Dokumentacja

Piotr Karamon, Kyrylo Iakymenko, Joanna Konieczny

Spis treści

1 Wkład poszczególnych osób

1. Schemat, funkcje użytkowników i systemu (50%)

   Kilka wspólnych spotkań online na których powstał dokument funkcji użytkowników i systemu oraz sam schemat bazy.

   - **Piotr Karamon** - (33.3%)
   - **Joanna Konieczny** - (33.3%)
   - **Kyrylo Iakymenko** - (33.3%)

2. Warunki integralnościowe (5%)

   - **Piotr Karamon** - pracownicy, płatności, użytkownicy plus dodanie opisów do wszystkich pozostałych (33.3%)
   - **Joanna Konieczny** - dla studiów (33.3%)
   - **Kyrylo Iakymenko** - dla kursów i webinarów (33.3%)

3. Generowanie danych (13%)

   - **Piotr Karamon** - generowanie danych dla webinarów (15%)
   - **Joanna Konieczny** - generowanie danych dla studiów (50%)
   - **Kyrylo Iakymenko** - generowanie danych dla kursów/użytkowników/pracowników. (35%)

4. Procedury, funkcje, triggery widoki (20%)

   - **Piotr Karamon** - Raporty finansowe, raport bilokacji, harmonogramy, wszystkie triggery, procedury/ funkcje związane z płatnościami, zapisami, webinarami (60%)
   - **Joanna Konieczny** - Część funkcji i procedur związanych ze studiami (10%)
   - **Kyrylo Iakymenko** - 3 widoki, 26 funkcji i procedur (30%)

5. Indeksy & uprawnienia (2%)

   - **Piotr Karamon** - Wszystko (100%)
   - **Joanna Konieczny** - (0%)
   - **Kyrylo Iakymenko** - (0%)

6. Przygotowanie dokumentacji (10%)

   - **Piotr Karamon** - Dodanie krótkich opisów do funkcji/procedur/indeksów/widoków. Stworzenie skryptu który z pliku `.sql` schematu bazy danych, pliku `.xml` z vertabelo i innych mniejszych plików zawierających procedury/funkcjie etc. generuje dokumentację do pliku `.org` który poźniej kompilowany jest do pliku `pdf`. (60%)
   - **Joanna Konieczny** - Opisanie tabel oraz ich wszystkich kolumn (w sekcji studiów) (20%)
   - **Kyrylo Iakymenko** - Opisanie tabel oraz ich wszystkich kolumn (wszystko poza studiami) (20%)

7. Sumaryczny wkład

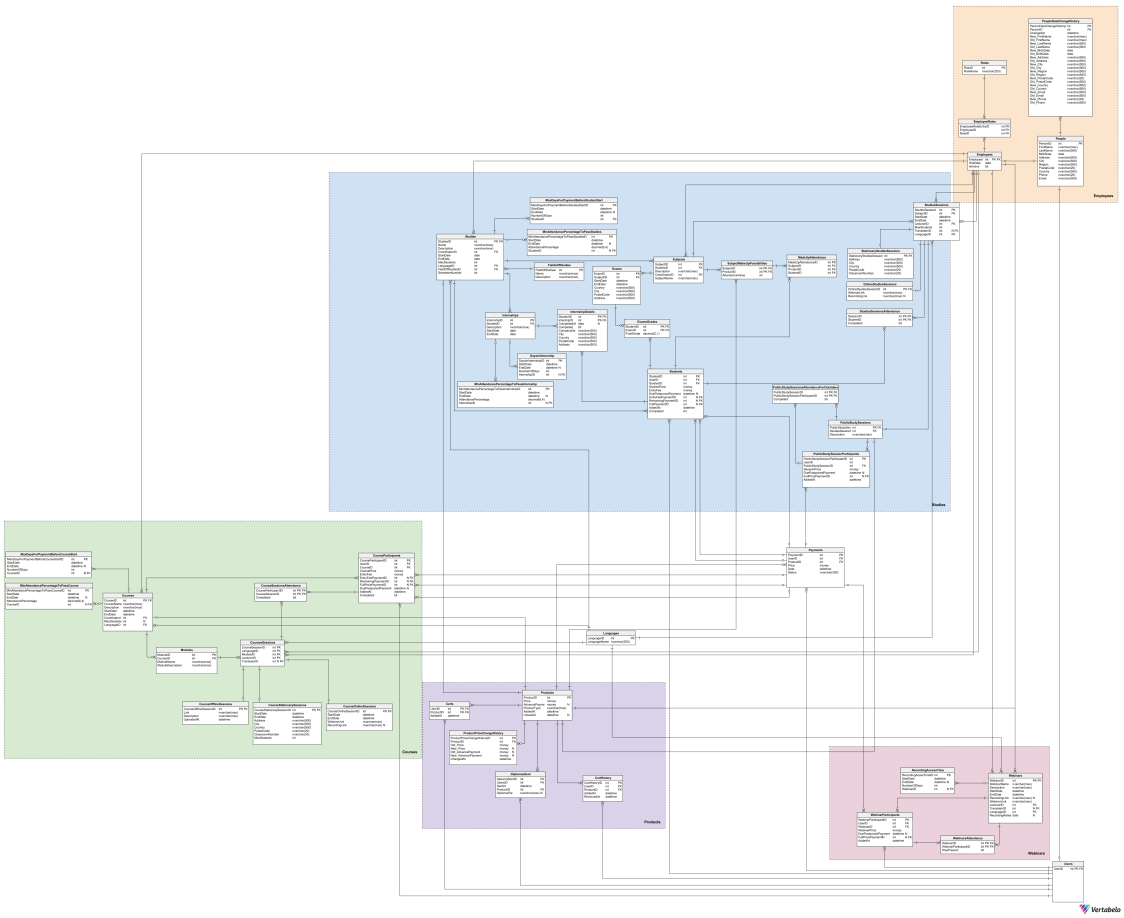
   - **Piotr Karamon** - (40.4%)
   - **Joanna Konieczny** - (28.8%)
   - **Kyrylo Iakymenko** - (30.8%)

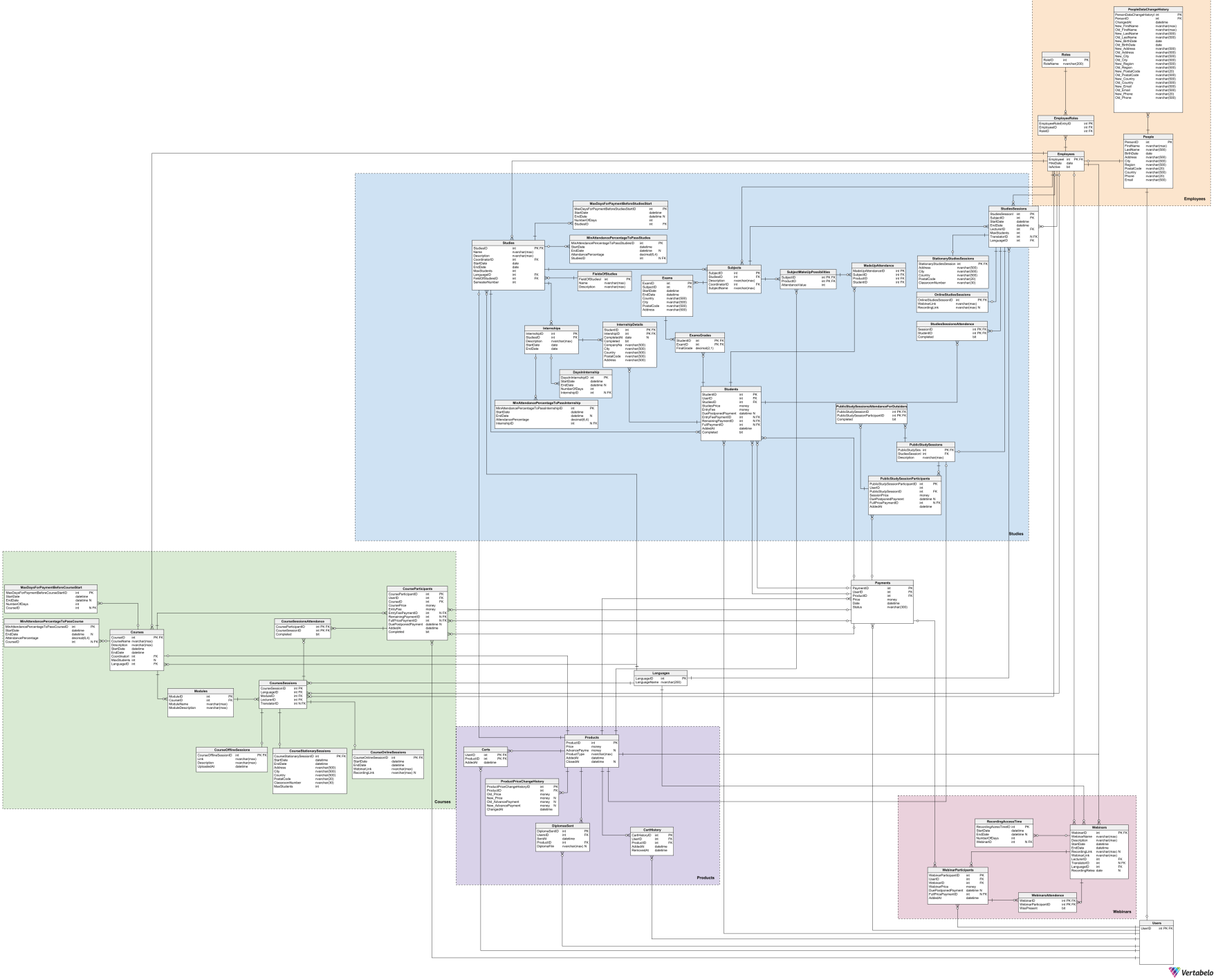
2 Funkcje użytkowników i systemu

- **Użytkownik niezarejestrowany**
  1. Przeglądanie oferty: Dostep do ogólnej oferty edukacyjnej bez możliwości rejestracji.
  2. Utworzenie konta.
- **Użytkownik zarejestrowany**
  1. Przeglądanie oferty edukacyjnej: Możliwość przeglądania dostepnych kursów, webinarów i studiów oraz pojedynczych spotkań studyjnych.
  2. Przeglądanie historii platnosci (nazwa zakupionego przedmiotu, kwota, data, informacja o tym, czy płatność się powiodła).
  3. Zarządanie kontem: Edycja danych osobowych.
  4. Zakup dostępu do płatnego webinaru.
  5. Zakup dostępu do nagrania z płatnego webinaru.
  6. Zapisanie się na kurs.
  7. Zapisanie się na studia.
  8. Zapisanie się na płatne pojedyncze spotkanie studyjne.
  9. Uczestnictwo w webinarze.
  10. Oglądanie udostępnionych nagrań z webinarów.
  11. Dodanie/usunięcie produktu z/do koszyka.
  12. Dokonanie płatności za produkt.
- **Uczestnik kursu**
  1. Przeglądanie modułów które składają sie na kurs.
  2. Dostep do harmonogramu: moduły stacjonarne - data, miejsce, prowadzący; moduly on-line synchroniczne - data, prowadzący, link do spotkania; moduły on-line asynchroniczne - link do nagrania; moduły hybrydowe - połączenie harmonogramów dla modułów stacjonarnych i online.
  3. Dostęp do informacji o postepie w kursie/zaliczeniu.
  4. Uczestnictwo w modułach on-line synchronicznych.
  5. Oglądanie nagrań (moduły on-line asynchroniczne).
  6. Oglądanie nagran z modułów on-line synchronicznych.
- **Student**
  1. Przeglądanie harmonogramu zajęć (kiedy, gdzie i z jakim prowadzącym).
  2. Przeglądanie programu studiów.
  3. Przeglądanie ocen.
  4. Dostęp do danych o frekwencji na zajeciach.
- **Księgowa**
  1. Zarządzanie płatnościami: Przeglądanie i monitorowanie płatności użytkowników.
  2. Generowanie raportów finansowych: Tworzenie zestawień przychodów z poszczególnych wydarzeń.
  3. Weryfikacja płatności odroczonych: Monitorowanie płatności odroczonych dla stałych klientów.
- **Wykładowca webinarów**
  1. Stworzenie webinaru.
  2. Modyfikacja danych webinaru.

- **Nauczyciel zwiazany z kursami**
    1. Tworzenie/modyfikacja kursów.
    2. Tworzenie/modyfikacja modułów.
    3. Tworzenie/modyfikacja zajęć.
    4. Wpisywanie obecności.
- **Nauczyciel akademicki**
    1. Tworzenie/modyfikacja przedmiotów.
    2. Tworzenie/modyfikacja egzaminów.
    3. Dodawanie/modyfikacja ocen z egzaminów.
    4. Tworzenie staży.
    5. Wpisywanie zaliczenia ze stażu.
    6. Aktualizacja danych dotyczących stażu danego studenta.
- **Sekretariat**
    1. Wysyłanie dyplomów.
    2. Raporty bilokacji.
    3. Informacje o harmonogramach pracowników i uczniów.
- **Tłumacz**
    1. Posiada dostep do wszystkich webinarów oraz kursów i studiów online.
    2. Dodawanie materiałów: może dodawać do bazy przetłumaczone materiały.
- **Administrator**
    1. Zarząda bazą danych.
    2. Ma nieograniczone uprawnienia.
- **Dyrektor szkoły**
    1. Odroczenie płatności.
    2. Dodanie pracowników.
    3. Nadawanie pracownikom ról.
    4. Utworzenie/modyfikacja studiów.
    5. Utworzenie/modyfikacja kierunku studiów.
    6. Dostep do raportów finansowych.
    7. Dostep do raportów dotyczacych pracowników.

3 Schemat




PeopleDataChangeHistory
Roles
EmployeeRoles
People
Employees
Employees
Studies
Courses
Products
Webinars
Users
Vertabelo

4 Tabele

4.1 `People`

Tabela People przechowuje informacje o osobach w systemie.

- `PersonID` - Unikalny identyfikator osoby, generowany automatycznie.
- `FirstName` - Imię osoby.
- `LastName` - Nazwisko osoby.
- `BirthDate` - Data urodzenia osoby.
- `Address` - Adres zamieszkania osoby.
- `City` - Miasto zamieszkania osoby.
- `Region` - Region zamieszkania osoby.
- `PostalCode` - Kod pocztowy zamieszkania osoby.
- `Country` - Kraj zamieszkania osoby.
- `Phone` - Numer telefonu osoby.
- `Email` - Adres e-mail osoby.

```
CREATE TABLE People (
    PersonID int  NOT NULL IDENTITY,
    FirstName nvarchar(max)  NOT NULL,
    LastName nvarchar(500)  NOT NULL,
    BirthDate date  NOT NULL,
    Address nvarchar(500)  NOT NULL,
    City nvarchar(500)  NOT NULL,
    Region nvarchar(500)  NOT NULL,
    PostalCode nvarchar(20)  NOT NULL,
    Country nvarchar(500)  NOT NULL,
    Phone nvarchar(20)  NOT NULL,
    Email nvarchar(500)  NOT NULL,
    CONSTRAINT Person_pk PRIMARY KEY  (PersonID)
);
```

Warunki integralnościowe:

- `People_EmailValid`

  Adres email musi zawierać znak '@'

  ```
  CONSTRAINT People_EmailValid CHECK
  (Email LIKE '%@%')
  ```

- `People_BirthDateValid`

  Data urodzenia nie może być z przyszłości

  ```
  CONSTRAINT People_BirthDateValid CHECK
  (BirthDate <= GetDate())
  ```

- `People_PhoneIsValid`

  Numer telefonu składa się z cyfr

  ```
  CONSTRAINT People_PhoneIsValid CHECK
  ((ISNUMERIC([Phone])=(1)))
  ```

- `People_PostalCodeIsValid`

  Kod pocztowy musi być w poprawnym formacie.

```
CONSTRAINT People_PostalCodeIsValid CHECK
(PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'
OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]'
OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
```

4.2 Employees

Tabela Employees przechowuje informacje o pracownikach systemu.

- `EmployeeID` - Unikalny identyfikator pracownika, stanowiący klucz główny tabeli.
- `HireDate` - Data zatrudnienia pracownika.
- `IsActive` - Flaga określająca, czy pracownik jest aktualnie zatrudniony.

```
CREATE TABLE Employees (
    EmployeeID int  NOT NULL,
    HireDate date  NOT NULL,
    IsActive bit  NOT NULL,
    CONSTRAINT id PRIMARY KEY  (EmployeeID)
);
ALTER TABLE Employees ADD CONSTRAINT Employees_People
    FOREIGN KEY (EmployeeID)
    REFERENCES People (PersonID);
```

4.3 Users

Tabela Users przechowuje podstawowe informacje o użytkownikach.

- `UserID` - Unikalny identyfikator użytkownika, stanowiący klucz główny tabeli.

```
CREATE TABLE Users (
    UserID int  NOT NULL,
    CONSTRAINT Users_pk PRIMARY KEY  (UserID)
);
ALTER TABLE Users ADD CONSTRAINT Users_People
    FOREIGN KEY (UserID)
    REFERENCES People (PersonID);
```

4.4 PeopleDataChangeHistory

Tabela PeopleDataChangeHistory przechowuje historię zmian danych osobowych w systemie.

- `PersonDataChangeHistoryID` - Unikalny identyfikator wpisu historii zmian danych osobowych, generowany automatycznie.
- `PersonID` - Identyfikator osoby, do której odnosi się historia zmian.
- `ChangedAt` - Data i czas dokonania zmiany.
- `New_FirstName` - Nowe imię.
- `Old_FirstName` - Stare imię.
- `New_LastName` - Nowe nazwisko.
- `Old_LastName` - Stare nazwisko.
- `New_BirthDate` - Nowa data urodzenia.
- `Old_BirthDate` - Stara data urodzenia.
- `New_Address` - Nowy adres zamieszkania.
- `Old_Address` - Stary adres zamieszkania.
- `New_City` - Nowe miasto zamieszkania.

- `Old_City` - Stare miasto zamieszkania.
- `New_Region` - Nowy region zamieszkania.
- `Old_Region` - Stary region zamieszkania.
- `New_PostalCode` - Nowy kod pocztowy zamieszkania.
- `Old_PostalCode` - Stary kod pocztowy zamieszkania.
- `New_Country` - Nowy kraj zamieszkania.
- `Old_Country` - Stary kraj zamieszkania.
- `New_Email` - Nowy adres e-mail.
- `Old_Email` - Stary adres e-mail.
- `New_Phone` - Nowy numer telefonu.
- `Old_Phone` - Stary numer telefonu.

```
CREATE TABLE PeopleDataChangeHistory (
    PersonDataChangeHistoryID int  NOT NULL IDENTITY,
    PersonID int  NOT NULL,
    ChangedAt datetime  NOT NULL,
    New_FirstName nvarchar(max)  NOT NULL,
    Old_FirstName nvarchar(max)  NOT NULL,
    New_LastName nvarchar(500)  NOT NULL,
    Old_LastName nvarchar(500)  NOT NULL,
    New_BirthDate date  NOT NULL,
    Old_BirthDate date  NOT NULL,
    New_Address nvarchar(500)  NOT NULL,
    Old_Address nvarchar(500)  NOT NULL,
    New_City nvarchar(500)  NOT NULL,
    Old_City nvarchar(500)  NOT NULL,
    New_Region nvarchar(500)  NOT NULL,
    Old_Region nvarchar(500)  NOT NULL,
    New_PostalCode nvarchar(20)  NOT NULL,
    Old_PostalCode nvarchar(500)  NOT NULL,
    New_Country nvarchar(500)  NOT NULL,
    Old_Country nvarchar(500)  NOT NULL,
    New_Email nvarchar(500)  NOT NULL,
    Old_Email nvarchar(500)  NOT NULL,
    New_Phone nvarchar(20)  NOT NULL,
    Old_Phone nvarchar(500)  NOT NULL,
    CONSTRAINT PersonDataChangeHistory_pk PRIMARY KEY  (PersonDataChangeHistoryID)
);
ALTER TABLE PeopleDataChangeHistory ADD CONSTRAINT PeopleDataChangeHistory_People
    FOREIGN KEY (PersonID)
    REFERENCES People (PersonID);
```

Warunki integralnościowe:

- `PeopleDataChangeHistory_ChangedAtIsValid`

  Data zmiany nie może być w przyszłości

  ```
  CONSTRAINT PeopleDataChangeHistory_ChangedAtIsValid CHECK
  (ChangedAt <= GetDate())
  ```

- `PeopleDataChangeHistory_NewPostalCodeIsValid`

  Kod pocztowy musi być w poprawnym formacie

  ```
  CONSTRAINT PeopleDataChangeHistory_NewPostalCodeIsValid CHECK
  (New_PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'
  OR New_PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]'
  OR New_PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
  ```

- `PeopleDataChangeHistory_New_EmailValid`

  Nowy adres email musi zawierać znak '@'

```
CONSTRAINT PeopleDataChangeHistory_New_EmailValid CHECK
(New_Email LIKE '%@%')
```

- `PeopleDataChangeHistory_New_BirthDate`

  Data urodzenia nie może być w przyszłości

```
CONSTRAINT PeopleDataChangeHistory_New_BirthDate CHECK
(New_BirthDate <= GetDate())
```

- `PeopleDataChangeHistory_New_Phone`

  Nowy numer telefonu składa się z cyfr

```
CONSTRAINT PeopleDataChangeHistory_New_Phone CHECK
(ISNUMERIC(New_Phone)=(1))
```

4.5 Roles

Tabela Roles przechowuje informacje o różnych rolach w systemie.

- `RoleID` - Unikalny identyfikator roli, generowany automatycznie.
- `RoleName` - Nazwa roli, opisująca jej funkcję lub uprawnienia w systemie.

```
CREATE TABLE Roles (
    RoleID int  NOT NULL IDENTITY,
    RoleName nvarchar(200)  NOT NULL,
    CONSTRAINT RoleName UNIQUE (RoleName),
    CONSTRAINT employeeType PRIMARY KEY  (RoleID)
);
```

4.6 EmployeeRoles

Tabela EmployeeRoles przechowuje informacje o rolach przypisanych pracownikom systemu.

- `EmployeeRoleEntryID` - Unikalny identyfikator wpisu roli pracownika, który jest generowany automatycznie.
- `EmployeeID` - Identyfikator pracownika, do którego przypisana jest rola. Jest to klucz obcy, odnoszący

się do kolumny EmployeeID w tabeli Employees.

- `RoleID` - Identyfikator roli przypisanej pracownikowi. Jest to klucz obcy, odnoszący się do kolumny RoleID w tabeli Roles.

```
CREATE TABLE EmployeeRoles (
    EmployeeRoleEntryID int  NOT NULL IDENTITY,
    EmployeeID int  NOT NULL,
    RoleID int  NOT NULL,
    CONSTRAINT EmployeeRoles_ak_1 UNIQUE (EmployeeRoleEntryID, RoleID),
    CONSTRAINT EmployeeRoles_pk PRIMARY KEY  (EmployeeRoleEntryID)
);
ALTER TABLE EmployeeRoles ADD CONSTRAINT EmployeeCategories_Employees
    FOREIGN KEY (EmployeeID)
    REFERENCES Employees (EmployeeID)
    ON DELETE  CASCADE;
ALTER TABLE EmployeeRoles ADD CONSTRAINT EmployeeRoles_Roles
    FOREIGN KEY (RoleID)
    REFERENCES Roles (RoleID)
    ON DELETE  CASCADE;
```

4.7 Products

Tabela Products przechowuje informacje o produktach w systemie.

- `ProductID` - Unikalny identyfikator produktu.
- `Price` - Cena produktu.
- `AdvancePayment` - Wartość zaliczki do zapłaty za produkt.
- `ProductType` - Rodzaj produktu.
- `AddedAt` - Data dodania produktu do systemu.
- `ClosedAt` - Data zamknięcia produktu, jeżeli produkt nie jest już dostępny.

```
CREATE TABLE Products (
    ProductID int  NOT NULL IDENTITY,
    Price money  NOT NULL,
    AdvancePayment money  NULL,
    ProductType nvarchar(max)  NOT NULL,
    AddedAt datetime  NOT NULL DEFAULT GETDATE(),
    ClosedAt datetime  NULL,
    CONSTRAINT Products_pk PRIMARY KEY  (ProductID)
);
```

Warunki integralnościowe:

- `Products_PriceIsValid`

  Cena musi być większa lub równa 0

  ```
  CONSTRAINT Products_PriceIsValid CHECK
  (Price >= 0)
  ```

- `Products_AdvancePaymentIsValid`

  Zaliczka musi być większa od 0 i nie może być większa od ceny całkowitej, lub być NULL

  ```
  CONSTRAINT Products_AdvancePaymentIsValid CHECK
  ((AdvancePayment > 0 AND AdvancePayment < Price)
  OR (AdvancePayment IS NULL))
  ```

- `Products_ProductTypeIsValid`

  Produkt może być webinarem, kursem, studia mi, albo pojedyńczym postkaniem studyjnym

  ```
  CONSTRAINT Products_ProductTypeIsValid CHECK
  (ProductType IN ('studies', 'course','webinar', 'public study session'))
  ```

- `Products_AddedAtIsValid`

  AddedAt nie może być w przyszłości

  ```
  CONSTRAINT Products_AddedAtIsValid CHECK
  (AddedAt <= GetDate())
  ```

- `Products_ClosedAtIsValid`

  ClosedAt musi być po AddedAt

  ```
  CONSTRAINT Products_ClosedAtIsValid CHECK
  (ClosedAt <= GetDate() AND ClosedAt >= AddedAt)
  ```

4.8 ProductPriceChangeHistory

Tabela ProductPriceChangeHistory przechowuje historię zmian cen produktów.

- `ProductPriceChangeHistoryID` - Unikalny identyfikator historii zmian cen produktów.
- `ProductID` - Identyfikator produktu, którego cena uległa zmianie.
- `Old_Price` - Stara cena produktu przed zmianą.
- `New_Price` - Nowa cena produktu po zmianie.
- `Old_AdvancePayment` - Stara wartość zaliczki przed zmianą.
- `New_AdvancePayment` - Nowa wartość zaliczki po zmianie.
- `ChangedAt` - Data dokonania zmiany.

```
CREATE TABLE ProductPriceChangeHistory (
    ProductPriceChangeHistoryID int  NOT NULL IDENTITY,
    ProductID int  NOT NULL,
    Old_Price money  NOT NULL,
    New_Price money  NULL,
    Old_AdvancePayment money  NULL,
    New_AdvancePayment money  NULL,
    ChangedAt datetime  NOT NULL DEFAULT GETDATE(),
    CONSTRAINT ProductPriceChangeHistory_pk PRIMARY KEY  (ProductPriceChangeHistoryID)
);
ALTER TABLE ProductPriceChangeHistory ADD CONSTRAINT ProductHistory_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID)
    ON DELETE  CASCADE;
```

Warunki integralnościowe:

- `ProductHistory_ChangedAtIsValid`

  ChangedAt nie może być z przyszłości

  ```
  CONSTRAINT ProductHistory_ChangedAtIsValid CHECK
  (ChangedAt <= GetDate())
  ```

- `ProductHIstory_NewPriceIsValid`

  Nowa cenu musi być nieujemna

  ```
  CONSTRAINT ProductHIstory_NewPriceIsValid CHECK
  (New_price >= 0)
  ```

- `ProductHistory_NewAdvancePaymentIsValid`

  Jeżeli wpisano zaliczkę to musi być ona większa od zera

  ```
  CONSTRAINT ProductHistory_NewAdvancePaymentIsValid CHECK
  (New_AdvancePayment > 0)
  ```

4.9 Payments

Tabela Payments przechowuje informacje o płatnościach dokonanych przez użytkowników.

- `PaymentID` - Unikalny identyfikator płatności.
- `UserID` - Identyfikator użytkownika, który dokonał płatności.
- `ProductID` - Identyfikator produktu, na który użytkownik dokonał płatności.
- `Price` - Kwota płatności.
- `Date` - Data dokonania płatności.
- `Status` - Status płatności “Successful” albo “Failed”

```
CREATE TABLE Payments (
    PaymentID int  NOT NULL IDENTITY,
    UserID int  NOT NULL,
    ProductID int  NOT NULL,
    Price money  NOT NULL,
    Date datetime  NOT NULL,
    Status nvarchar(300)  NOT NULL,
    CONSTRAINT Payments_pk PRIMARY KEY  (PaymentID)
);
ALTER TABLE Payments ADD CONSTRAINT OrderHistory_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
ALTER TABLE Payments ADD CONSTRAINT OrderHistory_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Warunki integralnościowe:

- `Payments_Price`

  Kwota musi być >= 0

  ```
  CONSTRAINT Payments_Price CHECK
  (Price >= 0)
  ```

- `Payments_Status`

  Możliwe wartości dla statusu płatności: “Successful”, “Failed”

  ```
  CONSTRAINT Payments_Status CHECK
  (Status in ('Successful', 'Failed'))
  ```

- `Payments_Date`

  Data płatności nie może być z przyszłości

  ```
  CONSTRAINT Payments_Date CHECK
  (Date <= GetDate())
  ```

4.10 `Carts`

Tabela Carts przechowuje informacje o koszykach zakupowych użytkowników.

- `UserID` - Identyfikator użytkownika, do którego przypisany jest koszyk.
- `ProductID` - Identyfikator produktu, który został dodany do koszyka.
- `AddedAt` - Data i godzina dodania produktu do koszyka. Wartość domyślna to bieżąca data i czas.

```
CREATE TABLE Carts (
    UserID int  NOT NULL,
    ProductID int  NOT NULL,
    AddedAt datetime  NOT NULL DEFAULT GEtDATE(),
    CONSTRAINT Carts_pk PRIMARY KEY  (UserID,ProductID)
);
ALTER TABLE Carts ADD CONSTRAINT Carts_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
ALTER TABLE Carts ADD CONSTRAINT Carts_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Warunki integralnościowe:

- `Carts_AddedAtIsValid`

  AddedAt nie może być w przyszłości

```
CONSTRAINT Carts_AddedAtIsValid CHECK
(AddedAt <= GetDate())
```

4.11 CartHistory

Tabela CartHistory przechowuje historię zmian w koszyku zakupowym.

- `CartHistoryID` - Unikalny identyfikator historii koszyka.
- `UserID` - Identyfikator użytkownika, do którego przypisana jest historia koszyka.
- `ProductID` - Identyfikator produktu, który był dodany do koszyka.
- `AddedAt` - Data dodania produktu do koszyka.
- `RemovedAt` - Data usunięcia produktu z koszyka.

```
CREATE TABLE CartHistory (
    CartHistoryID int  NOT NULL IDENTITY,
    UserID int  NOT NULL,
    ProductID int  NOT NULL,
    AddedAt datetime  NOT NULL,
    RemovedAt datetime  NOT NULL,
    CONSTRAINT CartHistory_pk PRIMARY KEY  (CartHistoryID)
);
ALTER TABLE CartHistory ADD CONSTRAINT CartHistory_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
ALTER TABLE CartHistory ADD CONSTRAINT CartHistory_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Warunki integralnościowe:

- `CartHistory_AddedAt`

  AddedAt nie może być w przyszłości

  ```
  CONSTRAINT CartHistory_AddedAt CHECK
  (AddedAt <= GetDate())
  ```

- `CartHIstory_RemovedAt`

  RemovedAt musi być po AddedAt

  ```
  CONSTRAINT CartHIstory_RemovedAt CHECK
  (RemovedAt >= AddedAt AND RemovedAt <= GetDate())
  ```

4.12 Languages

Tabela Languages przechowuje informacje o dostępnych językach w systemie.

- `LanguageID` - Unikalny identyfikator języka, generowany automatycznie.
- `LanguageName` - Nazwa języka, opisująca konkretny język używany w systemie.

```
CREATE TABLE Languages (
    LanguageID int  NOT NULL IDENTITY,
    LanguageName nvarchar(200)  NOT NULL,
    CONSTRAINT LanguageName UNIQUE (LanguageName),
    CONSTRAINT Languages_pk PRIMARY KEY  (LanguageID)
);
```

4.13 Webinars

Tabela Webinars przechowuje informacje dotyczące webinarów oferowanych w systemie.

- `WebinarID` - Unikalny identyfikator webinaru.
- `WebinarName` - Nazwa webinaru.
- `Description` - Opis webinaru, zawierający informacje na temat treści i celów.
- `StartDate` - Data rozpoczęcia webinaru.
- `EndDate` - Data zakończenia webinaru.
- `RecordingLink` - Link do nagrania webinaru (opcjonalny).
- `WebinarLink` - Link do udziału w webinarze.
- `LecturerID` - Identyfikator prowadzącego webinar, który jest pracownikiem systemu.
- `TranslatorID` - Identyfikator tłumacza webinaru (opcjonalny).
- `LanguageID` - Identyfikator języka, w jakim prowadzony jest webinar.
- `RecordingReleaseDate` - Data udostępnienia nagrania webinaru (opcjonalna).

```
CREATE TABLE Webinars (
    WebinarID int  NOT NULL,
    WebinarName nvarchar(max)  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    RecordingLink nvarchar(max)  NULL,
    WebinarLink nvarchar(max)  NOT NULL,
    LecturerID int  NOT NULL,
    TranslatorID int  NULL,
    LanguageID int  NOT NULL,
    RecordingReleaseDate date  NULL,
    CONSTRAINT Webinars_pk PRIMARY KEY  (WebinarID)
);
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Lecturers
    FOREIGN KEY (TranslatorID)
    REFERENCES Employees (EmployeeID);
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Products
    FOREIGN KEY (WebinarID)
    REFERENCES Products (ProductID);
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Translators
    FOREIGN KEY (LecturerID)
    REFERENCES Employees (EmployeeID);
```

Warunki integralnościowe:

- `Webinars_RecodingReleaseDateValid`

  RecodingReleaseData musi być po EndDate

  ```
  CONSTRAINT Webinars_RecodingReleaseDateValid CHECK
  (RecordingReleaseDate >= EndDate)
  ```

- `Webinars_RecodingLinkRelationWithRecordingReleaseDate`

  Jeżeli jest nagranie to RecordingReleaseDate nie może być nullem, jeżeli nie ma to RecordingReleaseDate musi być nullem

  ```
  CONSTRAINT Webinars_RecodingLinkRelationWithRecordingReleaseDate CHECK
  ((RecordingReleaseDate IS NULL AND RecordingLink IS NULL)
  OR
  ```

```
(RecordingReleaseDate IS NOT NULL AND RecordingLink IS NOT NULL))
```

- `Webinars_DateRangeIsValid`

  StartDate musi być przed EndDate

```
CONSTRAINT Webinars_DateRangeIsValid CHECK
(StartDate < EndDate)
```

4.14 `WebinarParticipants`

Tabela WebinarParticipants przechowuje informacje o uczestnikach webinarów.

- `WebinarParticipantID` - Unikalny identyfikator uczestnika webinaru.
- `UserID` - Identyfikator użytkownika, który jest uczestnikiem webinaru.
- `WebinarID` - Identyfikator webinaru, do którego przypisany jest uczestnik.
- `WebinarPrice` - Cena uczestnictwa w webinarze.
- `DuePostponedPayment` - Data odroczonego terminu płatności.
- `FullPricePaymentID` - Identyfikator pełnej płatności.
- `AddedAt` - Data dodania uczestnika do webinaru.

```
CREATE TABLE WebinarParticipants (
    WebinarParticipantID int  NOT NULL IDENTITY,
    UserID int  NOT NULL,
    WebinarID int  NOT NULL,
    WebinarPrice money  NOT NULL,
    DuePostponedPayment datetime  NULL,
    FullPricePaymentID int  NULL,
    AddedAt datetime  NOT NULL DEFAULT GETDATE(),
    CONSTRAINT WebinarParticipants_pk PRIMARY KEY  (WebinarParticipantID)
);
ALTER TABLE WebinarParticipants ADD CONSTRAINT WebinarParticipants_Payments
    FOREIGN KEY (FullPricePaymentID)
    REFERENCES Payments (PaymentID)
    ON DELETE  CASCADE;
ALTER TABLE WebinarParticipants ADD CONSTRAINT WebinarParticipants_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID)
    ON DELETE  CASCADE;
ALTER TABLE WebinarParticipants ADD CONSTRAINT WebinarParticipants_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES Webinars (WebinarID)
    ON DELETE  CASCADE;
```

Warunki integralnościowe:

- `WebinarParticipants_WebinarPrice`

  Cena za webinar musi być większa lub równa zero

```
CONSTRAINT WebinarParticipants_WebinarPrice CHECK
(WebinarPrice >= 0)
```

- `WebinarParticipants_FulPricePaymentID`

  FullPricePaymentID może być NULL gdy istnieje DuePostponedPayment lub gdy WebinarPrice = 0

```
CONSTRAINT WebinarParticipants_FulPricePaymentID CHECK
(FullPricePaymentID IS NOT NULL OR
 (DuePostponedPayment IS NOT NULL OR
     WebinarPrice = 0))
```

4.15 WebinarsAttendence

Tabela WebinarsAttendance przechowuje informacje dotyczące uczestnictwa w webinarze.

- `WebinarID` - Identyfikator webinaru, do którego odnosi się uczestnictwo.
- `WebinarParticipantID` - Identyfikator uczestnika webinaru.
- `WasPresent` - Wartość logiczna określająca, czy uczestnik był obecny na webinarze (1 - obecny, 0 - nieobecny).

```
CREATE TABLE WebinarsAttendence (
    WebinarID int  NOT NULL,
    WebinarParticipantID int  NOT NULL,
    WasPresent bit  NOT NULL,
    CONSTRAINT WebinarsAttendence_pk PRIMARY KEY  (WebinarID,WebinarParticipantID)
);
ALTER TABLE WebinarsAttendence ADD CONSTRAINT WebinarsAttendence_WebinarParticipants
    FOREIGN KEY (WebinarParticipantID)
    REFERENCES WebinarParticipants (WebinarParticipantID);
ALTER TABLE WebinarsAttendence ADD CONSTRAINT WebinarsAttendence_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES Webinars (WebinarID)
    ON DELETE  CASCADE;
```

4.16 Courses

Tabela Courses przechowuje informacje o kursach dostępnych w systemie.

- `CourseID` - Unikalny identyfikator kursu.
- `CourseName` - Nazwa kursu.
- `Description` - Opis kursu, zawierający informacje dotyczące treści i celów.
- `StartDate` - Data rozpoczęcia kursu.
- `EndDate` - Data zakończenia kursu.
- `CoordinatorID` - Identyfikator koordynatora kursu, który jest pracownikiem systemu.
- `MaxStudents` - Maksymalna liczba studentów, którzy mogą uczestniczyć w kursie.
- `LanguageID` - Identyfikator języka, w jakim prowadzony jest kurs.

```
CREATE TABLE Courses (
    CourseID int  NOT NULL,
    CourseName nvarchar(max)  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    CoordinatorID int  NOT NULL,
    MaxStudents int  NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT Courses_pk PRIMARY KEY  (CourseID)
);
ALTER TABLE Courses ADD CONSTRAINT Courses_Employees
    FOREIGN KEY (CoordinatorID)
    REFERENCES Employees (EmployeeID);
ALTER TABLE Courses ADD CONSTRAINT Courses_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
ALTER TABLE Courses ADD CONSTRAINT Courses_Products
    FOREIGN KEY (CourseID)
    REFERENCES Products (ProductID);
```

Warunki integralnościowe:

- `Course_MaxStudents`

  MaxStudents może być NULL, jeżeli np. jest to kurs wyłącznie online, w przeciwnym wypadku musi być > 0

```
CONSTRAINT Course_MaxStudents CHECK
(MaxStudents is NULL OR
(MaxStudents > 0) )
```

- `Course_DateIntervalIsValid`

  EndDate musi być po StartDate

```
CONSTRAINT Course_DateIntervalIsValid CHECK
(StartDate < EndDate)
```

4.17 CourseParticipants

Tabela CourseParticipants przechowuje informacje o uczestnikach kursów.

- `CourseParticipantID` - Unikalny identyfikator uczestnika kursu.
- `UserID` - Identyfikator użytkownika, który jest jednocześnie kluczem obcym powiązanym z tabelą Users.
- `CourseID` - Identyfikator kursu, który jest jednocześnie kluczem obcym powiązanym z tabelą Courses.
- `CoursePrice` - Cena kursu dla uczestnika.
- `EntryFee` - Cena zaliczki dla tego kursu.
- `EntryFeePaymentID` - Identyfikator płatności za zaliczkę, który jest jednocześnie kluczem obcym powiązanym z tabelą Payments.
- `RemainingPaymentID` - Identyfikator pozostałej płatności, który jest jednocześnie kluczem obcym powiązanym z tabelą Payments.
- `FullPricePaymentID` - Identyfikator pełnej płatności, który jest jednocześnie kluczem obcym powiązanym z tabelą Payments.
- `DuePostponedPayment` - Data, do której została odroczona płatność.
- `AddedAt` - Data dodania uczestnika do kursu.
- `Completed` - Wartość logiczna określająca, czy uczestnik ukończył kurs (1 - ukończono, 0 - nie ukończono).

```
CREATE TABLE CourseParticipants (
    CourseParticipantID int  NOT NULL IDENTITY,
    UserID int  NOT NULL,
    CourseID int  NOT NULL,
    CoursePrice money  NOT NULL,
    EntryFee money  NOT NULL,
    EntryFeePaymentID int  NULL,
    RemainingPaymentID int  NULL,
    FullPricePaymentID int  NULL,
    DuePostponedPayment datetime  NULL,
    AddedAt datetime  NOT NULL DEFAULT GETDATE(),
    Completed bit  NOT NULL,
    CONSTRAINT CourseParticipants_pk PRIMARY KEY  (CourseParticipantID)
);
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseParticipants_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseParticipants_EntryFeePayments
    FOREIGN KEY (EntryFeePaymentID)
    REFERENCES Payments (PaymentID);
```

```
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseParticipants_FullPricePayments
    FOREIGN KEY (FullPricePaymentID)
    REFERENCES Payments (PaymentID);
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseParticipants_RemainingPayments
    FOREIGN KEY (RemainingPaymentID)
    REFERENCES Payments (PaymentID);
ALTER TABLE CourseParticipants ADD CONSTRAINT CourseParticipants_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Warunki integralnościowe:

- `CourseParticipants_PriceCheck`

  Cena musi być >=0

  ```
  CONSTRAINT CourseParticipants_PriceCheck CHECK
  (CoursePrice >= 0)
  ```

- `CourseParticpants_EntryFeeCheck`

  Zaliczka nie może być ujemna oraz nie może być większa od całkowitej ceny

  ```
  CONSTRAINT CourseParticpants_EntryFeeCheck CHECK
  (EntryFee >= 0 and EntryFee <= CoursePrice)
  ```

4.18 Modules

Tabela Modules przechowuje informacje o modułach składających się na kursy w systemie.

- `ModuleID` - Unikalny identyfikator modułu, automatycznie generowany przez system.
- `CourseID` - Identyfikator kursu, do którego przypisany jest moduł.
- `ModuleName` - Nazwa modułu.
- `ModuleDescription` - Opis modułu, zawierający szczegółowe informacje na temat treści i celów.

```
CREATE TABLE Modules (
    ModuleID int  NOT NULL IDENTITY,
    CourseID int  NOT NULL,
    ModuleName nvarchar(max)  NOT NULL,
    ModuleDescription nvarchar(max)  NOT NULL,
    CONSTRAINT Modules_pk PRIMARY KEY  (ModuleID)
);
ALTER TABLE Modules ADD CONSTRAINT Modules_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

4.19 CoursesSessions

Tabela CoursesSessions przechowuje informacje o sesjach kursów.

- `CourseSessionID` - Unikalny identyfikator sesji kursu.
- `LanguageID` - Klucz obcy określający język, w jakim odbywa się sesja kursu.
- `ModuleID` - Klucz obcy wskazujący na moduł związany z daną sesją kursu.
- `LecturerID` - Klucz obcy wskazujący na wykładowcę prowadzącego daną sesję kursu.
- `TranslatorID` - Opcjonalny klucz obcy wskazujący na tłumacza przypisanego do sesji kursu.

```
CREATE TABLE CoursesSessions (
    CourseSessionID int  NOT NULL IDENTITY,
    LanguageID int  NOT NULL,
    ModuleID int  NOT NULL,
```

```
    LecturerID int  NOT NULL,
    TranslatorID int  NULL,
    CONSTRAINT CoursesSessions_pk PRIMARY KEY  (CourseSessionID)
);
ALTER TABLE CoursesSessions ADD CONSTRAINT CoursesSessions_Employees
    FOREIGN KEY (LecturerID)
    REFERENCES Employees (EmployeeID);
ALTER TABLE CoursesSessions ADD CONSTRAINT CoursesSessions_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
ALTER TABLE CoursesSessions ADD CONSTRAINT CoursesSessions_Modules
    FOREIGN KEY (ModuleID)
    REFERENCES Modules (ModuleID)
    ON DELETE  CASCADE;
ALTER TABLE CoursesSessions ADD CONSTRAINT CoursesSessions_Translators
    FOREIGN KEY (TranslatorID)
    REFERENCES Employees (EmployeeID);
```

4.20 `CourseOfflineSessions`

Tabela CourseOfflineSessions przechowuje informacje o sesjach kursów offline.

- `CourseOfflineSessionID` - Unikalny identyfikator sesji kursu offline.
- `Link` - Łącze do sesji kursu offline.
- `Description` - Opis sesji kursu offline, zawierający informacje na temat treści i celów.
- `UploadedAt` - Data przesłania informacji o sesji, domyślnie ustawiana na bieżącą datę.

```
CREATE TABLE CourseOfflineSessions (
    CourseOfflineSessionID int  NOT NULL,
    Link nvarchar(max)  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    UploadedAt datetime  NOT NULL DEFAULT GETDATE(),
    CONSTRAINT CourseOfflineSessions_pk PRIMARY KEY  (CourseOfflineSessionID)
);
ALTER TABLE CourseOfflineSessions ADD CONSTRAINT CourseOfflineSessions_CoursesSessions
    FOREIGN KEY (CourseOfflineSessionID)
    REFERENCES CoursesSessions (CourseSessionID)
    ON DELETE  CASCADE;
```

Warunki integralnościowe:

- `CourseOfflineSessions_UploadedAtIsValid`

  UploadedAt nie może być w przyszłości

  ```
  CONSTRAINT CourseOfflineSessions_UploadedAtIsValid CHECK
  (UploadedAt <= GETDATE() )
  ```

4.21 `CourseStationarySessions`

Tabela CourseStationarySessions przechowuje informacje o sesjach stacjonarnych kursów.

- `CourseStationarySessionID` - Unikalny identyfikator sesji stacjonarnej kursu.
- `StartDate` - Data i godzina rozpoczęcia sesji stacjonarnej.
- `EndDate` - Data i godzina zakończenia sesji stacjonarnej.
- `Address` - Adres, na którym odbywa się sesja stacjonarna.
- `City` - Miasto, w którym odbywa się sesja stacjonarna.
- `Country` - Kraj, w którym odbywa się sesja stacjonarna.
- `PostalCode` - Kod pocztowy sesji stacjonarnej, spełniający warunki poprawności.

- `ClassroomNumber` - Numer sali, w której odbywa się sesja stacjonarna.
- `MaxStudents` - Maksymalna liczba studentów, którzy mogą uczestniczyć w sesji stacjonarnej.

```
CREATE TABLE CourseStationarySessions (
    CourseStationarySessionID int  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    Address nvarchar(500)  NOT NULL,
    City nvarchar(500)  NOT NULL,
    Country nvarchar(500)  NOT NULL,
    PostalCode nvarchar(20)  NOT NULL,
    ClassroomNumber nvarchar(30)  NOT NULL,
    MaxStudents int  NOT NULL,
    CONSTRAINT CourseStationarySessions_pk PRIMARY KEY  (CourseStationarySessionID)
);
ALTER TABLE CourseStationarySessions ADD CONSTRAINT CourseStationarySessions_CoursesSessions
    FOREIGN KEY (CourseStationarySessionID)
    REFERENCES CoursesSessions (CourseSessionID)
    ON DELETE  CASCADE;
```

Warunki integralnościowe:

- `CourseStationarySessions_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT CourseStationarySessions_DateIntervalIsValid CHECK
  (StartDate < EndDate)
  ```

- `CourseStationarySessions_PostalCodeIsValid`

  Kod pocztowy ma być w poprawnym formacie

  ```
  CONSTRAINT CourseStationarySessions_PostalCodeIsValid CHECK
  (PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'
  OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]'
  OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
  ```

- `CourseStationarySessions_MaxStudentsIValid`

  MaxStudents musi być większy od zera

  ```
  CONSTRAINT CourseStationarySessions_MaxStudentsIValid CHECK
  (MaxStudents > 0)
  ```

4.22 CourseOnlineSessions

Tabela CourseOnlineSessions przechowuje informacje o sesjach kursów online.

- `CourseOnlineSessionID` - Unikalny identyfikator sesji kursu online.
- `StartDate` - Data rozpoczęcia sesji kursu online.
- `EndDate` - Data zakończenia sesji kursu online.
- `WebinarLink` - Link do platformy webinarowej, na której odbywa się sesja kursu online.
- `RecordingLink` - Link do nagrania sesji kursu online. Może być NULL w przypadku braku dostępnego nagrania.

```
CREATE TABLE CourseOnlineSessions (
    CourseOnlineSessionID int  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    WebinarLink nvarchar(max)  NOT NULL,
    RecordingLink nvarchar(max)  NULL,
    CONSTRAINT CourseOnlineSessions_pk PRIMARY KEY  (CourseOnlineSessionID)
);
```

```
ALTER TABLE CourseOnlineSessions ADD CONSTRAINT CourseOnlineSessions_CoursesSessions
    FOREIGN KEY (CourseOnlineSessionID)
    REFERENCES CoursesSessions (CourseSessionID)
    ON DELETE  CASCADE;
```

Warunki integralnościowe:

- `CourseOnlineSessions_DateIntervalCheck`

  EndDate musi być po StartDate

  ```
  CONSTRAINT CourseOnlineSessions_DateIntervalCheck CHECK
  (StartDate < EndDate)
  ```

4.23 CourseSessionsAttendance

Tabela CourseSessionsAttendance przechowuje informacje o uczestnictwie w sesjach kursu.

- `CourseParticipantID` - Identyfikator uczestnika kursu, który jest jednocześnie kluczem obcym powiązanym z tabelą CourseParticipants.
- `CourseSessionID` - Identyfikator sesji kursu, który jest jednocześnie kluczem obcym powiązanym z tabelą CourseSessions.
- `Completed` - Wartość logiczna określająca, czy uczestnik ukończył daną sesję kursu (1 - ukończono, 0 - nie ukończono).

```
CREATE TABLE CourseSessionsAttendance (
    CourseParticipantID int  NOT NULL,
    CourseSessionID int  NOT NULL,
    Completed bit  NOT NULL,
    CONSTRAINT CourseSessionsAttendance_pk PRIMARY KEY  (CourseSessionID,CourseParticipantID)
);
ALTER TABLE CourseSessionsAttendance ADD CONSTRAINT CourseSessionsAttendance_CourseParticipants
    FOREIGN KEY (CourseParticipantID)
    REFERENCES CourseParticipants (CourseParticipantID)
    ON DELETE  CASCADE;
ALTER TABLE CourseSessionsAttendance ADD CONSTRAINT CourseSessionsAttendance_CoursesSessions
    FOREIGN KEY (CourseSessionID)
    REFERENCES CoursesSessions (CourseSessionID)
    ON DELETE  CASCADE;
```

4.24 FieldsOfStudies

Tabela FieldsOfStudies przechowuje informacje o wszystkich dziedzinach oferowanych studiów.

- `FieldOfStudiesID` - Unikalny identyfikator dziedziny studiów.
- `Name` - Nazwa dziedziny studiów
- `Description` - Opis dziedziny studiów

```
CREATE TABLE FieldsOfStudies (
    FieldOfStudiesID int  NOT NULL IDENTITY,
    Name nvarchar(max)  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    CONSTRAINT FieldsOfStudies_pk PRIMARY KEY  (FieldOfStudiesID)
);
```

4.25 Studies

Tabela Studies przechowuje informacje o oferowanych programach studiów.

- `StudiesID` - Unikalny identyfikator studiów
- `Name` - Nazwa studiów

- `Description` - Opis studiów
- `CoordinatorID` - Identyfikator pracownika będącego koordynatorem studiów
- `StartDate` - Data rozpoczęcia studiów
- `EndDate` - Data zakończenia studiów
- `MaxStudents` - Maksymalna liczba studentów mogących zapisać się na studia
- `LanguageID` - ID języka, w którym będą prowadzone studia
- `FieldOfStudiesID` - ID dziedziny studiów
- `SemesterNumber` - Numer semestru studiów

```
CREATE TABLE Studies (
    StudiesID int  NOT NULL,
    Name nvarchar(max)  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    CoordinatorID int  NOT NULL,
    StartDate Date  NOT NULL,
    EndDate Date  NOT NULL,
    MaxStudents int  NOT NULL,
    LanguageID int  NOT NULL,
    FieldOfStudiesID int  NOT NULL,
    SemesterNumber int  NOT NULL,
    CONSTRAINT Studies_pk PRIMARY KEY  (StudiesID)
);
ALTER TABLE Studies ADD CONSTRAINT Studies_Employees
    FOREIGN KEY (CoordinatorID)
    REFERENCES Employees (EmployeeID);
ALTER TABLE Studies ADD CONSTRAINT Studies_FieldsOfStudies
    FOREIGN KEY (FieldOfStudiesID)
    REFERENCES FieldsOfStudies (FieldOfStudiesID);
ALTER TABLE Studies ADD CONSTRAINT Studies_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
ALTER TABLE Studies ADD CONSTRAINT Studies_Products
    FOREIGN KEY (StudiesID)
    REFERENCES Products (ProductID);
```

Warunki integralnościowe:

- `Studies_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT Studies_DateIntervalIsValid CHECK
  (StartDate < EndDate)
  ```

- `Studies_MaxStudentsIsValid`

  Maksymalna liczba studentów musi być większa od 0

  ```
  CONSTRAINT Studies_MaxStudentsIsValid CHECK
  (MaxStudents > 0)
  ```

- `Studies_SemesterIsValid`

  Numery semestrów zaczynają się od 1

  ```
  CONSTRAINT Studies_SemesterIsValid CHECK
  (SemesterNumber >= 1)
  ```

4.26 Students

Tabela Students przechowuje podstawowe informacje o studentach.

- `StudentID` - Unikalny identyfikator studenta
- `UserID` - ID użytkownika
- `StudiesID` - Identyfikator studiów
- `StudiesPrice` - Całkowita cena studiów
- `EntryFee` - Kwota zaliczki
- `DuePostponedPayment` - Data do której płatność została odroczona
- `EntryFeePaymentID` - Identyfikator płatności dla zaliczki
- `RemainingPaymentID` - Identyfikator spłaty pozostałej kwoty
- `FullPaymentID` - Identyfikator spłaty całości studiów
- `AddedAt` - Data dodania studenta do bazy
- `Completed` - Oznaczenie informujące o tym, czy student uzyskał zaliczenie studiów

```
CREATE TABLE Students (
    StudentID int  NOT NULL IDENTITY,
    UserID int  NOT NULL,
    StudiesID int  NOT NULL,
    StudiesPrice money  NOT NULL,
    EntryFee money  NOT NULL,
    DuePostponedPayment datetime  NULL,
    EntryFeePaymentID int  NULL,
    RemainingPaymentID int  NULL,
    FullPaymentID int  NULL,
    AddedAt datetime  NOT NULL DEFAULT GETDATE(),
    Completed bit  NOT NULL DEFAULT 0,
    CONSTRAINT Students_pk PRIMARY KEY  (StudentID)
);
ALTER TABLE Students ADD CONSTRAINT Students_FullPayments
    FOREIGN KEY (RemainingPaymentID)
    REFERENCES Payments (PaymentID);
ALTER TABLE Students ADD CONSTRAINT Students_Payments
    FOREIGN KEY (FullPaymentID)
    REFERENCES Payments (PaymentID);
ALTER TABLE Students ADD CONSTRAINT Students_RemainingPayments
    FOREIGN KEY (EntryFeePaymentID)
    REFERENCES Payments (PaymentID);
ALTER TABLE Students ADD CONSTRAINT Students_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
ALTER TABLE Students ADD CONSTRAINT Students_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

Warunki integralnościowe:

- `Students_PriceIsValid`

  Cena za studia musi być większa od zera

  ```
  CONSTRAINT Students_PriceIsValid CHECK
  (StudiesPrice > 0)
  ```

- `Students_EntryFeeIsValid`

  Zaliczka musi być większa od zera i mniejsza od całkowitej ceny studiów.

```
CONSTRAINT Students_EntryFeeIsValid CHECK
(EntryFee > 0 AND EntryFee < StudiesPrice)
```

4.27 Subjects

Tabela Subjects przechowuje informacje na temat wszystkich przedmiotów podpiętych pod wszystkie kierunki studiów.

- `SubjectID` - Unikalny identyfikator przedmiotu
- `StudiesID` - ID studiów, pod które podpięty jest przedmiot
- `Description` - Opis przedmiotu
- `CoordinatorID` - Identyfikator pracownika będącego koordynatorem przedmiotu
- `SubjectName` - Nazwa przedmiotu

```
CREATE TABLE Subjects (
    SubjectID int  NOT NULL IDENTITY,
    StudiesID int  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    CoordinatorID int  NOT NULL,
    SubjectName nvarchar(max)  NOT NULL,
    CONSTRAINT SubjectID PRIMARY KEY  (SubjectID)
);
ALTER TABLE Subjects ADD CONSTRAINT Studies_Subjects
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
ALTER TABLE Subjects ADD CONSTRAINT Subjects_Employees
    FOREIGN KEY (CoordinatorID)
    REFERENCES Employees (EmployeeID);
```

4.28 Exams

Tabela Exams przechowuje informacje o egzaminach.

- `ExamID` - Unikalny identyfikator egzaminu
- `SubjectID` - Identyfikator przedmiotu, do którego przeprowadzany jest egzamin
- `StartDate` - Data rozpoczęcia egzaminu
- `EndDate` - Data zakończenia egzaminu
- `Country` - Nazwa państwa, w którym przeprowadzany jest egzamin
- `City` - Nazwa miasta, w którym przeprowadzany jest egzamin
- `PostalCode` - Kod pocztowy adresu, w którym przeprowadzany jest egzamin
- `Address` - Dokładny adres przeprowadzania egzaminu

```
CREATE TABLE Exams (
    ExamID int  NOT NULL IDENTITY,
    SubjectID int  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    Country nvarchar(500)  NOT NULL,
    City nvarchar(500)  NOT NULL,
    PostalCode nvarchar(500)  NOT NULL,
    Address nvarchar(500)  NOT NULL,
    CONSTRAINT Exams_pk PRIMARY KEY  (ExamID)
);
ALTER TABLE Exams ADD CONSTRAINT Exams_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES Subjects (SubjectID);
```

Warunki integralnościowe:

- `Exams_DateInteralIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT Exams_DateInteralIsValid CHECK
  (StartDate < EndDate)
  ```

- `Exams_PostalCodeIsValid`

  Kod pocztowy ma być w poprawnym formacie

  ```
  CONSTRAINT Exams_PostalCodeIsValid CHECK
  (PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'
  OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]'
  OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
  ```

4.29 ExamsGrades

Tabela ExamGrades przechowuje informacje o ocenach z przeprowadzonych egzaminów.

- `StudentID` - Identyfikator egzaminowanego studenta
- `ExamID` - Identyfikator egzaminu, w którym student brał udział
- `FinalGrade` - Ocena końcowa uzyskana przez studenta z egzaminu

```
CREATE TABLE ExamsGrades (
    StudentID int  NOT NULL,
    ExamID int  NOT NULL,
    FinalGrade decimal(2,1)  NOT NULL,
    CONSTRAINT ExamsGrades_pk PRIMARY KEY  (StudentID,ExamID)
);
ALTER TABLE ExamsGrades ADD CONSTRAINT ExamsGrades_Exams
    FOREIGN KEY (ExamID)
    REFERENCES Exams (ExamID);
ALTER TABLE ExamsGrades ADD CONSTRAINT Grades_Students
    FOREIGN KEY (StudentID)
    REFERENCES Students (StudentID);
```

Warunki integralnościowe:

- `FinalExams_FinalGradeIsValid`

  Przyjmujemy skalę ocen jak na publicznej uczelni wyższej

  ```
  CONSTRAINT FinalExams_FinalGradeIsValid CHECK
  (FinalGrade IN (2.0, 3.0, 3.5, 4.0, 4.5, 5.0))
  ```

4.30 StudiesSessions

Tabela StudiesSessions przechowuje informacje o wszystkich zajęciach w ramach każdego z przedmiotów.

- `StudiesSessionID` - Unikalny identyfikator zajęć
- `SubjectID` - ID przedmiotu
- `StartDate` - Data rozpoczęcia zajęć
- `EndDate` - Data zakończenia zajęć
- `LecturerID` - Identyfikator pracownika prowadzącego zajęcia
- `MaxStudents` - Maksymalna liczba studentów, którzy mogą wziąć udział w zajęciach
- `TranslatorID` - W przypadku przedmiotu prowadzonego w innym języku - identyfikator tłumacza
- `LanguageID` - Identyfikator języku prowadzenia zajęć

```
CREATE TABLE StudiesSessions (
    StudiesSessionID int  NOT NULL IDENTITY,
    SubjectID int  NOT NULL,
    StartDate datetime  NOT NULL,
    EndDate datetime  NOT NULL,
    LecturerID int  NOT NULL,
    MaxStudents int  NOT NULL,
    TranslatorID int  NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT StudiesSessions_pk PRIMARY KEY  (StudiesSessionID)
);
ALTER TABLE StudiesSessions ADD CONSTRAINT StudiesSessions_Employees
    FOREIGN KEY (TranslatorID)
    REFERENCES Employees (EmployeeID);
ALTER TABLE StudiesSessions ADD CONSTRAINT StudiesSessions_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES Languages (LanguageID);
ALTER TABLE StudiesSessions ADD CONSTRAINT StudySessions_Employees
    FOREIGN KEY (LecturerID)
    REFERENCES Employees (EmployeeID);
ALTER TABLE StudiesSessions ADD CONSTRAINT StudySessions_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES Subjects (SubjectID);
```

Warunki integralnościowe:

- `StudiesSessions_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT StudiesSessions_DateIntervalIsValid CHECK
  (StartDate < EndDate)
  ```

- `MaxStudentsVerification`

  ```
  CONSTRAINT MaxStudentsVerification CHECK
  (MaxStudents > 0)
  ```

4.31 StationaryStudiesSessions

Tabela StationaryStudiesSessions przechowuje informacje na temat zajęć stacjonarnych.

- `StationaryStudiesSessionID` - Unikalny identyfikator zajęć stacjonarnych
- `Address` - Adres, pod którym odbywają się zajęcia.
- `City` - Nazwa miasta, w którym odbywają się zajęcia
- `Country` - Nazwa państwa, w którym odbywają się zajęcia
- `PostalCode` - Kod pocztowy do adresu, w którym odbywają się zajęcia
- `ClassroomNumber` - Nr sali, w której odbywają się zajęcia

```
CREATE TABLE StationaryStudiesSessions (
    StationaryStudiesSessionID int  NOT NULL,
    Address nvarchar(500)  NOT NULL,
    City nvarchar(500)  NOT NULL,
    Country nvarchar(500)  NOT NULL,
    PostalCode nvarchar(20)  NOT NULL,
    ClassroomNumber nvarchar(30)  NOT NULL,
    CONSTRAINT StationaryStudiesSessions_pk PRIMARY KEY  (StationaryStudiesSessionID)
);
ALTER TABLE StationaryStudiesSessions ADD CONSTRAINT StationaryStudiesSessions_StudySessions
    FOREIGN KEY (StationaryStudiesSessionID)
    REFERENCES StudiesSessions (StudiesSessionID)
    ON DELETE  CASCADE;
```

Warunki integralnościowe:

- `StationaryStudiesSessions_PostalCodeIsValid`

  Kod pocztowy jest w poprawny formacie

```
CONSTRAINT StationaryStudiesSessions_PostalCodeIsValid CHECK
(PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'
OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]'
OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
```

4.32 OnlineStudiesSessions

Tabela OnlineStudiesSessions przechowuje informacje o zajęciach odbywanych w formie zdalnej.

- `OnlineStudiesSessionID` - Unikalny identyfikator zajęć online
- `WebinarLink` - Link do spotkania na żywo
- `RecordingLink` - Link do nagrania, w przypadku gdy spotkanie było nagrywane

```
CREATE TABLE OnlineStudiesSessions (
    OnlineStudiesSessionID int  NOT NULL,
    WebinarLink nvarchar(max)  NOT NULL,
    RecordingLink nvarchar(max)  NULL,
    CONSTRAINT OnlineStudiesSessions_pk PRIMARY KEY  (OnlineStudiesSessionID)
);
ALTER TABLE OnlineStudiesSessions ADD CONSTRAINT OnlineStudiesSessions_StudySessions
    FOREIGN KEY (OnlineStudiesSessionID)
    REFERENCES StudiesSessions (StudiesSessionID)
    ON DELETE  CASCADE;
```

4.33 StudiesSessionsAttendence

Tabela StudiesSessionsAttendence przechowuje informacje o obecnościach studentów na zajęciach.

- `SessionID` - Identyfikator zajęć
- `StudentID` - ID studenta zapisaego na zajęcia
- `Completed` - Oznaczenie obecności studenta na zajęciach,

```
CREATE TABLE StudiesSessionsAttendence (
    SessionID int  NOT NULL,
    StudentID int  NOT NULL,
    Completed bit  NOT NULL,
    CONSTRAINT StudiesSessionsAttendence_pk PRIMARY KEY  (SessionID,StudentID)
);
ALTER TABLE StudiesSessionsAttendence ADD CONSTRAINT StudiesSessionsAttendence_Students
    FOREIGN KEY (StudentID)
    REFERENCES Students (StudentID);
ALTER TABLE StudiesSessionsAttendence ADD CONSTRAINT StudySessionsAttendence_StudySessions
    FOREIGN KEY (SessionID)
    REFERENCES StudiesSessions (StudiesSessionID)
    ON DELETE  CASCADE;
```

4.34 Internships

Tabela Internships przechowuje informacje o wszystkich oferowanych programach stażowych.

- `InternshipID` - Unikalny identyfikator programu stażowego.
- `StudiesID` - ID studiów, do których przypisany jest program stażowy
- `Description` - Opis stażu
- `StartDate` - Data rozpoczęcia stażu
- `EndDate` - Data zakończenia stażu

```
CREATE TABLE Internships (
    InternshipID int  NOT NULL IDENTITY,
    StudiesID int  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    StartDate date  NOT NULL,
    EndDate date  NOT NULL,
    CONSTRAINT Internships_pk PRIMARY KEY  (InternshipID)
);
ALTER TABLE Internships ADD CONSTRAINT Internships_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

Warunki integralnościowe:

- `Internships_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT Internships_DateIntervalIsValid CHECK
  (StartDate < EndDate)
  ```

4.35 InternshipDetails

Tabela InternshipDetails przechowuje informacje na temat przebiegu stażu dla każdego studenta.

- `StudentID` - ID studenta
- `IntershipID` - ID stażu
- `CompletedAt` - Data zaliczenia stażu, w przypadku gdy student go zaliczył
- `Completed` - Oznaczenie, czy student zaliczył staż
- `CompanyName` - Nazwa firmy oferującej staż
- `City` - Nazwa miasta, w którym zlokalizowana jest firma oferująca staż
- `Country` - Nazwa państwa, w którym zlokalizowana jest firma oferująca staż
- `PostalCode` - Kod pocztowy do adresu, w którym zlokalizowana jest firma oferująca staż
- `Address` - Adres firmy oferującej staż

```
CREATE TABLE InternshipDetails (
    StudentID int  NOT NULL,
    IntershipID int  NOT NULL,
    CompletedAt date  NULL,
    Completed bit  NOT NULL,
    CompanyName nvarchar(500)  NOT NULL,
    City nvarchar(500)  NOT NULL,
    Country nvarchar(500)  NOT NULL,
    PostalCode nvarchar(500)  NOT NULL,
    Address nvarchar(500)  NOT NULL,
    CONSTRAINT InternshipDetails_pk PRIMARY KEY  (IntershipID,StudentID)
);
ALTER TABLE InternshipDetails ADD CONSTRAINT InternshipAttendence_Internships
    FOREIGN KEY (IntershipID)
    REFERENCES Internships (InternshipID);
ALTER TABLE InternshipDetails ADD CONSTRAINT InternshipDetails_Students
    FOREIGN KEY (StudentID)
    REFERENCES Students (StudentID);
```

Warunki integralnościowe:

- `InternshipDetails_CompletedAtIsValid`

  CompletedAt nie może być w przyszłości

```
CONSTRAINT InternshipDetails_CompletedAtIsValid CHECK
(CompletedAt <= GetDate())
```

- `InternshipDetails_PostalCodeIsValid`

  Kod pocztowy ma być w prawidłowym formacie

```
CONSTRAINT InternshipDetails_PostalCodeIsValid CHECK
(PostalCode LIKE '[0-9][0-9]-[0-9][0-9][0-9]'
OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9]'
OR PostalCode LIKE '[0-9][0-9][0-9][0-9][0-9][0-9]')
```

4.36 PublicStudySessions

Tabela PublicStudySessions przechowuje informacje o zajęciach otwartych (tj. takich, w których użytkownik może uczestniczyć bez zapisywania się na studia).

- `PublicStudySessionID` - Unikalny identyfikator zajęć otwartych
- `StudiesSessionID` - Identyfikator zajęć
- `Description` - Opis zajęć otwartych

```
CREATE TABLE PublicStudySessions (
    PublicStudySessionID int  NOT NULL,
    StudiesSessionID int  NOT NULL,
    Description nvarchar(max)  NOT NULL,
    CONSTRAINT PublicStudySessions_ak_1 UNIQUE (StudiesSessionID),
    CONSTRAINT PublicStudySessions_pk PRIMARY KEY  (PublicStudySessionID)
);
ALTER TABLE PublicStudySessions ADD CONSTRAINT PublicStudySessions_Products
    FOREIGN KEY (PublicStudySessionID)
    REFERENCES Products (ProductID);
ALTER TABLE PublicStudySessions ADD CONSTRAINT PublicStudySessions_StudiesSessions
    FOREIGN KEY (StudiesSessionID)
    REFERENCES StudiesSessions (StudiesSessionID)
    ON DELETE  CASCADE;
```

4.37 PublicStudySessionParticipants

Tabela PublicStudySessionParticipants przechowuje informacje o uczestnikach zajęć otwartych.

- `PublicStudySessionParticipantID` - Unikalny identyfikator uczestnika zajęć otwartych
- `UserID` - ID użytkownika
- `PublicStudySessionID` - Identyfikator zajęć, w których uczestnik bierze udział
- `SessionPrice` - Cena uczestnictwa w zajęciach
- `DuePostponedPayment` - Data do której została odroczona płatność
- `FullPricePaymentID` - Identyfikator płatności, w przypadku jej uiszczenia
- `AddedAt` - Data dodania uczestnika zajęć otwartych do bazy

```
CREATE TABLE PublicStudySessionParticipants (
    PublicStudySessionParticipantID int  NOT NULL IDENTITY,
    UserID int  NOT NULL,
    PublicStudySessionID int  NOT NULL,
    SessionPrice money  NOT NULL,
    DuePostponedPayment datetime  NULL,
    FullPricePaymentID int  NULL,
    AddedAt datetime  NOT NULL DEFAULT GETDATE(),
    CONSTRAINT PublicStudySessionParticipants_pk PRIMARY KEY  (PublicStudySessionParticipantID)
);
ALTER TABLE PublicStudySessionParticipants ADD CONSTRAINT PublicStudySessionParticipants_Payments
    FOREIGN KEY (FullPricePaymentID)
```

```
    REFERENCES Payments (PaymentID);
ALTER TABLE PublicStudySessionParticipants ADD CONSTRAINT
↪  PublicStudySessionParticipants_PublicStudySessions
    FOREIGN KEY (PublicStudySessionID)
    REFERENCES PublicStudySessions (PublicStudySessionID);
```

Warunki integralnościowe:

- `PublicStudySessionParticipants_SessionPriceIsValid`

  ```
  CONSTRAINT PublicStudySessionParticipants_SessionPriceIsValid CHECK
  (SessionPrice > 0)
  ```

4.38 PublicStudySessionsAttendanceForOutsiders

Tabela PublicStudySessionsAttendanceForOutsiders przechowuje informacje o obecności użytkowników niezapisanych na studia w zajęciach otwartych.

- `PublicStudySessionID` - Identyfikator zajęć otwartych
- `PublicStudySessionParticipantID` - Identyfikator uczestnika zajęć otwartych
- `Completed` - Oznaczenie, czy użytkownik wziął udział w zajęciach otwartych.

```
CREATE TABLE PublicStudySessionsAttendanceForOutsiders (
    PublicStudySessionID int  NOT NULL,
    PublicStudySessionParticipantID int  NOT NULL,
    Completed bit  NOT NULL,
    CONSTRAINT PublicStudySessionsAttendanceForOutsiders_pk PRIMARY KEY
    ↪  (PublicStudySessionID,PublicStudySessionParticipantID)
);
ALTER TABLE PublicStudySessionsAttendanceForOutsiders ADD CONSTRAINT AttendanceForOutsiders
    FOREIGN KEY (PublicStudySessionParticipantID)
    REFERENCES PublicStudySessionParticipants (PublicStudySessionParticipantID);
ALTER TABLE PublicStudySessionsAttendanceForOutsiders ADD CONSTRAINT
↪  PublicStudySessionsAttendanceForOutsiders_PublicStudySessions
    FOREIGN KEY (PublicStudySessionID)
    REFERENCES PublicStudySessions (PublicStudySessionID);
```

4.39 SubjectMakeUpPossibilities

Tabela SubjectMakeUpPossibilities przechowuje informacje na temat wszystkich możliwych "zastępstw", które student może zrealizować w przypadku nieobecności na zajęciach z danego przedmiotu.

- `SubjectID` - Identyfikator przedmiotu
- `ProductID` - Identyfikator produktu, którego zakup oraz zrealizowanie pozwala odrobić nieobecności
- `AttendanceValue` - Ilość zajęć które odrabia zrealizowanie danego produktu.

```
CREATE TABLE SubjectMakeUpPossibilities (
    SubjectID int  NOT NULL,
    ProductID int  NOT NULL,
    AttendanceValue int  NOT NULL,
    CONSTRAINT SubjectMakeUpPossibilities_pk PRIMARY KEY  (SubjectID,ProductID)
);
ALTER TABLE SubjectMakeUpPossibilities ADD CONSTRAINT SubjectMakeUpPossibilities_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
ALTER TABLE SubjectMakeUpPossibilities ADD CONSTRAINT SubjectMakeUpPossibilities_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES Subjects (SubjectID);
```

Warunki integralnościowe:

- `SubjectMakeUpPossibilities_AttendanceValue`

  Jest to liczba odrobionych zajęć z przedmiotu, zatem musi być większa od zera

```
CONSTRAINT SubjectMakeUpPossibilities_AttendanceValue CHECK
(AttendanceValue > 0)
```

4.40 MadeUpAttendance

W tabeli MadeUpAttendance odnotowywane są wszystkie "zastępstwa" dla studentów odrabiających zajęcia z danego przedmiotu innym produktem.

- `MadeUpAttendanceID` - Unikalny identyfikator zrealizowanego "zastępstwa"
- `SubjectID` - Identyfikator przedmiotu
- `ProductID` - Identyfikator produktu
- `StudentID` - ID studenta

```
CREATE TABLE MadeUpAttendance (
    MadeUpAttendanceID int  NOT NULL IDENTITY,
    SubjectID int  NOT NULL,
    ProductID int  NOT NULL,
    StudentID int  NOT NULL,
    CONSTRAINT MadeUpAttendance_ak_1 UNIQUE (SubjectID, ProductID, StudentID),
    CONSTRAINT MadeUpAttendance_pk PRIMARY KEY  (MadeUpAttendanceID)
);
ALTER TABLE MadeUpAttendance ADD CONSTRAINT MadeUpAttendance_Students
    FOREIGN KEY (StudentID)
    REFERENCES Students (StudentID);
ALTER TABLE MadeUpAttendance ADD CONSTRAINT MadeUpAttendance_SubjectMakeUpPossibilities
    FOREIGN KEY (SubjectID,ProductID)
    REFERENCES SubjectMakeUpPossibilities (SubjectID,ProductID);
```

4.41 DiplomasSent

Tabela DiplomasSent przechowuje informacje o wysłanych dyplomach.

- `DiplomaSentID` - Unikalny identyfikator wysłanego dyplomu.
- `UserID` - Identyfikator użytkownika, któremu dyplom został wysłany.
- `SentAt` - Data wysłania dyplomu.
- `ProductID` - Identyfikator produktu związanego z dyplomem.
- `DiplomaFile` - Ścieżka do pliku dyplomu, jeżeli został załączony.

```
CREATE TABLE DiplomasSent (
    DiplomaSentID int  NOT NULL IDENTITY,
    UserID int  NOT NULL,
    SentAt datetime  NOT NULL DEFAULT GETDATE(),
    ProductID int  NOT NULL,
    DiplomaFile nvarchar(max)  NULL,
    CONSTRAINT DiplomasSent_pk PRIMARY KEY  (DiplomaSentID)
);
ALTER TABLE DiplomasSent ADD CONSTRAINT DiplomasSent_Products
    FOREIGN KEY (ProductID)
    REFERENCES Products (ProductID);
ALTER TABLE DiplomasSent ADD CONSTRAINT DiplomasSent_Users
    FOREIGN KEY (UserID)
    REFERENCES Users (UserID);
```

4.42 Tabele z parameterami

Jest to zestaw tabel które działają na identycznych zasadach. Przechowują one parametry biznesowe. Pozwalają zmieniać minimalne progi obecnośći w celu zaliczenia kursów, studiów, ilość dni w stażu etc.

Wszystkie tabele mają następującą strukturę:

- `EntryId` - id wpisu
- `StartDate` - początkowa data od której obowiązuje dane reguła
- `EndDate` - końcowa data od której reguła traci moc, jeżeli `EndDate` jest `NULL` to reguła obowiązuje do odwołania.
- `IdElementuKtóregoDotyczyReguła` - może to być np. `CourseID` albo `StudiesID`, jeżeli jest `NULL` to wtedy reguła dotyczy wszystkich obiektów(np. kursów lub semestrów studiów). To pole umożliwia wprowadzanie wyjątków. Gdy np. dla pewnego kursu chcemy zmniejszyć próg min. obecności do 50%. Przyjmujemy zasadę, że reguła szczególna uchyla regułę ogólną.

By ułatwić korzystanie z tych tabel stworzono funkcje typu `GetMinAttendancePercentageForStudies` które zwracają odpowiednią wielkość dla danego kursu, semestru studiów etc.

Napisano również triggery, które spełniają rolę warunków integralnościowych, dbają one o to, by nie miała miejsca sytuacja w której dla danego np. kursu mamy dwa obowiązujące progi obecności. Pamiętamy natomiast, że mogą obowiązywać dwie reguły na raz: ogólne i szczegółowa, wtedy szczegółowa uchyla ogólną.

4.42.1 `MinAttendancePercentageToPassInternship`

```
CREATE TABLE MinAttendancePercentageToPassInternship (
    MinAttendancePercentageToPassInternshipID int  NOT NULL IDENTITY,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    AttendancePercentage decimal(6,4)  NOT NULL,
    InternshipID int  NULL,
    CONSTRAINT MinAttendancePercentageToPassInternship_pk PRIMARY KEY
    ↪  (MinAttendancePercentageToPassInternshipID)
);
ALTER TABLE MinAttendancePercentageToPassInternship ADD CONSTRAINT
↪  MinAttendancePercentageToPassInternship_Internships
    FOREIGN KEY (InternshipID)
    REFERENCES Internships (InternshipID);
```

Warunki integralnościowe:

- `MinAttendancePercentageToPassInternship_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT MinAttendancePercentageToPassInternship_DateIntervalIsValid CHECK
  (StartDate < EndDate)
  ```

- `MinAttendancePercentageToPassInternship_PercentageIsValid`

  Procent obecności musi być w przedziale od 0 do 1 włącznie

  ```
  CONSTRAINT MinAttendancePercentageToPassInternship_PercentageIsValid CHECK
  (AttendancePercentage BETWEEN 0 AND 1.0)
  ```

4.42.2 `RecordingAccessTime`

```
CREATE TABLE RecordingAccessTime (
    RecordingAcessTimeID int  NOT NULL IDENTITY,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    NumberOfDays int  NOT NULL,
    WebinarID int  NULL,
```

```
    CONSTRAINT RecordingAccessTime_pk PRIMARY KEY  (RecordingAcessTimeID)
);
ALTER TABLE RecordingAccessTime ADD CONSTRAINT RecordingAccessTime_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES Webinars (WebinarID)
    ON DELETE  CASCADE;
```

Warunki integralnościowe:

- `RecordingAccessTime_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT RecordingAccessTime_DateIntervalIsValid CHECK
  (StartDate < EndDate)
  ```

- `RecordingAccessTime_NumberOfDaysIsValid`

  Liczba dni na którą udostępniamy nagrani musi być większa lub równa 0

  ```
  CONSTRAINT RecordingAccessTime_NumberOfDaysIsValid CHECK
  (NumberOfDays >= 0)
  ```

4.42.3 MaxDaysForPaymentBeforeStudiesStart

```
CREATE TABLE MaxDaysForPaymentBeforeStudiesStart (
    MaxDaysForPaymentBeforeStudiesStartID int  NOT NULL IDENTITY,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    NumberOfDays int  NOT NULL,
    StudiesID int  NULL,
    CONSTRAINT MaxDaysForPaymentBeforeStudiesStart_pk PRIMARY KEY
    ↪  (MaxDaysForPaymentBeforeStudiesStartID)
);
ALTER TABLE MaxDaysForPaymentBeforeStudiesStart ADD CONSTRAINT
↪  MaxDaysForPaymentBeforeStudiesStart_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

Warunki integralnościowe:

- `MaxDaysForPaymentBeforeStudiesStart_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT MaxDaysForPaymentBeforeStudiesStart_DateIntervalIsValid CHECK
  (EndDate > StartDate)
  ```

- `MaxDaysForPaymentBeforeStudiesStart_NumberOfDaysIsValid`

  Liczba dni przed rozpoczęciem musi być większa od 0

  ```
  CONSTRAINT MaxDaysForPaymentBeforeStudiesStart_NumberOfDaysIsValid CHECK
  (NumberOfDays > 0)
  ```

4.42.4 MinAttendancePercentageToPassCourse

```
CREATE TABLE MinAttendancePercentageToPassCourse (
    MinAttendancePercentageToPassCourseID int  NOT NULL IDENTITY,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    AttendancePercentage decimal(6,4)  NOT NULL,
    CourseID int  NULL,
    CONSTRAINT MinAttendancePercentageToPassCourse_pk PRIMARY KEY
    ↪  (MinAttendancePercentageToPassCourseID)
);
ALTER TABLE MinAttendancePercentageToPassCourse ADD CONSTRAINT
↪  MinAttendancePercentageToPassCourse_Courses
    FOREIGN KEY (CourseID)
```

```
    REFERENCES Courses (CourseID);
```

Warunki integralnościowe:

- `MinAttendancePercentageToPassCourse_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT MinAttendancePercentageToPassCourse_DateIntervalIsValid CHECK
  ((StartDate < EndDate))
  ```

- `MinAttendancePercentageToPassCourse_AttendencePercentageIsValid`

  Procent obecności musi być w przedziale od 0 do 1 włącznie

  ```
  CONSTRAINT MinAttendancePercentageToPassCourse_AttendencePercentageIsValid CHECK
  ((AttendancePercentage >= 0) and (AttendancePercentage <= 1))
  ```

4.42.5 DaysInInternship

```
CREATE TABLE DaysInInternship (
    DaysInInternshipID int  NOT NULL IDENTITY,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    NumberOfDays int  NOT NULL,
    InternshipID int  NULL,
    CONSTRAINT DaysInInternship_pk PRIMARY KEY  (DaysInInternshipID)
);
ALTER TABLE DaysInInternship ADD CONSTRAINT DaysOfPracticeLaws_Internships
    FOREIGN KEY (InternshipID)
    REFERENCES Internships (InternshipID);
```

Warunki integralnościowe:

- `DaysInInternship_DateIntervalIsValid`

  EndDate musi być po StartDate

  ```
  CONSTRAINT DaysInInternship_DateIntervalIsValid CHECK
  (StartDate < EndDate)
  ```

- `DaysInInternship_NumberOfDaysIsValid`

  Liczba dni stażu musi być większa od zera

  ```
  CONSTRAINT DaysInInternship_NumberOfDaysIsValid CHECK
  (NumberOfDays > 0)
  ```

4.42.6 MaxDaysForPaymentBeforeCourseStart

```
CREATE TABLE MaxDaysForPaymentBeforeCourseStart (
    MaxDaysForPaymentBeforeCourseStartID int  NOT NULL IDENTITY,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    NumberOfDays int  NOT NULL,
    CourseID int  NULL,
    CONSTRAINT MaxDaysForPaymentBeforeCourseStart_pk PRIMARY KEY
    ↪  (MaxDaysForPaymentBeforeCourseStartID)
);
ALTER TABLE MaxDaysForPaymentBeforeCourseStart ADD CONSTRAINT
↪  MaxDaysForPaymentBeforeCourseStart_Courses
    FOREIGN KEY (CourseID)
    REFERENCES Courses (CourseID);
```

Warunki integralnościowe:

- `MaxDaysForPaymentBeforeCourseStart_DateIntervalIsValid`

  EndDate musi być po StartDate

```
CONSTRAINT MaxDaysForPaymentBeforeCourseStart_DateIntervalIsValid CHECK
(StartDate < EndDate)
```

- `MaxDaysForPaymentBeforeCourseStart_NumberOfDaysIValid`

  Liczba dni przed rozpoczęciem musi być większa od 0

```
CONSTRAINT MaxDaysForPaymentBeforeCourseStart_NumberOfDaysIValid CHECK
(NumberOfDays > 0)
```

4.42.7 MinAttendancePercentageToPassStudies

```
CREATE TABLE MinAttendancePercentageToPassStudies (
    MinAttendancePercentageToPassStudiesID int  NOT NULL IDENTITY,
    StartDate datetime  NOT NULL,
    EndDate datetime  NULL,
    AttendancePercentage decimal(6,4)  NOT NULL,
    StudiesID int  NULL,
    CONSTRAINT MinAttendancePercentageToPassStudies_pk PRIMARY KEY
    ↪  (MinAttendancePercentageToPassStudiesID)
);
ALTER TABLE MinAttendancePercentageToPassStudies ADD CONSTRAINT
↪  MinAttendancePercentageToPassStudies_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES Studies (StudiesID);
```

Warunki integralnościowe:

- `MinAttendancePercentageToPassStudies_DateIntervalIsValid`

  EndDate musi być po StartDate

```
CONSTRAINT MinAttendancePercentageToPassStudies_DateIntervalIsValid CHECK
(StartDate < EndDate)
```

- `MinAttendancePercentageToPassStudies_PercentageIsValid`

  Procent obecności musi być w przedziale od 0 do 1 włącznie

```
CONSTRAINT MinAttendancePercentageToPassStudies_PercentageIsValid CHECK
(AttendancePercentage BETWEEN 0 AND 1)
```

5 Widoki

5.1 ActivityConflicts

Widok prezentujący konflikty terminów zajęć dla użytkowników

```
CREATE OR ALTER VIEW ActivityConflicts AS
WITH AUX AS (
    SELECT WP.UserID, W.StartDate, W.EndDate, 'webinar' as 'ActivityType', W.WebinarID as
    ↪  'ActivityID'
    FROM Webinars W
    JOIN WebinarParticipants WP ON WP.WebinarID = W.WebinarID
    UNION
    SELECT CP.UserID, ISNULL(CSS.StartDate, COS.StartDate), ISNULL(CSS.EndDate, COS.EndDate),
        'CourseSession', CS.CourseSessionID
    FROM CoursesSessions CS
    JOIN Modules M ON M.ModuleID = CS.ModuleID
    JOIN Courses C ON C.CourseID = M.CourseID
    JOIN CourseParticipants CP ON CP.CourseID = C.CourseID
    LEFT JOIN CourseStationarySessions CSS ON CSS.CourseStationarySessionID = CS.CourseSessionID
    LEFT JOIN CourseOnlineSessions COS ON COS.CourseOnlineSessionID = CS.CourseSessionID
    WHERE CSS.CourseStationarySessionID IS NOT NULL OR COS.CourseOnlineSessionID IS NOT NULL
    UNION
    SELECT STU.UserID, SS.StartDate, SS.EndDate, 'StudiesSession', SS.StudiesSessionID
    FROM StudiesSessions SS
    JOIN Subjects S ON S.SubjectID = SS.SubjectID
```

```
    JOIN Studies ST ON ST.StudiesID = S.StudiesID
    JOIN Students STU ON STU.StudiesID = ST.StudiesID
    UNION
    SELECT PSSP.UserID, SS.StartDate, SS.EndDate, 'PublicStudySession',
    ↪  PSSP.PublicStudySessionParticipantID
    FROM PublicStudySessions PSS
    JOIN PublicStudySessionParticipants PSSP ON PSSP.PublicStudySessionID =
    ↪  PSS.PublicStudySessionID
    JOIN StudiesSessions SS ON SS.StudiesSessionID = PSS.StudiesSessionID
)
SELECT
    a1.UserID,
    P.FirstName,
    P.LastName,
    P.Phone,
    P.Email,
    a1.StartDate AS 'Activity_1_Start',
    a1.EndDate AS 'Activity_1_End',
    a1.ActivityType as 'Activity_1_Type',
    a1.ActivityID as 'Activity_1_ID',
    a2.StartDate AS 'Activity_2_Start',
    a2.EndDate AS 'Activity_2_End',
    a2.ActivityType AS 'Activity_2_Type',
    a2.ActivityID AS 'Activity2_ID'
FROM
    AUX a1
JOIN
    AUX a2
ON
    a1.UserID = a2.UserID AND
    a1.StartDate <= a2.EndDate AND
    a1.EndDate >= a2.StartDate
    AND a1.ActivityID <> a2.ActivityID AND a1.ActivityType <> a2.ActivityType
    AND (a1.ActivityType + CAST(a1.ActivityID AS VARCHAR) < a2.ActivityType + CAST(a2.ActivityID
    ↪  AS VARCHAR))
JOIN People P ON P.PersonID = a1.UserID
WHERE
    a1.UserID = a2.UserID;
```

5.2 SchoolOffer

Widok prezentujący bieżącą ofertę edukacyjną (webinary, kursy, studia)

```
CREATE OR ALTER VIEW SchoolOffer AS
SELECT
        W.WebinarID as 'ProductID',
       'Webinar' as'ProductType',
        W.WebinarName as 'ProductName',
        W.Description,
        Price as 'TotalPrice',
        NULL as 'AdvancePayment',
        StartDate,
        EndDate
FROM Webinars W
JOIN Products P ON W.WebinarID = P.ProductID AND P.ClosedAt IS NULL
UNION
SELECT C.CourseID, 'Course', C.CourseName, C.[Description], P.Price, P.AdvancePayment,
↪  C.StartDate, C.EndDate
FROM Courses C
JOIN Products P ON P.ProductID = C.CourseID AND P.ClosedAt IS NULL AND C.StartDate > GETDATE()
UNION
SELECT S.StudiesID, 'Studies', S.Name, S.[Description], P.Price, P.AdvancePayment, S.StartDate,
↪  S.EndDate
FROM Studies S
JOIN Products P ON P.ProductID = S.StudiesID AND P.ClosedAt IS NULL
WHERE S.StartDate > GETDATE()
UNION
SELECT
        PSS.PublicStudySessionID,
```

```
       'Public Study Session',
       'Sesja również dla osób z zewnątrz' + S.[Description],
        S.[Description],
        P.Price,
        P.AdvancePayment,
        SS.StartDate,
        SS.EndDate
FROM PublicStudySessions PSS
JOIN Products P ON P.ProductID = PSS.PublicStudySessionID AND P.ClosedAt IS NULL
JOIN StudiesSessions SS ON SS.StudiesSessionID = PSS.StudiesSessionID
JOIN Subjects S ON S.SubjectID = SS.SubjectID
WHERE SS.StartDate > GETDATE();
```

5.3 EmployeeTimeTable

Widok prezentujący harmonogram pracy pracowników

```
CREATE OR ALTER VIEW EmployeeTimeTable AS
WITH EmployeeData AS (
    SELECT E.EmployeeID, P.FirstName + ' ' + P.LastName as 'FullName'
    FROM Employees E
    JOIN People P ON P.PersonID = E.EmployeeID
)
SELECT 'Stationary Course Session' as 'Session Type',
        CSS.CourseStationarySessionID 'SessionID',
        ED.EmployeeID as 'EmployeeID',
        ED.FullName as 'FullName',
        CSS.StartDate, CSS.EndDate
FROM CourseStationarySessions CSS
JOIN CoursesSessions CS ON CS.CourseSessionID = CSS.CourseStationarySessionID
JOIN EmployeeData ED ON ED.EmployeeID = CS.LecturerID
UNION
SELECT 'Online Course Session',
        COS.CourseOnlineSessionID,
        ED.EmployeeID,
        ED.FullName,
        COS.StartDate,
        COS.EndDate
FROM CourseOnlineSessions COS
JOIN CoursesSessions CS ON CS.CourseSessionID = COS.CourseOnlineSessionID
JOIN EmployeeData ED ON ED.EmployeeID = CS.LecturerID
UNION
SELECT 'Webinar', W.WebinarID, ED.EmployeeID, ED.FullName, W.StartDate, W.EndDate
FROM Webinars W
JOIN EmployeeData ED ON ED.EmployeeID = W.LecturerID
UNION
SELECT 'Studies Session',
        SS.StudiesSessionID,
        ED.EmployeeID,
        ED.FullName,
        SS.StartDate,
        SS.EndDate
FROM StudiesSessions SS
JOIN EmployeeData ED ON ED.EmployeeID = SS.LecturerID;
```

5.4 EmployeeStatistics

Widok prezentujący statystyki dotyczące aktywności pracowników

```
CREATE OR ALTER VIEW EmployeeStatistics AS
SELECT
    E.EmployeeID,
    P.FirstName,
    P.LastName,
    (
        SELECT COUNT(*)
        FROM Webinars W
        WHERE W.LecturerID = E.EmployeeID
```

```
    ) as 'WebinarsConducted',
    (
        SELECT COUNT(*)
        FROM Courses C
        WHERE C.CoordinatorID = E.EmployeeID
    ) as 'CoursesCoordinated',
    (
        SELECT COUNT(*)
        FROM Studies S
        WHERE S.CoordinatorID = E.EmployeeID
    ) as 'StudiesCoordinated',
    (
        SELECT COUNT(*)
        FROM StudiesSessions SS
        WHERE SS.LecturerID = E.EmployeeID AND SS.EndDate >  GETDATE()
    ) as 'StudiesSessionsConducted',
    (
        SELECT COUNT(COS.CourseOnlineSessionID) + COUNT(CSS.CourseStationarySessionID)
        FROM CoursesSessions CS
        LEFT JOIN CourseOnlineSessions COS ON
             COS.CourseOnlineSessionID = CS.CourseSessionID AND COS.StartDate < GETDATE()
        LEFT JOIN CourseStationarySessions CSS ON
             CSS.CourseStationarySessionID = CS.CourseSessionID AND CSS.EndDate < GETDATE()
        WHERE CS.LecturerID = E.EmployeeID
    ) as 'CourseSessionsConducted'

FROM Employees E
JOIN People P ON P.PersonID = E.EmployeeID;
```

5.5 TotalIncomeForProducts

Widok prezentujący łączne przychody z różnych produktów edukacyjnych

```
CREATE OR ALTER VIEW TotalIncomeForProducts AS
WITH ProductsIncome AS (
    SELECT
        W.WebinarID AS 'ProductID',
    'Webinar' AS 'ProductType',
        W.WebinarName AS 'ProductName',
        W.Description,
        W.StartDate AS 'Date',
        W.LecturerID AS 'MainEmployeeId'
    FROM Webinars W

    UNION

    SELECT
        C.CourseID,
    'Course',
        C.CourseName,
        C.Description,
        C.StartDate,
        C.CoordinatorID
    FROM Courses C

    UNION

    SELECT
        S.StudiesID,
    'Studies',
        S.Name,
        S.Description,
        CONVERT(datetime, S.StartDate) AS 'Date',
        S.CoordinatorID
    FROM Studies S

  UNION

  SELECT
```

```
    P.PublicStudySessionID,
    'Public study session',
    S.SubjectName,
    P.Description,
    SS.StartDate,
    SS.LecturerID
  FROM PublicStudySessions P
  JOIN StudiesSessions SS ON SS.StudiesSessionID = P.StudiesSessionID
  JOIN Subjects S ON SS.SubjectID = S.SubjectID
)
SELECT
    PI.ProductID,
  PI.ProductType,
    PI.ProductName,
  ISNULL(SUM(P.Price),0) as 'Income',
    PI.Description,
    PI.Date,
    PP.FirstName + ' ' + PP.LastName AS 'MainEmployee'
FROM ProductsIncome PI
JOIN Employees E ON E.EmployeeID = PI.MainEmployeeId
JOIN People PP ON PP.PersonID = E.EmployeeID
LEFT JOIN Payments P ON P.ProductID = PI.ProductID AND P.Status='Successful'
GROUP BY PI.ProductID, PI.ProductType, PI.ProductName, PI.Description, PI.Date, PP.FirstName + '
↪  ' + PP.LastName
HAVING PI.Date < GETDATE();
```

5.6 RevenueSummaryByProductType

Widok prezentujący miesięczne i roczne podsumowanie przychodów według typów produktów

```
CREATE OR ALTER VIEW RevenueSummaryByProductType AS
SELECT
    ISNULL(CAST(YEAR(P.Date) AS NVARCHAR(4)), 'Total') AS RevenueYear,
    CASE
        WHEN MONTH(P.Date) IS NULL THEN 'Total'
        ELSE CAST(MONTH(P.Date) AS NVARCHAR(2))
    END AS RevenueMonth,
    COALESCE(Pr.ProductType, 'All Types') AS ProductType,
    SUM(P.Price) AS TotalRevenue
FROM
    Payments P
INNER JOIN
    Products Pr ON P.ProductID = Pr.ProductID
WHERE
    P.Status = 'Successful'
GROUP BY
    YEAR(P.Date),
    MONTH(P.Date),
    ROLLUP(Pr.ProductType);
```

5.7 TimeTableForAllUsers

Widok prezentujący harmonogram zajęć dla wszystkich użytkowników

```
CREATE OR ALTER VIEW TimeTableForAllUsers AS
SELECT
    S.UserID,
    'studies session' as 'type',
    SS.StartDate as 'StartDate',
    SS.EndDate as 'EndDate',
    P.FirstName + ' ' + P.LastName as 'Lecturer',
    CASE
        WHEN SSS.StationaryStudiesSessionID IS NULL THEN 'online'
        ELSE SSS.Country + ' ' + SSS.PostalCode + ' ' + SSS.City + ' ' + SSS.Address + ' ' +
        ↪  SSS.ClassroomNumber
    END as 'Place'
FROM Students S
JOIN Studies ON Studies.StudiesID = S.StudiesID
```

```
JOIN Subjects ON Subjects.StudiesID = Studies.StudiesID
JOIN StudiesSessions SS ON SS.SubjectID = Subjects.SubjectID
JOIN Employees E ON E.EmployeeID = SS.LecturerID
JOIN People P ON E.EmployeeID = P.PersonID
LEFT JOIN StationaryStudiesSessions SSS ON SSS.StationaryStudiesSessionID = SS.StudiesSessionID
LEFT JOIN OnlineStudiesSessions OSS ON OSS.OnlineStudiesSessionID = SS.StudiesSessionID
UNION
SELECT
    Wp.UserID,
    'webinar',
    W.StartDate,
    W.EndDate,
    P.FirstName + ' ' + P.LastName,
    'online'
FROM Webinars W
JOIN WebinarParticipants WP ON WP.WebinarID = W.WebinarID
JOIN Employees E ON W.LecturerID = E.EmployeeID
JOIN People P ON P.PersonID = E.EmployeeID
UNION
SELECT
    CP.UserID,
    'course session',
    ISNULL(CSS.StartDate, CONS.StartDate),
    ISNULL(CSS.EndDate, CONS.EndDate),
    P.FirstName + ' ' + P.LastName,
    CASE
        WHEN CSS.CourseStationarySessionID IS NOT NULL THEN
            CSS.Country + ' ' + CSS.PostalCode + ' ' + CSS.City + ' ' + CSS.Address + ' ' +
            ↪  CSS.ClassroomNumber
        ELSE 'online'
    END
FROM CourseParticipants CP
JOIN Courses C ON C.CourseID = CP.CourseID
JOIN Modules M ON M.CourseID = C.CourseID
JOIN CoursesSessions CS ON CS.ModuleID = M.ModuleID
JOIN Employees E ON E.EmployeeID = CS.LecturerID
JOIN People P ON P.PersonID = E.EmployeeID
LEFT JOIN CourseStationarySessions CSS ON CSS.CourseStationarySessionID = CS.CourseSessionID
LEFT JOIN CourseOnlineSessions CONS ON CONS.CourseOnlineSessionID = CS.CourseSessionID
WHERE CSS.CourseStationarySessionID IS NOT NULL OR CONS.CourseOnlineSessionID IS NOT NULL
UNION
SELECT
    PSSP.UserID,
    'public study session',
    SS.StartDate as 'StartDate',
    SS.EndDate as 'EndDate',
    P.FirstName + ' ' + P.LastName,
    CASE
        WHEN SSS.StationaryStudiesSessionID IS NULL THEN 'online'
        ELSE SSS.Country + ' ' + SSS.PostalCode + ' ' + SSS.City + ' ' + SSS.Address + ' ' +
        ↪  SSS.ClassroomNumber
    END as 'Place'
FROM PublicStudySessionParticipants PSSP
JOIN PublicStudySessions PSS ON PSS.PublicStudySessionID = PSSP.PublicStudySessionID
JOIN StudiesSessions SS ON SS.StudiesSessionID = PSS.StudiesSessionID
LEFT JOIN StationaryStudiesSessions SSS ON SSS.StationaryStudiesSessionID = SS.StudiesSessionID
LEFT JOIN OnlineStudiesSessions OSS ON OSS.OnlineStudiesSessionID = SS.StudiesSessionID
JOIN Employees E ON E.EmployeeID = SS.LecturerID
JOIN People P ON P.PersonID = E.EmployeeID;
```

5.8 Loaners

Widok prezentujący listę dłużników

```
CREATE OR ALTER VIEW Loaners AS
WITH UserDetails AS (
    SELECT
        U.UserID,
        P.FirstName + ' ' + P.LastName AS 'FullName',
```

```
        P.Email,
        P.Phone
    FROM
        Users U
    JOIN People P ON P.PersonID = U.UserID
)
SELECT
    UD.UserID,
    UD.FullName,
    UD.Email,
    UD.Phone,
    WP.WebinarID AS 'ProductIDToPay',
    'Webinar' AS 'ProductType',
    W.WebinarName AS 'ProductName',
    WP.WebinarPrice AS 'LoanAmount',
    WP.DuePostponedPayment AS 'PaymentDue'
FROM
    WebinarParticipants WP
JOIN UserDetails UD ON UD.UserID = WP.UserID
JOIN Webinars W ON W.WebinarID = WP.WebinarID
WHERE
    WP.DuePostponedPayment IS NOT NULL AND WP.FullPricePaymentID IS NULL

UNION

SELECT
    UD.UserID,
    UD.FullName,
    UD.Email,
    UD.Phone,
    CP.CourseID AS 'ProductIDToPay',
    'Course' AS 'ProductType',
    C.CourseName AS 'ProductName',
    CP.CoursePrice AS 'LoanAmount',
    CP.DuePostponedPayment AS 'PaymentDue'
FROM
    CourseParticipants CP
JOIN UserDetails UD ON UD.UserID = CP.UserID
JOIN Courses C ON C.CourseID = CP.CourseID
WHERE
    CP.DuePostponedPayment IS NOT NULL AND CP.FullPricePaymentID IS NULL AND CP.EntryFeePaymentID
    ↪ IS NULL AND CP.RemainingPaymentID IS NULL

UNION

SELECT
    UD.UserID,
    UD.FullName,
    UD.Email,
    UD.Phone,
    S.StudiesID AS 'ProductIDToPay',
    'Studies' AS 'ProductType',
    SS.Name AS 'ProductName',
    S.StudiesPrice - ISNULL(EFP.Price, 0) - ISNULL(RPP.Price, 0) AS 'LoanAmount',
    S.DuePostponedPayment AS 'PaymentDue'
FROM
    Students S
JOIN UserDetails UD ON UD.UserID = S.UserID
JOIN Studies SS ON SS.StudiesID = S.StudiesID
LEFT JOIN Payments EFP ON EFP.PaymentID = S.EntryFeePaymentID
LEFT JOIN Payments RPP ON RPP.PaymentID = S.RemainingPaymentID
WHERE
    S.DuePostponedPayment IS NOT NULL AND (S.EntryFeePaymentID IS NULL OR S.RemainingPaymentID IS
    ↪ NULL)

UNION

SELECT
    UD.UserID,
```

```
    UD.FullName,
    UD.Email,
    UD.Phone,
    P.PublicStudySessionID AS 'ProductIDToPay',
    'Public Study Session' AS 'ProductType',
    S.SubjectName AS 'ProductName',
    P.SessionPrice AS 'LoanAmount',
    P.DuePostponedPayment AS 'PaymentDue'
FROM
    PublicStudySessionParticipants P
JOIN UserDetails UD ON UD.UserID = P.UserID
JOIN PublicStudySessions PS ON PS.PublicStudySessionID = P.PublicStudySessionID
JOIN StudiesSessions SS ON SS.StudiesSessionID = PS.StudiesSessionID
JOIN Subjects S ON S.SubjectID = SS.SubjectID
WHERE
    P.DuePostponedPayment IS NOT NULL AND P.FullPricePaymentID IS NULL;
```

5.9 AttendanceListForEachSession

Widok prezentujący frekwencje na zajęciach

```
CREATE OR ALTER VIEW AttendanceListForEachSession AS
WITH SessionsAttendance AS
(SELECT
  'Studies Session' as 'SessionType',
  S.UserID,
  SS.StudiesSessionID as 'SessionID',
  SS.StartDate,
  SS.EndDate,
  SA.Completed
FROM StudiesSessions SS
JOIN StudiesSessionsAttendence SA ON SA.SessionID = SS.StudiesSessionID
JOIN Students S ON S.StudentID = SA.StudentID
JOIN Subjects SUB ON SUB.SubjectID = SS.SubjectID
WHERE SS.EndDate < GETDATE()
UNION
SELECT
  'Public Study Session',
  PSP.UserID,
  PSP.PublicStudySessionID,
  SS.StartDate,
  SS.EndDate,
  PA.Completed
FROM PublicStudySessions PS
JOIN StudiesSessions SS ON SS.StudiesSessionID = PS.StudiesSessionID
JOIN PublicStudySessionsAttendanceForOutsiders PA
  ON PA.PublicStudySessionID=PS.PublicStudySessionID
JOIN PublicStudySessionParticipants PSP
  ON PSP.PublicStudySessionParticipantID = PA.PublicStudySessionParticipantID
UNION
SELECT
  'Course Offline Session',
  CP.UserID,
  CS.CourseOfflineSessionID,
  CS.UploadedAt,
  NULL,
  CA.Completed
FROM CourseOfflineSessions CS
JOIN CourseSessionsAttendance CA ON CA.CourseSessionID = CS.CourseOfflineSessionID
JOIN CourseParticipants CP ON CP.CourseParticipantID = CA.CourseParticipantID
UNION
SELECT
  'Course Online Session',
  CP.UserID,
  CS.CourseOnlineSessionID,
  CS.StartDate,
  CS.EndDate,
  CA.Completed
FROM CourseOnlineSessions CS
```

```
JOIN CourseSessionsAttendance CA ON CA.CourseSessionID = CS.CourseOnlineSessionID
JOIN CourseParticipants CP ON CP.CourseParticipantID = CA.CourseParticipantID
UNION
SELECT
  'Course Stationary Session',
  CP.UserID,
  CS.CourseStationarySessionID,
  CS.StartDate,
  CS.EndDate,
  CA.Completed
FROM CourseStationarySessions CS
JOIN CourseSessionsAttendance CA ON CA.CourseSessionID = CS.CourseStationarySessionID
JOIN CourseParticipants CP ON CP.CourseParticipantID = CA.CourseParticipantID
UNION
SELECT
  'Webinar',
  WP.UserID,
  W.WebinarID,
  W.StartDate,
  W.EndDate,
  WA.WasPresent
FROM Webinars W
JOIN WebinarsAttendence WA ON WA.WebinarID = W.WebinarID
JOIN WebinarParticipants WP ON WP.WebinarParticipantID = WA.WebinarParticipantID)
SELECT
  P.FirstName,
  P.LastName,
  SA.UserID,
  SA.SessionType,
  SA.SessionID,
  SA.StartDate,
  SA.EndDate,
  SA.Completed
FROM SessionsAttendance SA
JOIN People P ON SA.UserID = P.PersonID;
```

5.10 GeneralAttendance

Widok prezentujący ogólne statystyki frekwencji na zajęciach

```
CREATE OR ALTER VIEW GeneralAttendance AS
WITH StudiesSessionsInfo AS (
  SELECT
    SS.StudiesSessionID,
    CASE
      WHEN PS.StudiesSessionID IS NULL THEN 'Study Session'
      ELSE 'Study Session & Public'
    END as 'type',
    S.SubjectName + ' ' + CAST(SS.StudiesSessionID as VARCHAR(10)) as SessionName,

    (
      SELECT COUNT(SU.StudentID)
      FROM Subjects SUB
      JOIN Studies ST ON  SUB.StudiesID = ST.StudiesID
      JOIN Students SU ON SU.StudiesID = ST.StudiesID
      WHERE SUB.SubjectID = S.SubjectID
    ) + (
      SELECT COUNT(*)
      FROM PublicStudySessionParticipants PS2
      WHERE PS2.PublicStudySessionID = PS.PublicStudySessionID
    ) as 'PeopleEnlisted',
    (
      SELECT COUNT(*)
      FROM StudiesSessionsAttendence SA
      WHERE SA.SessionID = SS.StudiesSessionID AND Completed=1
    ) + (
      SELECT COUNT(*)
      FROM PublicStudySessionsAttendanceForOutsiders PSA
      WHERE PSA.PublicStudySessionID = PS.PublicStudySessionID AND Completed=1
```

```
    ) as 'NumberOfPeoplePresent'
  FROM StudiesSessions SS
  LEFT JOIN PublicStudySessions PS ON PS.StudiesSessionID = SS.StudiesSessionID
  JOIN Subjects S ON S.SubjectID = SS.SubjectID
  WHERE SS.EndDate < GETDATE()
)
SELECT 'Webinar' as Type, Web.WebinarName as SessionName, COUNT(*) as PeopleEnlisted,
↪  SUM(CAST(W.WasPresent as INT)) as NumOfPeoplePresent, ROUND(100*SUM(CAST(W.WasPresent as
↪  float))/COUNT(*), 2) as Percentage
FROM WebinarsAttendence As W
INNER JOIN Webinars as Web on Web.WebinarID = W.WebinarID
WHERE Web.EndDate < GETDATE()
GROUP BY W.WebinarID, Web.WebinarName
UNION
SELECT 'Course Session' as Type, M.ModuleName as SessionName, COUNT(*) as PeopleEnlisted,
↪  SUM(CAST(C.Completed as INT)) as NumOfPeoplePresent, ROUND(100*SUM(CAST(C.Completed as
↪  float))/COUNT(*), 2) as Percentage
FROM CourseSessionsAttendance As C
INNER JOIN CoursesSessions as CS on CS.CourseSessionID = C.CourseSessionID
INNER JOIN Modules as M on CS.ModuleID = M.ModuleID
GROUP BY C.CourseSessionID, M.ModuleName
UNION
SELECT I.type,I.SessionName,I.PeopleEnlisted,I.NumberOfPeoplePresent,
  CASE
  WHEN I.PeopleEnlisted > 0 THEN
    ROUND(100*(CAST(I.NumberOfPeoplePresent AS FLOAT))/ CAST(I.PeopleEnlisted AS FLOAT), 2)
  ELSE
    0
  END
FROM StudiesSessionsInfo I;
```

5.11 NumberOfPeopleRegisteredForEvents

Widok prezentujący liste osób zapisanych na przyszłe wydarzenia

```
CREATE OR ALTER VIEW NumberOfPeopleRegisteredForEvents AS
SELECT CSS.StartDate as'StartDate',
     CSS.EndDate,
       'Stationary' as 'StationaryOrOnline',
       'course session' as 'Type',
       P.FirstName + ' ' + P.LastName as 'Lecturer',
       COUNT(CP.CourseParticipantID) as 'PeopleRegistered'
FROM CourseStationarySessions CSS
JOIN CoursesSessions CS ON CS.CourseSessionID=CSS.CourseStationarySessionID
JOIN Employees E ON E.EmployeeID = CS.LecturerID
JOIN People P ON P.PersonID = E.EmployeeID
JOIN Modules M ON M.ModuleID = CS.ModuleID
JOIN Courses C ON C.CourseID = M.CourseID
JOIN CourseParticipants CP ON CP.CourseID = C.CourseID
WHERE CSS.StartDate > GETDATE()
GROUP BY CSS.StartDate,CSS.EndDate,  P.FirstName + ' ' + P.LastName
UNION
SELECT COS.StartDate,
     COS.EndDate,
       'Online',
       'course session',
       P.FirstName + ' ' + P.LastName,
       COUNT(CP.CourseParticipantID)
FROM CourseOnlineSessions COS
JOIN CoursesSessions CS ON CS.CourseSessionID=COS.CourseOnlineSessionID
JOIN Employees E ON E.EmployeeID = CS.LecturerID
JOIN People P ON P.PersonID = E.EmployeeID
JOIN Modules M ON M.ModuleID = CS.ModuleID
JOIN Courses C ON C.CourseID = M.CourseID
JOIN CourseParticipants CP ON CP.CourseID = C.CourseID
WHERE COS.StartDate > GETDATE()
GROUP BY COS.CourseOnlineSessionID, COS.StartDate,COS.EndDate, P.FirstName + ' ' + P.LastName
UNION
SELECT W.StartDate,
```

```
     W.EndDate,
        'Online',
        'webinar',
        P.FirstName + ' ' + P.LastName,
        COUNT(WP.WebinarParticipantID)
FROM Webinars W
JOIN WebinarParticipants WP ON WP.WebinarID = W.WebinarID
JOIN Employees E ON E.EmployeeID = W.LecturerID
JOIN People P ON P.PersonID = E.EmployeeID
WHERE W.EndDate > GETDATE()
GROUP BY W.WebinarID, W.StartDate, W.EndDate, P.FirstName + ' ' + P.LastName
UNION
SELECT SS.StartDate, SS.EndDate,
    CASE
        WHEN OSS.OnlineStudiesSessionID  IS NULL THEN 'Stationary'
        ELSE 'Online'
    END,
    'studies sessions',
    P.FirstName + ' ' + P.LastName
    'studies session',
    COUNT(Students.StudentID) + COUNT(SSS.StationaryStudiesSessionID)
FROM StudiesSessions SS
LEFT JOIN PublicStudySessions PSS ON PSS.StudiesSessionID = SS.StudiesSessionID
LEFT JOIN PublicStudySessionParticipants PSSP ON PSSP.PublicStudySessionID =
↪  PSS.PublicStudySessionID
LEFT JOIN OnlineStudiesSessions OSS ON OSS.OnlineStudiesSessionID = SS.StudiesSessionID
LEFT JOIN StationaryStudiesSessions SSS ON SSS.StationaryStudiesSessionID = SS.StudiesSessionID
JOIN Employees E ON E.EmployeeID = SS.LecturerID
JOIN People P ON E.EmployeeID = P.PersonID
JOIN Subjects S ON S.SubjectID = SS.SubjectID
JOIN Studies ON Studies.StudiesID = S.StudiesID
JOIN Students ON Students.StudiesID = Studies.StudiesID
WHERE SS.EndDate > GETDATE()
GROUP BY SS.StudiesSessionID, SS.StartDate,SS.EndDate, SS.StudiesSessionID,
↪  OSS.OnlineStudiesSessionID, P.FirstName + ' ' + P.LastName;
```

6 Funkcje i procedury

6.1 dbo.GetRecordingAccessDays

Funkcja pobierająca liczbę dni dostępu do nagrań webinarów.

```
CREATE OR ALTER FUNCTION dbo.GetRecordingAccessDays(@WebinarID INT)
RETURNS INT
AS
BEGIN
    DECLARE @NumberOfDays INT;

    -- Try to find a specific rule for the given WebinarID
    SELECT @NumberOfDays = NumberOfDays
    FROM RecordingAccessTime
    WHERE WebinarID = @WebinarID
          AND StartDate <= GETDATE()
          AND (EndDate IS NULL OR EndDate > GETDATE());

    -- If no specific rule found, look for a general rule
    IF @NumberOfDays IS NULL
    BEGIN
        SELECT @NumberOfDays = NumberOfDays
        FROM RecordingAccessTime
        WHERE WebinarID IS NULL
              AND StartDate <= GETDATE()
              AND (EndDate IS NULL OR EndDate > GETDATE());
    END

    -- Return the number of days
    RETURN @NumberOfDays;
END;
```

```
GO
```

6.2 dbo.GetMinAttendancePercentageForCourse

Funkcja pobierająca minimalny procent obecności wymagany do zaliczenia kursu.

```
CREATE OR ALTER FUNCTION dbo.GetMinAttendancePercentageForCourse(@CourseID INT)
RETURNS DECIMAL(6, 4)
AS
BEGIN
    DECLARE @MinAttendancePercentage DECIMAL(6, 4);

    -- Try to find a specific rule for the given CourseID
    SELECT @MinAttendancePercentage = AttendancePercentage
    FROM MinAttendancePercentageToPassCourse
    WHERE CourseID = @CourseID
          AND StartDate <= GETDATE()
          AND (EndDate IS NULL OR EndDate > GETDATE());

    -- If no specific rule found, look for a general rule (CourseID is NULL)
    IF @MinAttendancePercentage IS NULL
    BEGIN
        SELECT @MinAttendancePercentage = AttendancePercentage
        FROM MinAttendancePercentageToPassCourse
        WHERE CourseID IS NULL
              AND StartDate <= GETDATE()
              AND (EndDate IS NULL OR EndDate > GETDATE());
    END

    -- Return the minimum attendance percentage
    RETURN @MinAttendancePercentage;
END;
GO
```

6.3 dbo.GetMinAttendancePercentageForInternship

Funkcja pobierająca minimalny procent obecności wymagany do zaliczenia stażu.

```
CREATE OR ALTER FUNCTION dbo.GetMinAttendancePercentageForInternship(@InternshipID INT)
RETURNS DECIMAL(6, 4)
AS
BEGIN
    DECLARE @MinAttendancePercentage DECIMAL(6, 4);

    -- Try to find a specific rule for the given InternshipID
    SELECT @MinAttendancePercentage = AttendancePercentage
    FROM MinAttendancePercentageToPassInternship
    WHERE InternshipID = @InternshipID
          AND StartDate <= GETDATE()
          AND (EndDate IS NULL OR EndDate > GETDATE());

    -- If no specific rule found, look for a general rule (InternshipID is NULL)
    IF @MinAttendancePercentage IS NULL
    BEGIN
        SELECT @MinAttendancePercentage = AttendancePercentage
        FROM MinAttendancePercentageToPassInternship
        WHERE InternshipID IS NULL
              AND StartDate <= GETDATE()
              AND (EndDate IS NULL OR EndDate > GETDATE());
    END

    -- Return the minimum attendance percentage
    RETURN @MinAttendancePercentage;
END;
GO
```

6.4 dbo.GetMinAttendancePercentageForStudies

Funkcja pobierająca minimalny procent obecności wymagany do zaliczenia studiów.

```
CREATE OR ALTER FUNCTION dbo.GetMinAttendancePercentageForStudies(@StudiesID INT)
RETURNS DECIMAL(6, 4)
AS
BEGIN
    DECLARE @MinAttendancePercentage DECIMAL(6, 4);

    -- Try to find a specific rule for the given StudiesID
    SELECT @MinAttendancePercentage = AttendancePercentage
    FROM MinAttendancePercentageToPassStudies
    WHERE StudiesID = @StudiesID
          AND StartDate <= GETDATE()
          AND (EndDate IS NULL OR EndDate > GETDATE());

    -- If no specific rule found, look for a general rule (StudiesID is NULL)
    IF @MinAttendancePercentage IS NULL
    BEGIN
        SELECT @MinAttendancePercentage = AttendancePercentage
        FROM MinAttendancePercentageToPassStudies
        WHERE StudiesID IS NULL
              AND StartDate <= GETDATE()
              AND (EndDate IS NULL OR EndDate > GETDATE());
    END

    -- Return the minimum attendance percentage
    RETURN @MinAttendancePercentage;
END;
GO
```

6.5 dbo.GetMaxDaysForPaymentBeforeCourseStart

Funkcja pobierająca maksymalną liczbę dni na dokonanie płatności przed rozpoczęciem kursu.

```
CREATE OR ALTER FUNCTION dbo.GetMaxDaysForPaymentBeforeCourseStart(@CourseID INT)
RETURNS INT
AS
BEGIN
    DECLARE @MaxDaysForPayment INT;

    -- Try to find a specific rule for the given CourseID
    SELECT @MaxDaysForPayment = NumberOfDays
    FROM MaxDaysForPaymentBeforeCourseStart
    WHERE CourseID = @CourseID
          AND StartDate <= GETDATE()
          AND (EndDate IS NULL OR EndDate > GETDATE());

    -- If no specific rule found, look for a general rule (CourseID is NULL)
    IF @MaxDaysForPayment IS NULL
    BEGIN
        SELECT @MaxDaysForPayment = NumberOfDays
        FROM MaxDaysForPaymentBeforeCourseStart
        WHERE CourseID IS NULL
              AND StartDate <= GETDATE()
              AND (EndDate IS NULL OR EndDate > GETDATE());
    END

    -- Return the maximum days for payment
    RETURN @MaxDaysForPayment;
END;
GO
```

6.6 dbo.GetMaxDaysForPaymentBeforeStudiesStart

Funkcja pobierająca maksymalną liczbę dni na dokonanie płatności przed rozpoczęciem studiów.

```sql
CREATE OR ALTER FUNCTION dbo.GetMaxDaysForPaymentBeforeStudiesStart(@StudiesID INT)
RETURNS INT
AS
BEGIN
    DECLARE @MaxDaysForPayment INT;

    -- Try to find a specific rule for the given StudiesID
    SELECT @MaxDaysForPayment = NumberOfDays
    FROM MaxDaysForPaymentBeforeStudiesStart
    WHERE StudiesID = @StudiesID
          AND StartDate <= GETDATE()
          AND (EndDate IS NULL OR EndDate > GETDATE());

    -- If no specific rule found, look for a general rule (StudiesID is NULL)
    IF @MaxDaysForPayment IS NULL
    BEGIN
        SELECT @MaxDaysForPayment = NumberOfDays
        FROM MaxDaysForPaymentBeforeStudiesStart
        WHERE StudiesID IS NULL
              AND StartDate <= GETDATE()
              AND (EndDate IS NULL OR EndDate > GETDATE());
    END

    -- Return the maximum days for payment
    RETURN @MaxDaysForPayment;
END;
GO
```

6.7 dbo.GetDaysInInternship

Funkcja pobierająca liczbę dni trwania stażu.

```sql
CREATE OR ALTER FUNCTION dbo.GetDaysInInternship(@InternshipID INT)
RETURNS INT
AS
BEGIN
    DECLARE @DaysInInternship INT;

    -- Try to find a specific rule for the given InternshipID
    SELECT @DaysInInternship = NumberOfDays
    FROM DaysInInternship
    WHERE InternshipID = @InternshipID
          AND StartDate <= GETDATE()
          AND (EndDate IS NULL OR EndDate > GETDATE());

    -- If no specific rule found, look for a general rule (InternshipID is NULL)
    IF @DaysInInternship IS NULL
    BEGIN
        SELECT @DaysInInternship = NumberOfDays
        FROM DaysInInternship
        WHERE InternshipID IS NULL
              AND StartDate <= GETDATE()
              AND (EndDate IS NULL OR EndDate > GETDATE());
    END

    -- Return the number of days in the internship
    RETURN @DaysInInternship;
END;
GO
```

6.8 AddFieldOfStudy

Procedura do dodawania nowego kierunku studiów.

```sql
CREATE OR ALTER PROCEDURE AddFieldOfStudy
  @Name NVARCHAR(MAX),
  @Description NVARCHAR(MAX)
AS
```

```
BEGIN
  SET NOCOUNT ON;

  -- Add a new field of study
  INSERT INTO FieldsOfStudies (Name, Description)
  VALUES (@Name, @Description);
  RETURN 0;
END;
GO
```

6.9 DeleteFieldOfStudies

Procedura do usuwania kierunku studiów.

```
CREATE OR ALTER PROCEDURE DeleteFieldOfStudies
  @FieldOfStudiesID INT
AS
BEGIN
  SET NOCOUNT ON;

-- Check if there are studies in the given field of study
IF NOT EXISTS (SELECT 1 FROM Studies WHERE FieldOfStudiesID = @FieldOfStudiesID)
  BEGIN
      BEGIN TRANSACTION;
      -- Usuń kierunek studiów
      DELETE FROM FieldsOfStudies WHERE FieldOfStudiesID = @FieldOfStudiesID;
      COMMIT;
      RETURN 0;
  END
  ELSE
  BEGIN
      PRINT('Field of studies cannot be removed because there are studies in it.')
  END
  RETURN 1;
END;
GO
```

6.10 CreateSemesterOfStudies

Procedura do tworzenia semestru studiów.

```
CREATE OR ALTER PROCEDURE CreateSemesterOfStudies
    @Price money,
    @AdvancePayment money,
    @Name nvarchar(max),
    @Description nvarchar(max),
    @CoordinatorID int,
    @StartDate date,
    @EndDate date,
    @MaxStudents int,
    @LanguageID int,
    @FieldOfStudiesID int,
    @SemesterNumber int
AS
BEGIN
    -- Insert into Products table with 'studies' as the ProductType
    INSERT INTO Products (Price, AdvancePayment, ProductType)
    VALUES (@Price, @AdvancePayment, 'studies')

    -- Capture the newly inserted ProductID
    DECLARE @NewProductID int = SCOPE_IDENTITY()

    -- Insert into Studies table
    INSERT INTO Studies (StudiesID, Name, Description, CoordinatorID, StartDate, EndDate,
    ↪  MaxStudents, LanguageID, FieldOfStudiesID, SemesterNumber)
    VALUES (@NewProductID, @Name, @Description, @CoordinatorID, @StartDate, @EndDate,
    ↪  @MaxStudents, @LanguageID, @FieldOfStudiesID, @SemesterNumber)
END;
```

```
GO
```

6.11 ModifyStudies

Procedura do modyfikacji danych studiów.

```
CREATE OR ALTER PROCEDURE ModifyStudies
  @StudiesID int,
  @Name nvarchar(max) = NULL,
  @Description nvarchar(max) = NULL,
  @CoordinatorID int = NULL,
  @StartDate Date = NULL,
  @EndDate Date = NULL,
  @MaxStudents int = NULL,
  @LanguageID int = NULL,
  @FieldOfStudiesID int = NULL,
  @SemesterNumber int = NULL
AS
BEGIN
  -- Update the Studies table
  UPDATE Studies
  SET
      Name = ISNULL(@Name, Name),
      Description = ISNULL(@Description, Description),
      CoordinatorID = ISNULL(@CoordinatorID, CoordinatorID),
      StartDate = ISNULL(@StartDate, StartDate),
      EndDate = ISNULL(@EndDate, EndDate),
      MaxStudents = ISNULL(@MaxStudents, MaxStudents),
      LanguageID = ISNULL(@LanguageID, LanguageID),
      FieldOfStudiesID = ISNULL(@FieldOfStudiesID, FieldOfStudiesID),
      SemesterNumber = ISNULL(@SemesterNumber, SemesterNumber)
  WHERE StudiesID = @StudiesID;
END;
GO
```

6.12 DeleteSession

Procedura do usuwania sesji.

```
CREATE OR ALTER PROCEDURE DeleteSession (
  @SessionID INT
)
AS
BEGIN
  IF EXISTS(
      SELECT PublicStudySessionID FROM PublicStudySessions WHERE PublicStudySessionID =
      ↪  @SessionID
  )
  BEGIN
      RAISERROR ('Cannot delete public sessions', 10, 1)
  END
  IF NOT EXISTS(
      SELECT StudiesSessionID FROM StudiesSessions WHERE StudiesSessionID = @SessionID
  )
  BEGIN
      RAISERROR ('Study session does not exist', 10, 1)
  END
  DELETE FROM StudiesSessions WHERE StudiesSessionID = @SessionID
  PRINT 'Successfully deleted data for study session ' + CAST(@SessionID AS NVARCHAR(10));
END;
GO
```

6.13 ModifyStudySession

Procedura do modyfikacji sesji studiów.

```sql
CREATE OR ALTER PROCEDURE ModifyStudySession
  @StudiesSessionID int,
  @SubjectID int = NULL,
  @StartDate datetime = NULL,
  @EndDate datetime = NULL,
  @LecturerID int = NULL,
  @MaxStudents int = NULL,
  @TranslatorID int = NULL,
  @LanguageID int = NULL
AS
BEGIN
  UPDATE StudiesSessions
  SET
      SubjectID = COALESCE(@SubjectID, SubjectID),
      StartDate = COALESCE(@StartDate, StartDate),
      EndDate = COALESCE(@EndDate, EndDate),
      LecturerID = COALESCE(@LecturerID, LecturerID),
      MaxStudents = COALESCE(@MaxStudents, MaxStudents),
      TranslatorID = COALESCE(@TranslatorID, TranslatorID),
      LanguageID = COALESCE(@LanguageID, LanguageID)
  WHERE StudiesSessionID = @StudiesSessionID;
END;
GO
```

6.14 CreateStationaryStudySession

Procedura do dodawania sesji stacjonarnych studiów.

```sql
CREATE OR ALTER PROCEDURE CreateStationaryStudySession
  @SubjectID int,
  @StartDate datetime,
  @EndDate datetime,
  @LecturerID int,
  @MaxStudents int,
  @TranslatorID int = NULL,
  @LanguageID int,
  @Address nvarchar(500),
  @City nvarchar(500),
  @Country nvarchar(500),
  @PostalCode nvarchar(20),
  @ClassroomNumber nvarchar(30)
AS
BEGIN
  -- Insert into StudiesSessions table
  INSERT INTO StudiesSessions (SubjectID, StartDate, EndDate, LecturerID, MaxStudents,
  ↪  TranslatorID, LanguageID)
  VALUES (@SubjectID, @StartDate, @EndDate, @LecturerID, @MaxStudents, @TranslatorID,
  ↪  @LanguageID);

  -- Capture the newly created StudiesSession ID
  DECLARE @NewSessionID int;
  SET @NewSessionID = SCOPE_IDENTITY();

  -- Insert the location details into StationaryStudiesSessions
  INSERT INTO StationaryStudiesSessions (StationaryStudiesSessionID, Address, City, Country,
  ↪  PostalCode, ClassroomNumber)
  VALUES (@NewSessionID, @Address, @City, @Country, @PostalCode, @ClassroomNumber);
END;
GO
```

6.15 ModifyStationaryStudySession

Procedura do modyfikacji sesji stacjonarnych studiów.

```sql
CREATE OR ALTER PROCEDURE ModifyStationaryStudySession
  @StudiesSessionID int,
  @SubjectID int = NULL,
  @StartDate datetime = NULL,
```

```
  @EndDate datetime = NULL,
  @LecturerID int = NULL,
  @MaxStudents int = NULL,
  @TranslatorID int = NULL,
  @LanguageID int = NULL,
  @Address nvarchar(500) = NULL,
  @City nvarchar(500) = NULL,
  @Country nvarchar(500) = NULL,
  @PostalCode nvarchar(20) = NULL,
  @ClassroomNumber nvarchar(30) = NULL
AS
BEGIN
  -- Update the StudiesSessions table
  EXEC ModifyStudySession @StudiesSessionID, @SubjectID, @StartDate, @EndDate, @LecturerID,
  ↪  @MaxStudents, @TranslatorID, @LanguageID;

  -- Update the StationaryStudiesSessions table
  UPDATE StationaryStudiesSessions
  SET
      Address = COALESCE(@Address, Address),
      City = COALESCE(@City, City),
      Country = COALESCE(@Country, Country),
      PostalCode = COALESCE(@PostalCode, PostalCode),
      ClassroomNumber = COALESCE(@ClassroomNumber, ClassroomNumber)
  WHERE StationaryStudiesSessionID = @StudiesSessionID;
END;
GO
```

6.16 AddOnlineStudySession

Procedura do dodawania sesji online studiów.

```
CREATE OR ALTER PROCEDURE AddOnlineStudySession
  @SubjectID int,
  @StartDate datetime,
  @EndDate datetime,
  @LecturerID int,
  @MaxStudents int,
  @TranslatorID int = NULL,
  @LanguageID int,
  @WebinarLink nvarchar(max),
  @RecordingLink nvarchar(max) = NULL
AS
BEGIN
  -- Insert into StudiesSessions table
  INSERT INTO StudiesSessions (SubjectID, StartDate, EndDate, LecturerID, MaxStudents,
  ↪  TranslatorID, LanguageID)
  VALUES (@SubjectID, @StartDate, @EndDate, @LecturerID, @MaxStudents, @TranslatorID,
  ↪  @LanguageID);

  -- Capture the newly created StudiesSession ID
  DECLARE @NewSessionID int;
  SET @NewSessionID = SCOPE_IDENTITY();

  -- Insert the online studies session details
  INSERT INTO OnlineStudiesSessions (OnlineStudiesSessionID, WebinarLink, RecordingLink)
  VALUES (@NewSessionID, @WebinarLink, @RecordingLink);
END;
GO
```

6.17 ModifyOnlineStudySession

Procedura do modyfikacji sesji online studiów.

```
CREATE OR ALTER PROCEDURE ModifyOnlineStudySession
  @OnlineStudiesSessionID int,
  @WebinarLink nvarchar(max) = NULL,
  @RecordingLink nvarchar(max) = NULL,
```

```
  -- Parameters for modifying StudiesSessions fields
  @SubjectID int = NULL,
  @StartDate datetime = NULL,
  @EndDate datetime = NULL,
  @LecturerID int = NULL,
  @MaxStudents int = NULL,
  @TranslatorID int = NULL,
  @LanguageID int = NULL
AS
BEGIN
  -- Update the OnlineStudiesSessions table
  UPDATE OnlineStudiesSessions
  SET
      WebinarLink = COALESCE(@WebinarLink, WebinarLink),
      RecordingLink = COALESCE(@RecordingLink, RecordingLink)
  WHERE OnlineStudiesSessionID = @OnlineStudiesSessionID;

  -- Update the StudiesSessions table
  IF @SubjectID IS NOT NULL OR @StartDate IS NOT NULL OR @EndDate IS NOT NULL OR @LecturerID IS
  ↪  NOT NULL OR @MaxStudents IS NOT NULL OR @TranslatorID IS NOT NULL OR @LanguageID IS NOT
  ↪  NULL
  BEGIN
      EXEC ModifyStudySession
          @StudiesSessionID = @OnlineStudiesSessionID,
          @SubjectID = @SubjectID,
          @StartDate = @StartDate,
          @EndDate = @EndDate,
          @LecturerID = @LecturerID,
          @MaxStudents = @MaxStudents,
          @TranslatorID = @TranslatorID,
          @LanguageID = @LanguageID;
  END
END;
GO
```

6.18 ModifyOnlineStudiesSessionRecording

Procedura do modyfikacji nagarania ze studyjnego spotkania online.

```
CREATE OR ALTER PROCEDURE ModifyOnlineStudiesSessionRecording
    @OnlineStudiesSessionID INT,
    @NewRecordingLink NVARCHAR(MAX)
AS
BEGIN
    UPDATE OnlineStudiesSessions
    SET RecordingLink = @NewRecordingLink
    WHERE OnlineStudiesSessionID = @OnlineStudiesSessionID;
END;
GO
```

6.19 CreateSubject

Procedura do tworzenia przedmiotu.

```
CREATE OR ALTER PROCEDURE CreateSubject
  @StudiesID int,
  @Description nvarchar(max),
  @CoordinatorID int,
  @SubjectName nvarchar(max)
AS
BEGIN
  -- Insert the new subject into the Subjects table
  INSERT INTO Subjects (StudiesID, Description, CoordinatorID, SubjectName)
  VALUES (@StudiesID, @Description, @CoordinatorID, @SubjectName);
END;
GO
```

6.20 ModifySubject

Procedura do modyfikacji przedmiotu.

```
CREATE OR ALTER PROCEDURE ModifySubject
  @SubjectID int,
  @StudiesID int = NULL,
  @Description nvarchar(max) = NULL,
  @CoordinatorID int = NULL,
  @SubjectName nvarchar(max) = NULL
AS
BEGIN
  -- Update the Subjects table
  UPDATE Subjects
  SET
      StudiesID = COALESCE(@StudiesID, StudiesID),
      Description = COALESCE(@Description, Description),
      CoordinatorID = COALESCE(@CoordinatorID, CoordinatorID),
      SubjectName = COALESCE(@SubjectName, SubjectName)
  WHERE SubjectID = @SubjectID;
END;
GO
```

6.21 AddExam

Procedura do dodawania egzaminu.

```
CREATE OR ALTER PROCEDURE AddExam
  @SubjectID int,
  @StartDate datetime,
  @EndDate datetime,
  @Country nvarchar(500),
  @City nvarchar(500),
  @PostalCode nvarchar(500),
  @Address nvarchar(500)
AS
BEGIN
  INSERT INTO Exams (SubjectID, StartDate, EndDate, Country, City, PostalCode, Address)
  VALUES (@SubjectID, @StartDate, @EndDate, @Country, @City, @PostalCode, @Address);
END;
GO
```

6.22 ModifyExam

Procedura do modyfikacji egzaminu.

```
CREATE OR ALTER PROCEDURE ModifyExam
  @ExamID int,
  @SubjectID int = NULL,
  @StartDate datetime = NULL,
  @EndDate datetime = NULL,
  @Country nvarchar(500) = NULL,
  @City nvarchar(500) = NULL,
  @PostalCode nvarchar(500) = NULL,
  @Address nvarchar(500) = NULL
AS
BEGIN
  UPDATE Exams
  SET SubjectID = COALESCE(@SubjectID, SubjectID),
      StartDate = COALESCE(@StartDate, StartDate),
      EndDate = COALESCE(@EndDate, EndDate),
      Country = COALESCE(@Country, Country),
      City = COALESCE(@City, City),
      PostalCode = COALESCE(@PostalCode, PostalCode),
      Address = COALESCE(@Address, Address)
  WHERE ExamID = @ExamID;
END;
GO
```

6.23 UpdateExamGrade

Procedura do aktualizacji oceny z egzaminu.

```
CREATE OR ALTER PROCEDURE UpdateExamGrade
  @StudentID int,
  @ExamID int,
  @FinalGrade decimal(2,1)
AS
BEGIN
  IF EXISTS (SELECT 1 FROM ExamsGrades WHERE StudentID = @StudentID AND ExamID = @ExamID)
  BEGIN
      UPDATE ExamsGrades
      SET FinalGrade = @FinalGrade
      WHERE StudentID = @StudentID AND ExamID = @ExamID;
  END
  ELSE
  BEGIN
      INSERT INTO ExamsGrades (StudentID, ExamID, FinalGrade)
      VALUES (@StudentID, @ExamID, @FinalGrade);
  END
END;
GO
```

6.24 DeleteExamGrade

Procedura do usuwania oceny z egzaminu.

```
CREATE OR ALTER PROCEDURE DeleteExamGrade
  @StudentID int,
  @ExamID int
AS
BEGIN
  DELETE FROM ExamsGrades WHERE StudentID = @StudentID AND ExamID = @ExamID;
END;
GO
```

6.25 AddInternship

Procedura do dodawania stażu.

```
CREATE OR ALTER PROCEDURE AddInternship
  @StudiesID int,
  @Description nvarchar(max),
  @StartDate date,
  @EndDate date
AS
BEGIN
  INSERT INTO Internships (StudiesID, Description, StartDate, EndDate)
  VALUES (@StudiesID, @Description, @StartDate, @EndDate);

  PRINT 'Internship added successfully.';
END;
GO
```

6.26 ModifyInternship

Procedura do modyfikacji danych stażu.

```
CREATE OR ALTER PROCEDURE ModifyInternship
  @InternshipID int,
  @NewDescription nvarchar(max) = NULL,
  @NewStartDate date = NULL,
  @NewEndDate date = NULL
AS
BEGIN
  UPDATE Internships
```

```
  SET Description = COALESCE(@NewDescription, Description),
      StartDate = COALESCE(@NewStartDate, StartDate),
      EndDate = COALESCE(@NewEndDate, EndDate)
  WHERE InternshipID = @InternshipID;

  PRINT 'Internship modified successfully.';
END;
GO
```

6.27 UpdateInternshipDetail

Procedura do aktualizacji szczegółów stażu dla danego studenta.

```
CREATE OR ALTER PROCEDURE UpdateInternshipDetail
  @StudentID int,
  @InternshipID int,
  @CompletedAt date = NULL,
  @Completed bit,
  @CompanyName nvarchar(500),
  @City nvarchar(500),
  @Country nvarchar(500),
  @PostalCode nvarchar(500),
  @Address nvarchar(500)
AS
BEGIN
  IF EXISTS (SELECT 1 FROM InternshipDetails WHERE StudentID = @StudentID AND IntershipID =
  ↪ @InternshipID)
  BEGIN
      UPDATE InternshipDetails
      SET CompletedAt = @CompletedAt,
          Completed = @Completed,
          CompanyName = @CompanyName,
          City = @City,
          Country = @Country,
          PostalCode = @PostalCode,
          Address = @Address
      WHERE StudentID = @StudentID AND IntershipID = @InternshipID;
  END
  ELSE
  BEGIN
      INSERT INTO InternshipDetails (StudentID, IntershipID, CompletedAt, Completed, CompanyName,
      ↪ City, Country, PostalCode, Address)
      VALUES (@StudentID, @InternshipID, @CompletedAt, @Completed, @CompanyName, @City, @Country,
      ↪ @PostalCode, @Address);
  END

  PRINT 'Internship detail updated successfully.';
END;
GO
```

6.28 DeleteInternshipDetail

Procedura do usuwania szczegółów stażu.

```
CREATE OR ALTER PROCEDURE DeleteInternshipDetail
  @StudentID int,
  @InternshipID int
AS
BEGIN
  DELETE FROM InternshipDetails WHERE StudentID = @StudentID AND IntershipID= @InternshipID;

  PRINT 'Internship detail deleted successfully.';
END;
GO
```

6.29 InsertMadeUpAttendance

Procedura do dodawania udziału w zajęciach zamiennych.

```
CREATE OR ALTER PROCEDURE InsertMadeUpAttendance (
  @StudentID INT,
  @ProductID INT,
  @SubjectID INT
)
AS
BEGIN
  IF NOT EXISTS(
      SELECT * FROM SubjectMakeUpPossibilities
      WHERE ProductID = @ProductID AND SubjectID = @SubjectID
  )
  BEGIN
      RAISERROR ('This is not a valid make-up possibility.', 10, 1)
  END
  IF EXISTS(
      SELECT * FROM MadeUpAttendance
      WHERE StudentID = @StudentID AND ProductID=@ProductID
  )
  BEGIN
      RAISERROR ('This student already made up the attendance using this product', 10, 1)
  END

  DECLARE @attendance BIT
DECLARE @UserID INT

SELECT @UserID = UserID FROM
Students WHERE Students.StudentID = @StudentID;

  SET @attendance = dbo.CheckIfUserCompletedProduct(@UserID, @ProductID)

  IF (@attendance = 1)
  BEGIN
      INSERT INTO MadeUpAttendance (SubjectID, ProductID, StudentID)
      VALUES (@SubjectID, @ProductID, @StudentID)
  END
  PRINT 'Succesfully made-up attendance'
END;
GO
```

6.30 CheckIfUserCompletedProduct

Funkcja sprawdzająca, czy użytkownik ukończył dany produkt (kurs, webinar, etc.).

```
CREATE OR ALTER FUNCTION CheckIfUserCompletedProduct (
@UserID INT,
@ProductID INT
)
RETURNS BIT
AS
BEGIN
IF EXISTS (
    SELECT * FROM Students
    WHERE UserID = @UserID AND StudiesID = @ProductID  AND Completed = 1
)
BEGIN
    RETURN 1
END
IF EXISTS(
    SELECT * FROM CourseParticipants
    WHERE UserID = @UserID AND CourseID = @ProductID  AND Completed = 1
)
BEGIN
    RETURN 1
END
IF EXISTS(
    SELECT * FROM WebinarsAttendence WA
JOIN WebinarParticipants WP ON WP.WebinarParticipantID = WA.WebinarParticipantID
    WHERE WP.UserID = @UserID AND WP.WebinarID = @ProductID AND WA.WasPresent = 1
)
```

```
BEGIN
    RETURN 1
END
IF EXISTS(
    SELECT * FROM PublicStudySessionsAttendanceForOutsiders A
    JOIN PublicStudySessionParticipants PS ON
    PS.PublicStudySessionParticipantID = A.PublicStudySessionParticipantID
    WHERE UserID = @UserID AND PS.PublicStudySessionParticipantID = @ProductID AND Completed = 1
)
BEGIN
    RETURN 1
END
RETURN 0
END;
GO
```

6.31 CloseStudies

Procedura do zamykania semestru studiów. Sprawdza czy dany student zdał egzaminy, zaliczył staże, ma odpowiednią obecność i jeżeli tak to wpisuje zaliczenie semestru studiów.

```
CREATE OR ALTER PROCEDURE CloseStudies(@StudiesID INT)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION

        -- Declare variables
        DECLARE @MinAttendancePercentage DECIMAL(6,4);

        -- Get minimum attendance percentage for the studies from the function
        SELECT @MinAttendancePercentage = dbo.GetMinAttendancePercentageForStudies(@StudiesID);

        -- Temporary table to store students' data
        CREATE TABLE #StudentData (
            StudentID INT,
            SubjectID INT,
            TotalSessions INT,
            AttendedSessions INT,
            MadeUpSessions INT,
            EffectiveAttendance DECIMAL(6,4),
            HasPassed BIT,
            InternshipCompleted BIT,
            ExamsPassed BIT
        );

        -- Populate the temporary table with initial data
        INSERT INTO #StudentData (StudentID, SubjectID, TotalSessions, AttendedSessions,
        ↪  MadeUpSessions, EffectiveAttendance, HasPassed)
        SELECT
            s.StudentID,
            ss.SubjectID,
            (SELECT COUNT(*) FROM StudiesSessions ss2 WHERE ss2.SubjectID = ss.SubjectID) AS
            ↪  TotalSessions,
            SUM(CASE WHEN ssa.Completed = 1 THEN 1 ELSE 0 END) AS AttendedSessions,
            0 AS MadeUpSessions, -- Initial value, will be updated later
            0.0 AS EffectiveAttendance, -- Initial value, will be updated later
            0 AS HasPassed -- Initial value, will be updated later
        FROM Students s
        LEFT JOIN StudiesSessionsAttendence ssa ON s.StudentID = ssa.StudentID
        LEFT JOIN StudiesSessions ss ON ssa.SessionID = ss.StudiesSessionID
        WHERE s.StudiesID = @StudiesID
        GROUP BY s.StudentID, ss.SubjectID;


        -- Update the table with made up sessions
        UPDATE #StudentData
        SET MadeUpSessions = (
```

```
            SELECT SUM(smup.AttendanceValue)
            FROM MadeUpAttendance mup
            JOIN SubjectMakeUpPossibilities smup ON mup.SubjectID = smup.SubjectID AND
            ↪  mup.ProductID = smup.ProductID
            WHERE mup.StudentID = #StudentData.StudentID AND mup.SubjectID =
            ↪  #StudentData.SubjectID
        );

        -- Calculate effective attendance for each student in each subject
        UPDATE #StudentData
        SET EffectiveAttendance = (CAST(AttendedSessions AS DECIMAL) + CAST(MadeUpSessions AS
        ↪  DECIMAL)) / CAST(TotalSessions AS DECIMAL);

        -- Determine if the student has passed based on the effective attendance and minimum
        ↪  required attendance
        UPDATE #StudentData
        SET HasPassed = CASE WHEN EffectiveAttendance >= @MinAttendancePercentage THEN 1 ELSE 0
        ↪  END;

        -- Check if each student has completed all internships
        UPDATE #StudentData
        SET InternshipCompleted = CASE
            WHEN EXISTS (
                SELECT 1
                FROM InternshipDetails id
                JOIN Internships i ON id.IntershipID = i.InternshipID
                WHERE id.StudentID = #StudentData.StudentID AND i.StudiesID = @StudiesID AND
                ↪  id.Completed = 1
                GROUP BY id.StudentID
                HAVING COUNT(id.IntershipID) = (SELECT COUNT(InternshipID) FROM Internships WHERE
                ↪  StudiesID = @StudiesID)
            ) THEN 1
            ELSE 0
        END;

        -- Check if each student has passed all exams with a grade >= 3.0
        UPDATE #StudentData
        SET ExamsPassed = CASE
            WHEN NOT EXISTS (
                SELECT 1
                FROM ExamsGrades eg
                JOIN Exams e ON eg.ExamID = e.ExamID
                JOIN Subjects sub ON e.SubjectID = sub.SubjectID
                WHERE eg.StudentID = #StudentData.StudentID AND sub.StudiesID = @StudiesID AND
                ↪  eg.FinalGrade < 3.0
            ) THEN 1
            ELSE 0
        END;

        -- Update the Students table to set Completed = 1 for those who passed everything
        UPDATE s
        SET s.Completed = 1
        FROM Students s
        JOIN #StudentData sd ON s.StudentID = sd.StudentID
        WHERE sd.HasPassed = 1 AND sd.InternshipCompleted = 1 AND sd.ExamsPassed = 1 AND
        ↪  s.StudiesID = @StudiesID;

        -- Final output of the procedure: StudentID, SubjectID, HasPassed, InternshipCompleted,
        ↪  ExamsPassed
        SELECT * FROM #StudentData;

        -- Clean up temporary table
        DROP TABLE #StudentData;

        COMMIT TRANSACTION
    END TRY
    BEGIN CATCH
        -- Error handling and rollback
        ROLLBACK TRANSACTION;
```

```
        THROW;
    END CATCH
END;
GO
```

6.32 AddOrModifyPublicStudySessionAttendance

Procedura do dodawania lub modyfikacji obecności na pojedynczym spotkaniu studyjnym

```
CREATE OR ALTER PROCEDURE AddOrModifyPublicStudySessionAttendance(
  @ParticipantID INT,
  @SessionID INT,
  @Completed BIT
)
AS
BEGIN
  IF EXISTS(
      SELECT * FROM PublicStudySessionsAttendanceForOutsiders
      WHERE PublicStudySessionID = @SessionID AND PublicStudySessionParticipantID =
      ↪ @ParticipantID
  )
  BEGIN
      UPDATE PublicStudySessionsAttendanceForOutsiders
      SET
          Completed = @Completed
      WHERE
          PublicStudySessionParticipantID = @ParticipantID
      AND PublicStudySessionID = @SessionID
      PRINT 'Attendance already exists. Successfuly modified attendance'
  END
  ELSE
  BEGIN
      INSERT INTO PublicStudySessionsAttendanceForOutsiders (PublicStudySessionID,
      ↪ PublicStudySessionParticipantID, Completed)
      VALUES (@SessionID, @ParticipantID, @Completed)
      PRINT 'Successfully inserted attendance'
  END
END;
GO
```

6.33 DeletePublicStudySessionAttendance

Procedura do usuwania obecności na pojedynczym spotkaniu studyjnym

```
CREATE OR ALTER PROCEDURE DeletePublicStudySessionAttendance(
  @ParticipantID INT,
  @SessionID INT
)
AS
BEGIN
  IF EXISTS(
      SELECT * FROM PublicStudySessionsAttendanceForOutsiders
      WHERE PublicStudySessionID = @SessionID AND PublicStudySessionParticipantID =
      ↪ @ParticipantID
  )
  BEGIN
      DELETE FROM PublicStudySessionsAttendanceForOutsiders
      WHERE PublicStudySessionID = @SessionID AND PublicStudySessionParticipantID =
      ↪ @ParticipantID
      PRINT 'Succesfully deleted attendance.'
  END
  ELSE
  BEGIN
      PRINT 'This attendance does not exist.'
  END
END;
GO
```

6.34 getStudiesAttendance

Funkcja zwracająca szczegółową listę obecności dla danego semestru studiów.

```
CREATE OR ALTER FUNCTION getStudiesAttendance(@StudiesID INT)
RETURNS TABLE
AS
RETURN (
    SELECT
        s.StudentID,
        s.UserID,
        p.FirstName,
        p.LastName,
        sub.SubjectID,
        sub.SubjectName,
        ss.StudiesSessionID AS SessionID,
        CASE
            WHEN sss.StationaryStudiesSessionID IS NOT NULL THEN 'Stationary'
            WHEN oss.OnlineStudiesSessionID IS NOT NULL THEN 'Online'
            ELSE 'Unknown'
        END AS SessionType,
        ss.StartDate,
        ss.EndDate,
        ssa.Completed
    FROM
        StudiesSessions ss
    INNER JOIN
        StudiesSessionsAttendence ssa ON ss.StudiesSessionID = ssa.SessionID
    INNER JOIN
        Students s ON ssa.StudentID = s.StudentID
    INNER JOIN
        People p ON s.UserID = p.PersonID
    INNER JOIN
        Subjects sub ON ss.SubjectID = sub.SubjectID
    LEFT JOIN
        StationaryStudiesSessions sss ON ss.StudiesSessionID = sss.StationaryStudiesSessionID
    LEFT JOIN
        OnlineStudiesSessions oss ON ss.StudiesSessionID = oss.OnlineStudiesSessionID
    WHERE
        sub.StudiesID = @StudiesID
);
GO
```

6.35 CreateCourse

Procedura tworząca kurs.

```
CREATE OR ALTER PROCEDURE CreateCourse
    @CourseName nvarchar(max),
    @Description nvarchar(max),
    @StartDate datetime,
    @EndDate datetime,
    @CoordinatorID int,
    @MaxStudents int = NULL,  -- Nullable
    @LanguageID int,
    @Price money,
    @AdvancePayment money
AS
BEGIN
    -- Insert into Products table and capture the new ProductID
    DECLARE @NewProductID int;

    INSERT INTO Products (Price, AdvancePayment, ProductType)
    VALUES (@Price, @AdvancePayment, 'course');

    SET @NewProductID = SCOPE_IDENTITY();

    -- Insert into Courses table using the new ProductID
```

```
    INSERT INTO Courses (CourseID, CourseName, Description, StartDate, EndDate, CoordinatorID,
    ↪  MaxStudents, LanguageID)
    VALUES (@NewProductID, @CourseName, @Description, @StartDate, @EndDate, @CoordinatorID,
    ↪  @MaxStudents, @LanguageID);

    -- Print confirmation message
    PRINT 'Course created successfully.';
END;
GO
```

6.36 ModifyCourse

Procedura modyfikująca kurs.

```
CREATE OR ALTER PROCEDURE ModifyCourse
    @CourseID int,
    @NewCourseName nvarchar(max) = NULL,
    @NewDescription nvarchar(max) = NULL,
    @NewStartDate datetime = NULL,
    @NewEndDate datetime = NULL,
    @NewCoordinatorID int = NULL,
    @NewMaxStudents int = NULL,
    @NewLanguageID int = NULL
AS
BEGIN
    -- Update the Courses table
    UPDATE Courses
    SET CourseName = COALESCE(@NewCourseName, CourseName),
        Description = COALESCE(@NewDescription, Description),
        StartDate = COALESCE(@NewStartDate, StartDate),
        EndDate = COALESCE(@NewEndDate, EndDate),
        CoordinatorID = COALESCE(@NewCoordinatorID, CoordinatorID),
        MaxStudents = COALESCE(@NewMaxStudents, MaxStudents),
        LanguageID = COALESCE(@NewLanguageID, LanguageID)
    WHERE CourseID = @CourseID;

    -- Print confirmation message
    PRINT 'Course data updated successfully.';
END;
GO
```

6.37 CreateModule

Procedura tworząca moduł kursu.

```
CREATE OR ALTER PROCEDURE CreateModule
    @CourseID int,
    @ModuleName nvarchar(max),
    @ModuleDescription nvarchar(max)
AS
BEGIN
    -- Insert the new module into the Modules table
    INSERT INTO Modules (CourseID, ModuleName, ModuleDescription)
    VALUES (@CourseID, @ModuleName, @ModuleDescription);

    -- Print confirmation message
    PRINT 'Module created successfully.';
END;
GO
```

6.38 ModifyModule

Procedura modyfikująca moduł kursu.

```
CREATE OR ALTER PROCEDURE ModifyModule
    @ModuleID int,
    @NewModuleName nvarchar(max) = NULL,
```

```
    @NewModuleDescription nvarchar(max) = NULL
AS
BEGIN
    -- Update the Modules table
    UPDATE Modules
    SET ModuleName = COALESCE(@NewModuleName, ModuleName),
        ModuleDescription = COALESCE(@NewModuleDescription, ModuleDescription)
    WHERE ModuleID = @ModuleID;

    -- Print confirmation message
    PRINT 'Module updated successfully.';
END;
GO
```

6.39 DeleteModule

Procedura usuwająca moduł kursu.

```
CREATE OR ALTER PROCEDURE DeleteModule
    @ModuleID int
AS
BEGIN
    -- Delete the specified module from the Modules table
    DELETE FROM Modules
    WHERE ModuleID = @ModuleID;

    -- Print confirmation message
    PRINT 'Module deleted successfully.';
END;
GO
```

6.40 ModifyOnlineCourseSession

Procedura modyfikująca sesję online kursu.

```
CREATE OR ALTER PROCEDURE ModifyOnlineCourseSession
    @CourseSessionID int,
    @LanguageID int = NULL,
    @ModuleID int = NULL,
    @LecturerID int = NULL,
    @TranslatorID int = NULL,
    @StartDate datetime = NULL,
    @EndDate datetime = NULL,
    @WebinarLink nvarchar(max) = NULL,
    @RecordingLink nvarchar(max) = NULL
AS
BEGIN
    -- Update the general course session details
    UPDATE CoursesSessions
    SET LanguageID = COALESCE(@LanguageID, LanguageID),
        ModuleID = COALESCE(@ModuleID, ModuleID),
        LecturerID = COALESCE(@LecturerID, LecturerID),
        TranslatorID = COALESCE(@TranslatorID, TranslatorID)
    WHERE CourseSessionID = @CourseSessionID;

    -- Update the online-specific details
    UPDATE CourseOnlineSessions
    SET StartDate = COALESCE(@StartDate, StartDate),
        EndDate = COALESCE(@EndDate, EndDate),
        WebinarLink = COALESCE(@WebinarLink, WebinarLink),
        RecordingLink = COALESCE(@RecordingLink, RecordingLink)
    WHERE CourseOnlineSessionID = @CourseSessionID;

    PRINT 'Online course session modified successfully.';
END;
GO
```

6.41 ModifyOfflineCourseSession

Procedura modyfikująca sesję offline kursu.

```
CREATE OR ALTER PROCEDURE ModifyOfflineCourseSession
    @CourseSessionID int,
    @LanguageID int = NULL,
    @ModuleID int = NULL,
    @LecturerID int = NULL,
    @TranslatorID int = NULL,
    @Link nvarchar(max) = NULL,
    @Description nvarchar(max) = NULL
AS
BEGIN
    -- Update the general course session details
    UPDATE CoursesSessions
    SET LanguageID = COALESCE(@LanguageID, LanguageID),
        ModuleID = COALESCE(@ModuleID, ModuleID),
        LecturerID = COALESCE(@LecturerID, LecturerID),
        TranslatorID = COALESCE(@TranslatorID, TranslatorID)
    WHERE CourseSessionID = @CourseSessionID;

    -- Update the offline-specific details
    UPDATE CourseOfflineSessions
    SET Link = COALESCE(@Link, Link),
        Description = COALESCE(@Description, Description)
    WHERE CourseOfflineSessionID = @CourseSessionID;

    PRINT 'Offline course session modified successfully.';
END;
GO
```

6.42 ModifyStationaryCourseSession

Procedura modyfikująca stacjonarną sesję kursu.

```
CREATE OR ALTER PROCEDURE ModifyStationaryCourseSession
    @CourseSessionID int,
    @LanguageID int = NULL,
    @ModuleID int = NULL,
    @LecturerID int = NULL,
    @TranslatorID int = NULL,
    @StartDate datetime = NULL,
    @EndDate datetime = NULL,
    @Address nvarchar(500) = NULL,
    @City nvarchar(500) = NULL,
    @Country nvarchar(500) = NULL,
    @PostalCode nvarchar(20) = NULL,
    @ClassroomNumber nvarchar(30) = NULL,
    @MaxStudents int = NULL
AS
BEGIN
    -- Update the general course session details
    UPDATE CoursesSessions
    SET LanguageID = COALESCE(@LanguageID, LanguageID),
        ModuleID = COALESCE(@ModuleID, ModuleID),
        LecturerID = COALESCE(@LecturerID, LecturerID),
        TranslatorID = COALESCE(@TranslatorID, TranslatorID)
    WHERE CourseSessionID = @CourseSessionID;

    -- Update the stationary-specific details
    UPDATE CourseStationarySessions
    SET StartDate = COALESCE(@StartDate, StartDate),
        EndDate = COALESCE(@EndDate, EndDate),
        Address = COALESCE(@Address, Address),
        City = COALESCE(@City, City),
        Country = COALESCE(@Country, Country),
        PostalCode = COALESCE(@PostalCode, PostalCode),
        ClassroomNumber = COALESCE(@ClassroomNumber, ClassroomNumber),
```

```
        MaxStudents = COALESCE(@MaxStudents, MaxStudents)
    WHERE CourseStationarySessionID = @CourseSessionID;

    PRINT 'Stationary course session modified successfully.';
END;
GO
```

6.43 DeleteCourseSession

Procedura usuwająca sesję kursu.

```
CREATE OR ALTER PROCEDURE DeleteCourseSession
    @CourseSessionID int
AS
BEGIN
    -- Delete the course session from the CoursesSessions table
    -- Associated records in other session tables will be deleted automatically due to ON DELETE
    ↪  CASCADE constraints
    DELETE FROM CoursesSessions WHERE CourseSessionID = @CourseSessionID;

    PRINT 'Course session and its associated records deleted successfully.';
END;
GO
```

6.44 UpdateAttendance

Procedura aktualizująca obecność na kursie.

```
CREATE OR ALTER PROCEDURE UpdateAttendance
    @CourseParticipantID int,
    @CourseSessionID int,
    @WasPresent bit
AS
BEGIN
    IF EXISTS (SELECT 1 FROM CourseSessionsAttendance WHERE CourseParticipantID =
    ↪  @CourseParticipantID AND CourseSessionID = @CourseSessionID)
    BEGIN
        -- Update existing record
        UPDATE CourseSessionsAttendance
        SET Completed = @WasPresent
        WHERE CourseParticipantID = @CourseParticipantID AND CourseSessionID = @CourseSessionID;
    END
    ELSE
    BEGIN
        -- Insert new record
        INSERT INTO CourseSessionsAttendance (CourseParticipantID, CourseSessionID, Completed)
        VALUES (@CourseParticipantID, @CourseSessionID, @WasPresent);
    END

    PRINT 'Attendance record updated successfully.';
END;
GO
```

6.45 DeleteAttendance

Procedura usuwająca obecność na kursie.

```
CREATE OR ALTER PROCEDURE DeleteAttendance
    @CourseParticipantID int,
    @CourseSessionID int
AS
BEGIN
    -- Delete the attendance record
    DELETE FROM CourseSessionsAttendance
    WHERE CourseParticipantID = @CourseParticipantID AND CourseSessionID = @CourseSessionID;

    PRINT 'Attendance record deleted successfully.';
```

```
END;
GO
```

6.46 PlayOfflineSessionRecording

Procedura odtwarzająca nagranie sesji offline.

```
CREATE OR ALTER PROCEDURE PlayOfflineSessionRecording
    @CourseParticipantID int,
    @CourseOfflineSessionID int,
    @SessionLink nvarchar(max) OUTPUT
AS
BEGIN
    -- Check if attendance already exists
    IF NOT EXISTS (SELECT 1 FROM CourseSessionsAttendance WHERE CourseParticipantID =
    ↪  @CourseParticipantID AND CourseSessionID = @CourseOfflineSessionID)
    BEGIN
        -- Insert attendance record as 'present' if it doesn't exist
        INSERT INTO CourseSessionsAttendance (CourseParticipantID, CourseSessionID, Completed)
        VALUES (@CourseParticipantID, @CourseOfflineSessionID, 1);
    END

    -- Retrieve the link to the offline session
    SELECT @SessionLink = Link FROM CourseOfflineSessions WHERE CourseOfflineSessionID =
    ↪  @CourseOfflineSessionID;

    -- Return the session link
    RETURN;
END;
GO
```

6.47 GetOnlineSessionRecordingLink

Funkcja pobierająca link do nagrania sesji online.

```
CREATE OR ALTER FUNCTION GetOnlineSessionRecordingLink
(
    @CourseOnlineSessionID int
)
RETURNS nvarchar(max)
AS
BEGIN
    DECLARE @RecordingLink nvarchar(max);

    -- Retrieve the link to the online session recording
    SELECT @RecordingLink = RecordingLink
    FROM CourseOnlineSessions
    WHERE CourseOnlineSessionID = @CourseOnlineSessionID;

    -- Return the session recording link
    RETURN @RecordingLink;
END;
GO
```

6.48 UpdateCourseSessionAttendance

Procedura aktualizująca obecność na sesji kursu.

```
CREATE OR ALTER PROCEDURE UpdateCourseSessionAttendance
    @CourseParticipantID int,
    @CourseSessionID int,
    @WasPresent bit
AS
BEGIN
    IF EXISTS (SELECT 1 FROM CourseSessionsAttendance WHERE CourseParticipantID =
    ↪  @CourseParticipantID AND CourseSessionID = @CourseSessionID)
    BEGIN
```

```
        -- Update existing record
        UPDATE CourseSessionsAttendance
        SET Completed = @WasPresent
        WHERE CourseParticipantID = @CourseParticipantID AND CourseSessionID = @CourseSessionID;
    END
    ELSE
    BEGIN
        -- Insert new record
        INSERT INTO CourseSessionsAttendance (CourseParticipantID, CourseSessionID, Completed)
        VALUES (@CourseParticipantID, @CourseSessionID, @WasPresent);
    END
END;
GO
```

6.49 DeleteCourseSessionAttendance

Usuń wpis do listy obecności.

```
CREATE OR ALTER PROCEDURE DeleteCourseSessionAttendance
    @CourseParticipantID int,
    @CourseSessionID int
AS
BEGIN
    -- Delete the attendance record
    DELETE FROM CourseSessionsAttendance
    WHERE CourseParticipantID = @CourseParticipantID AND CourseSessionID = @CourseSessionID;
END;
GO
```

6.50 CloseCourse

Procedura zamykająca kurs. Sprawdza ona na podstawie obecności kto zaliczył kurs i wpisuje zaliczenie.

```
CREATE OR ALTER PROCEDURE CloseCourse(@CourseID INT)
AS
BEGIN
    -- Start the transaction
    BEGIN TRANSACTION

    BEGIN TRY
        -- Update the ClosedAt date for the course in Products table
        UPDATE Products
        SET ClosedAt = GETDATE()
        FROM Products
        INNER JOIN Courses ON Products.ProductID = Courses.CourseID
        WHERE Courses.CourseID = @CourseID

        -- Get the minimum attendance percentage for the course
        DECLARE @MinAttendancePercentage DECIMAL(6, 4)
        SELECT @MinAttendancePercentage = dbo.GetMinAttendancePercentageForCourse(@CourseID)

        -- Temporary table to store module attendance for each participant
        DECLARE @ModuleAttendanceStats TABLE (CourseParticipantID INT, ModuleID INT,
        ↪  AllSessionsCompleted BIT)

        -- Insert module attendance stats for each participant
        INSERT INTO @ModuleAttendanceStats (CourseParticipantID, ModuleID, AllSessionsCompleted)
        SELECT
            cp.CourseParticipantID,
            cs.ModuleID,
            CASE WHEN COUNT(csa.CourseSessionID) = SUM(CASE WHEN csa.Completed = 1 THEN 1 ELSE 0
            ↪  END) THEN 1 ELSE 0 END
        FROM
            CourseParticipants cp
    INNER JOIN
            CourseSessionsAttendance csa ON cp.CourseParticipantID = csa.CourseParticipantID
    INNER JOIN
```

```
            CoursesSessions cs ON cs.CourseSessionID = csa.CourseSessionID
        WHERE
            cp.CourseID = @CourseID
        GROUP BY
            cp.CourseParticipantID, cs.ModuleID

        -- Update the CourseParticipants table for those who have completed the course
        UPDATE cp
        SET Completed = 1
        FROM CourseParticipants cp
        WHERE cp.CourseID = @CourseID
        AND
        (
            SELECT CAST(CAST(SUM(CAST(mat.AllSessionsCompleted AS INT)) AS DECIMAL) /
            ↪  NULLIF(COUNT(mat.ModuleID), 0) AS DECIMAL(5, 2))
            FROM @ModuleAttendanceStats mat
            WHERE mat.CourseParticipantID = cp.CourseParticipantID
        ) >= @MinAttendancePercentage

        -- Commit the transaction
        COMMIT
    END TRY
    BEGIN CATCH
        -- Rollback the transaction in case of error
        ROLLBACK

        PRINT('Couldnt close the course');
    END CATCH
END;
GO
```

6.51 ModifyCourseOnlineSessionRecording

Modyfikuj nagranie z CourseOnlineSession

```
CREATE OR ALTER PROCEDURE ModifyCourseOnlineSessionRecording
    @CourseOnlineSessionID INT,
    @NewRecordingLink NVARCHAR(MAX)
AS
BEGIN
    UPDATE CourseOnlineSessions
    SET RecordingLink = @NewRecordingLink
    WHERE CourseOnlineSessionID = @CourseOnlineSessionID;
END;
GO
```

6.52 AddWebinar

Procedura do tworzenia nowego webinaru.

```
CREATE OR ALTER PROCEDURE AddWebinar
    @WebinarName nvarchar(max),
    @Description nvarchar(max),
    @StartDate datetime,
    @EndDate datetime,
    @WebinarLink nvarchar(max),
    @LecturerID int,
    @TranslatorID int,
    @LanguageID int,
    @Price money
AS
BEGIN
    -- Insert into Products table
    INSERT INTO Products(Price, AdvancePayment, ProductType)
    VALUES (@Price, NULL, 'webinar')

    -- Get the last inserted ProductID
    DECLARE @ProductID int
```

```
    SET @ProductID = SCOPE_IDENTITY()

    -- Insert into Webinars table
    INSERT INTO Webinars(WebinarID, WebinarName, Description, StartDate, EndDate, WebinarLink,
    ↪  LecturerID, TranslatorID, LanguageID)
    VALUES (@ProductID, @WebinarName, @Description, @StartDate, @EndDate, @WebinarLink,
    ↪  @LecturerID, @TranslatorID, @LanguageID)
END;
GO
```

6.53 ModifyWebinarData

Procedura do modyfikacji danych webinaru.

```
CREATE OR ALTER PROCEDURE ModifyWebinarData
    @WebinarID INT,
    @WebinarName NVARCHAR(MAX),
    @Description NVARCHAR(MAX),
    @StartDate DATETIME,
    @EndDate DATETIME,
    @RecordingLink NVARCHAR(MAX) = NULL,
    @WebinarLink NVARCHAR(MAX),
    @LecturerID INT,
    @LanguageID INT,
    @RecordingReleaseDate DATE = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Check if the WebinarID exists in the Webinars table
    IF EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID)
    BEGIN
        -- Update the Webinars table with the provided data
        UPDATE Webinars
        SET
            WebinarName = @WebinarName,
            Description = @Description,
            StartDate = @StartDate,
            EndDate = @EndDate,
            RecordingLink = CASE WHEN @RecordingLink = '' THEN NULL ELSE @RecordingLink END,
            WebinarLink = @WebinarLink,
            LecturerID = @LecturerID,
            LanguageID = @LanguageID,
            RecordingReleaseDate = CASE WHEN @RecordingReleaseDate = '' THEN NULL ELSE
            ↪  @RecordingReleaseDate END
        WHERE WebinarID = @WebinarID;

        -- Return success message or handle any additional logic as needed
        PRINT 'Webinar data has been modified successfully.';
    END
    ELSE
    BEGIN
        -- Handle the case where the WebinarID does not exist
        PRINT 'Webinar with ID ' + CAST(@WebinarID AS NVARCHAR(MAX)) + ' does not exist.';
    END
END;
GO
```

6.54 DeleteWebinar

Procedura do usuwania webinaru.

```
CREATE OR ALTER PROCEDURE DeleteWebinar @WebinarID INT
AS
BEGIN
    -- Transaction ensures all or nothing operation
    BEGIN TRANSACTION;
```

```
    BEGIN TRY
        -- Step 1: Delete from WebinarsAttendence
        DELETE FROM WebinarsAttendence
        WHERE WebinarID = @WebinarID;

        -- Step 2: Delete from WebinarParticipants
        DELETE FROM WebinarParticipants
        WHERE WebinarID = @WebinarID;

        -- Step 3: Delete the webinar from Webinars
        DELETE FROM Webinars
        WHERE WebinarID = @WebinarID;

        -- If everything is okay, commit the transaction
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        -- If there is an error, roll back the transaction
        ROLLBACK TRANSACTION;
        THROW; -- Re-throw the caught exception to the caller
    END CATCH
END;
GO
```

6.55 OpenWebinar

Procedura do otwarcia webinaru przez uczestnika.

```
CREATE OR ALTER PROCEDURE OpenWebinar @WebinarParticipantID INT
AS
BEGIN
    DECLARE @WebinarID INT;
    DECLARE @CurrentTime DATETIME = GETDATE();
    DECLARE @WebinarLink NVARCHAR(MAX);

    -- Find the WebinarID associated with the participant
    SELECT @WebinarID = WebinarID FROM WebinarParticipants WHERE WebinarParticipantID =
    ↪  @WebinarParticipantID;

    -- Check if the Webinar is currently ongoing
    IF EXISTS (SELECT 1 FROM Webinars WHERE WebinarID = @WebinarID AND StartDate <= @CurrentTime
    ↪  AND EndDate >= @CurrentTime)
    BEGIN
        -- Check if the participant is not already marked as present
        IF NOT EXISTS (SELECT 1 FROM WebinarsAttendence WHERE WebinarParticipantID =
        ↪  @WebinarParticipantID AND WebinarID = @WebinarID)
        BEGIN
            -- Mark the participant as present
            INSERT INTO WebinarsAttendence (WebinarID, WebinarParticipantID, WasPresent)
            VALUES (@WebinarID, @WebinarParticipantID, 1);
        END

        -- Get the webinar link
        SELECT @WebinarLink = WebinarLink FROM Webinars WHERE WebinarID = @WebinarID;

    -- Return the link to the caller or perform any other needed action with it
    SELECT @WebinarLink AS WebinarLink;
END
ELSE
BEGIN
    -- Handle the case where the webinar is not ongoing
    RAISERROR('The webinar is not currently ongoing or does not exist.', 16, 1);
END
END;
GO
```

6.56 DisplayWebinarRecording

Procedura do wyświetlania nagrania webinaru.

```
CREATE OR ALTER PROCEDURE DisplayWebinarRecording @WebinarParticipantID INT
AS
BEGIN
    DECLARE @WebinarID INT;
    DECLARE @RecordingReleaseDate DATE;
    DECLARE @AddedAt DATETIME;
    DECLARE @RecordingLink NVARCHAR(MAX);
    DECLARE @AccessDays INT;
    DECLARE @AccessStartDate DATETIME;
    DECLARE @AccessEndDate DATETIME;

    -- Find the WebinarID and AddedAt for the participant
    SELECT @WebinarID = WebinarID, @AddedAt = AddedAt
    FROM WebinarParticipants
    WHERE WebinarParticipantID = @WebinarParticipantID;

    -- Get the recording release date and recording link for the webinar
    SELECT @RecordingReleaseDate = RecordingReleaseDate, @RecordingLink = RecordingLink
    FROM Webinars
    WHERE WebinarID = @WebinarID;

    -- Check if recording is available
    IF @RecordingReleaseDate IS NOT NULL AND @RecordingLink IS NOT NULL
    BEGIN
        -- Get the number of access days
        SET @AccessDays = dbo.GetRecordingAccessDays(@WebinarID);

        -- Calculate the start date for recording access using the later of RecordingReleaseDate
        ↪  or AddedAt
        SET @AccessStartDate = CASE
                                WHEN @RecordingReleaseDate > @AddedAt THEN @RecordingReleaseDate
                                ELSE @AddedAt
                               END;

        -- Calculate the end date for recording access
        SET @AccessEndDate = DATEADD(DAY, @AccessDays, @AccessStartDate);

        -- Check if the current date is within the access period
        IF GETDATE() <= @AccessEndDate
        BEGIN
            -- Participant is within the access period, return the recording link
            SELECT @RecordingLink AS RecordingLink;
        END
        ELSE
        BEGIN
            -- Participant is not within the access period or other conditions not met
            RAISERROR('Access to the webinar recording is either not available or the access
            ↪  period has expired.', 16, 1
);
END
END
ELSE
BEGIN
-- Recording is not available for this webinar
RAISERROR('There is no recording available for this webinar.', 16, 1);
END
END;
GO
```

6.57 UpdateMissingWebinarAttendance

Procedura do aktualizacji brakującej obecności na webinarze.

```
CREATE OR ALTER PROCEDURE UpdateMissingWebinarAttendance
    @WebinarID INT
AS
BEGIN
    -- Insert attendance records only for participants who were not present (WasPresent = 0)
    INSERT INTO WebinarsAttendence (WebinarID, WebinarParticipantID, WasPresent)
```

```
    SELECT @WebinarID, WP.WebinarParticipantID, 0
    FROM WebinarParticipants WP
    LEFT JOIN WebinarsAttendence WA ON WP.WebinarParticipantID = WA.WebinarParticipantID AND
    ↪  WA.WebinarID = @WebinarID
    WHERE WP.WebinarID = @WebinarID
    AND WA.WebinarParticipantID IS NULL; -- Exclude participants who are already in the
    ↪  WebinarsAttendence table

    -- Print confirmation message for updating attendance list
    PRINT 'Updated attendance list for Webinar ' + CAST(@WebinarID AS NVARCHAR(10));
END;
GO
```

6.58 CloseWebinar

Procedura do zamknięcia webinaru.

```
CREATE OR ALTER PROCEDURE CloseWebinar
    @WebinarID INT
AS
BEGIN
    -- Update the ClosedAt column in the Products table for the specified WebinarID
    UPDATE Products
    SET ClosedAt = GETDATE()
    WHERE ProductID = @WebinarID;

    -- Print confirmation message
    PRINT 'Webinar with ID ' + CAST(@WebinarID AS NVARCHAR(10)) + ' has been closed.';
END;
GO
```

6.59 ModifyWebinarRecording

Procedura do modyfikacji nagrania z Webinaru.

```
CREATE OR ALTER PROCEDURE ModifyWebinarRecording
    @WebinarID INT,
    @NewRecordingLink NVARCHAR(MAX)
AS
BEGIN
    UPDATE Webinars
    SET RecordingLink = @NewRecordingLink
    WHERE WebinarID = @WebinarID;
END;
GO
```

6.60 CreateNewPerson

Procedura do tworzenia nowej osoby w bazie danych.

```
CREATE OR ALTER PROCEDURE CreateNewPerson
    @FirstName NVARCHAR(MAX),
    @LastName NVARCHAR(500),
    @BirthDate DATE,
    @Address NVARCHAR(500),
    @City NVARCHAR(500),
    @Region NVARCHAR(500),
    @PostalCode NVARCHAR(20),
    @Country NVARCHAR(500),
    @Phone NVARCHAR(20),
    @Email NVARCHAR(500)
AS
BEGIN
    INSERT INTO People (
        FirstName,
        LastName,
        BirthDate,
```

```
        Address,
        City,
        Region,
        PostalCode,
        Country,
        Phone,
        Email
    )
    VALUES (
        @FirstName,
        @LastName,
        @BirthDate,
        @Address,
        @City,
        @Region,
        @PostalCode,
        @Country,
        @Phone,
        @Email
    );

    -- Print confirmation message
    PRINT 'New person added successfully.';
END;
GO
```

6.61 UpdatePersonData

Procedura do aktualizacji danych osobowych istniejącej osoby.

```
CREATE OR ALTER PROCEDURE UpdatePersonData
    @PersonID INT,
    @NewFirstName NVARCHAR(MAX) = NULL,
    @NewLastName NVARCHAR(500) = NULL,
    @NewBirthDate DATE = NULL,
    @NewAddress NVARCHAR(500) = NULL,
    @NewCity NVARCHAR(500) = NULL,
    @NewRegion NVARCHAR(500) = NULL,
    @NewPostalCode NVARCHAR(20) = NULL,
    @NewCountry NVARCHAR(500) = NULL,
    @NewPhone NVARCHAR(20) = NULL,
    @NewEmail NVARCHAR(500) = NULL
AS
BEGIN
    -- Update only the fields that are provided (non-null)
    UPDATE People
    SET FirstName = COALESCE(@NewFirstName, FirstName),
        LastName = COALESCE(@NewLastName, LastName),
        BirthDate = COALESCE(@NewBirthDate, BirthDate),
        Address = COALESCE(@NewAddress, Address),
        City = COALESCE(@NewCity, City),
        Region = COALESCE(@NewRegion, Region),
        PostalCode = COALESCE(@NewPostalCode, PostalCode),
        Country = COALESCE(@NewCountry, Country),
        Phone = COALESCE(@NewPhone, Phone),
        Email = COALESCE(@NewEmail, Email)
    WHERE PersonID = @PersonID;

    -- Print confirmation message
    PRINT 'Person data updated successfully.';
END;
GO
```

6.62 RemovePerson

Procedura do usunięcia osoby z bazy danych.

```sql
CREATE OR ALTER PROCEDURE RemovePerson
    @PersonID INT
AS
BEGIN
    DELETE FROM People
    WHERE PersonID = @PersonID;

    -- Print confirmation message
    PRINT 'Person removed successfully.';
END;
GO
```

6.63 AddUser

Procedura do dodawania nowego użytkownika.

```sql
CREATE OR ALTER PROCEDURE AddUser
    @FirstName NVARCHAR(MAX),
    @LastName NVARCHAR(500),
    @BirthDate DATE,
    @Address NVARCHAR(500),
    @City NVARCHAR(500),
    @Region NVARCHAR(500),
    @PostalCode NVARCHAR(20),
    @Country NVARCHAR(500),
    @Phone NVARCHAR(20),
    @Email NVARCHAR(500),
    @UserID INT OUTPUT  -- Output parameter to return the generated UserID
AS
BEGIN
    BEGIN TRY
        -- Insert user data into People table
        INSERT INTO People (
            FirstName, LastName, BirthDate, Address, City, Region, PostalCode, Country, Phone,
            ↪ Email
        )
        VALUES (
            @FirstName, @LastName, @BirthDate, @Address, @City, @Region, @PostalCode, @Country,
            ↪ @Phone, @Email
        );

        -- Get the generated UserID (same as PersonID)
        SET @UserID = SCOPE_IDENTITY();

        -- Insert user data into Users table
        INSERT INTO Users (UserID)
        VALUES (@UserID);

        -- Print confirmation message
        PRINT 'User added successfully.';
    END TRY
    BEGIN CATCH
        -- Handle any errors (e.g., check constraint violations)
        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END;
GO
```

6.64 AddEmployee

Procedura do dodawania nowego pracownika.

```sql
CREATE OR ALTER PROCEDURE AddEmployee
    @FirstName NVARCHAR(MAX),
    @LastName NVARCHAR(500),
    @BirthDate DATE,
    @Address NVARCHAR(500),
    @City NVARCHAR(500),
```

```
    @Region NVARCHAR(500),
    @PostalCode NVARCHAR(20),
    @Country NVARCHAR(500),
    @Phone NVARCHAR(20),
    @Email NVARCHAR(500),
    @HireDate DATE,
    @EmployeeID INT OUTPUT  -- Output parameter to return the generated EmployeeID
AS
BEGIN
    BEGIN TRY
        -- Insert employee data into People table
        INSERT INTO People (
            FirstName, LastName, BirthDate, Address, City, Region, PostalCode, Country, Phone,
            ↪ Email
        )
        VALUES (
            @FirstName, @LastName, @BirthDate, @Address, @City, @Region, @PostalCode, @Country,
            ↪ @Phone, @Email
        );

        -- Get the generated EmployeeID (same as PersonID)
        SET @EmployeeID = SCOPE_IDENTITY();

        -- Insert employee data into Employees table
        INSERT INTO Employees (EmployeeID, HireDate, IsActive)
        VALUES (@EmployeeID, @HireDate, 1); -- IsActive is set to 1 by default

        -- Print confirmation message
        PRINT 'Employee added successfully.';
    END TRY
    BEGIN CATCH
        -- Handle any errors (e.g., check constraint violations)
        PRINT 'Error: ' + ERROR_MESSAGE();
    END CATCH;
END;
GO
```

6.65 AddProductToCart

Procedura do dodawania produktu do koszyka użytkownika.

```
CREATE OR ALTER PROCEDURE AddProductToCart
    @UserID INT,
    @ProductID INT
AS
BEGIN
    -- Check if the product is not already in the cart
    IF NOT EXISTS (SELECT 1 FROM Carts WHERE UserID = @UserID AND ProductID = @ProductID)
    BEGIN
        -- Insert the product into the cart if it doesn't already exist
        INSERT INTO Carts (UserID, ProductID, AddedAt)
        VALUES (@UserID, @ProductID, GETDATE());

        PRINT 'Product added to the cart successfully.';
    END
    ELSE
    BEGIN
        PRINT 'Product already exists in the cart.';
    END;
END;
GO
```

6.66 RemoveProductFromCart

Procedura do usuwania produktu z koszyka użytkownika.

```sql
CREATE OR ALTER PROCEDURE RemoveProductFromCart
    @UserID INT,
    @ProductID INT
AS
BEGIN
    -- Delete the product from the cart (Carts table)
    DELETE FROM Carts
    WHERE UserID = @UserID AND ProductID = @ProductID;

    PRINT 'Product removed from the cart successfully.';
END;
GO
```

6.67 SendDiploma

Procedura do wysyłania dyplomu.

```sql
CREATE OR ALTER PROCEDURE SendDiploma
    @UserID int,
    @ProductID int,
    @DiplomaData nvarchar(max)  -- This parameter represents the diploma data
AS
BEGIN
    -- Insert the diploma sending record into the DiplomasSent table
    INSERT INTO DiplomasSent (UserID, ProductID, DiplomaFile)
    VALUES (@UserID, @ProductID, @DiplomaData);

    -- Print confirmation message
    PRINT 'Diploma sent successfully.';
END;
GO
```

6.68 AddRole

Procedura do dodawania nowej roli.

```sql
CREATE OR ALTER PROCEDURE AddRole
    @RoleName nvarchar(200)
AS
BEGIN
    -- Insert the new role into the Roles table
    INSERT INTO Roles (RoleName)
    VALUES (@RoleName);

    -- Print confirmation message
    PRINT 'Role added successfully.';
END;
GO
```

6.69 ModifyRole

Procedura do modyfikacji roli.

```sql
CREATE OR ALTER PROCEDURE ModifyRole
    @RoleID int,
    @NewRoleName nvarchar(200)
AS
BEGIN
    -- Update the Roles table
    UPDATE Roles
    SET RoleName = @NewRoleName
    WHERE RoleID = @RoleID;

    -- Print confirmation message
    PRINT 'Role updated successfully.';
END;
GO
```

6.70 AddEmployeeRole

Procedura do dodawania roli pracownikowi.

```
CREATE OR ALTER PROCEDURE AddEmployeeRole
    @EmployeeID int,
    @RoleID int
AS
BEGIN
    -- Insert a new association of role and employee into the EmployeeRoles table
    INSERT INTO EmployeeRoles (EmployeeID, RoleID)
    VALUES (@EmployeeID, @RoleID);

    -- Print confirmation message
    PRINT 'Role added to employee successfully.';
END;
GO
```

6.71 RemoveEmployeeRole

Procedura do usuwania roli od pracownika.

```
CREATE OR ALTER PROCEDURE RemoveEmployeeRole
    @EmployeeID int,
    @RoleID int
AS
BEGIN
    -- Delete the role-employee association from the EmployeeRoles table
    DELETE FROM EmployeeRoles
    WHERE EmployeeID = @EmployeeID AND RoleID = @RoleID;

    -- Print confirmation message
    PRINT 'Role removed from employee successfully.';
END;
GO
```

6.72 ChangeProductPrice

Procedura do zmiany ceny produktu.

```
CREATE OR ALTER PROCEDURE ChangeProductPrice
    @ProductID int,
    @NewPrice money,
    @NewAdvancePayment money = NULL
AS
BEGIN
    -- Update the product price in the Products table
    UPDATE Products
    SET Price = @NewPrice,
        AdvancePayment = @NewAdvancePayment
    WHERE ProductID = @ProductID;

    -- Print confirmation message
    PRINT 'Product price updated successfully.';
END;
GO
```

6.73 getCourseAttendance

Funkcja zwracająca historię płatności dla danego użytkownika.

```
CREATE OR ALTER FUNCTION getCourseAttendance(@CourseID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
```

```
        cp.CourseParticipantID,
        u.UserID,
        pe.FirstName AS UserFirstName,
        pe.LastName AS UserLastName,
        CASE
            WHEN cos.CourseOfflineSessionID IS NOT NULL THEN 'Offline'
            WHEN con.CourseOnlineSessionID IS NOT NULL THEN 'Online'
            WHEN cst.CourseStationarySessionID IS NOT NULL THEN 'Stationary'
            ELSE 'Unknown'
        END AS SessionType,
        COALESCE(NULL, con.StartDate, cst.StartDate) AS StartDate,
        COALESCE(NULL, con.EndDate, cst.EndDate) AS EndDate,
        cs.ModuleID,
        m.ModuleName,
        e.PersonID AS LecturerID,
        e.FirstName AS LecturerFirstName,
        e.LastName AS LecturerLastName,
    ca.Completed as 'Completed'
    FROM
        CourseParticipants cp
    INNER JOIN
        Users u ON cp.UserID = u.UserID
    INNER JOIN
        People pe ON u.UserID = pe.PersonID
  INNER JOIN
      Courses co ON co.CourseID = cp.CourseID
    INNER JOIN
    Modules m ON  co.CourseID = m.CourseID
    INNER JOIN
        CoursesSessions cs ON cs.ModuleID = m.ModuleID
    LEFT JOIN
        CourseOfflineSessions cos ON cs.CourseSessionID = cos.CourseOfflineSessionID
    LEFT JOIN
        CourseOnlineSessions con ON cs.CourseSessionID = con.CourseOnlineSessionID
    LEFT JOIN
        CourseStationarySessions cst ON cs.CourseSessionID = cst.CourseStationarySessionID
    LEFT JOIN
        People e ON cs.LecturerID = e.PersonID
  LEFT JOIN
    CourseSessionsAttendance ca ON ca.CourseSessionID = cs.CourseSessionID AND
    ↪  ca.CourseParticipantID = cp.CourseParticipantID
    WHERE
        cp.CourseID = @CourseID
);
GO
```

6.74 EnrollUserToFreeWebinar

Procedura zapisująca użytkownika na bezpłatny webinar.

```
CREATE OR ALTER PROCEDURE EnrollUserToFreeWebinar
    @UserID INT,
    @WebinarID INT
AS
BEGIN
    -- Check if the webinar is free
    IF EXISTS (
        SELECT 1
        FROM Products P
        JOIN Webinars W ON W.WebinarID = P.ProductID
        WHERE W.WebinarID = @WebinarID AND P.Price = 0
    )
    BEGIN
        -- Check if the user is already enrolled in the webinar
        IF NOT EXISTS (
            SELECT 1
            FROM WebinarParticipants
            WHERE UserID = @UserID AND WebinarID = @WebinarID
        )
```

```
        BEGIN
            -- Insert user into WebinarParticipants
            INSERT INTO WebinarParticipants (UserID, WebinarID, WebinarPrice)
            VALUES (@UserID, @WebinarID, 0)
        END
        ELSE
        BEGIN
            -- Raise an error if the user is already enrolled
            RAISERROR ('User is already enrolled in this webinar.', 16, 1);
        END
    END
    ELSE
    BEGIN
        -- Raise an error if the webinar is not free
        RAISERROR ('The specified webinar is not free.', 16, 1);
    END
END;
GO
```

6.75 dbo.CanUserPurchasePaidWebinar

Funkcja sprawdzająca, czy użytkownik może zakupić płatny webinar.

```
CREATE OR ALTER  FUNCTION dbo.CanUserPurchasePaidWebinar
(
    @UserID INT,
    @WebinarID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @StartDate DATETIME
    DECLARE @RecordingReleaseDate DATE
    DECLARE @CanPurchase BIT = 0

    -- Retrieve Webinar Start Date and Recording Release Date
    SELECT
        @StartDate = StartDate,
        @RecordingReleaseDate = RecordingReleaseDate
    FROM
        Webinars
    WHERE
        WebinarID = @WebinarID

    -- Check if current date is before the Webinar's Start Date
    IF GETDATE() < @StartDate
    BEGIN
        -- Check if the user is not already enrolled
        IF NOT EXISTS (
            SELECT 1
            FROM WebinarParticipants
            WHERE UserID = @UserID AND WebinarID = @WebinarID
        )
        BEGIN
            -- User can purchase access to the webinar
            SET @CanPurchase = 1
        END
    END
    ELSE
    BEGIN
        -- After the Webinar has started, check if the recording is released
        IF @RecordingReleaseDate IS NOT NULL AND GETDATE() >= @RecordingReleaseDate
        BEGIN
            -- User can purchase access to the webinar recording
            SET @CanPurchase = 1
        END
    END

    -- Return the result
```

```
    RETURN @CanPurchase
END;
GO
```

6.76 ProcessWebinarPayment

Procedura przetwarzająca płatność za webinar.

```
CREATE OR ALTER PROCEDURE ProcessWebinarPayment
    @UserID INT,
    @WebinarID INT,
    @Price MONEY,
    @Status NVARCHAR(300)
AS
BEGIN
    -- Insert the payment record
    INSERT INTO Payments (UserID, ProductID, Price, Date, Status)
    VALUES (@UserID, @WebinarID, @Price, GETDATE(), @Status)

    -- Get the last inserted PaymentID
    DECLARE @PaymentID INT
    SELECT @PaymentID = SCOPE_IDENTITY()

    -- If the payment failed, just exit the procedure
    IF @Status = 'Failed'
    BEGIN
        RETURN
    END

  DECLARE @DuePostponedPayment datetime;
  DECLARE @FullPricePaymentID int;
  DECLARE @UsersPrice money;
  SELECT
    @FullPricePaymentID=FullPricePaymentID,
    @DuePostponedPayment = DuePostponedPayment,
    @UsersPrice=WebinarPrice
  FROM WebinarParticipants WHERE UserID=@UserID AND WebinarID=@WebinarID;

  IF @DuePostponedPayment IS NOT NULL AND @FullPricePaymentID IS NULL AND @UsersPrice=@Price
  BEGIN
    UPDATE WebinarParticipants
    SET FullPricePaymentID=@PaymentID
    WHERE UserID=@UserID AND WebinarID=@WebinarID;
    RETURN;
  END


    -- Check if the price matches the actual webinar price
    IF NOT EXISTS (
        SELECT 1
        FROM Products P
        JOIN Webinars W ON W.WebinarID = P.ProductID
        WHERE W.WebinarID = @WebinarID AND P.Price = @Price
    )
    BEGIN
        RAISERROR('The specified price does not match the actual webinar price.', 16, 1)
        RETURN
    END


    -- Double check using CanUserPurchasePaidWebinar
    IF dbo.CanUserPurchasePaidWebinar(@UserID, @WebinarID) = 1
    BEGIN
        -- Enroll the user to the webinar
        INSERT INTO WebinarParticipants (UserID, WebinarID, WebinarPrice, FullPricePaymentID)
        VALUES (@UserID, @WebinarID, @Price, @PaymentID)
    END
    ELSE
    BEGIN
```

```
        RAISERROR('The user cannot purchase the webinar at this time.', 16, 1)
    END
END;
GO
```

6.77 CanUserPurchaseCourse

Funkcja sprawdzająca, czy użytkownik może zakupić kurs.

```
CREATE OR ALTER  FUNCTION CanUserPurchaseCourse
(
    @UserID INT,
    @CourseID INT
)
RETURNS BIT
AS
BEGIN
    DECLARE @CanPurchase BIT = 0
    DECLARE @CourseStartDate DATETIME
    DECLARE @MaxStudents INT
    DECLARE @CurrentEnrollmentCount INT
    DECLARE @MaxDaysForPayment INT
    DECLARE @CourseClosed BIT

    -- Get course start date and MaxStudents
    SELECT
        @CourseStartDate = StartDate,
        @MaxStudents = MaxStudents,
        @CourseClosed = CASE WHEN ClosedAt IS NULL THEN 0 ELSE 1 END
    FROM Courses
  JOIN Products P ON P.ProductID = Courses.CourseID
    WHERE CourseID = @CourseID

    -- Get the current number of enrolled students
    SELECT @CurrentEnrollmentCount = COUNT(*)
    FROM CourseParticipants
    WHERE CourseID = @CourseID

    -- Check if it's too late to make the payment
    SELECT @MaxDaysForPayment = dbo.GetMaxDaysForPaymentBeforeCourseStart(@CourseID)

    -- Check if user is already enrolled
    IF NOT EXISTS (
        SELECT 1
        FROM CourseParticipants
        WHERE UserID = @UserID AND CourseID = @CourseID
    )
    BEGIN
        -- Check if course is not closed, within the payment window, and not full
        IF @CourseClosed = 0 AND
           GETDATE() < DATEADD(DAY, -@MaxDaysForPayment, @CourseStartDate) AND
           (@MaxStudents IS NULL OR @CurrentEnrollmentCount < @MaxStudents)
        BEGIN
            SET @CanPurchase = 1
        END
    END

    RETURN @CanPurchase
END;
GO
```

6.78 ProcessCoursePayment

Procedura przetwarzająca płatność za kurs.

```
CREATE OR ALTER PROCEDURE ProcessCoursePayment
    @UserID INT,
    @CourseID INT,
```

```
    @Price MONEY,
    @Status NVARCHAR(300)
AS
BEGIN
    -- Declare variable to store the new PaymentID
    DECLARE @NewPaymentID INT;

    -- Insert the payment record and store the new PaymentID
    INSERT INTO Payments (UserID, ProductID, Price, Date, Status)
    VALUES (@UserID, @CourseID, @Price, GETDATE(), @Status);

    -- Get the last inserted PaymentID
    SET @NewPaymentID = SCOPE_IDENTITY();

    -- If payment failed, do nothing more
    IF @Status = 'Failed'
        RETURN;

    -- Check if user is already enrolled in the course
    IF NOT EXISTS (SELECT * FROM CourseParticipants WHERE UserID = @UserID AND CourseID =
    ↪  @CourseID)
    BEGIN

        -- User is not enrolled, check if they can purchase the course
        DECLARE @CanPurchase BIT = dbo.CanUserPurchaseCourse(@UserID, @CourseID);
        IF @CanPurchase = 1
        BEGIN
      -- Get course details
      DECLARE @CoursePrice MONEY, @EntryFee MONEY;
      SELECT @CoursePrice = Price, @EntryFee = AdvancePayment FROM Products WHERE ProductID =
      ↪  @CourseID;

            -- Check if payment is full price or advance payment
            IF @Price IN (@CoursePrice, @EntryFee)
            BEGIN
                -- Insert into CourseParticipants
                INSERT INTO CourseParticipants (UserID, CourseID, CoursePrice, EntryFee,
                ↪  EntryFeePaymentID, FullPricePaymentID, AddedAt, Completed)
                VALUES (@UserID, @CourseID, @CoursePrice, @EntryFee,
                        CASE WHEN @Price = @EntryFee THEN @NewPaymentID ELSE NULL END,
                        CASE WHEN @Price = @CoursePrice THEN @NewPaymentID ELSE NULL END,
                        GETDATE(), 0);
            END
            ELSE
            BEGIN
                -- Raise error: Price does not match
                RAISERROR('Payment amount does not match course fees.', 16, 1);
                RETURN;
            END
        END
        ELSE
        BEGIN
            -- Raise error: Cannot purchase course
            RAISERROR('User cannot purchase the course.', 16, 1);
            RETURN;
        END
    END
    ELSE
    BEGIN
        -- User is already enrolled, check remaining payment
        DECLARE @RemainingPrice MONEY, @MaxDaysForPayment INT, @CourseStartDate DATETIME,
        ↪  @UserCoursePrice MONEY;
    SELECT @RemainingPrice = CoursePrice-EntryFee,
         @UserCoursePrice=CoursePrice
    FROM CourseParticipants WHERE UserID=@UserID AND CourseID=@CourseID;
        SELECT @CourseStartDate = StartDate
        FROM Courses
        WHERE CourseID = @CourseID;
        SET @MaxDaysForPayment = dbo.GetMaxDaysForPaymentBeforeCourseStart(@CourseID);
```

```
    DECLARE @FullPaymentID int;
    DECLARE @EntryFeePaymentID int;
    DECLARE @RemainingPaymentID int;
    DECLARE @DuePostponedPayment datetime;

    SELECT @FullPaymentID = FullPricePaymentID, @EntryFeePaymentID=EntryFeePaymentID,
        @RemainingPaymentID=RemainingPaymentID, @DuePostponedPayment = DuePostponedPayment
    FROM CourseParticipants
    WHERE UserID=@UserID AND CourseID = @CourseID;


    IF @FullPaymentID IS NULL AND @EntryFeePaymentID IS NULL AND @RemainingPaymentID IS NULL AND
      @DuePostponedPayment IS NOT NULL AND @Price=@UserCoursePrice
    BEGIN
      UPDATE CourseParticipants
      SET FullPricePaymentID=@NewPaymentID
      WHERE UserID=@UserID AND CourseID=@CourseID;
      RETURN;
    END


        IF @Price = @RemainingPrice AND GETDATE() < DATEADD(DAY, -@MaxDaysForPayment,
        ↪  @CourseStartDate) AND
           NOT EXISTS (SELECT * FROM CourseParticipants WHERE UserID = @UserID AND CourseID =
           ↪  @CourseID AND RemainingPaymentID IS NOT NULL)
        BEGIN
            -- Update CourseParticipants with remaining payment
            UPDATE CourseParticipants
            SET RemainingPaymentID = @NewPaymentID
            WHERE UserID = @UserID AND CourseID = @CourseID;
        END
        ELSE
        BEGIN
            -- Raise error: Invalid payment or conditions not met
            RAISERROR('Invalid payment amount or conditions for remaining payment not met.', 16,
            ↪  1);
            RETURN;
        END
    END
END;
GO
```

6.79 dbo.CanUserPurchasePublicStudySession

Funkcja sprawdzająca, czy użytkownik może zakupić pojedyńcze spotkanie studyjne.

```
CREATE OR ALTER FUNCTION dbo.CanUserPurchasePublicStudySession(@UserID INT, @PublicStudySessionID
↪  INT)
RETURNS BIT
AS
BEGIN
    DECLARE @CanEnroll BIT = 1; -- Default: user can enroll

    -- Check if the session is still open
    IF EXISTS (SELECT 1 FROM Products WHERE ProductID = @PublicStudySessionID AND ClosedAt IS NOT
    ↪  NULL)
    OR GETDATE() >= (SELECT StartDate FROM PublicStudySessions PSS
          JOIN StudiesSessions SS ON PSS.StudiesSessionID = SS.StudiesSessionID
          WHERE PSS.PublicStudySessionID=@PublicStudySessionID)
    BEGIN
        SET @CanEnroll = 0; -- Session is closed
    END

    -- Check if user is already enrolled
    IF @CanEnroll = 1 AND EXISTS (SELECT 1 FROM PublicStudySessionParticipants WHERE UserID =
    ↪  @UserID AND PublicStudySessionID = @PublicStudySessionID)
    BEGIN
        SET @CanEnroll = 0; -- User is already enrolled
```

```
    END

    DECLARE @MaxStudents INT, @ExternalEnrollments INT, @OrdinaryEnrollments INT;

    -- Get the MaxStudents value for this session
    SELECT @MaxStudents = ss.MaxStudents
    FROM PublicStudySessions pss
    INNER JOIN StudiesSessions ss ON pss.StudiesSessionID = ss.StudiesSessionID
    WHERE pss.PublicStudySessionID = @PublicStudySessionID;

    -- Count the number of external participants enrolled
    SELECT @ExternalEnrollments = COUNT(*)
    FROM PublicStudySessionParticipants
    WHERE PublicStudySessionID = @PublicStudySessionID;

    -- Count the number of ordinary students enrolled
    SELECT @OrdinaryEnrollments = COUNT(*)
    FROM Students s
    INNER JOIN Studies st ON st.StudiesID = s.StudiesID
    INNER JOIN Subjects sub ON sub.StudiesID = st.StudiesID
    INNER JOIN StudiesSessions ss ON ss.SubjectID = sub.SubjectID
    INNER JOIN PublicStudySessions pss ON pss.StudiesSessionID = ss.StudiesSessionID
    WHERE pss.PublicStudySessionID = @PublicStudySessionID;

    -- Check if the maximum number of students has been reached
    DECLARE @TotalEnrollments INT = @ExternalEnrollments + @OrdinaryEnrollments;
    IF @TotalEnrollments >= @MaxStudents
    BEGIN
        SET @CanEnroll = 0; -- Enrollment limit reached
    END

    RETURN @CanEnroll;
END;
GO
```

6.80 ProcessPublicStudySessionPayment

Procedura przetwarzająca płatność za publiczną sesję studiów.

```
CREATE OR ALTER PROCEDURE ProcessPublicStudySessionPayment
    @UserID INT,
    @PublicStudySessionID INT,
    @Price MONEY,
    @Status NVARCHAR(300)
AS
BEGIN
    -- Insert into Payments and store the new PaymentID
    DECLARE @NewPaymentID INT;
    INSERT INTO Payments (UserID, ProductID, Price, Date, Status)
    VALUES (@UserID, @PublicStudySessionID, @Price, GETDATE(), @Status);

    SET @NewPaymentID = SCOPE_IDENTITY();

    -- If the payment is unsuccessful, end the procedure
    IF @Status = 'Failed'
        RETURN;

    -- Check if the price matches the PublicStudySession price
    DECLARE @SessionPrice MONEY;
    SELECT @SessionPrice = P.Price
    FROM PublicStudySessions PSS
    JOIN Products P ON PSS.PublicStudySessionID = P.ProductID
    WHERE PSS.PublicStudySessionID = @PublicStudySessionID;

    IF @Price != @SessionPrice
    BEGIN
        RAISERROR('Incorrect price for the Public Study Session', 16, 1);
        RETURN;
    END
```

```
  DECLARE @DuePostponedPayment datetime;
  DECLARE @FullPaymentID int;
  DECLARE @UsersSessionPrice int;
  SELECT @DuePostponedPayment = DuePostponedPayment,
       @FullPaymentID=FullPricePaymentID,
       @UsersSessionPrice = SessionPrice
  FROM PublicStudySessionParticipants
  WHERE UserID=@UserID AND PublicStudySessionID=@PublicStudySessionID;

  IF @DuePostponedPayment IS NOT NULL AND @FullPaymentID IS NULL AND @UsersSessionPrice=@Price
  BEGIN
    UPDATE PublicStudySessionParticipants
    SET FullPricePaymentID=@NewPaymentID
    WHERE UserID=@UserID AND PublicStudySessionID=@PublicStudySessionID
    RETURN;
  END

    -- Check if the user can purchase the public study session
    IF dbo.CanUserPurchasePublicStudySession(@UserID, @PublicStudySessionID) = 0
    BEGIN
        RAISERROR('User cannot purchase this Public Study Session', 16, 1);
        RETURN;
    END

    -- Insert into PublicStudySessionParticipants and link the payment
    INSERT INTO PublicStudySessionParticipants (UserID, PublicStudySessionID, SessionPrice,
    ↪  AddedAt, FullPricePaymentID)
    VALUES (@UserID, @PublicStudySessionID, @Price, GETDATE(), @NewPaymentID);
END;
GO
```

6.81 dbo.CanUserPurchaseStudies

Funkcja sprawdzająca, czy użytkownik może zakupić studia.

```
CREATE OR ALTER FUNCTION dbo.CanUserPurchaseStudies(@UserID INT, @StudiesID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @CanPurchase BIT = 1; -- Default: user can purchase the studies

    -- Check if the studies are still open
    IF EXISTS (SELECT 1 FROM Products WHERE ProductID = @StudiesID AND ClosedAt IS NOT NULL)
       OR GETDATE() > (SELECT StartDate FROM Studies WHERE StudiesID = @StudiesID)
    BEGIN
        SET @CanPurchase = 0; -- Studies are closed
    END

    -- Check if user is already enrolled in the studies
    IF @CanPurchase = 1 AND EXISTS (SELECT 1 FROM Students WHERE UserID = @UserID AND StudiesID =
    ↪  @StudiesID)
    BEGIN
        SET @CanPurchase = 0; -- User is already enrolled
    END

    -- Check if the maximum number of students has been reached
    IF @CanPurchase = 1
    BEGIN
        DECLARE @MaxStudents INT;
        SELECT @MaxStudents = MaxStudents FROM Studies WHERE StudiesID = @StudiesID;

        DECLARE @CurrentEnrollments INT;
        SELECT @CurrentEnrollments = COUNT(*) FROM Students WHERE StudiesID = @StudiesID;

        IF @CurrentEnrollments >= @MaxStudents
        BEGIN
            SET @CanPurchase = 0; -- Maximum number of students reached
        END
```

```
    END

    -- Check if it's not too late to purchase the studies
    IF @CanPurchase = 1
    BEGIN
        DECLARE @MaxDaysForPayment INT;
        SET @MaxDaysForPayment = dbo.GetMaxDaysForPaymentBeforeStudiesStart(@StudiesID);
        DECLARE @StudiesStartDate DATETIME;
        SELECT @StudiesStartDate = StartDate FROM Studies WHERE StudiesID = @StudiesID;

        IF GETDATE() > DATEADD(DAY, -@MaxDaysForPayment, @StudiesStartDate)
        BEGIN
            SET @CanPurchase = 0; -- Too late to enroll
        END
    END

    -- Check if user has completed the previous semester (if applicable)
    IF @CanPurchase = 1
    BEGIN
        DECLARE @SemesterNumber INT, @FieldOfStudiesID INT;
        SELECT @SemesterNumber = SemesterNumber, @FieldOfStudiesID = FieldOfStudiesID FROM
        ↪  Studies WHERE StudiesID = @StudiesID;

        IF @SemesterNumber > 1
        BEGIN
            DECLARE @PreviousSemesterID INT;
            SELECT @PreviousSemesterID = StudiesID FROM Studies
            WHERE FieldOfStudiesID = @FieldOfStudiesID AND SemesterNumber = @SemesterNumber - 1;

            IF NOT EXISTS (SELECT 1 FROM Students WHERE UserID = @UserID AND StudiesID =
            ↪  @PreviousSemesterID AND Completed = 1)
            BEGIN
                SET @CanPurchase = 0; -- User did not complete the previous semester
            END
        END
    END

    RETURN @CanPurchase;
END;
GO
```

6.82 ProcessStudiesPayment

Procedura przetwarzająca płatność za studia.

```
CREATE OR ALTER PROCEDURE ProcessStudiesPayment
    @UserID int,
    @StudiesID int,
    @Price money,
    @Status nvarchar(300)
AS
BEGIN
    -- Declare a variable to store the newly created payment ID
    DECLARE @NewPaymentID int;

    -- Insert the payment into Payments table and store its ID
    INSERT INTO Payments (UserID, ProductID, Price, Date, Status)
    VALUES (@UserID, @StudiesID, @Price, GETDATE(), @Status);

    SET @NewPaymentID = SCOPE_IDENTITY();

    -- Exit if payment status is 'Failed'
    IF @Status = 'Failed' RETURN;

    -- Check if the user is not already a student for the specified studies
    IF NOT EXISTS (SELECT * FROM Students WHERE UserID = @UserID AND StudiesID = @StudiesID)
    BEGIN
        -- Check if the price is either EntryFee or Full Price
        DECLARE @EntryFee money, @FullPrice money;
```

```
    SELECT @EntryFee = AdvancePayment, @FullPrice = Price
    FROM Products
    WHERE ProductID = @StudiesID;

    -- Perform additional check
    IF dbo.CanUserPurchaseStudies(@UserID, @StudiesID) = 1
    BEGIN
        IF @Price = @EntryFee OR @Price = @FullPrice
        BEGIN
            -- Insert record into Students
            INSERT INTO Students (UserID, StudiesID, StudiesPrice, EntryFee, AddedAt,
            ↪  FullPaymentID, EntryFeePaymentID)
            VALUES (@UserID, @StudiesID, @FullPrice, @EntryFee, GETDATE(),
                    CASE WHEN @Price = @FullPrice THEN @NewPaymentID ELSE NULL END,
                    CASE WHEN @Price = @EntryFee THEN @NewPaymentID ELSE NULL END);
        END
        ELSE
        BEGIN
            RAISERROR('Invalid price amount.', 16, 1);
            RETURN;
        END
    END
    ELSE
    BEGIN
        RAISERROR('User cannot purchase these studies.', 16, 1);
        RETURN;
    END
END
ELSE
BEGIN
    DECLARE @RemainingPrice money;
DECLARE @StudiesPrice money;
DECLARE @DuePostponedPayment datetime;
    SELECT @RemainingPrice = StudiesPrice - EntryFee ,
     @StudiesPrice = StudiesPrice,
     @DuePostponedPayment = DuePostponedPayment
    FROM Students
    WHERE UserID = @UserID AND StudiesID = @StudiesID;

    -- Consult GetMaxDaysForPaymentBeforeStudiesStart
    DECLARE @MaxDaysForPayment int, @StudiesStartDate datetime, @RemainingPaymentID int,
      @EntryFeePaymentID int, @FullPaymentID int
    SELECT @MaxDaysForPayment = dbo.GetMaxDaysForPaymentBeforeStudiesStart(@StudiesID);
    SELECT @StudiesStartDate = (SELECT StartDate FROM Studies WHERE StudiesID=@StudiesID);
    SELECT @RemainingPaymentID = RemainingPaymentID,
           @EntryFeePaymentID = EntryFeePaymentID,
           @FullPaymentID = FullPaymentID
      FROM Students WHERE UserID = @UserID AND StudiesID=@StudiesID;

IF @Price = @StudiesPrice AND @FullPaymentID IS NULL AND @EntryFeePaymentID IS NULL
  AND @RemainingPaymentID IS NULL AND @DuePostponedPayment IS NOT NULL
    BEGIN
    UPDATE Students
    SET FullPaymentID = @NewPaymentID
    WHERE UserID = @UserID AND StudiesID = @StudiesID;
  RETURN;
END


    IF @Price = @RemainingPrice AND GETDATE() < DATEADD(DAY, -@MaxDaysForPayment,
    ↪  @StudiesStartDate)
    AND @RemainingPaymentID IS NULL
    BEGIN
        -- Link up the remaining price payment
        UPDATE Students
        SET RemainingPaymentID = @NewPaymentID
        WHERE UserID = @UserID AND StudiesID = @StudiesID;
    END
    ELSE
```

```
        BEGIN
            RAISERROR('Payment amount does not match the remaining price or it is too late for
            ↪  payment.', 16, 1);
            RETURN;
        END

    END
END;
GO
```

6.83 ProcessPayment

Procedura ogólna przetwarzająca płatność, delegująca do odpowiedniej procedury w zależności od rodzaju produktu.

```
CREATE OR ALTER PROCEDURE ProcessPayment
    @UserID int,
    @ProductID int,
    @Price money,
    @Status nvarchar(300)
AS
BEGIN
    -- Declare variable to store the product type
    DECLARE @ProductType nvarchar(max);

    -- Get the product type
    SELECT @ProductType = ProductType
    FROM Products
    WHERE ProductID = @ProductID;

    -- Check the product type and delegate to the appropriate procedure
    IF @ProductType = 'webinar'
    BEGIN
        -- Call ProcessWebinarPayment procedure
        EXEC ProcessWebinarPayment @UserID, @ProductID, @Price, @Status;
    END
    ELSE IF @ProductType = 'course'
    BEGIN
        -- Call ProcessCoursePayment procedure
        EXEC ProcessCoursePayment @UserID, @ProductID, @Price, @Status;
    END
    ELSE IF @ProductType = 'studies'
    BEGIN
        -- Call ProcessStudiesPayment procedure
        EXEC ProcessStudiesPayment @UserID, @ProductID, @Price, @Status;
    END
    ELSE IF @ProductType = 'public study session'
    BEGIN
        -- Call ProcessPublicStudySessionPayment procedure
        EXEC ProcessPublicStudySessionPayment @UserID, @ProductID, @Price, @Status;
    END
    ELSE
    BEGIN
        -- Handle unknown product types
        RAISERROR('Unknown product type.', 16, 1);
    END
END;
GO
```

6.84 EnrollUserWithoutImmediatePayment

Procedura umożliwiająca odroczenie płatności za produkt.

```
CREATE OR ALTER PROCEDURE EnrollUserWithoutImmediatePayment
    @UserID int,
    @ProductID int,
    @DuePostponedPayment datetime
AS
```

```
BEGIN
    -- Variables for product details
    DECLARE @ProductPrice money
    DECLARE @AdvancePayment money
    DECLARE @ProductType nvarchar(max)
    DECLARE @Exists bit

    -- Initialize the variable to check existence
    SET @Exists = 0

    -- Retrieve the price, advance payment, and type of the product from the Products table
    SELECT
        @ProductPrice = Price,
        @AdvancePayment = ISNULL(AdvancePayment, 0),
        @ProductType = ProductType
    FROM Products
    WHERE ProductID = @ProductID

    -- Check if the user is already enrolled in the product
    IF @ProductType = 'studies'
    BEGIN
        IF EXISTS (SELECT 1 FROM Students WHERE UserID = @UserID AND StudiesID = @ProductID)
            SET @Exists = 1
    END
    ELSE IF @ProductType = 'course'
    BEGIN
        IF EXISTS (SELECT 1 FROM CourseParticipants WHERE UserID = @UserID AND CourseID =
        ↪  @ProductID)
            SET @Exists = 1
    END
    ELSE IF @ProductType = 'webinar'
    BEGIN
        IF EXISTS (SELECT 1 FROM WebinarParticipants WHERE UserID = @UserID AND WebinarID =
        ↪  @ProductID)
            SET @Exists = 1
    END
    ELSE IF @ProductType = 'public study session'
    BEGIN
        IF EXISTS (SELECT 1 FROM PublicStudySessionParticipants WHERE UserID = @UserID AND
        ↪  PublicStudySessionID = @ProductID)
            SET @Exists = 1
    END

    -- Insert the record only if the user is not already enrolled
    IF @Exists = 0
    BEGIN
        -- Insert logic based on product type
        IF @ProductType = 'studies'
        BEGIN
            INSERT INTO Students (UserID, StudiesID, StudiesPrice, EntryFee, DuePostponedPayment,
            ↪  Completed)
            VALUES (@UserID, @ProductID, @ProductPrice, @AdvancePayment, @DuePostponedPayment, 0)
        END
        ELSE IF @ProductType = 'course'
        BEGIN
            INSERT INTO CourseParticipants (UserID, CourseID, CoursePrice, EntryFee,
            ↪  DuePostponedPayment, Completed)
            VALUES (@UserID, @ProductID, @ProductPrice, @AdvancePayment, @DuePostponedPayment, 0)
        END
        ELSE IF @ProductType = 'webinar'
        BEGIN
            INSERT INTO WebinarParticipants (UserID, WebinarID, WebinarPrice,
            ↪  DuePostponedPayment)
            VALUES (@UserID, @ProductID, @ProductPrice, @DuePostponedPayment)
        END
        ELSE IF @ProductType = 'public study session'
        BEGIN
            INSERT INTO PublicStudySessionParticipants (UserID, PublicStudySessionID,
            ↪  SessionPrice, DuePostponedPayment)
```

```
            VALUES (@UserID, @ProductID, @ProductPrice, @DuePostponedPayment)
        END
    END
    ELSE
    BEGIN
        -- Handle the case where the user is already enrolled
        PRINT 'User is already enrolled in this product.'
    END
END;
GO
```

6.85 GetEmployeeTimetable

Funkcja zwracająca harmonogram pracownika w określonym przedziale czasowym

```
CREATE OR ALTER FUNCTION GetEmployeeTimetable
(
    @EmployeeID INT,
    @StartDate DATETIME,
    @EndDate DATETIME
)
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM EmployeeTimeTable
    WHERE EmployeeID = @EmployeeID
      AND StartDate >= @StartDate
      AND EndDate <= @EndDate
);
GO
```

6.86 GetUserTimeTable

Funkcja zwracająca harmonogram zajęć użytkownika w określonym przedziale czasowym

```
CREATE OR ALTER FUNCTION GetUserTimeTable
(
    @UserID INT,
    @StartDate DATETIME,
    @EndDate DATETIME
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        T.*
    FROM
        TimeTableForAllUsers T
    WHERE
        T.UserID = @UserID
        AND T.StartDate >= @StartDate
        AND T.EndDate <= @EndDate
);
GO
```

7 Triggery

7.1 tr_People_AfterUpdate

Trigger zapisujący w tabeli PeopleDataChangeHistory zmiany danych osób, takie jak imię, nazwisko, data urodzenia, adres, miasto, region, kod pocztowy, kraj, email i telefon.

```sql
CREATE OR ALTER TRIGGER tr_People_AfterUpdate
ON People
AFTER UPDATE
AS
BEGIN
    -- Insert the changed data into the PeopleDataChangeHistory table
    INSERT INTO PeopleDataChangeHistory (
        PersonID,
        ChangedAt,
        New_FirstName,
        Old_FirstName,
        New_LastName,
        Old_LastName,
        New_BirthDate,
        Old_BirthDate,
        New_Address,
        Old_Address,
        New_City,
        Old_City,
        New_Region,
        Old_Region,
        New_PostalCode,
        Old_PostalCode,
        New_Country,
        Old_Country,
        New_Email,
        Old_Email,
        New_Phone,
        Old_Phone
    )
    SELECT
        i.PersonID,
        GETDATE(),
        i.FirstName,
        d.FirstName,
        i.LastName,
        d.LastName,
        i.BirthDate,
        d.BirthDate,
        i.Address,
        d.Address,
        i.City,
        d.City,
        i.Region,
        d.Region,
        i.PostalCode,
        d.PostalCode,
        i.Country,
        d.Country,
        i.Email,
        d.Email,
        i.Phone,
        d.Phone
    FROM
        inserted i
    JOIN
        deleted d ON i.PersonID = d.PersonID
    WHERE
        i.FirstName <> d.FirstName
        OR i.LastName <> d.LastName
        OR i.BirthDate <> d.BirthDate
        OR i.Address <> d.Address
        OR i.City <> d.City
        OR i.Region <> d.Region
        OR i.PostalCode <> d.PostalCode
        OR i.Country <> d.Country
        OR i.Email <> d.Email
        OR i.Phone <> d.Phone;
END;
```

```
GO
```

7.2 trg_RemoveProductFromCart

Trigger rejestrujący w tabeli CartHistory usunięcie produktu z koszyka, zawierający informacje o użytkowniku, produkcie, dacie dodania i usunięcia.

```
CREATE OR ALTER TRIGGER trg_RemoveProductFromCart
ON Carts
AFTER DELETE
AS
BEGIN
    INSERT INTO CartHistory (UserID, ProductID, AddedAt, RemovedAt)
    SELECT d.UserID, d.ProductID, d.AddedAt, GETDATE()
    FROM deleted d;
END;
GO
```

7.3 RecordPriceChange

Trigger zapisujący w tabeli ProductPriceChangeHistory zmiany ceny i zaliczki produktów, zawierający identyfikator produktu, starą i nową cenę, starą i nową zaliczkę oraz datę zmiany.

```
CREATE OR ALTER TRIGGER RecordPriceChange
ON Products
AFTER UPDATE
AS
BEGIN
    -- Insert the price change into the ProductPriceChangeHistory table
    INSERT INTO ProductPriceChangeHistory (ProductID, Old_Price, New_Price, Old_AdvancePayment,
    ↪  New_AdvancePayment, ChangedAt)
    SELECT
        d.ProductID,
        d.Price, -- Old price
        i.Price, -- New price
        d.AdvancePayment, -- Old advance payment
        i.AdvancePayment, -- New advance payment
        GETDATE()
    FROM deleted d
    INNER JOIN inserted i ON d.ProductID = i.ProductID
    WHERE d.Price <> i.Price OR d.AdvancePayment <> i.AdvancePayment;
END;
GO
```

7.4 trg_CheckOverlap_MinAttendanceCourse

Trigger sprawdzający nakładanie się zakresów dat w wymaganiach dotyczących minimalnej frekwencji na kursach.

```
CREATE OR ALTER TRIGGER trg_CheckOverlap_MinAttendanceCourse
ON MinAttendancePercentageToPassCourse
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM MinAttendancePercentageToPassCourse m
        INNER JOIN inserted i ON m.CourseID = i.CourseID OR (m.CourseID IS NULL AND i.CourseID IS
        ↪  NULL)
        WHERE
            m.MinAttendancePercentageToPassCourseID <> i.MinAttendancePercentageToPassCourseID
            ↪  AND
            (m.StartDate < COALESCE(i.EndDate, '9999-12-31') AND COALESCE(m.EndDate,
            ↪  '9999-12-31') > i.StartDate)
    )
    BEGIN
        RAISERROR ('Date range overlaps with existing entry.', 16, 1);
```

```
        ROLLBACK TRANSACTION;
    END
END;
GO
```

7.5 trg_CheckOverlap_MinAttendanceInternship

Trigger zapobiegający nakładaniu się zakresów dat w wymaganiach dotyczących minimalnej frekwencji na stażach.

```
CREATE OR ALTER TRIGGER trg_CheckOverlap_MinAttendanceInternship
ON MinAttendancePercentageToPassInternship
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM MinAttendancePercentageToPassInternship m
        INNER JOIN inserted i ON m.InternshipID = i.InternshipID OR (m.InternshipID IS NULL AND
        ↪  i.InternshipID IS NULL)
        WHERE
            m.MinAttendancePercentageToPassInternshipID <>
            ↪  i.MinAttendancePercentageToPassInternshipID AND
            (m.StartDate < COALESCE(i.EndDate, '9999-12-31') AND COALESCE(m.EndDate,
            ↪  '9999-12-31') > i.StartDate)
    )
    BEGIN
        RAISERROR ('Date range overlaps with existing entry.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

7.6 trg_CheckOverlap_MinAttendanceStudies

Trigger kontrolujący nakładanie się zakresów dat w wymaganiach dotyczących minimalnej frekwencji na studiach.

```
CREATE OR ALTER TRIGGER trg_CheckOverlap_MinAttendanceStudies
ON MinAttendancePercentageToPassStudies
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM MinAttendancePercentageToPassStudies m
        INNER JOIN inserted i ON m.StudiesID = i.StudiesID OR (m.StudiesID IS NULL AND
        ↪  i.StudiesID IS NULL)
        WHERE
            m.MinAttendancePercentageToPassStudiesID <> i.MinAttendancePercentageToPassStudiesID
            ↪  AND
            (m.StartDate < COALESCE(i.EndDate, '9999-12-31') AND COALESCE(m.EndDate,
            ↪  '9999-12-31') > i.StartDate)
    )
    BEGIN
        RAISERROR ('Date range overlaps with existing entry.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

7.7 trg_CheckOverlap_MaxDaysPaymentCourse

Trigger sprawdzający nakładanie się zakresów dat w regulaminie maksymalnego czasu na zapłatę przed rozpoczęciem kursu.

```
CREATE OR ALTER TRIGGER trg_CheckOverlap_MaxDaysPaymentCourse
ON MaxDaysForPaymentBeforeCourseStart
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM MaxDaysForPaymentBeforeCourseStart m
        INNER JOIN inserted i ON m.CourseID = i.CourseID OR (m.CourseID IS NULL AND i.CourseID IS
        ↪  NULL)
        WHERE
            m.MaxDaysForPaymentBeforeCourseStartID <> i.MaxDaysForPaymentBeforeCourseStartID AND
            (m.StartDate < COALESCE(i.EndDate, '9999-12-31') AND COALESCE(m.EndDate,
            ↪  '9999-12-31') > i.StartDate)
    )
    BEGIN
        RAISERROR ('Date range overlaps with existing entry.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

7.8 trg_CheckOverlap_MaxDaysPaymentStudies

Trigger zapobiegający nakładaniu się zakresów dat dotyczących maksymalnego czasu na zapłatę przed rozpoczęciem studiów.

```
CREATE OR ALTER TRIGGER trg_CheckOverlap_MaxDaysPaymentStudies
ON MaxDaysForPaymentBeforeStudiesStart
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM MaxDaysForPaymentBeforeStudiesStart m
        INNER JOIN inserted i ON m.StudiesID = i.StudiesID
        WHERE
            m.MaxDaysForPaymentBeforeStudiesStartID <> i.MaxDaysForPaymentBeforeStudiesStartID
            ↪  AND
            (m.StartDate < COALESCE(i.EndDate, '9999-12-31') AND COALESCE(m.EndDate,
            ↪  '9999-12-31') > i.StartDate)
    )
    BEGIN
        RAISERROR ('Date range overlaps with existing entry.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

7.9 trg_CheckOverlap_RecordingAccessTime

Trigger kontrolujący nakładanie się zakresów dat w dostępie do nagrań webinarów.

```
CREATE OR ALTER TRIGGER trg_CheckOverlap_RecordingAccessTime
ON RecordingAccessTime
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM RecordingAccessTime m
        INNER JOIN inserted i ON m.WebinarID = i.WebinarID OR (m.WebinarID IS NULL AND
        ↪  i.WebinarID IS NULL)
        WHERE
            m.RecordingAcessTimeID <> i.RecordingAcessTimeID AND
            (m.StartDate < COALESCE(i.EndDate, '9999-12-31') AND COALESCE(m.EndDate,
            ↪  '9999-12-31') > i.StartDate)
    )
```

```
    BEGIN
        RAISERROR ('Date range overlaps with existing entry.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

7.10 trg_CheckOverlap_DaysInInternship

Trigger sprawdzający nakładanie się zakresów dat w określonych dniach stażu.

```
CREATE OR ALTER TRIGGER trg_CheckOverlap_DaysInInternship
ON DaysInInternship
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM DaysInInternship m
        INNER JOIN inserted i ON m.InternshipID = i.InternshipID OR (m.InternshipID IS NULL AND
        ↪  i.InternshipID IS NULL)
        WHERE
            m.DaysInInternshipID <> i.DaysInInternshipID AND
            (m.StartDate < COALESCE(i.EndDate, '9999-12-31') AND COALESCE(m.EndDate,
            ↪  '9999-12-31') > i.StartDate)
    )
    BEGIN
        RAISERROR ('Date range overlaps with existing entry.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
GO
```

8 Indeksy

8.1 idx_webinars

Indeksy w tabeli WebinarParticipants: użytkownik, webinar, ID płatności pełnej ceny

```
CREATE INDEX idx_webinars ON Webinars (LecturerID, TranslatorID, LanguageID, StartDate, EndDate);
```

8.2 idx_webinarparticipants

Indeksy w tabeli Courses: data rozpoczęcia, data zakończenia, koordynator, język

```
CREATE INDEX idx_webinarparticipants ON WebinarParticipants (UserID, WebinarID,
↪  FullPricePaymentID);
```

8.3 idx_courses

Indeksy w tabeli Payments: użytkownik, produkt, data

```
CREATE INDEX idx_courses ON Courses (StartDate, EndDate, CoordinatorID, LanguageID);
```

8.4 idx_payments

Indeksy w tabeli CoursesSessions: wykładowca, tłumacz, moduł

```
CREATE INDEX idx_payments ON Payments (UserID, ProductID, Date);
```

8.5 idx_coursessessions

Indeksy w tabeli CourseOnlineSessions: data rozpoczęcia, data zakończenia

```
CREATE INDEX idx_coursessessions ON CoursesSessions (LecturerID, TranslatorID, ModuleID);
```

8.6 idx_courseonlinesessions

Indeksy w tabeli CourseStationarySessions: data rozpoczęcia, data zakończenia

```
CREATE INDEX idx_courseonlinesessions ON CourseOnlineSessions (StartDate, EndDate);
```

8.7 idx_coursestationarysessions

Indeksy w tabeli CourseParticipants: kurs, użytkownik, data dodania

```
CREATE INDEX idx_coursestationarysessions ON CourseStationarySessions (StartDate, EndDate);
```

8.8 idx_courseparticipants

Indeksy w tabeli DaysInInternship: data rozpoczęcia, data zakończenia, staż

```
CREATE INDEX idx_courseparticipants ON CourseParticipants (CourseID, UserID, AddedAt);
```

8.9 idx_daysininternship

Indeksy w tabeli DiplomasSent: użytkownik, data wysłania

```
CREATE INDEX idx_daysininternship ON DaysInInternship (StartDate, EndDate, InternshipID);
```

8.10 idx_diplomassent

Indeksy w tabeli EmployeeRoles: pracownik, rola

```
CREATE INDEX idx_diplomassent ON DiplomasSent (UserID, SentAt);
```

8.11 idx_employeeroles

Indeksy w tabeli Exams: przedmiot, data rozpoczęcia, data zakończenia

```
CREATE INDEX idx_employeeroles ON EmployeeRoles (EmployeeID, RoleID);
```

8.12 idx_exams

Indeksy w tabeli ExamsGrades: student, egzamin

```
CREATE INDEX idx_exams ON Exams (SubjectID, StartDate, EndDate);
```

8.13 idx_examsgrades

Indeksy w tabeli InternshipDetails: student, data ukończenia

```
CREATE INDEX idx_examsgrades ON ExamsGrades (StudentID, ExamID);
```

8.14 idx_internshipdetails

Indeksy w tabeli Internships: studia, data rozpoczęcia, data zakończenia

```
CREATE INDEX idx_internshipdetails ON InternshipDetails (StudentID, CompletedAt);
```

8.15 idx_internships

Indeksy w tabeli MaxDaysForPaymentBeforeCourseStart: data rozpoczęcia, data zakończenia, kurs

```
CREATE INDEX idx_internships ON Internships (StudiesID, StartDate, EndDate);
```

8.16 idx_maxdayspaymentcourse

Indeksy w tabeli MaxDaysForPaymentBeforeStudiesStart: data rozpoczęcia, data zakończenia, studia

```
CREATE INDEX idx_maxdayspaymentcourse ON MaxDaysForPaymentBeforeCourseStart (StartDate, EndDate,
↪  CourseID);
```

8.17 idx_maxdayspaymentstudies

Indeksy w tabeli MinAttendancePercentageToPassCourse: data rozpoczęcia, data zakończenia, kurs

```
CREATE INDEX idx_maxdayspaymentstudies ON MaxDaysForPaymentBeforeStudiesStart (StartDate,
↪  EndDate, StudiesID);
```

8.18 idx_minattendancecourse

Indeksy w tabeli MinAttendancePercentageToPassInternship: data rozpoczęcia, data zakończenia, staż

```
CREATE INDEX idx_minattendancecourse ON MinAttendancePercentageToPassCourse (StartDate, EndDate,
↪  CourseID);
```

8.19 idx_minattendanceinternship

Indeksy w tabeli MinAttendancePercentageToPassStudies: data rozpoczęcia, data zakończenia, studia

```
CREATE INDEX idx_minattendanceinternship ON MinAttendancePercentageToPassInternship (StartDate,
↪  EndDate, InternshipID);
```

8.20 idx_minattendancestudies

Indeksy w tabeli People: data urodzenia

```
CREATE INDEX idx_minattendancestudies ON MinAttendancePercentageToPassStudies (StartDate,
↪  EndDate, StudiesID);
```

8.21 idx_people

Indeksy w tabeli Products: cena, data dodania, data zamknięcia

```
CREATE INDEX idx_people ON People (BirthDate);
```

8.22 idx_products

Indeksy w tabeli RecordingAccessTime: data rozpoczęcia, data zakończenia, webinar

```
CREATE INDEX idx_products ON Products (Price, AddedAt, ClosedAt);
```

8.23 idx_recordingaccesstime

Indeksy w tabeli Students: użytkownik, studia, data dodania

```
CREATE INDEX idx_recordingaccesstime ON RecordingAccessTime (StartDate, EndDate, WebinarID);
```

8.24 idx_students

Indeksy w tabeli Studies: data rozpoczęcia, data zakończenia, koordynator, język

```
CREATE INDEX idx_students ON Students (UserID, StudiesID, AddedAt);
```

8.25 idx_studies

Indeksy w tabeli StudiesSessions: przedmiot, data rozpoczęcia, data zakończenia, wykładowca, język

```
CREATE INDEX idx_studies ON Studies (StartDate, EndDate, CoordinatorID, LanguageID);
```

8.26 idx_studiessessions

Indeksy w tabeli Subjects: studia, koordynator

```
CREATE INDEX idx_studiessessions ON StudiesSessions (SubjectID, StartDate, EndDate, LecturerID,
↪  LanguageID);
```

8.27 idx_subjects

```
CREATE INDEX idx_subjects ON Subjects (StudiesID, CoordinatorID);
```

9 Roles

9.1 Nauczyciel akademicki

Może tworzyć przedmioty/zajęcia, egzaminy, dodawać oceny z egzaminów, wpisywać obecności, tworzyć staże, wpisywać zaliczenia ze stażu, aktualizować ich dane

```
CREATE ROLE [AcademicTeacherRole];

GRANT EXECUTE ON [CreateSubject] TO [AcademicTeacherRole];
GRANT EXECUTE ON [ModifySubject] TO [AcademicTeacherRole];
GRANT EXECUTE ON [CreateSemesterOfStudies] TO [AcademicTeacherRole];
GRANT EXECUTE ON [AddExam] TO [AcademicTeacherRole];
GRANT EXECUTE ON [ModifyExam] TO [AcademicTeacherRole];
GRANT EXECUTE ON [UpdateExamGrade] TO [AcademicTeacherRole];
GRANT EXECUTE ON [AddInternship] TO [AcademicTeacherRole];
GRANT EXECUTE ON [ModifyInternship] TO [AcademicTeacherRole];
GRANT EXECUTE ON [UpdateInternshipDetail] TO [AcademicTeacherRole];
GRANT EXECUTE ON [UpdateAttendance] TO [AcademicTeacherRole];

GRANT SELECT ON [AttendanceListForEachSession] TO [AcademicTeacherRole];
GRANT SELECT ON [GeneralAttendance] TO [AcademicTeacherRole];
```

9.2 Księgowa

Ma dostęp do widoków związanych z finansami

```
CREATE ROLE [AccountantRole];
GRANT SELECT ON [dbo].[TotalIncomeForProducts] TO [AccountantRole];
GRANT SELECT ON [dbo].[RevenueSummaryByProductType] TO [AccountantRole];
GRANT SELECT ON [dbo].[Loaners] TO [AccountantRole];
```

9.3 Administrator - zarządza bazą danych, nieograniczone uprawnienia

```
CREATE ROLE AdministratorRole;

GRANT CONTROL ON DATABASE::u_karamon TO [AdministratorRole];
GRANT VIEW DEFINITION TO [AdministratorRole];
GRANT VIEW DATABASE STATE TO [AdministratorRole];
```

9.4 Nauczyciel związany z kursami

Może tworzyć modyfikować kursy, zajęcia, moduły, wpisywać obecności

```
CREATE ROLE [CoursesTeacherRole];

GRANT EXECUTE ON [CreateCourse] TO [CoursesTeacherRole];
GRANT EXECUTE ON [ModifyCourse] TO [CoursesTeacherRole];
GRANT EXECUTE ON [CreateModule] TO [CoursesTeacherRole];
GRANT EXECUTE ON [ModifyModule] TO [CoursesTeacherRole];
GRANT EXECUTE ON [DeleteModule] TO [CoursesTeacherRole];

GRANT EXECUTE ON [ModifyOnlineCourseSession] TO [CoursesTeacherRole];
GRANT EXECUTE ON [ModifyOfflineCourseSession] TO [CoursesTeacherRole];
GRANT EXECUTE ON [ModifyStationaryCourseSession] TO [CoursesTeacherRole];
GRANT EXECUTE ON [DeleteCourseSession] TO [CoursesTeacherRole];

GRANT EXECUTE ON [UpdateAttendance] TO [CoursesTeacherRole];
GRANT EXECUTE ON [DeleteAttendance] TO [CoursesTeacherRole];
GRANT EXECUTE ON [UpdateCourseSessionAttendance] TO [CoursesTeacherRole];
GRANT EXECUTE ON [DeleteCourseSessionAttendance] TO [CoursesTeacherRole];
```

9.5 Dyrektor

Odroczenie płatności, widoki dotyczące finansów, wyników pracowników, nadawanie pracownikom ról. Utworzenie/modyfikacja studiów. Utworzenie/modyfikacja kierunku studiów. Dostęp do raportów finansowych dostęp do raportów dotyczących pracowników.

```
CREATE ROLE HeadMasterRole;

GRANT SELECT ON TotalIncomeForProducts TO HeadMasterRole;
GRANT SELECT ON RevenueSummaryByProductType TO HeadMasterRole;

GRANT EXECUTE ON AddEmployee TO HeadMasterRole;
GRANT EXECUTE ON AddRole TO HeadMasterRole;
GRANT EXECUTE ON ModifyRole TO HeadMasterRole;
GRANT EXECUTE ON AddEmployeeRole TO HeadMasterRole;
GRANT EXECUTE ON RemoveEmployeeRole TO HeadMasterRole;

GRANT EXECUTE ON CreateSemesterOfStudies TO HeadMasterRole;
GRANT EXECUTE ON ModifyStudies TO HeadMasterRole;
GRANT EXECUTE ON AddFieldOfStudy TO HeadMasterRole;
GRANT EXECUTE ON DeleteFieldOfStudies TO HeadMasterRole;

GRANT SELECT ON EmployeeStatistics TO HeadMasterRole;
GRANT SELECT ON EmployeeTimeTable TO HeadMasterRole;
GRANT SELECT ON ActivityConflicts TO HeadMasterRole;

GRANT EXECUTE ON EnrollUserWithoutImmediatePayment TO HeadMasterRole;
GRANT EXECUTE ON ChangeProductPrice TO HeadMasterRole;
```

9.6 Sekretariat

Wysyłanie dyplomów, raporty bilokacji, informacje o pracownikach i uczniach

```
CREATE ROLE [SecretariatRole];

GRANT EXECUTE ON [dbo].[SendDiploma] TO [SecretariatRole];

GRANT SELECT ON [dbo].[EmployeeStatistics] TO [SecretariatRole];
GRANT SELECT ON [dbo].[Loaners] TO [SecretariatRole];
GRANT SELECT ON [dbo].[AttendanceListForEachSession] TO [SecretariatRole];
GRANT SELECT ON [dbo].[GeneralAttendance] TO [SecretariatRole];
GRANT SELECT ON [dbo].[NumberOfPeopleRegisteredForEvents] TO [SecretariatRole];
```

9.7 Tłumacz

Ddodawanie/modyfikacja nagrania w celu dodania tłumaczenia

```
CREATE ROLE TranslatorRole;

GRANT EXECUTE ON ModifyOnlineCourseSession TO TranslatorRole;
GRANT EXECUTE ON ModifyOnlineStudySession TO TranslatorRole;
GRANT EXECUTE ON ModifyOnlineStudiesSessionRecording TO TranslatorRole;
GRANT EXECUTE ON ModifyWebinarRecording TO TranslatorRole;

GRANT SELECT ON Webinars TO TranslatorRole;
GRANT SELECT ON CoursesSessions TO TranslatorRole;
GRANT SELECT ON CourseOnlineSessions TO TranslatorRole;
GRANT SELECT ON CourseOfflineSessions TO TranslatorRole;
GRANT SELECT ON StudiesSessions TO TranslatorRole;
GRANT SELECT ON OnlineStudiesSessions TO TranslatorRole;
```

9.8 Wykładowcy webinariów

Tworzenie/modyfikacja webinarów

```
CREATE ROLE [WebinarLecturerRole];

GRANT EXECUTE ON [dbo].[AddWebinar] TO [WebinarLecturerRole];
GRANT EXECUTE ON [dbo].[ModifyWebinarData] TO [WebinarLecturerRole];
GRANT EXECUTE ON [dbo].[DeleteWebinar] TO [WebinarLecturerRole];
GRANT EXECUTE ON [dbo].[ModifyWebinarRecording] TO [WebinarLecturerRole];
GRANT EXECUTE ON [dbo].[CloseWebinar] TO [WebinarLecturerRole];
```

10 Generowanie danych

Dane do bazy są generowane przez trzy skrypty napisane w języku Python.

- `courses.py` dla kursów oraz dla pracowników/użytkowników
- `studies.py` dla studiów
- `webinars.py` dla webinarów

Kod do generowanie danych został odłączony oddzielnie od tej dokumentacji.