [IX_CourseModuleTeachers] ON
[dbo].[CourseModuleTeachers]
(
      [CourseModuleID] ASC,
      [TeacherID] ASC
)
```

- Indeks na *OrderID, PostponeStartDate, PostponeEndDate* w tabeli Postponements

```
CREATE NONCLUSTERED INDEX [IX_Postponements] ON [dbo].[Postponements]
(
      [OrderID] ASC,
      [PostponeStartDate] ASC,
      [PostponeEndDate] ASC
)
```

- Indeks na *OrderID, PaymentStatusID* w tabeli Payments

```
CREATE NONCLUSTERED INDEX [IX_Payments] ON [dbo].[Payments]
(
      [OrderID] ASC,
      [PaymentStatusID] ASC
)
```

Uprawnienia

W bazie utworzone są role: CoordinatorUser, NonRegisteredUser, RegisteredUser, SecretariatEmployeeUser, SystemUser, TeacherUser. Zostały im nadane uprawnienia opisane w funkcjonalności bazy danych.

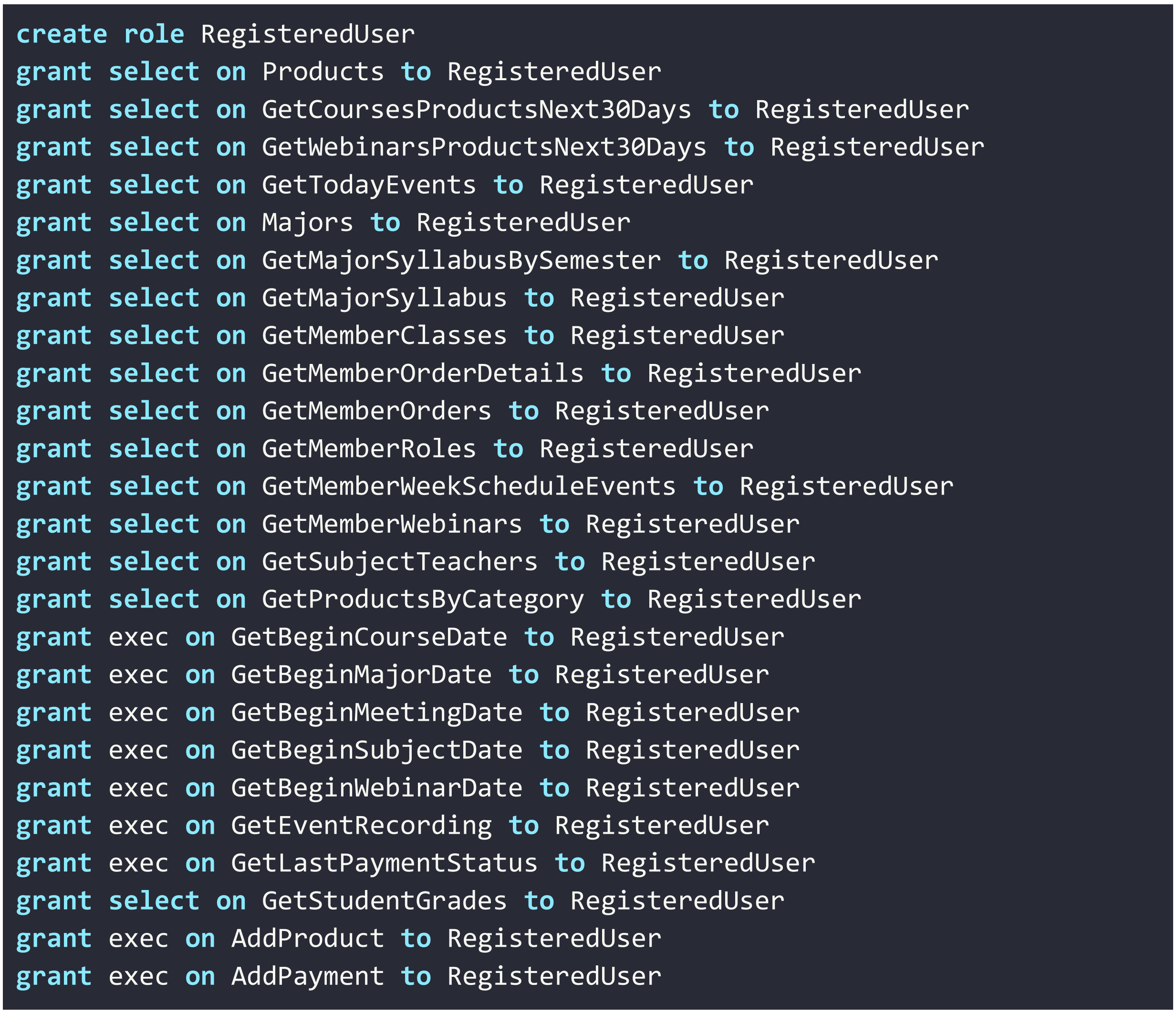
- AdministratorUser

```
create role administrator
 grant all privileges
```

- NonRegisteredUser

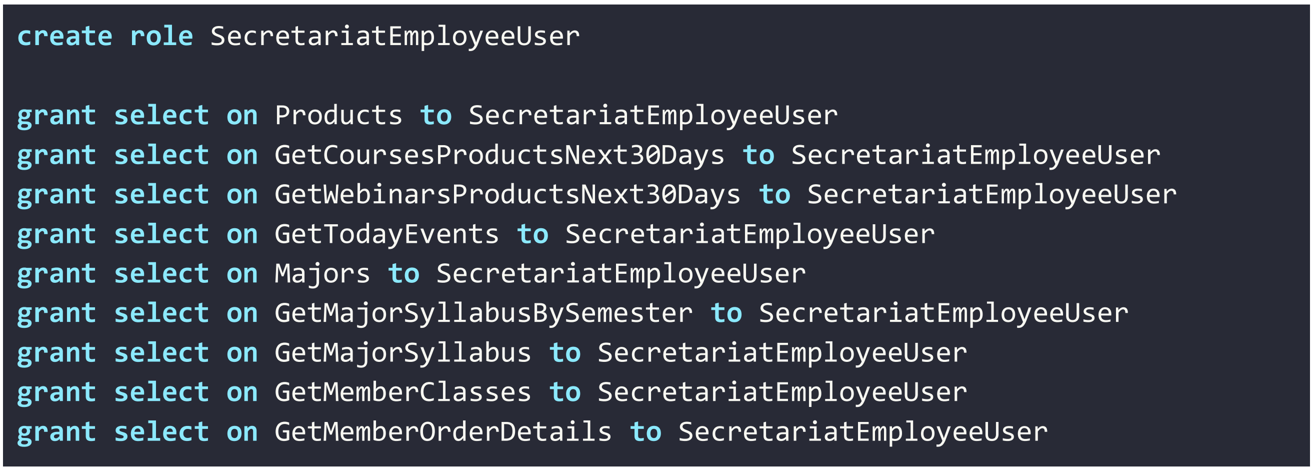
```
create role NonRegisteredUser
grant exec on AddMember to NonRegisteredUser
grant exec on AddCredential to NonRegisteredUser
grant select on Products to NonRegisteredUser
grant select on Majors to  NonRegisteredUser
grant select on GetMajorSyllabusBySemester to NonRegisteredUser
grant select on GetMajorSyllabus to NonRegisteredUser
grant select on GetCoursesProductsNext30Days to NonRegisteredUser
grant select on GetProductsByCategory to NonRegisteredUser
grant select on GetWebinarsProductsNext30Days to NonRegisteredUser
```

- RegisteredUser

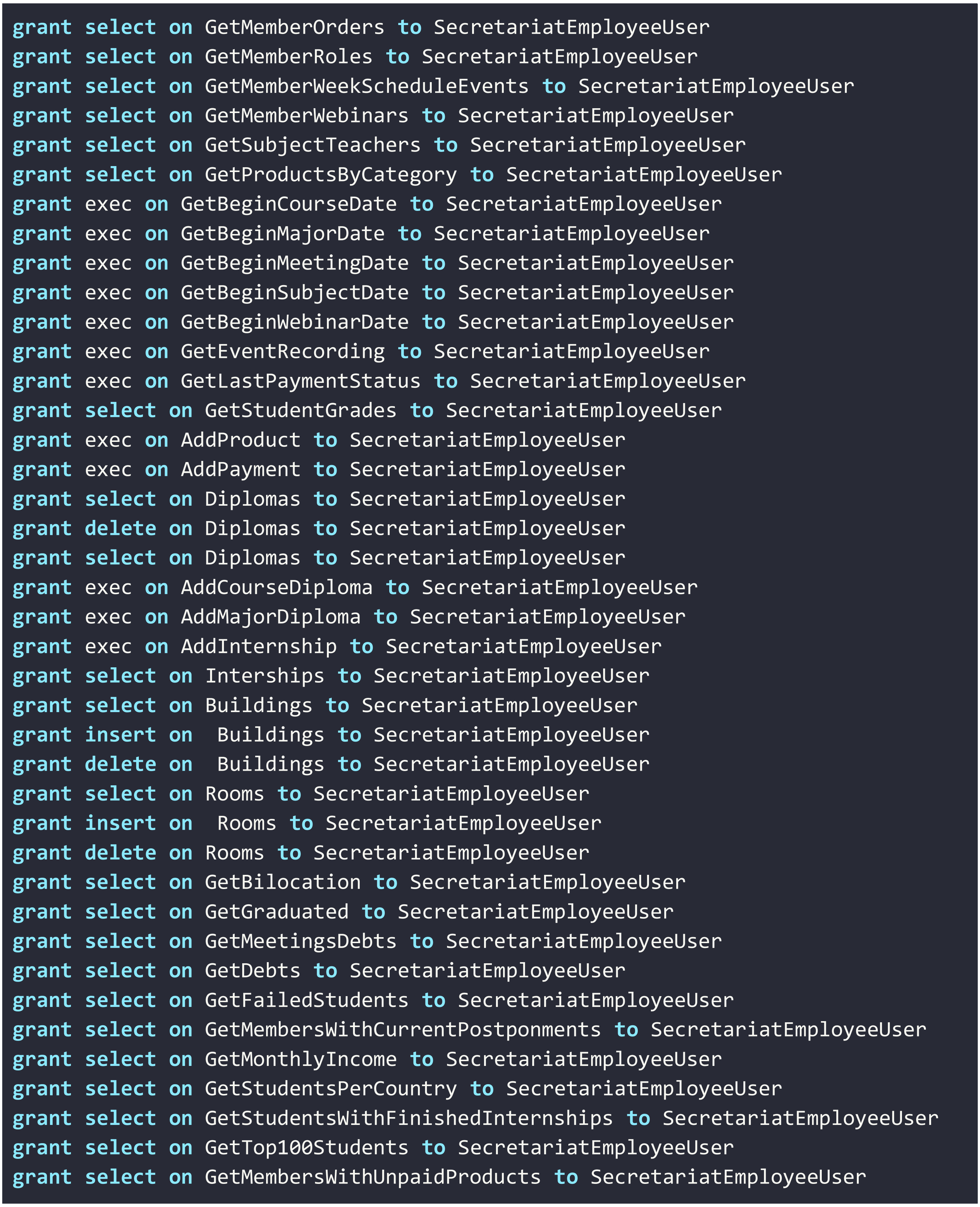


```sql
create role RegisteredUser
grant select on Products to RegisteredUser
grant select on GetCoursesProductsNext30Days to RegisteredUser
grant select on GetWebinarsProductsNext30Days to RegisteredUser
grant select on GetTodayEvents to RegisteredUser
grant select on Majors to RegisteredUser
grant select on GetMajorSyllabusBySemester to RegisteredUser
grant select on GetMajorSyllabus to RegisteredUser
grant select on GetMemberClasses to RegisteredUser
grant select on GetMemberOrderDetails to RegisteredUser
grant select on GetMemberOrders to RegisteredUser
grant select on GetMemberRoles to RegisteredUser
grant select on GetMemberWeekScheduleEvents to RegisteredUser
grant select on GetMemberWebinars to RegisteredUser
grant select on GetSubjectTeachers to RegisteredUser
grant select on GetProductsByCategory to RegisteredUser
grant exec on GetBeginCourseDate to RegisteredUser
grant exec on GetBeginMajorDate to RegisteredUser
grant exec on GetBeginMeetingDate to RegisteredUser
grant exec on GetBeginSubjectDate to RegisteredUser
grant exec on GetBeginWebinarDate to RegisteredUser
grant exec on GetEventRecording to RegisteredUser
grant exec on GetLastPaymentStatus to RegisteredUser
grant select on GetStudentGrades to RegisteredUser
grant exec on AddProduct to RegisteredUser
grant exec on AddPayment to RegisteredUser
```

- SecretariatEmployeeUser

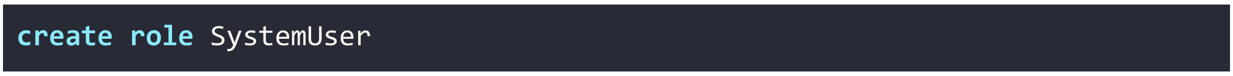


```sql
create role SecretariatEmployeeUser

grant select on Products to SecretariatEmployeeUser
grant select on GetCoursesProductsNext30Days to SecretariatEmployeeUser
grant select on GetWebinarsProductsNext30Days to SecretariatEmployeeUser
grant select on GetTodayEvents to SecretariatEmployeeUser
grant select on Majors to SecretariatEmployeeUser
grant select on GetMajorSyllabusBySemester to SecretariatEmployeeUser
grant select on GetMajorSyllabus to SecretariatEmployeeUser
grant select on GetMemberClasses to SecretariatEmployeeUser
grant select on GetMemberOrderDetails to SecretariatEmployeeUser
```



```
grant select on GetMemberOrders to SecretariatEmployeeUser
grant select on GetMemberRoles to SecretariatEmployeeUser
grant select on GetMemberWeekScheduleEvents to SecretariatEmployeeUser
grant select on GetMemberWebinars to SecretariatEmployeeUser
grant select on GetSubjectTeachers to SecretariatEmployeeUser
grant select on GetProductsByCategory to SecretariatEmployeeUser
grant exec on GetBeginCourseDate to SecretariatEmployeeUser
grant exec on GetBeginMajorDate to SecretariatEmployeeUser
grant exec on GetBeginMeetingDate to SecretariatEmployeeUser
grant exec on GetBeginSubjectDate to SecretariatEmployeeUser
grant exec on GetBeginWebinarDate to SecretariatEmployeeUser
grant exec on GetEventRecording to SecretariatEmployeeUser
grant exec on GetLastPaymentStatus to SecretariatEmployeeUser
grant select on GetStudentGrades to SecretariatEmployeeUser
grant exec on AddProduct to SecretariatEmployeeUser
grant exec on AddPayment to SecretariatEmployeeUser
grant select on Diplomas to SecretariatEmployeeUser
grant delete on Diplomas to SecretariatEmployeeUser
grant select on Diplomas to SecretariatEmployeeUser
grant exec on AddCourseDiploma to SecretariatEmployeeUser
grant exec on AddMajorDiploma to SecretariatEmployeeUser
grant exec on AddInternship to SecretariatEmployeeUser
grant select on Interships to SecretariatEmployeeUser
grant select on Buildings to SecretariatEmployeeUser
grant insert on  Buildings to SecretariatEmployeeUser
grant delete on  Buildings to SecretariatEmployeeUser
grant select on Rooms to SecretariatEmployeeUser
grant insert on  Rooms to SecretariatEmployeeUser
grant delete on Rooms to SecretariatEmployeeUser
grant select on GetBilocation to SecretariatEmployeeUser
grant select on GetGraduated to SecretariatEmployeeUser
grant select on GetMeetingsDebts to SecretariatEmployeeUser
grant select on GetDebts to SecretariatEmployeeUser
grant select on GetFailedStudents to SecretariatEmployeeUser
grant select on GetMembersWithCurrentPostponments to SecretariatEmployeeUser
grant select on GetMonthlyIncome to SecretariatEmployeeUser
grant select on GetStudentsPerCountry to SecretariatEmployeeUser
grant select on GetStudentsWithFinishedInternships to SecretariatEmployeeUser
grant select on GetTop100Students to SecretariatEmployeeUser
grant select on GetMembersWithUnpaidProducts to SecretariatEmployeeUser
```

- SystemUser



```
create role SystemUser
```

```
grant select to SystemUser
grant exec on AddAccessToCourse to SystemUser
grant exec on AddAccessToWebinar to SystemUser
grant exec on AddAddress to SystemUser
grant exec on AddCity to SystemUser
grant exec on AddCategory to SystemUser
grant exec on AddDiscount to SystemUser
grant exec on AddPaymentStatus to SystemUser
grant exec on AddScheduledEvent to SystemUser
grant exec on CreateBackup to SystemUser
grant exec on EnrollStudent to SystemUser
grant exec on SetProductAdvancePrice to SystemUser
```

- TeacherUser

```
create role TeacherUser
grant select on Products to TeacherUser
grant select on GetCoursesProductsNext30Days to TeacherUser
grant select on GetWebinarsProductsNext30Days to TeacherUser
grant select on GetTodayEvents to TeacherUser
grant select on Majors to TeacherUser
grant select on GetMajorSyllabusBySemester to TeacherUser
grant select on GetMajorSyllabus to TeacherUser
grant select on GetMemberClasses to TeacherUser
grant select on GetMemberOrderDetails to TeacherUser
grant select on GetMemberOrders to TeacherUser
grant select on GetMemberRoles to TeacherUser
grant select on GetMemberWeekScheduleEvents to TeacherUser
grant select on GetMemberWebinars to TeacherUser
grant select on GetSubjectTeachers to TeacherUser
grant select on GetProductsByCategory to TeacherUser
grant exec on GetBeginCourseDate to TeacherUser
grant exec on GetBeginMajorDate to TeacherUser
grant exec on GetBeginMeetingDate to TeacherUser
grant exec on GetBeginSubjectDate to TeacherUser
grant exec on GetBeginWebinarDate to TeacherUser
grant exec on GetEventRecording to TeacherUser
grant exec on GetLastPaymentStatus to TeacherUser
grant select on GetStudentGrades to TeacherUser
grant exec on AddProduct to TeacherUser
grant exec on AddPayment to TeacherUser

grant exec on AddWebinar to TeacherUser
grant insert on Webinars to TeacherUser
grant delete on Webinars to TeacherUser
grant exec on AddCourse to TeacherUser
grant insert on Courses to TeacherUser
grant delete on Courses to TeacherUser
grant exec on AddCourseModule to TeacherUser
grant insert on CourseModules to TeacherUser
grant delete on CourseModules to TeacherUser
grant exec on AddCourseModuleAttendance to TeacherUser
grant exec on AddStudentAttendance to TeacherUser
grant exec on AddStudentGrade to TeacherUser
grant exec on CancelEvent to TeacherUser
grant exec on SetLectureGrade to TeacherUser
grant exec on SetFinalGrade to TeacherUser
grant exec on MoveEvent to TeacherUser
grant exec on SetEventInterpreter to TeacherUser
grant exec on SetEventLanguage to TeacherUser
grant exec on AddRecording to TeacherUser
grant select on ListClassAttendance to TeacherUser
grant select on ListCourseModuleAttendance to TeacherUser
grant select on ListClassesAttendance to TeacherUser
grant select on ListCourseMembers to TeacherUser
grant select on ListWebinarMembers to TeacherUser
```

- CoordinatorUser

```
create role CoordinatorUser
grant select on Products to CoordinatorUser
grant select on GetCoursesProductsNext30Days to CoordinatorUser
grant select on GetWebinarsProductsNext30Days to CoordinatorUser
grant select on GetTodayEvents to CoordinatorUser
grant select on Majors to CoordinatorUser
grant select on GetMajorSyllabusBySemester to CoordinatorUser
grant select on GetMajorSyllabus to CoordinatorUser
grant select on GetMemberClasses to CoordinatorUser
grant select on GetMemberOrderDetails to CoordinatorUser
grant select on GetMemberOrders to CoordinatorUser
grant select on GetMemberRoles to CoordinatorUser
grant select on GetMemberWeekScheduleEvents to CoordinatorUser
grant select on GetMemberWebinars to CoordinatorUser
grant select on GetSubjectTeachers to CoordinatorUser
grant select on GetProductsByCategory to CoordinatorUser
grant exec on GetBeginCourseDate to CoordinatorUser
```

```sql
grant exec on GetBeginMajorDate to CoordinatorUser
grant exec on GetBeginMeetingDate to CoordinatorUser
grant exec on GetBeginSubjectDate to CoordinatorUser
grant exec on GetBeginWebinarDate to CoordinatorUser
grant exec on GetEventRecording to CoordinatorUser
grant exec on GetLastPaymentStatus to CoordinatorUser
grant select on GetStudentGrades to CoordinatorUser
grant exec on AddProduct to CoordinatorUser
grant exec on AddPayment to CoordinatorUser
grant exec on AddClass to CoordinatorUser
grant exec on AddCourse to CoordinatorUser
grant exec on AddMajor to CoordinatorUser
grant exec on AddSubject to CoordinatorUser
grant exec on AddTeacherToCourseModule to CoordinatorUser
grant exec on AddTeacherToSubject to CoordinatorUser
```