Projekt bazy danych firmy edukacyjnej

Przedmiot: Podstawy Baz Danych Autorzy: Maciej Nowakowski, Zuzanna Stajniak, Mateusz Lampert

Użytkownicy bazy danych:

1. Administrator
2. Dyrektor Placówki
3. Pracownik biura administracji
4. Pracownik biura dydaktyki
5. Prowadzący
6. Tłumacz
7. Student

Funkcje Użytkowników:

1. Administrator
    - Usuwanie nagrań
2. Dyrektor placówki
    - Dodawanie pracowników
    - Indywidualne zmienianie terminów opłat
3. Pracownik biura administracji
    - Generowanie raportów
    - Układanie planu zajęć
    - Zmiana harmonogramu zajęć z przyczyn losowych
4. Pracownik biura dydaktyki
    - Generowanie raportów
    - Generowanie i wysyłanie dyplomu
    - Weryfikowanie czy użytkownik zaliczył dany kurs lub studia
5. Prowadzący
    - Tworzenie sylabusa nowego przedmiotu
    - Tworzenie webinarów
    - Zakładanie kursów
    - Sprawdzanie obecności na stacjonarnych zajęciach
    - Weryfikacja odrabiania nieobecności na studiach.
6. Tłumacz
7. Student
    - Dodawanie usług do koszyka
    - Opłacanie usług w koszyku
    - Zapis na zajęcia do odrabiania nieobecności na studiach
8. Każdy użytkownik
    - Możliwość przeglądania oferty kursów

Funkcje Systemowe:

- Sprawdzanie obecności zdalnej
- Sprawdzanie czy użytkownik ma dostęp do usługi
- Sprawdzenie czy zapis na daną usługę jest możliwy.

Schemat bazy danych

Opis tabel

Clients

Tabela przechowuje podstawowe dane o kliencie. Klient to osoba, która może złożyć zamówienie i korzystać z usług szkoły. Zawiera identyfikator klienta (clientID), imię oraz nazwisko (firstName, lastName) oraz dane adresowe (adress, city, region).

Klucz główny: clientID

Orders

Tabela przechowuje podstawowe dane o zamówieniu. Można zamówić wiele usług w jednym zamówieniu. Można zamówić webinar, kurs, studia, zjazd na studiach lub pojedyncze spotkanie w ramach studiów. Tablica zawiera identyfikator zamówienia (orderID), identyfikator klienta (clientID), datę zamówienia (orderDate), link do płatności (paymentLink) oraz datę przyjęcia płatności (receiptDate).

Klucz główny: orderID

Klucz obcy: clientID (z tabelą clients)

OrderDetails

Tabela przechowuje szczegółowe dane o zamówieniu. Można zamówić webinar, kurs, studia, zjazd na studiach lub pojedyncze spotkanie w ramach studiów. Zawiera identyfikator zamówienia (orderID), identyfikator usługi w koszyku (serviceID) oraz cenę za tą usługę (price).

Klucze główne: orderID, serviceID

Klucze obce: orderID (z tabelą **Orders**), serviceID (z tabelą **Services**)

Services

Tabela przechowuje podstawowe dane o dostępnych usługach edukacyjnych. Usługi to webinar, kurs, studia, zjazd na studiach lub pojedyncze spotkanie w ramach studiów. Tabela zawiera identyfikator usługi (serviceID), nazwę usługi (serviceName), typ usługi (type - webinar, kurs, studia, zjazd, pojedyńcze spotkanie), datę rozpoczęcia (startDate) oraz cenę(price).

Klucz główny: serviceID

Studies

Tabela przechowuje podstawowe dane o studiach. Studia to wieloletnie formy kształcenia składające się z praktyk oraz zjazdów, które składają się z przedmiotów. Tabela zawiera identyfikator studiów (studiesID), nazwę (name) oraz liczbę zjazdów (terms).

Klucz główny: studiesID

Klucz obcy: studiesID (z tabelą **Services**)

Terms

Tabela przechowuje podstawowe dane o zjeździe na studiach. Zjazd na studiach to część studiów. Zjazdy składają się z przedmiotów. Tabela zawiera identyfikator zjazdu(termID), identyfikator studiów (studiesID), datę rozpoczęcia i zakończenia(startDate, endDate).

Klucz główny: termID

Klucze obce: studiesID (z tabelą **Studies**), termID (z tabelą **Practices**)

Subjects

Tabela zawiera szczegółowe informacje o przedmiotach realizowanych podczas poszczególnych zjazdów. Zawiera identyfikator przedmiotu(subjectID), identyfikator zjazdu (termID), nazwę przedmiotu (subjectName), kategorię przedmiotu (type)

Klucz główny: subjectID

Klucz obcy: termID (z tabelą **Terms**)

SubjectGrades

Tabela przechowuje informację o ocenach oraz zaliczeniu przedmiotu ze studiów. Zawiera identyfikator oceny(gradeID), identyfikator przedmiotu (subjectID), identyfikator klienta (clientID) oraz informację o ocenie (grade)

Klucz główny: gradeID

Klucze obce: subjectID (z tabelą **Subjects**), clientID (z tabelą **Clients**)

Employees

Tabela przechowuje podstawowe dane o pracownikach. Pracownik to osoba zatrudniona przez szkołę. Są to pracownicy naukowi, tłumacze, pracownicy administracji, pracownicy ds nauki. Tabela zawiera identyfikator pracownika (employeeID), imię oraz nazwisko (firstName, lastName) oraz stanowisko (position).

Klucz główny: employeeID

Interpreters

Tabela przechowuje podstawowe dane o tłumaczu. Tłumacz to pracownik szkoły. Tabela zawiera identyfikator tłumacza (interpreterID), identyfikator pracownika (employeeID) oraz język, którym się posługuje (language)

Klucz główny: interpreterID

Klucz obcy: employeeID (z tabelą **Employees**)

Meetings

Tabela przechowuje dane o spotkaniu. Spotkanie może odbywać się w ramach webinaru, kursu lub studiów. Zawiera identyfikator spotkania (meetingID), identyfikator tłumacza (interpreterID), datę i czas spotkania (datetime), link do spotkania online (meetingLink), język wykładowy (language) oraz identyfikator nagrania (recordID).

Klucz główny: meetingID

Klucze obce: interpreterID (z tabelą **Interpreters**), recordID (z tabelą **Recordings**), tutorID (z tabelą **Employees**)

Attendees

Tabela przechowuje dane o obecności na spotkaniach. Zawiera identyfikator obecności (attendanceID), identyfikator spotkania (meetingID), identyfikator klienta (meetingID), informację o obecności (present) oraz indentyfikator zajęć na których odrabiano nieobecność (substituteMeetingID).

Klucz główny: attendanceID

Klucz obcy: substituteAttendanceID (z tabelą **Attendees**), meetingID (z tabelą **Meetings**), clientID (z tabelą **Clients**)

Recordings

Tabela przechowuje linki do nagrań. Zawiera identyfikator nagrania (recordID) oraz link do nagrania (recordLInk).

Klucz główny: recordID

Access

Tabela przechowuje dane dostępów do nagrań. Zawiera identyfikator klienta (clientID), identyfikator spotkania (meetingID), datę końca dostępu (lastDay) oraz status obejrzenia (status).

Klucze główne: clientID, recordID

Klucze obce: clientID (z tabelą **Clients**), recordID (z tabelą **Recordings**)

Practices

Tabela przechowuje informacje o praktykach przypisanych do studiów. Praktyki odbywają się dwa razy do roku w ramach studiów. Zawiera identyfikator praktyk (practiceID), identyfikator semestru (termID)

Klucze główne: practiceID

Klucze obce: termID (z tabelą **Terms**)

IndividualInternship

Tabela przechowuje informacje o poszczególnych praktykach odbywanych przez poszczególnych studentów. Zawiera identyfikator pojedyńczych praktyk (InternshipID), identyfikator studenta (clientID), identyfikator praktyk (practiceID), datę rozpoczęcia oraz zakończenia (startDate oraz endDate) oraz informację o zaliczeniu praktyk (passed)

Klucz główny: internshipID

Klucze główne: clientID (z tabelą **Clients**), practiceID (z tabelą **Practices**)

Webinars

Tabela zawiera informację o webinarach. Webinar to pojedyncze spotkanie. Tabela zawiera identyfikator webinaru (serviceID), maksymalną możliwą ilość uczestników (maxParticipants) oraz identyfikator spotkania, do którego jest przypisany (meetingID)

Klucz główny: serviceID

Klucze obce: serviceID (z tabelą **Services**), meetingID (z tabelą **Meetings**)

Courses

Tabela zawiera informację o kursach. Kurs może mieć formę zdalną, stacjonarną lub hybrydową. Każdy kurs ma określoną liczbę spotkań na żywo, które odbywają się stacjonarnie lub zdalnie, oraz nagrań offline. Tabela zawiera identyfikator kursu (serviceID), maksymalną możliwą ilość uczestników (maxParticipants), łączną ilość spotkań na żywo w ramach kursu (totalMeetings), łączną ilość nagrań w ramach kursu (totalRecords)

Klucz główny: serviceID

Klucz obcy: serviceID (z tabelą **Services**)

CoursesMeetings

Tabela zawiera informacje o poszczególnych spotkaniach na żywo w ramach kursu. Zawiera identyfikator spotkania z kursu (courseMeetingID), identyfikator spotkania (meetingID) oraz identyfikator kursu (serviceID)

Klucz główny: courseMeetingID

Klucze obce: meetingID (z tabelą **Meetings**), serviceID (z tabelą **Courses**)

CoursesRecords

Tabela zawiera informacje o poszczególnych nagraniach w ramach kursu. Zawiera identyfikator nagrania z kursu (courseRecordID), identyfikator nagrania (recordID) oraz identyfikator kursu (serviceID)

Klucz główny: courseRecordID

Klucze obce: recordID (z tabelą **Recordings**), serviceID (z tabelą **Courses**)

CoursesGrades

Tabela zawiera informacje o zaliczeniu kursu. Zawiera identyfikator oceny (gradeID), identyfikator kursu (serviceID), identyfikator klienta (clientID), informację o zaliczeniu (passed).

Klucz główny: gradeID

Klucze obce: serviceID (z tabelą **Courses**), clientID (z tabelą **Clients**)

SubjectsMeetings

Tabela zawiera informacje o poszczególnych spotkaniach w ramach przedmiotu. Zawiera identyfikator spotkania w ramach przedmiotu (subjectMeetingID), identyfikator spotkania (meetingID) oraz identyfikator przedmiotu (subjectID)

Klucz główny: subjectMeetingID

Klucze obce: meetingID (z tabelą **Meetings**), subjectID (z tabelą **Subjects**)

SubjectsRecordings

Tabela zawiera informacje o poszczególnych nagraniach w ramach przedmiotu na studiach. Zawiera identyfikator nagrania z przedmiotu (subjectRecordingID), identyfikator nagrania (recordID) oraz identyfikator przedmiotu (subjectID)

Klucz główny: courseRecordID

Klucze obce: recordID (z tabelą **Recordings**), serviceID (z tabelą **Courses**)

Implementacja bazy danych

Do utworzenia bazy danych wraz z wszystkimi połączeniami użyta została następująca implementacja:

```
-- Table: Access
CREATE TABLE [dbo].[Access](
    [clientID] [int] NOT NULL,
    [recordID] [int] NOT NULL,
    [lastDay] [date] NOT NULL,
    [status] [bit] NULL,
 CONSTRAINT [Access_pk] PRIMARY KEY CLUSTERED
(
    [clientID] ASC,
    [recordID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Access]  WITH CHECK ADD  CONSTRAINT [Access_Clients] FOREIGN KEY([clientID])
REFERENCES [dbo].[Clients] ([clientID])
GO

ALTER TABLE [dbo].[Access] CHECK CONSTRAINT [Access_Clients]
GO

ALTER TABLE [dbo].[Access]  WITH CHECK ADD  CONSTRAINT [access_recordings] FOREIGN KEY([recordID])
REFERENCES [dbo].[Recordings] ([recordID])
GO

ALTER TABLE [dbo].[Access] CHECK CONSTRAINT [access_recordings]
GO

-- Reference: Access_Clients (table: Access)
ALTER TABLE Access ADD CONSTRAINT Access_Clients
    FOREIGN KEY (clientID)
    REFERENCES Clients (clientID);

-- Reference: access_recordings (table: Access)
ALTER TABLE Access ADD CONSTRAINT access_recordings
    FOREIGN KEY (recordID)
    REFERENCES Recordings (recordID);

-- Table: Attendees
CREATE TABLE [dbo].[Attendees](
    [attendanceID] [int] IDENTITY(1,1) NOT NULL,
    [clientID] [int] NOT NULL,
    [meetingID] [int] NOT NULL,
    [present] [bit] NULL,
    [substituteAttendanceID] [int] NULL,
 CONSTRAINT [Attendees_pk] PRIMARY KEY CLUSTERED
(
    [attendanceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY],
 CONSTRAINT [Unique_Attendee_Per_Meeting_Client] UNIQUE NONCLUSTERED
(
    [clientID] ASC,
    [meetingID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Attendees]  WITH CHECK ADD  CONSTRAINT [attendees_attendees] FOREIGN KEY([substituteAttendanceID])
REFERENCES [dbo].[Attendees] ([attendanceID])
GO

ALTER TABLE [dbo].[Attendees] CHECK CONSTRAINT [attendees_attendees]
GO

ALTER TABLE [dbo].[Attendees]  WITH CHECK ADD  CONSTRAINT [Attendees_Clients] FOREIGN KEY([clientID])
REFERENCES [dbo].[Clients] ([clientID])
GO

ALTER TABLE [dbo].[Attendees] CHECK CONSTRAINT [Attendees_Clients]
GO

ALTER TABLE [dbo].[Attendees]  WITH CHECK ADD  CONSTRAINT [attendees_meeting] FOREIGN KEY([meetingID])
REFERENCES [dbo].[Meetings] ([meetingID])
GO

ALTER TABLE [dbo].[Attendees] CHECK CONSTRAINT [attendees_meeting]
GO

-- Reference: attendees_attendees (table: Attendees)
ALTER TABLE Attendees ADD CONSTRAINT attendees_attendees
    FOREIGN KEY (substituteAttendanceID)
```

```
    REFERENCES Attendees (attendanceID);

-- Reference: attendees_meeting (table: Attendees)
ALTER TABLE Attendees ADD CONSTRAINT attendees_meeting
    FOREIGN KEY (meetingID)
    REFERENCES Meetings (meetingID);

-- Reference: Attendees_Clients (table: Attendees)
ALTER TABLE Attendees ADD CONSTRAINT Attendees_Clients
    FOREIGN KEY (clientID)
    REFERENCES Clients (clientID);

-- Table: Clients
CREATE TABLE [dbo].[Clients](
    [clientID] [int] IDENTITY(1,1) NOT NULL,
    [firstName] [nvarchar](30) NOT NULL,
    [lastName] [nvarchar](30) NOT NULL,
    [address] [nvarchar](100) NOT NULL,
    [city] [nvarchar](50) NOT NULL,
    [region] [nvarchar](20) NOT NULL,
 CONSTRAINT [Clients_pk] PRIMARY KEY CLUSTERED
(
    [clientID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

-- Table: CourseGrade
CREATE TABLE [dbo].[CourseGrade](
    [gradeID] [int] IDENTITY(1,1) NOT NULL,
    [serviceID] [char](4) NOT NULL,
    [clientID] [int] NOT NULL,
    [passed] [bit] NULL,
 CONSTRAINT [CourseGrade_pk] PRIMARY KEY CLUSTERED
(
    [gradeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CourseGrade]  WITH CHECK ADD  CONSTRAINT [CourseGrade_Clients] FOREIGN KEY([clientID])
REFERENCES [dbo].[Clients] ([clientID])
GO

ALTER TABLE [dbo].[CourseGrade] CHECK CONSTRAINT [CourseGrade_Clients]
GO

ALTER TABLE [dbo].[CourseGrade]  WITH CHECK ADD  CONSTRAINT [CourseGrade_Courses] FOREIGN KEY([serviceID])
REFERENCES [dbo].[Courses] ([serviceID])
GO

ALTER TABLE [dbo].[CourseGrade] CHECK CONSTRAINT [CourseGrade_Courses]
GO

-- Reference: CourseGrade_Courses (table: CourseGrade)
ALTER TABLE CourseGrade ADD CONSTRAINT CourseGrade_Courses
    FOREIGN KEY (serviceID)
    REFERENCES Courses (serviceID);

-- Reference: CourseGrade_Clients (table: CourseGrade)
ALTER TABLE CourseGrade ADD CONSTRAINT CourseGrade_Clients
    FOREIGN KEY (clientID)
    REFERENCES Clients (clientID);

-- Table: CourseRecords
CREATE TABLE [dbo].[CourseRecords](
    [courseRecordID] [char](4) NOT NULL,
    [recordID] [int] NOT NULL,
    [serviceID] [char](4) NOT NULL,
 CONSTRAINT [CourseRecords_pk] PRIMARY KEY CLUSTERED
(
    [courseRecordID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CourseRecords]  WITH CHECK ADD  CONSTRAINT [Courses_CourseRecords] FOREIGN KEY([serviceID])
REFERENCES [dbo].[Courses] ([serviceID])
GO

ALTER TABLE [dbo].[CourseRecords] CHECK CONSTRAINT [Courses_CourseRecords]
GO

ALTER TABLE [dbo].[CourseRecords]  WITH CHECK ADD  CONSTRAINT [Recordings_CourseRecords] FOREIGN KEY([recordID])
REFERENCES [dbo].[Recordings] ([recordID])
GO
```

```
ALTER TABLE [dbo].[CourseRecords] CHECK CONSTRAINT [Recordings_CourseRecords]
GO

-- Reference: Recordings_CourseRecords (table: CourseRecords)
ALTER TABLE CourseRecords ADD CONSTRAINT Recordings_CourseRecords
    FOREIGN KEY (recordID)
    REFERENCES Recordings (recordID);

-- Reference: Courses_CourseRecords (table: CourseRecords)
ALTER TABLE CourseRecords ADD CONSTRAINT Courses_CourseRecords
    FOREIGN KEY (serviceID)
    REFERENCES Courses (serviceID);

-- Table: Courses
CREATE TABLE [dbo].[Courses](
    [serviceID] [char](4) NOT NULL,
    [totalMeetings] [int] NULL,
    [totalRecords] [int] NULL,
 CONSTRAINT [Courses_pk] PRIMARY KEY CLUSTERED
(
    [serviceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [courseDetails_services] FOREIGN KEY([serviceID])
REFERENCES [dbo].[Services] ([serviceID])
GO

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [coursesContentCheck] CHECK  ((NOT ([totalMeetings] IS NULL AND [totalRecords] IS NULL)
AND ([totalMeetings] IS NULL AND [totalRecords]>(0) OR [totalRecords] IS NULL AND [totalMeetings]>(0) OR [totalRecords] IS NOT NULL AND
[totalMeetings] IS NOT NULL AND ([totalRecords]+[totalMeetings])>(0))))
GO

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

ALTER TABLE [dbo].[Courses]  WITH CHECK ADD  CONSTRAINT [CoursesServiceIDLetterCheck] CHECK  ((isnumeric(left([serviceID],(1)))<>(1)))
GO

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

-- Reference: courseDetails_services (table: Courses)
ALTER TABLE Courses ADD CONSTRAINT courseDetails_services
    FOREIGN KEY (serviceID)
    REFERENCES Services (serviceID);

-- Table: CoursesMeetings
CREATE TABLE [dbo].[CoursesMeetings](
    [courseMeetingID] [char](4) NOT NULL,
    [meetingID] [int] NOT NULL,
    [serviceID] [char](4) NOT NULL,
 CONSTRAINT [CoursesMeetings_pk] PRIMARY KEY CLUSTERED
(
    [courseMeetingID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[CoursesMeetings]  WITH CHECK ADD  CONSTRAINT [Meetings_Workshops] FOREIGN KEY([meetingID])
REFERENCES [dbo].[Meetings] ([meetingID])
GO

ALTER TABLE [dbo].[CoursesMeetings] CHECK CONSTRAINT [Meetings_Workshops]
GO

ALTER TABLE [dbo].[CoursesMeetings]  WITH CHECK ADD  CONSTRAINT [Workshops_ServiceDetails] FOREIGN KEY([serviceID])
REFERENCES [dbo].[Courses] ([serviceID])
GO

ALTER TABLE [dbo].[CoursesMeetings] CHECK CONSTRAINT [Workshops_ServiceDetails]
GO

-- Reference: Workshops_ServiceDetails (table: CoursesMeetings)
ALTER TABLE CoursesMeetings ADD CONSTRAINT Workshops_ServiceDetails
    FOREIGN KEY (serviceID)
    REFERENCES Courses (serviceID);

-- Reference: Meetings_Workshops (table: CoursesMeetings)
ALTER TABLE CoursesMeetings ADD CONSTRAINT Meetings_Workshops
    FOREIGN KEY (meetingID)
    REFERENCES Meetings (meetingID);

-- Table: Employees
```

```
CREATE TABLE [dbo].[Employees](
    [employeeID] [int] IDENTITY(1,1) NOT NULL,
    [firstName] [nvarchar](30) NOT NULL,
    [lastName] [nvarchar](30) NOT NULL,
    [position] [nvarchar](100) NOT NULL,
 CONSTRAINT [Employees_pk] PRIMARY KEY CLUSTERED
(
    [employeeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

-- Table: IndividualInternship
CREATE TABLE [dbo].[IndividualInternship](
    [InternshipID] [int] IDENTITY(1,1) NOT NULL,
    [ClientID] [int] NOT NULL,
    [PracticeID] [int] NOT NULL,
    [StartDate] [datetime] NOT NULL,
    [EndDate] [datetime] NOT NULL,
    [Passed] [bit] NULL,
 CONSTRAINT [IndividualInternship_pk] PRIMARY KEY CLUSTERED
(
    [InternshipID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[IndividualInternship]  WITH CHECK ADD  CONSTRAINT [Clients_IndividualInternship] FOREIGN KEY([ClientID])
REFERENCES [dbo].[Clients] ([clientID])
GO

ALTER TABLE [dbo].[IndividualInternship] CHECK CONSTRAINT [Clients_IndividualInternship]
GO

ALTER TABLE [dbo].[IndividualInternship]  WITH CHECK ADD  CONSTRAINT [IndividualInternship_Practices] FOREIGN KEY([PracticeID])
REFERENCES [dbo].[Practices] ([PracticeID])
GO

ALTER TABLE [dbo].[IndividualInternship] CHECK CONSTRAINT [IndividualInternship_Practices]
GO

ALTER TABLE [dbo].[IndividualInternship]  WITH CHECK ADD  CONSTRAINT [lengthOfInternshipCheck] CHECK  ((datediff(day,[startdate],[enddate])=
(14)))
GO

ALTER TABLE [dbo].[IndividualInternship] CHECK CONSTRAINT [lengthOfInternshipCheck]
GO

-- Reference: Clients_IndividualInternship (table: IndividualInternship)
ALTER TABLE IndividualInternship ADD CONSTRAINT Clients_IndividualInternship
    FOREIGN KEY (ClientID)
    REFERENCES Clients (clientID);

-- Reference: IndividualInternship_Practices (table: IndividualInternship)
ALTER TABLE IndividualInternship ADD CONSTRAINT IndividualInternship_Practices
    FOREIGN KEY (PracticeID)
    REFERENCES Practices (PracticeID);

-- Table: Interpreters
CREATE TABLE [dbo].[Interpreters](
    [interpreterID] [int] IDENTITY(1,1) NOT NULL,
    [language] [varchar](30) NOT NULL,
    [employeeID] [int] NOT NULL,
 CONSTRAINT [Interpreters_pk] PRIMARY KEY CLUSTERED
(
    [interpreterID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Interpreters]  WITH CHECK ADD  CONSTRAINT [interpreters_employees] FOREIGN KEY([employeeID])
REFERENCES [dbo].[Employees] ([employeeID])
GO

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

-- Reference: interpreters_employees (table: Interpreters)
ALTER TABLE Interpreters ADD CONSTRAINT interpreters_employees
    FOREIGN KEY (employeeID)
    REFERENCES Employees (employeeID);

-- Table: Meetings
CREATE TABLE [dbo].[Meetings](
    [meetingID] [int] NOT NULL,
    [tutorID] [int] NOT NULL,
    [interpreterID] [int] NULL,
```

```
    [datetime] [datetime] NOT NULL,
    [meetingLink] [varchar](100) NULL,
    [Language] [varchar](20) NOT NULL,
    [recordID] [int] NULL,
 CONSTRAINT [Meetings_pk] PRIMARY KEY CLUSTERED
(
    [meetingID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Meetings]  WITH CHECK ADD  CONSTRAINT [meeting_interpreters] FOREIGN KEY([interpreterID])
REFERENCES [dbo].[Interpreters] ([interpreterID])
GO

ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [meeting_interpreters]
GO

ALTER TABLE [dbo].[Meetings]  WITH CHECK ADD  CONSTRAINT [Meetings_Employees] FOREIGN KEY([tutorID])
REFERENCES [dbo].[Employees] ([employeeID])
GO

ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [Meetings_Employees]
GO

ALTER TABLE [dbo].[Meetings]  WITH CHECK ADD  CONSTRAINT [recordings_meeting] FOREIGN KEY([recordID])
REFERENCES [dbo].[Recordings] ([recordID])
GO

ALTER TABLE [dbo].[Meetings] CHECK CONSTRAINT [recordings_meeting]
GO

-- Reference: meeting_interpreters (table: Meetings)
ALTER TABLE Meetings ADD CONSTRAINT meeting_interpreters
    FOREIGN KEY (interpreterID)
    REFERENCES Interpreters (interpreterID);

-- Reference: recordings_meeting (table: Meetings)
ALTER TABLE Meetings ADD CONSTRAINT recordings_meeting
    FOREIGN KEY (recordID)
    REFERENCES Recordings (recordID);


-- Reference: Meetings_Employees (table: Meetings)
ALTER TABLE Meetings ADD CONSTRAINT Meetings_Employees
    FOREIGN KEY (tutorID)
    REFERENCES Employees (employeeID);

-- Table: OrderDetails
CREATE TABLE [dbo].[OrderDetails](
    [orderID] [uniqueidentifier] NOT NULL,
    [serviceID] [char](4) NOT NULL,
    [price] [money] NOT NULL,
 CONSTRAINT [OrderDetails_pk] PRIMARY KEY CLUSTERED
(
    [orderID] ASC,
    [serviceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[OrderDetails] ADD  DEFAULT (newid()) FOR [orderID]
GO

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

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

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [orderDetails_services] FOREIGN KEY([serviceID])
REFERENCES [dbo].[Services] ([serviceID])
GO

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

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

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

ALTER TABLE [dbo].[OrderDetails]  WITH CHECK ADD  CONSTRAINT [OrderDetailsServicesIDLetterCheck] CHECK  ((isnumeric(left([serviceID],(1)))<>
(1)))
GO
```

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

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

-- Reference: orderDetails_services (table: OrderDetails)
ALTER TABLE OrderDetails ADD CONSTRAINT orderDetails_services
    FOREIGN KEY (serviceID)
    REFERENCES Services (serviceID);

-- Table: Orders
CREATE TABLE [dbo].[Orders](
    [orderID] [uniqueidentifier] NOT NULL,
    [clientID] [int] NOT NULL,
    [orderDate] [datetime] NULL,
    [paymentLink] [varchar](100) NOT NULL,
    [receiptDate] [datetime] NULL,
 CONSTRAINT [Orders_pk] PRIMARY KEY CLUSTERED
(
    [orderID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Orders] ADD  CONSTRAINT [DF_Orders_ID]  DEFAULT (newid()) FOR [orderID]
GO

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [orders_clients] FOREIGN KEY([clientID])
REFERENCES [dbo].[Clients] ([clientID])
GO

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

ALTER TABLE [dbo].[Orders]  WITH CHECK ADD  CONSTRAINT [CheckOrderDates] CHECK  (([orderDate] IS NULL AND [receiptDate] IS NULL OR [receiptDate]
IS NULL OR [receiptDate]>=[orderDate]))
GO

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

-- Reference: orders_clients (table: Orders)
ALTER TABLE Orders ADD CONSTRAINT orders_clients
    FOREIGN KEY (clientID)
    REFERENCES Clients (clientID);

-- Table: Practices
CREATE TABLE [dbo].[Practices](
    [PracticeID] [int] IDENTITY(1,1) NOT NULL,
    [studiesID] [char](4) NULL,
 CONSTRAINT [Practices_pk] PRIMARY KEY CLUSTERED
(
    [PracticeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Practices]  WITH CHECK ADD  CONSTRAINT [Studies_Pracices] FOREIGN KEY([studiesID])
REFERENCES [dbo].[Studies] ([studiesID])
GO

ALTER TABLE [dbo].[Practices] CHECK CONSTRAINT [Studies_Pracices]
GO

-- Reference: Studies_Practices (table: Practices)
ALTER TABLE Practices ADD CONSTRAINT Studies_Practices
    FOREIGN KEY (studiesID)
    REFERENCES Studies (studiesID);

-- Table: Recordings
CREATE TABLE [dbo].[Recordings](
    [recordID] [int] IDENTITY(1,1) NOT NULL,
    [recordLink] [varchar](100) NOT NULL,
 CONSTRAINT [Recordings_pk] PRIMARY KEY CLUSTERED
(
    [recordID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

-- Table: Services
CREATE TABLE [dbo].[Services](
    [serviceID] [char](4) NOT NULL,
    [serviceName] [nvarchar](50) NOT NULL,
```

```
    [type] [varchar](10) NOT NULL,
    [startDate] [datetime] NOT NULL,
    [price] [money] NOT NULL,
    [availability] [bit] NOT NULL,
    [maxParticipants] [int] NULL,
 CONSTRAINT [Services_pk] PRIMARY KEY CLUSTERED
(
    [serviceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Services]  WITH CHECK ADD  CONSTRAINT [CheckParticipants] CHECK  (([maxParticipants]>=(0)))
GO

ALTER TABLE [dbo].[Services] CHECK CONSTRAINT [CheckParticipants]
GO

ALTER TABLE [dbo].[Services]  WITH CHECK ADD  CONSTRAINT [CheckPrice] CHECK  (([price]>=(0)))
GO

ALTER TABLE [dbo].[Services] CHECK CONSTRAINT [CheckPrice]
GO

ALTER TABLE [dbo].[Services]  WITH CHECK ADD  CONSTRAINT [ServicesIDLetterCheck] CHECK  ((isnumeric(left([serviceID],(1)))<>(1)))
GO

ALTER TABLE [dbo].[Services] CHECK CONSTRAINT [ServicesIDLetterCheck]
GO

-- Table: Studies
CREATE TABLE [dbo].[Studies](
    [studiesID] [char](4) NOT NULL,
    [name] [nvarchar](50) NOT NULL,
    [terms] [int] NOT NULL,
 CONSTRAINT [Studies_pk] PRIMARY KEY CLUSTERED
(
    [studiesID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [studies_services] FOREIGN KEY([studiesID])
REFERENCES [dbo].[Services] ([serviceID])
GO

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

ALTER TABLE [dbo].[Studies]  WITH CHECK ADD  CONSTRAINT [StudiesIDLetterCheck] CHECK  ((isnumeric(left([studiesID],(1)))<>(1)))
GO

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

-- Reference: studies_services (table: Studies)
ALTER TABLE Studies ADD CONSTRAINT studies_services
    FOREIGN KEY (studiesID)
    REFERENCES Services (serviceID);

-- Table: SubjectGrades
CREATE TABLE [dbo].[SubjectGrades](
    [gradeID] [int] IDENTITY(1,1) NOT NULL,
    [subjectID] [char](4) NOT NULL,
    [clientID] [int] NOT NULL,
    [grade] [int] NULL,
 CONSTRAINT [SubjectGrades_pk] PRIMARY KEY CLUSTERED
(
    [gradeID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[SubjectGrades]  WITH CHECK ADD  CONSTRAINT [Clients_Grades] FOREIGN KEY([clientID])
REFERENCES [dbo].[Clients] ([clientID])
GO

ALTER TABLE [dbo].[SubjectGrades] CHECK CONSTRAINT [Clients_Grades]
GO

ALTER TABLE [dbo].[SubjectGrades]  WITH CHECK ADD  CONSTRAINT [grades_Course] FOREIGN KEY([subjectID])
REFERENCES [dbo].[Subjects] ([subjectID])
GO

ALTER TABLE [dbo].[SubjectGrades] CHECK CONSTRAINT [grades_Course]
GO
```

```
ALTER TABLE [dbo].[SubjectGrades]  WITH CHECK ADD  CONSTRAINT [rangeOfGradesCheck] CHECK  (([grade] IS NULL OR [grade]>(1) AND [grade]<(6)))
GO

ALTER TABLE [dbo].[SubjectGrades] CHECK CONSTRAINT [rangeOfGradesCheck]
GO

-- Reference: Clients_Grades (table: SubjectGrades)
ALTER TABLE SubjectGrades ADD CONSTRAINT Clients_Grades
    FOREIGN KEY (clientID)
    REFERENCES Clients (clientID);

    -- Reference: grades_Course (table: SubjectGrades)
ALTER TABLE SubjectGrades ADD CONSTRAINT grades_Course
    FOREIGN KEY (subjectID)
    REFERENCES Subjects (subjectID);

-- Table: Subjects
CREATE TABLE [dbo].[Subjects](
    [subjectID] [char](4) NOT NULL,
    [termID] [char](4) NOT NULL,
    [SubjectName] [varchar](100) NOT NULL,
    [type] [varchar](3) NOT NULL,
 CONSTRAINT [Subjects_pk] PRIMARY KEY CLUSTERED
(
    [subjectID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Subjects]  WITH CHECK ADD  CONSTRAINT [Terms_Subjects] FOREIGN KEY([termID])
REFERENCES [dbo].[Terms] ([termID])
GO

ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [Terms_Subjects]
GO

ALTER TABLE [dbo].[Subjects]  WITH CHECK ADD  CONSTRAINT [SubjectIDLetterCheck] CHECK  ((isnumeric(left([subjectID],(1)))<>(1)))
GO

ALTER TABLE [dbo].[Subjects] CHECK CONSTRAINT [SubjectIDLetterCheck]
GO

-- Reference: Terms_Subjects (table: Subjects)
ALTER TABLE Subjects ADD CONSTRAINT Terms_Subjects
    FOREIGN KEY (termID)
    REFERENCES Terms (termID);

-- Table: SubjectsMeetings
CREATE TABLE [dbo].[SubjectsMeetings](
    [subjectMeetingID] [char](4) NOT NULL,
    [subjectID] [char](4) NOT NULL,
    [meetingID] [int] NOT NULL,
 CONSTRAINT [SubjectsMeetings_pk] PRIMARY KEY CLUSTERED
(
    [subjectMeetingID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[SubjectsMeetings]  WITH CHECK ADD  CONSTRAINT [Lectures_Services] FOREIGN KEY([subjectMeetingID])
REFERENCES [dbo].[Services] ([serviceID])
GO

ALTER TABLE [dbo].[SubjectsMeetings] CHECK CONSTRAINT [Lectures_Services]
GO

ALTER TABLE [dbo].[SubjectsMeetings]  WITH CHECK ADD  CONSTRAINT [Meetings_Lectures] FOREIGN KEY([meetingID])
REFERENCES [dbo].[Meetings] ([meetingID])
GO

ALTER TABLE [dbo].[SubjectsMeetings] CHECK CONSTRAINT [Meetings_Lectures]
GO

ALTER TABLE [dbo].[SubjectsMeetings]  WITH CHECK ADD  CONSTRAINT [Subjects_Lectures] FOREIGN KEY([subjectID])
REFERENCES [dbo].[Subjects] ([subjectID])
GO

ALTER TABLE [dbo].[SubjectsMeetings] CHECK CONSTRAINT [Subjects_Lectures]
GO

-- Reference: Meetings_Lectures (table: SubjectsMeetings)
ALTER TABLE SubjectsMeetings ADD CONSTRAINT Meetings_Lectures
    FOREIGN KEY (meetingID)
    REFERENCES Meetings (meetingID);

-- Reference: Lectures_Services (table: SubjectsMeetings)
ALTER TABLE SubjectsMeetings ADD CONSTRAINT Lectures_Services
    FOREIGN KEY (subjectMeetingID)
```

```
    REFERENCES Services (serviceID);

-- Reference: Subjects_Lectures (table: SubjectsMeetings)
ALTER TABLE SubjectsMeetings ADD CONSTRAINT Subjects_Lectures
    FOREIGN KEY (subjectID)
    REFERENCES Subjects (subjectID);

-- Table: SubjectsRecordings

CREATE TABLE [dbo].[SubjectsRecordings](
    [subjectRecordingID] [char](4) NOT NULL,
    [subjectID] [char](4) NOT NULL,
    [recordID] [int] NOT NULL,
 CONSTRAINT [SubjectsRecordings_pk] PRIMARY KEY CLUSTERED
(
    [subjectRecordingID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[SubjectsRecordings]  WITH CHECK ADD  CONSTRAINT [Recordings_SubjectsRecordings] FOREIGN KEY([recordID])
REFERENCES [dbo].[Recordings] ([recordID])
GO

ALTER TABLE [dbo].[SubjectsRecordings] CHECK CONSTRAINT [Recordings_SubjectsRecordings]
GO

ALTER TABLE [dbo].[SubjectsRecordings]  WITH CHECK ADD  CONSTRAINT [SubjectsRecordings_Subjects] FOREIGN KEY([subjectID])
REFERENCES [dbo].[Subjects] ([subjectID])
GO

ALTER TABLE [dbo].[SubjectsRecordings] CHECK CONSTRAINT [SubjectsRecordings_Subjects]
GO

-- Reference: Recordings_SubjectsRecordings (table: SubjectsRecordings)
ALTER TABLE SubjectsRecordings ADD CONSTRAINT Recordings_SubjectsRecordings
    FOREIGN KEY (recordID)
    REFERENCES Recordings (recordID);

-- Reference: SubjectsRecordings_Subjects (table: SubjectsRecordings)
ALTER TABLE SubjectsRecordings ADD CONSTRAINT SubjectsRecordings_Subjects
    FOREIGN KEY (subjectID)
    REFERENCES Subjects (subjectID);

-- Table: Terms
CREATE TABLE [dbo].[Terms](
    [termID] [char](4) NOT NULL,
    [studiesID] [char](4) NOT NULL,
    [startDate] [date] NOT NULL,
    [endDate] [date] NOT NULL,
 CONSTRAINT [Terms_pk] PRIMARY KEY CLUSTERED
(
    [termID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Terms]  WITH CHECK ADD  CONSTRAINT [conventions_studies] FOREIGN KEY([studiesID])
REFERENCES [dbo].[Studies] ([studiesID])
GO

ALTER TABLE [dbo].[Terms] CHECK CONSTRAINT [conventions_studies]
GO

ALTER TABLE [dbo].[Terms]  WITH CHECK ADD  CONSTRAINT [Services_Terms] FOREIGN KEY([termID])
REFERENCES [dbo].[Services] ([serviceID])
GO

ALTER TABLE [dbo].[Terms] CHECK CONSTRAINT [Services_Terms]
GO

ALTER TABLE [dbo].[Terms]  WITH CHECK ADD  CONSTRAINT [TermsIDLetterCheck] CHECK  ((isnumeric(left([termID],(1)))<>(1)))
GO

ALTER TABLE [dbo].[Terms] CHECK CONSTRAINT [TermsIDLetterCheck]
GO

ALTER TABLE [dbo].[Terms]  WITH CHECK ADD  CONSTRAINT [timeOfTermCheck] CHECK  (([endDate]>[startDate]))
GO

ALTER TABLE [dbo].[Terms] CHECK CONSTRAINT [timeOfTermCheck]
GO

-- Reference: Services_Terms (table: Terms)
ALTER TABLE Terms ADD CONSTRAINT Services_Terms
    FOREIGN KEY (termID)
    REFERENCES Services (serviceID);
```

```
-- Reference: conventions_studies (table: Terms)
ALTER TABLE Terms ADD CONSTRAINT conventions_studies
    FOREIGN KEY (studiesID)
    REFERENCES Studies (studiesID);

-- Table: Webinars
CREATE TABLE [dbo].[Webinars](
    [serviceID] [char](4) NOT NULL,
    [meetingID] [int] NOT NULL,
 CONSTRAINT [Webinars_pk] PRIMARY KEY CLUSTERED
(
    [serviceID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON, ALLOW_PAGE_LOCKS = ON,
OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT [Meetings_Webinars] FOREIGN KEY([meetingID])
REFERENCES [dbo].[Meetings] ([meetingID])
GO

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

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT [Webinars_Services] FOREIGN KEY([serviceID])
REFERENCES [dbo].[Services] ([serviceID])
GO

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

ALTER TABLE [dbo].[Webinars]  WITH CHECK ADD  CONSTRAINT [WebinarServiceIDLetterCheck] CHECK  ((isnumeric(left([serviceID],(1)))<>(1)))
GO

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

-- Reference: Meetings_Webinars (table: Webinars)
ALTER TABLE Webinars ADD CONSTRAINT Meetings_Webinars
    FOREIGN KEY (meetingID)
    REFERENCES Meetings (meetingID);

-- Reference: Webinars_Services (table: Webinars)
ALTER TABLE Webinars ADD CONSTRAINT Webinars_Services
    FOREIGN KEY (serviceID)
    REFERENCES Services (serviceID);

-- End of file.
```

Widoki

Przykładowa implementacja widoku, który dla każdej dostępnej usługi zwraca informację o łącznych przychodach z jej tytułu:

```
-- widok zwracający przychody dla kazdego oferowanego serwisu
create view RaportFinansowy as
    select serviceName, t.serviceID, przychody
    from Services as s
    join (select serviceID, sum(price) as przychody from OrderDetails
    group by serviceID) as t on t.serviceID=s.serviceID
```

Raport zwracający informację o liczbie zapisanych osób na przyszłe spotkania z kursów.

```
CREATE VIEW [dbo].[enrolledForCoursesMeetings]
AS
SELECT 'cm' + CAST(CM.meetingID AS char(6)) AS eventID, S.serviceName AS EventName, COUNT(A.attendanceID) AS enrolled, 'CMeeting' AS type
FROM            dbo.Attendees AS A INNER JOIN
                         dbo.Meetings AS M ON M.meetingID = A.meetingID INNER JOIN
                         dbo.CoursesMeetings AS CM ON CM.meetingID = M.meetingID INNER JOIN
                         dbo.Courses AS C ON C.serviceID = CM.serviceID INNER JOIN
                         dbo.Services AS S ON S.serviceID = C.serviceID
WHERE        (M.datetime > GETDATE())
GROUP BY CM.meetingID, S.serviceName
```

Raport zwracający informację o liczbie zapisanych osób na przyszłe spotkania ze studiów.

```
CREATE VIEW [dbo].[enrolledForStudiesMeetings]
AS
SELECT 'sm' + CAST(SM.meetingID AS char(4)) AS eventID, S.serviceName, COUNT(A.attendanceID) AS enrolled, 'SMeeting' AS type
FROM dbo.Attendees AS A INNER JOIN
    dbo.Meetings AS M ON M.meetingID = A.meetingID INNER JOIN
    dbo.SubjectsMeetings AS SM ON SM.meetingID = M.meetingID INNER JOIN
    dbo.Services AS S ON S.serviceID = SM.subjectMeetingID
```

```
WHERE (M.datetime > GETDATE())
GROUP BY SM.meetingID, S.serviceName
```

Raport zwracający informację o liczbie zapisanych osób na przyszłe studia, semestry, kursy, webinary.

```
CREATE VIEW [dbo].[PeopleEnrolledForFutureStudiesCoursesWebinars]
AS
SELECT S.serviceID AS eventID, S.serviceName AS EventName, COUNT(O.orderID) AS enrolled, S.type
FROM dbo.Orders AS O INNER JOIN
    dbo.OrderDetails AS OD ON OD.orderID = O.orderID RIGHT OUTER JOIN
    dbo.Services AS S ON S.serviceID = OD.serviceID
WHERE (S.startDate > GETDATE()) AND (S.type <> 'pSpotkanie')
GROUP BY S.serviceID, S.serviceName, S.type
GO
```

Raport zwracający informację o liczbie zapisanych osób na przyszłe wydarzenia.

```
CREATE VIEW PeopleEnrolledForFutureEvents as
select *
from PeopleEnrolledForFutureStudiesCoursesWebinars
union
select *
from enrolledForCoursesMeetings
union
select *
from enrolledForStudiesMeetings
```

Lista dostępnych (opłaconych) spotkań w ramach webinarów.

```
create view  AvalibleWebinarMeetings as
    select c.clientID, c.firstName+' '+c.lastName as name , m.meetingLink, m.datetime  from Clients as c
    join Orders as o on o.clientID=c.clientID and receiptDate is not null
    join OrderDetails as od on o.orderID=od.orderID
    join Services as s on s.serviceID=od.serviceID
    join webinars as w on s.serviceID=w.serviceID
    join Meetings as m on m.meetingID=w.meetingID
```

Lista dostępnych (opłaconych) spotkań w ramach przedmiotów w ramach studiów.

```
create view AvalibleSubjectMeetings as
    select c.clientID, c.firstName+' '+c.lastName as name , m.meetingLink, m.datetime  from Clients as c
    join Orders as o on o.clientID=c.clientID and receiptDate is not null
    join OrderDetails as od on o.orderID=od.orderID
    join Services as s on s.serviceID=od.serviceID
    join Terms as t on t.termID=s.serviceID
    join Subjects as su on su.termID=t.termID
    join SubjectsMeetings as sm on sm.subjectID=su.subjectID
    join Meetings as m on m.meetingID=sm.meetingID

    union all

    select c.clientID, c.firstName+' '+c.lastName as name , m.meetingLink, m.datetime from Clients as c
    join Orders as o on o.clientID=c.clientID and receiptDate is not null
    join OrderDetails as od on o.orderID=od.orderID
    join Services as s on s.serviceID=od.serviceID
    join SubjectsMeetings as sm on sm.subjectMeetingID=s.serviceID
    join Meetings as m on m.meetingID=sm.meetingID
```

Lista dostępnych (opłaconych) spotkań w ramach kursów.

```
create view AvalibleCourseMeetings as
    select c.clientID, c.firstName+' '+c.lastName as name , m.meetingLink, m.datetime  from Clients as c
    join Orders as o on o.clientID=c.clientID and receiptDate is not null
    join OrderDetails as od on o.orderID=od.orderID
    join Services as s on s.serviceID=od.serviceID
    join Courses as cs on s.serviceID=cs.serviceID
    join CoursesMeetings as cm on cm.serviceID=cs.serviceID
    join Meetings as m on m.meetingID=cm.meetingID
```

Wszystkie dostępne (opłacone) spotkania.

```
create view AllAvalibleMeetings as
    select * from AvalibleSubjectMeetings
    union all
    select * from AvalibleCourseMeetings
    union all
    select * from AvalibleWebinarMeetings
```

Raport kolizji ze względu na czas spotkań.

```
Create VIEW CollisionReport as
    SELECT Distinct A.clientID, A.name, A.dateTime AS CollisionDate
    FROM AllAvalibleMeetings A
    JOIN AllAvalibleMeetings B ON A.clientID = B.clientID AND A.dateTime <= B.dateTime
    AND A.dateTime + DATEADD(HOUR, 1, '1900-01-01T00:00:00') > B.dateTime and A.meetingLink!=B.meetingLink
```

Lista klientów wraz z zdanymi kursami.

```
create view PassedCourses as
    select c.ClientID, firstName, LastName, address from clients as c
    join CourseGrade as g on g.clientID=c.clientID and
    passed is not Null and passed = 1
```

Lista klientów, którzy zaliczyli wszystkie przedmioty na studiach.

```
create view PassedStudiesGrades as
    with temp as
        (select c.ClientID, firstName, LastName, s.subjectID,
        s.subjectName, t.studiesID, grade, name as studiesName, address+' '+city+' '+region address
        from clients as c
        join SubjectGrades as g on g.clientID=c.clientID
        join Subjects as s on s.subjectID=g.subjectID
        join Terms as t on t.termID=s.termID
        join studies on studies.studiesID=t.studiesID),
        amountOfSubjects as
            (select s.StudiesID, count(*) as amount from Studies as s
            join terms as t on t.studiesID=s.studiesID
            join Subjects on Subjects.termID=t.termID
            group by s.studiesID)

select a.StudiesID, StudiesName, ClientID, firstName, LastName, address
from amountOfSubjects as a
    join (select ClientID, studiesName, studiesID,firstName, LastName, address, count(*) as amountOfPassed from temp
    where grade is not null and grade>2
    group by ClientID, studiesName, studiesID,firstName, LastName, address) as t
    on t.studiesID=a.studiesID and amount=amountOfPassed
```

Lista klientów, który zaliczyli wszystkie praktyki.

```
create view PassedAllInternships as
    with temp as
        (select c.ClientID, firstName, LastName, i.InternshipID,
        s.studiesID, passed, name as studiesName, address+' '+city+' '+region address
        from clients as c
        join IndividualInternship as i on i.clientID=c.clientID
        join Practices as p on p.PracticeID=i.PracticeID
        join studies as s on s.studiesID=p.studiesID),
        amountOfPractices as
            (select s.StudiesID, count(*) as amount from Studies as s
            join Practices as p on p.studiesID=s.studiesID
            group by s.studiesID)

    select a.StudiesID, StudiesName, ClientID, firstName, LastName, address from amountOfPractices as a
    join (select ClientID, studiesName, studiesID,firstName, LastName, address, count(*) as amountOfPassed from temp
    where Passed is not null and passed=1 group by ClientID, studiesName, studiesID,firstName, LastName, address) as t
    on t.studiesID=a.studiesID and amount=amountOfPassed
```

Lista klientów, który zakończyli studia pozytywnie.

```
create view PassedStudies as
    select g.clientID, g.firstName, g.lastName, g.address from PassedStudiesGrades as g
    join PassedAllInternships as i
    on i.clientID=g.clientID
```

Frekwencja pojedynczych spotkań.

```
create view SingleMeetingFrequentionReport as
    select m.meetingID, count(*) as AmountOfRegistered, sum(present) as AmountOfPresent, format((cast(sum(present) as float)/count(*))*100,
'0.00')+'%' as frequention, datetime from Meetings as m
    join Attendees as a on m.meetingID=a.meetingID and datetime<=GETDATE()
    group by m.meetingID, datetime
```

Frekwencja w ramach przedmiotów.

```
create view SubjectFrequentionReport as
    select s.subjectID, SubjectName, endDate, t.termID, studiesID,
    min(cast(replace((f.frequention),'%','') as decimal(5,2))) as 'smallestFrequention',
    max(cast(replace((f.frequention),'%','') as decimal(5,2))) as 'biggestFrequention',
    cast(avg(cast(replace((f.frequention),'%','') as decimal(5,2))) as decimal(5,2)) as 'avarageFrequention'
    from Subjects as s
    join terms as t on t.termID=s.termID and (endDate<GETDATE() or subjectID='p001')
    join SubjectsMeetings sm on s.subjectID=sm.subjectID
    join SingleMeetingFrequentionReport as f on f.meetingID=sm.meetingID
    group by s.subjectID, SubjectName,endDate, t.termID, studiesID
```

Frekwencja w ramach zjazdu.

```
create view TermFrequentionReport as
    with tempDesc as
        (select top 1 subjectID, subjectName, termID, avarageFrequention
        from SubjectFrequentionReport as sr
        order by avarageFrequention desc
        ),
        tempAsc as (select top 1 subjectID, subjectName, termID, avarageFrequention
        from SubjectFrequentionReport as sr
        order by avarageFrequention asc),
        tempAvg as(
        select sr.termID, avg(avarageFrequention) as avarage
        from  SubjectFrequentionReport as sr
        group by sr.termID
        )

select t.termID, maksi.subjectID as IdOfMostPopularSubject, maksi.subjectName as NameOfMostPopularSubject, maksi.avarageFrequention as
frequentionOfMostPopular,
        mini.subjectID as IdOfLeastPopularSubject, mini.subjectName as NameOfLeastPopularSubject, mini.avarageFrequention as
frequentionOfLeastPopular,
        a.avarage as avarageFrequentionOnThisTerm
    from terms as t
    join (select * from tempDesc) as maksi on maksi.termID=t.termID
    join (select * from tempAsc) as mini on mini.termID=t.termID
    join (select termID, avarage from tempAvg ) as a on a.termID=t.termID
```

Lista osób, które zapłaciły za usługi

```
CREATE VIEW PayedServices
AS
SELECT firstName, lastName, c.clientID, serviceID
FROM Clients c
join Orders o ON c.clientID = o.clientID
join OrderDetails d ON o.orderID = d.orderID
WHERE o.receiptDate IS NOT NULL;
```

Lista osób, które korzystają z kursów których nie opłaciły

```
CREATE VIEW DebtorsCourses
AS
SELECT a.clientID, r.serviceID
FROM Access a
join CourseRecords r ON a.recordID = r.recordID
left join PayedServices p ON a.clientID = p.clientID and r.serviceID = p.serviceID
WHERE p.clientID is null and p.serviceID is null

UNION

SELECT a.clientID, C.serviceID
FROM Attendees a
JOIN CoursesMeetings c ON a.meetingID = c.meetingID
LEFT JOIN PayedServices p ON a.clientID = p.clientID and c.serviceID = p.serviceID
WHERE p.clientID is null and p.serviceID is null;
```

Lista osób, które korzystają ze spotkań na studiach za które nie zapłaciły

```
CREATE VIEW DebtorsSubjectMeetings
AS
SELECT a.clientID, s.termID
FROM Access a
JOIN SubjectsRecordings r ON a.recordID = r.recordID
JOIN Subjects s ON r.subjectID = s.subjectID
LEFT JOIN PayedServices p ON a.clientID = p.clientID and s.termID = p.serviceID
WHERE p.clientID IS NULL and p.serviceID IS NULL
```

```
UNION

SELECT a.clientID, m.subjectMeetingID
FROM Attendees a
JOIN SubjectsMeetings m ON a.meetingID = m.meetingID
JOIN Subjects s ON m.subjectID = s.subjectID
LEFT JOIN PayedServices p on a.clientID = p.clientID and (m.subjectMeetingID = p.serviceID OR s.termID = p.serviceID)
WHERE p.clientID IS NULL
```

Lista osób, które korzystają z webinarów za które nie zapłaciły

```
CREATE VIEW DebtorsWebinars
AS
SELECT a.clientID, w.serviceID
FROM Attendees a
join Webinars w ON a.meetingID = w.meetingID
left join PayedServices p ON a.clientID = p.clientID and w.serviceID = p.serviceID
WHERE p.clientID IS NULL and p.serviceID IS NULL
```

Lista osób, które mają dostęp do usług za które nie zapłaciły.

```
CREATE VIEW ListOfDebtors
AS
SELECT c.clientID, c.firstName, c.lastName, d.serviceID
FROM Clients c
JOIN (
    SELECT * FROM DebtorsCourses
    UNION
    SELECT * FROM DebtorsWebinars
    UNION
    SELECT * FROM DebtorsSubjectMeetings
) AS d ON c.clientID = d.clientID;
```

Funkcje

Funkcja zwracająca tabelę z wszystkimi ID meetingów dla danego kursu (potrzebne do innych procedur)

```
CREATE FUNCTION MeetingsDuringCourse(@serviceID CHAR(4))
RETURNS TABLE
AS
RETURN (
    select M.meetingID
    from Services S
        join Courses C on C.serviceID = S.serviceID
        join CoursesMeetings CM on CM.serviceID = C.serviceID
        join Meetings M on M.meetingID = CM.meetingID
    where S.serviceID = @serviceID
);
```

Funkcja zwracająca tabelę z wszystkimi ID meetingów dla danego webinaru (potrzebne do innych procedur)

```
CREATE FUNCTION MeetingsDuringWebinar(@serviceID CHAR(4))
RETURNS TABLE
AS
RETURN (
    select M.meetingID
    from Services S
        join Webinars W on W.serviceID = S.serviceID
        join Meetings M on M.meetingID = W.meetingID
    where S.serviceID = @serviceID
);
```

Funkcja zwracająca tabelę z wszystkimi ID meetingów dla danego spotkania w ramach studiów(potrzebne do innych procedur)

```
CREATE FUNCTION MeetingsDuringSubject(@serviceID CHAR(4))
RETURNS TABLE
AS
RETURN (
    select M.meetingID
    from Services S
        join SubjectsMeetings SM on SM.subjectMeetingID = S.serviceID
        join Meetings M on M.meetingID = SM.meetingID
    where S.serviceID = @serviceID
);
```

Funkcja zwracająca tabelę z wszystkimi ID meetingów dla danego semestru (zjazdu) w ramach studiów(potrzebne do innych procedur)

```
CREATE FUNCTION MeetingsDuringTerm(@serviceID CHAR(4))
RETURNS TABLE
AS
RETURN (
    select M.meetingID
    from Services S
        join Terms T on T.termID = S.serviceID
        join Subjects on Subjects.termID = T.termID
        join SubjectsMeetings SM on SM.subjectID = Subjects.subjectID
        join Meetings M on M.meetingID = SM.meetingID
    where S.serviceID = @serviceID
);
```

Funkcja zwracająca tabelę z wszystkimi ID meetingów dla danych studiów (potrzebne do innych procedur)

```
CREATE FUNCTION MeetingsDuringStudies(@serviceID CHAR(4))
RETURNS TABLE
AS
RETURN (
    select M.meetingID
    from Services S
        join Studies on Studies.studiesID = S.serviceID
        join Terms T on T.studiesID = Studies.studiesID
        join Subjects on Subjects.termID = T.termID
        join SubjectsMeetings SM on SM.subjectID = Subjects.subjectID
        join Meetings M on M.meetingID = SM.meetingID
    where S.serviceID = @serviceID
);
```

Funkcja zwracająca tabelę z wszystkimi ID meetingów dla dowolnej usługi (potrzebne do innych procedur)

```
CREATE FUNCTION MeetingsDuringService(@serviceID CHAR(4))
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM
    (
        SELECT * FROM MeetingsDuringCourse(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'kurs'
        UNION ALL
        SELECT * FROM MeetingsDuringWebinar(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'webinar'
        UNION ALL
        SELECT * FROM MeetingsDuringStudies(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'studia'
        UNION ALL
        SELECT * FROM MeetingsDuringSubject(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'pSpotkanie'
        UNION ALL
        SELECT * FROM MeetingsDuringTerm(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'semestr'
    ) AS MergedMeetings
);
```

Funkcja zwracająca wszystkie nagrania dla danego kursu

```
create function RecordingsDuringCourse(@serviceID CHAR(4))
returns TABLE
as
return (
    select CR.recordID
    from Services S
        join Courses C on C.serviceID = S.serviceID
        join CourseRecords CR on CR.serviceID = C.serviceID
    where S.serviceID = @serviceID
)
```

Funkcja zwracająca wszystkie nagrania dla danego semestru

```
create function RecordingsDuringTerm(@serviceID CHAR(4))
returns TABLE
as
return (
    select SR.recordID
    from Services S
        join Terms T on T.termID = S.serviceID
        join Subjects on Subjects.termID = T.termID
        join SubjectsRecordings SR on SR.subjectID = Subjects.subjectID
    where S.serviceID = @serviceID
)
```

Funkcja zwracająca wszystkie nagrania dla danych studiów

```
create function RecordingsDuringStudies(@serviceID CHAR(4))
returns TABLE
as
return (
    select SR.recordID
    from Services S
        join Studies on Studies.studiesID = S.serviceID
        join Terms T on T.studiesID = Studies.studiesID
        join Subjects on Subjects.termID = T.termID
        join SubjectsRecordings SR on SR.subjectID = Subjects.subjectID
    where S.serviceID = @serviceID
)
```

Funkcja zwracająca wszystkie nagrania dla dowolnej usługi

```
CREATE FUNCTION RecordingsDuringService(@serviceID CHAR(4))
RETURNS TABLE
AS
RETURN
(
    SELECT *
    FROM
    (
        SELECT * FROM RecordingsDuringCourse(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'kurs'
        UNION ALL
        SELECT * FROM RecordingsDuringStudies(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'studia'
        UNION ALL
        SELECT * FROM RecordingsDuringTerm(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'semestr'
    ) AS MergedMeetings
);
```

Procedury

Przykładowa implementacja procedury dodające nowej klienta:

```
Procedura dodająca nowego klienta. Jako argumenty przyjmuje imię, nazwisko, adres, miasto oraz województwo

CREATE PROCEDURE addNewClient
    @p_firstName NVARCHAR(30),
    @p_lastName NVARCHAR(30),
    @p_address NVARCHAR(100),
    @p_city NVARCHAR(50),
    @p_region NVARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO Clients (firstName, lastName, address, city, region)
    VALUES (@p_firstName, @p_lastName, @p_address, @p_city, @p_region);
END;
```

Przykładowa implementacja procedury dodającej nowego pracownika:

```
Procedura dodająca nowego pracownika. Jako argumenty przyjmuje imię, nazwisko oraz stanowisko pracy.

CREATE PROCEDURE addNewEmployee
    @p_firstName NVARCHAR(30),
    @p_lastName NVARCHAR(30),
    @p_position NVARCHAR(100)
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO Employees(firstName,lastName,position)
    VALUES (@p_firstName, @p_lastName, @p_position);
END;
```

Procedura dodająca obecności dla danego klienta dla danego serwisu

```
CREATE PROCEDURE addCustomerToServiceAttendance
    @ClientID INTEGER,
    @ServiceID CHAR(4)
AS
BEGIN
    insert into Attendees (clientID, meetingID, present, substituteAttendanceID)
    select @ClientID, MDS.meetingID, NULL, NULL
```

```
    from MeetingsDuringService(@ServiceID) as MDS
END;
```

Procedura przyznająca dostęp do nagrań w ramach kursu, studiów lub semestru

```
CREATE PROCEDURE giveAccessToCourseOrStudiesRecordings
    @ServiceID CHAR(4),
    @ClientID INTEGER
AS
BEGIN
    insert into Access (clientID, recordID, lastDay, status)
    select @ClientID, RDS.recordID, DATEADD(DAY, 30, GETDATE()), NULL
    from RecordingsDuringService(@ServiceID) as RDS
END;
```

Procedura dzięki której można manualnie przyznać dostęp użytkownikowi

```
CREATE PROCEDURE enrollManually
    @ServiceID CHAR(4),
    @ClientID INTEGER
AS
BEGIN
    DECLARE @Type VARCHAR(10);
    SET @Type = (select type from Services where serviceID = @ServiceID)

    IF @Type IN ('kurs', 'studia', 'semestr')
    BEGIN
        EXEC giveAccessToCourseOrStudiesRecordings @ClientID, @ServiceID;
    END
    EXEC addCustomerToServiceAttendance @ClientID, @ServiceID;
END;
```

Przyznanie dostępu konkretnemu klientowi do konkretnego nagrania

```
create procedure giveClientAccessToRecording
    @clientID INTEGER,
    @recordID INTEGER
as
begin
    insert into Access(recordID, clientID, lastDay, status)
    values (@recordID, @clientID, DATEADD(day, 30, getdate()), NULL)
end;
```

Dodawanie nowego nagrania spotkania

```
CREATE PROCEDURE [dbo].[createRecording]
    @MeetingID INTEGER
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        DECLARE @meetingLink VARCHAR(100);
        DECLARE @recordLink VARCHAR(100);
        DECLARE @newRecordID INTEGER;

        SET @meetingLink = (SELECT meetingLink FROM Meetings WHERE MeetingID = @MeetingID);
        SET @recordLink = 'https://www.educewave.com/r' + SUBSTRING(@meetingLink, 28, 6);

        INSERT INTO Recordings (recordLink)
        VALUES (@recordLink);

        SET @newRecordID = SCOPE_IDENTITY();

        UPDATE Recordings
        SET recordLink = @recordLink + CAST(@newRecordID AS VARCHAR(5))
        WHERE recordID = @newRecordID;

        UPDATE Meetings
        SET recordID = @newRecordID
        WHERE meetingID = @MeetingID;

        COMMIT;
    END TRY
    BEGIN CATCH
        ROLLBACK;
    END CATCH;
END;
GO
```

Tworzenie nowego nagrania i automatyczne przydzielanie dostępu wszystkim uprawnionym.

```
create procedure giveAccessToNewRecording
    @MeetingID INTEGER
as
begin
    exec createRecording @MeetingID;

    declare @newRecordingID INTEGER;
    set @newRecordingID = (select max(recordID) from Recordings)

    insert into Access(clientID, recordID, lastDay, status)
    select A.clientID, @newRecordingID, DATEADD(day, 30, GETDATE()), NULL
    from Attendees A
    where A.meetingID = @MeetingID

end;
```

Zmiana daty spotkania.

```
CREATE PROCEDURE changeMeetingDate
@meetingID INT,
@newDate DATETIME
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Meetings WHERE meetingID = @meetingID)
    BEGIN
        UPDATE Meetings
        SET datetime = @newDate
        WHERE meetingID = @meetingID;
    END
    ELSE
    BEGIN
    RAISERROR('Meeting %s does not exist.', 16, 1, @meetingID);
    END
END;
```

Zmiana dostępności danej usługi (włączenie oraz wyłączenie)

```
CREATE PROCEDURE changeAvailability
@serviceID CHAR(4)
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Services WHERE serviceID = @serviceID)
    BEGIN
        UPDATE Services
        SET availability = ~availability
        WHERE serviceID = @serviceID;
    END
    ELSE
    BEGIN
        RAISERROR('Service %s does not exist.', 16, 1, @serviceID);
    END
END;
```

****Triggery

Jak ktoś kupuje dostęp do usługi, to dodawany na listę obecności

```
CREATE TRIGGER AddToAttendanceList
ON Orders
AFTER UPDATE
AS
BEGIN
    IF UPDATE(ReceiptDate)
    BEGIN
        DECLARE @ReceiptDateBefore DATETIME;
        DECLARE @ServiceID CHAR(4);
        DECLARE @ClientID INTEGER;
        DECLARE @ReceiptDateAfter DATETIME;

        -- Pobierz stare i nowe wartości statusu
        SELECT @ReceiptDateBefore = D.receiptDate,
               @ServiceID = OD.serviceID,
               @ClientID = D.clientID,
               @ReceiptDateAfter = I.receiptDate
        FROM deleted D
        JOIN inserted I ON D.orderID = I.orderID
        JOIN OrderDetails OD ON OD.orderID = I.orderID;

        IF @ReceiptDateBefore IS NULL AND @ReceiptDateAfter IS NOT NULL
        BEGIN
```

```
            EXEC addCustomerToServiceAttendance @ClientID, @ServiceID;
        END
    END
END;
```

Trigger dający użytkownikowi dostęp do obowiązkowych nagrań w momencie zaksięgowania płatności

```
CREATE TRIGGER AddToAccessList
ON Orders
AFTER UPDATE
AS
BEGIN
    IF UPDATE(ReceiptDate)
    BEGIN
        DECLARE @ReceiptDateBefore DATETIME;
        DECLARE @ServiceID CHAR(4);
        DECLARE @ClientID INTEGER;
        DECLARE @ReceiptDateAfter DATETIME;

        SELECT @ReceiptDateBefore = D.receiptDate,
               @ServiceID = OD.serviceID,
               @ClientID = D.clientID,
               @ReceiptDateAfter = I.receiptDate
        FROM deleted D
        JOIN inserted I ON d.orderID = I.orderID
        JOIN OrderDetails OD ON OD.orderID = D.orderID
        JOIN Services S ON S.serviceID = OD.serviceID and S.type IN ('kurs', 'studia', 'semestr');

        IF @ReceiptDateBefore IS NULL AND @ReceiptDateAfter IS NOT NULL
        BEGIN
            EXEC giveAccessToCourseOrStudiesRecordings @ClientID, @ServiceID;
        END
    END
END;
```

Trigger wstawiający nowe nagrania kursów jeżeli nie przekraczają limitu spotkań oraz aktywujący dostępność usługi jeżeli zostały wprowadzone wszystkie spotkania i nagrania.

```
CREATE TRIGGER AfterInsertCoursesRecords
ON CourseRecords
AFTER INSERT
AS
BEGIN
    DECLARE @currentServiceID CHAR(4), @totalRecords INT, @addedRecords INT;

    DECLARE serviceID_cursor CURSOR FOR
        SELECT c.serviceID, c.totalRecords, COUNT(*) as addedRecords
        FROM inserted i
        JOIN Courses c ON i.serviceID = c.serviceID
        JOIN CourseRecords r ON c.serviceID = r.serviceID
        GROUP BY c.serviceID, c.totalRecords;

    OPEN serviceID_cursor;
    FETCH NEXT FROM serviceID_cursor INTO @currentServiceID, @totalRecords, @addedRecords;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (@totalRecords is NULL)
        BEGIN
            DELETE FROM CourseRecords
            WHERE courseRecordID IN (SELECT courseRecordID FROM inserted WHERE serviceID = @currentServiceID);
            RAISERROR('There should be no records for ServiceID %s', 16, 1, @currentServiceID);
        END
        ELSE IF (@totalRecords > @addedRecords)
        BEGIN
            RAISERROR('Too few records for ServiceID %s', 16, 1, @currentServiceID);

        END
        ELSE IF (@totalRecords < @addedRecords)
        BEGIN
            DELETE FROM CourseRecords
            WHERE courseRecordID IN (SELECT courseRecordID FROM inserted WHERE serviceID = @currentServiceID);
            RAISERROR('Too many records for ServiceID %s', 16, 1, @currentServiceID);
        END
        ELSE
        BEGIN
            EXEC UpdateAvailability @serviceID = @currentServiceID;
        END

        FETCH NEXT FROM serviceID_cursor INTO @currentServiceID, @totalRecords, @addedRecords;
    END

    CLOSE serviceID_cursor;
    DEALLOCATE serviceID_cursor;
END;
```

Trigger wstawiający nowe spotkania kursów jeżeli nie przekraczają limitu spotkań oraz aktywujący dostępność usługi jeżeli zostały wprowadzone wszystkie spotkania i nagrania.

```
CREATE TRIGGER AfterInsertCoursesMeetings
ON CoursesMeetings
AFTER INSERT
AS
BEGIN
    DECLARE @currentServiceID CHAR(4), @totalMeetings INT, @plannedMeetings INT;

    DECLARE serviceID_cursor CURSOR FOR
        SELECT c.serviceID, c.totalMeetings, COUNT(*) as plannedMeetings
        FROM inserted i
        JOIN Courses c ON i.serviceID = c.serviceID
        JOIN CoursesMeetings m ON c.serviceID = m.serviceID
        GROUP BY c.serviceID, c.totalMeetings;

    OPEN serviceID_cursor;
    FETCH NEXT FROM serviceID_cursor INTO @currentServiceID, @totalMeetings, @plannedMeetings;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        IF (@totalMeetings is NULL)
        BEGIN
            DELETE FROM CoursesMeetings
            WHERE courseMeetingID IN (SELECT courseMeetingID FROM inserted WHERE serviceID = @currentServiceID);
            RAISERROR('There should be no meetings for ServiceID %s', 16, 1, @currentServiceID);
        END
        ELSE IF (@totalMeetings > @plannedMeetings)
        BEGIN
            RAISERROR('Too few meetings for ServiceID %s', 16, 1, @currentServiceID);

        END
        ELSE IF (@totalMeetings < @plannedMeetings)
        BEGIN
            DELETE FROM CoursesMeetings
            WHERE courseMeetingID IN (SELECT courseMeetingID FROM inserted WHERE serviceID = @currentServiceID);
            RAISERROR('Too many meetings for ServiceID %s', 16, 1, @currentServiceID);
        END
        ELSE
        BEGIN
            EXEC UpdateAvailability @serviceID = @currentServiceID;
        END

        FETCH NEXT FROM serviceID_cursor INTO @currentServiceID, @totalMeetings, @plannedMeetings;
    END

    CLOSE serviceID_cursor;
    DEALLOCATE serviceID_cursor;
END;
```

Indeksy

Access

Na identyfikatorze klienta (clientID)

```
CREATE NONCLUSTERED INDEX Access_ClientID ON Access (clientID);
```

Na identyfikatorze nagrania (recordID)

```
CREATE NONCLUSTERED INDEX Access_RecordID ON Access (recordID);
```

Attendees

Na identyfikatorze klienta (clientID)

```
CREATE NONCLUSTERED INDEX Attendees_clientID ON Attendees (clientID);
```

Na identyfikatorze spotkania (meetingID)

```
CREATE NONCLUSTERED INDEX Attendees_meetingID ON Attendees(meetingID);
```

Clients

Na identyfikatorze klienta (clientID)

```
CREATE UNIQUE NONCLUSTERED INDEX Clients_clientID ON Clients (clientID);
```

Na imieniu i nazwisku (firstName, lastName)

```
CREATE NONCLUSTERED INDEX Clients_name ON Clients (firstName, lastName);
```

Na adresie klienta (city, region)

```
CREATE NONCLUSTERED INDEX Clients_location ON Clients (city, region);
```

CourseGrade

Na identyfikatorze klienta (clientID)

```
CREATE NONCLUSTERED INDEX CourseGrades_clientID ON CourseGrade (clientID);
```

Na statusie kursu (passed)

```
CREATE NONCLUSTERED INDEX CourseGrades_passed ON CourseGrade (passed);
```

Na identyfikatorze usługi (serviceID)

```
CREATE NONCLUSTERED INDEX CourseGrades_serviceID ON CourseGrade (serviceID);
```

Employees

Na identyfikatorze pracownika (employeeID)

```
CREATE UNIQUE NONCLUSTERED INDEX Employees_employeeID ON Employees (employeeID);
```

Na imieniu i nazwisku (firstName, lastName)

```
CREATE UNIQUE NONCLUSTERED INDEX Employees_employeeID ON Employees (employeeID);
```

Na pozycji pracownika (position)

```
CREATE NONCLUSTERED INDEX Employees_position ON Employees (position);
```

IndividualInternship

Na identyfikatorze klienta (clientID)

```
CREATE NONCLUSTERED INDEX Internships_clientID ON IndividualInternship (ClientID);
```

Na datach rozpoczęcia i zakończenia stażu (StartDate, EndDate)

```
CREATE NONCLUSTERED INDEX Internships_date ON IndividualInternship (StartDate, EndDate);
```

Interpreters

Na identyfikatorze pracownika (employeeID)

```
CREATE NONCLUSTERED INDEX Interpreters_EmployeeID ON Interpreters (employeeID);
```

Na język, którym posługuje się tłumacz (language)

```
CREATE NONCLUSTERED INDEX Interpreters_Language ON Interpreters (language);
```

Meetings

Na dacie spotkania (datetime)

```
CREATE NONCLUSTERED INDEX Meetings_date ON Meetings (datetime);
```

Na język, w którym jest prowadzone spotkanie (language)

```
CREATE NONCLUSTERED INDEX Meetings_language ON Meetings (language);
```

Na indentyfikator spotkania (meetingID)

```
CREATE NONCLUSTERED INDEX Meetings_meetingID ON Meetings (meetingID);
```

SubjectGrades

Na identyfikatorze klienta (clientID)

```
CREATE NONCLUSTERED INDEX SubjectGrades_clientID ON SubjectGrades (clientID);
```

Na oceny przedmiotów ze studiów (grade)

```
CREATE NONCLUSTERED INDEX SubjectGrades_grades ON SubjectGrades (grade);
```

Na indentyfikator przedmiotu ze studiów (subjectID)

```
CREATE NONCLUSTERED INDEX SubjectGrades_subjectID ON SubjectGrades (subjectID);
```

Uprawnienia do poszczególnych tabel i funkcjonalności

Utworzenie ról, do których zostaną przypisani konkretni użytkownicy bazy

```
create role admin
create role administration_office_employee
create role director
create role education_office_employee
create role interpreter
create role student
create role tutor
```

Raport listy dłużników - pracownik administracji

```
grant select on ListOfDebtors to administration_office_employee
grant select on DebtorsCourses to administration_office_employee
grant select on DebtorsSubjectMeetings to administration_office_employee
grant select on DebtorsWebinars to administration_office_employee
```

Raport frekwencji - pracownik administracji

```
grant select on SingleMeetingFrequentionReport to administration_office_employee
grant select on SubjectFrequentionReport to administration_office_employee
grant select on TermFrequentionReport to administration_office_employee
```

Raport osób zapisanych na przyszłe wydarzenia - pracownik administracji

```
grant select on enrolledForStudiesMeetings to administration_office_employee
grant select on enrolledForCoursesMeetings to administration_office_employee
grant select on PeopleEnrolledForFutureEvents to administration_office_employee
grant select on PeopleEnrolledForFutureStudiesCoursesWebinars to administration_office_employee
```

Raport osób, które zdały studia lub kurs - pracownik administracji

```
grant select on PassedStudies to administration_office_employee
grant select on PassedStudiesGrades to administration_office_employee
grant select on PassedAllInternships to administration_office_employee
```

Raport kolizji - pracownik administracji

```
grant select on AvalibleSubjectMeetings to administration_office_employee
grant select on AvalibleCourseMeetings to administration_office_employee
grant select on AvalibleWebinarMeetings to administration_office_employee
grant select on AllAvalibleMeetings to administration_office_employee
```

Sprawdzanie obecności

```
grant select, insert, update on Attendees to tutor
```

Usuwanie nagrań - administrator

```
grant delete on Recordings to admin
```

Zmiana daty spotkania z przyczyn losowych + dodawanie spotkań (układanie harmonogramu zajęć) - pracownik adminisracji<

```
grant insert, alter on Meetings to administration_office_employee
grant execute on changeMeetingDate to administration_office_employee
```

Manualne dodawanie klienta do spotkań - odroczenie płatności

```
grant select on RecordingsDuringCourse to director
grant select on RecordingsDuringStudies to director
grant select on RecordingsDuringTerm to director
grant select on RecordingsDuringService to director
grant select on RecordingsDuringService to director
grant execute on giveAccessToCourseOrStudiesRecordings to director
grant execute on addCustomerToServiceAttendance to director
grant execute on enrollManually to director
```

Dodawanie toku studiów - pracownik biura dydaktyki

```
grant insert on Services to education_office_employee
grant insert on Studies to education_office_employee
grant insert on Terms to education_office_employee
grant insert on Subjects to education_office_employee
grant insert on SubjectsMeetings to education_office_employee
grant insert on SubjectsRecordings to education_office_employee
```

Dodawanie nowych webinarów - pracownik biura dydaktyki

```
grant insert on Webinars to education_office_employee
```

Dodawanie nowych kursów - pracownik biura dydaktyki

```
grant insert on Courses to education_office_employee
grant insert on CoursesMeetings to education_office_employee
grant insert on CourseRecords to education_office_employee
```

Możliwość przeglądania oferty - wszyscy

```
grant select on Services to public
```

Dodawanie i usuwanie pracowników

```
grant insert, delete on Employees to director
grant insert, delete on Interpreters to director
```

Raport finansowy - pracownik biura administracji

```
grant select on FinancialReport to administration_office_employee
```

Dostęp do poszczególnych tabel (jeśli nie zostało to wcześniej uwzględnione)

```
grant select on Orders to director, administration_office_employee
grant select on OrderDetails to director, administration_office_employee
grant select, update on Access to tutor
grant select, update on Attendees to tutor
grant select on Clients to director, administration_office_employee, education_office_employee
grant select on CourseGrade to education_office_employee
grant select, insert, update on CourseGrade to tutor
grant select, insert on Courses to education_office_employee
grant select, update on Employees to administration_office_employee
```

```
grant select, insert, delete, update on Employees to director
grant select, insert, update on IndividualInternship to education_office_employee
grant select, insert, delete, update on Interpreters to director
grant select, update on Interpreters to administration_office_employee
grant select, update, insert on Meetings to tutor, director, interpreter, education_office_employee
grant select, update, insert on Practices to education_office_employee
grant select, update, select on Recordings to education_office_employee, tutor
grant select on Studies to public
grant select on Webinars to public
grant select on Courses to public
grant select on Terms to public
grant select on Subjects to public
grant select on SubjectGrades to tutor
grant select, insert, update on SubjectGrades to tutor
grant select, insert on SubjectsMeetings to tutor, education_office_employee
grant select, insert on SubjectsRecordings to tutor, education_office_employee
```