

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 sylabusu 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

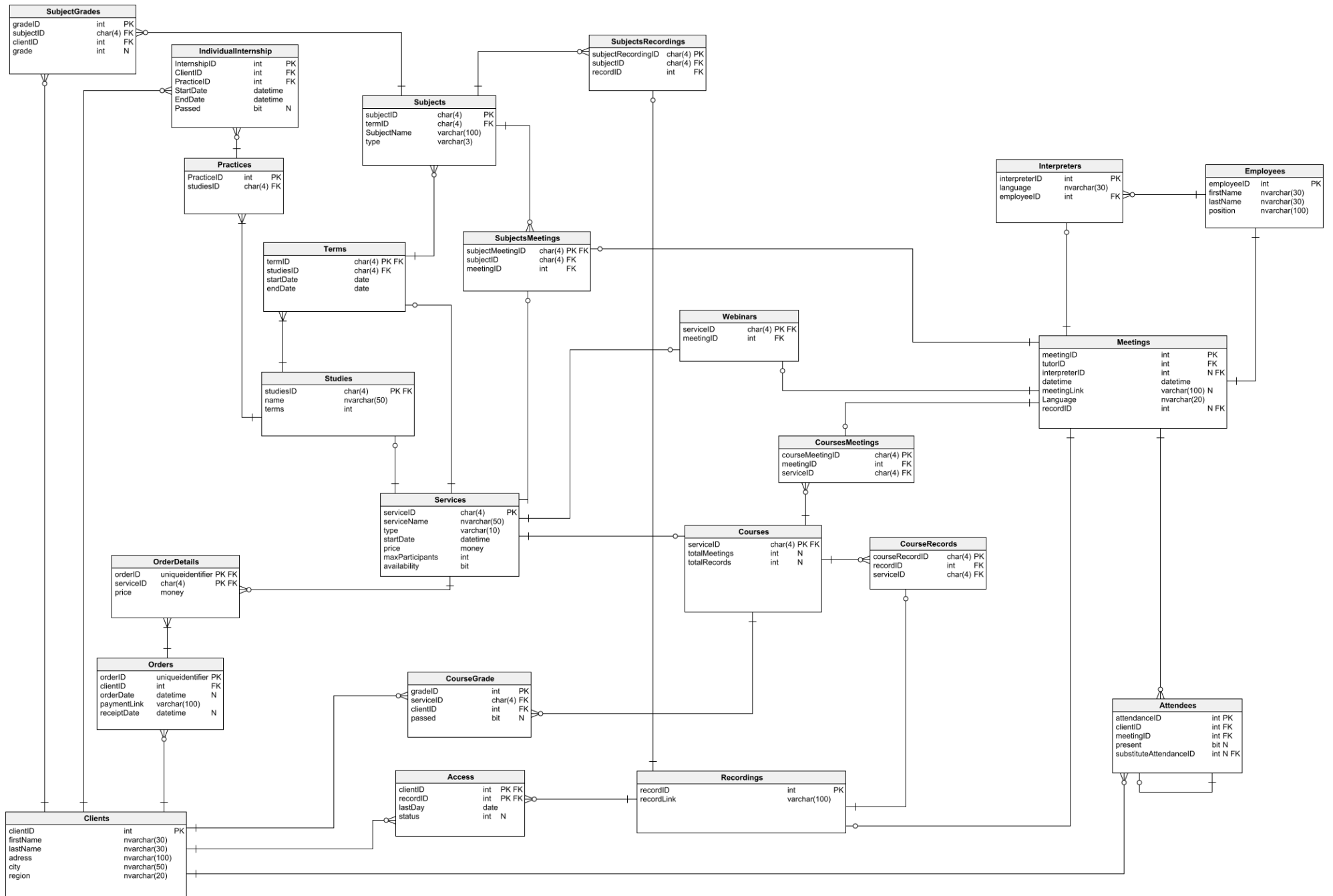
8. Każdy użytkownik

- 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. Zawiera identyfikator klienta (clientID), imię oraz nazwisko (firstName, lastName) oraz dane adresowe (address, city, region).

Klucz główny: clientID

Orders

Tabela przechowuje podstawowe dane o zamówieniu. 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. 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. Zawiera identyfikator usługi (serviceID), nazwę usługi (serviceName), typ usługi (type - webinar, kurs, studia, zjazd, pojedyncze spotkanie), datę rozpoczęcia (startDate) oraz cenę(price).

Klucz główny: serviceID

Studies

Tabela przechowuje podstawowe dane o studiach. 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. 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. 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. Zawiera identyfikator pracownika (employeeID), imię oraz nazwisko (firstName, lastName) oraz stanowisko (position).

Klucz główny: employeeID

Interpreters

Tabela przechowuje podstawowe dane o tłumaczu. 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. 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 danego semestru. 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 pojedynczych 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. 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. 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 spotkania 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**)

Implementacja bazy danych

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

```
-- Table: Access
CREATE TABLE Access (
  clientID int NOT NULL,
  recordID int NOT NULL,
  lastDay date NOT NULL,
  status int NULL,
  CONSTRAINT Access_pk PRIMARY KEY (clientID,recordID)
);

-- 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 Attendees (
  attendanceID int NOT NULL,
  clientID int NOT NULL,
  meetingID int NOT NULL,
  present int NULL,
  substituteAttendanceID int NULL,
  CONSTRAINT Attendees_pk PRIMARY KEY (attendanceID)
);

-- 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 Clients (
  clientID int NOT NULL IDENTITY(1, 1),
  firstName varchar(30) NOT NULL,
  lastName varchar(30) NOT NULL,
  adress varchar(100) NOT NULL,
  city varchar(50) NOT NULL,
  region varchar(20) NOT NULL,
  CONSTRAINT Clients_pk PRIMARY KEY (clientID)
);

-- Table: CourseGrade
CREATE TABLE CourseGrade (
  gradeID int NOT NULL IDENTITY(1, 1),
  serviceID char(4) NOT NULL,
  clientID int NOT NULL,
  passed varchar(8) NULL,
  CONSTRAINT CourseGrade_pk PRIMARY KEY (gradeID)
);

-- 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 CourseRecords (
  courseRecordID char(4) NOT NULL,
  recordID int NOT NULL,
  serviceID char(4) NOT NULL,
  CONSTRAINT CourseRecords_pk PRIMARY KEY (courseRecordID)
);

-- 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 Courses (

```

```

serviceID char(4) NOT NULL,
maxParticipants int NULL,
totalMeetings int NULL,
totalRecords int NULL,
CONSTRAINT Courses_pk PRIMARY KEY (serviceID)
);

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

-- Table: CoursesMeetings
CREATE TABLE CoursesMeetings (
    courseMeetingID char(4) NOT NULL,
    meetingID int NOT NULL,
    serviceID char(4) NOT NULL,
    CONSTRAINT CoursesMeetings_pk PRIMARY KEY (courseMeetingID)
);

-- 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 Employees (
    employeeID int NOT NULL IDENTITY(1, 1),
    firstName varchar(30) NOT NULL,
    lastName varchar(30) NOT NULL,
    position varchar(100) NOT NULL,
    CONSTRAINT Employees_pk PRIMARY KEY (employeeID)
);

-- Table: IndividualInternship
CREATE TABLE IndividualInternship (
    InternshipID int NOT NULL IDENTITY(1, 1),
    ClientID int NOT NULL,
    PracticeID int NOT NULL,
    StartDate datetime NOT NULL,
    EndDate datetime NOT NULL,
    Passed int NOT NULL,
    CONSTRAINT IndividualInternship_pk PRIMARY KEY (InternshipID)
);

```

```

-- 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 Interpreters (
    interpreterID int NOT NULL IDENTITY(1, 1),
    language varchar(30) NOT NULL,
    employeeID int NOT NULL,
    CONSTRAINT Interpreters_pk PRIMARY KEY (interpreterID)
);

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

-- Table: Meetings
CREATE TABLE 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 (meetingID)
);

-- 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 OrderDetails (
    orderID uniqueidentifier NOT NULL DEFAULT NEWID(),
    serviceID char(4) NOT NULL,
    price money NOT NULL,
    CONSTRAINT OrderDetails_pk PRIMARY KEY (orderID,serviceID)
);

-- 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 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 (orderID)
);

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

-- Table: Practices
CREATE TABLE Practices (
    PracticeID int NOT NULL IDENTITY(1, 1),
    termID char(4) NOT NULL,
    CONSTRAINT Practices_pk PRIMARY KEY (PracticeID)
);

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

-- Table: Recordings

```

```

CREATE TABLE Recordings (
    recordID int NOT NULL IDENTITY(1, 1),
    recordLink varchar(100) NOT NULL,
    CONSTRAINT Recordings_pk PRIMARY KEY (recordID)
);

<<<<<<< HEAD:dokumentacja.md
-- Table: Services
CREATE TABLE Services (
    serviceID char(4) NOT NULL,
    serviceName varchar(50) NOT NULL,
    type varchar(10) NOT NULL,
    startDate datetime NOT NULL,
    price money NOT NULL,
    CONSTRAINT Services_pk PRIMARY KEY (serviceID)
);

-- Table: Studies
CREATE TABLE Studies (
    studiesID char(4) NOT NULL,
    name varchar(50) NOT NULL,
    terms int NOT NULL,
    CONSTRAINT Studies_pk PRIMARY KEY (studiesID)
);

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

-- Table: SubjectGrades
CREATE TABLE SubjectGrades (
    gradeID int NOT NULL IDENTITY(1, 1),
    subjectID char(4) NOT NULL,
    clientID int NOT NULL,
    grade int NULL,
    CONSTRAINT SubjectGrades_pk PRIMARY KEY (gradeID)
);

-- 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 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 (subjectID)
);

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

-- Table: SubjectsMeetings
CREATE TABLE SubjectsMeetings (
    subjectMeetingID char(4) NOT NULL,
    subjectID char(4) NOT NULL,
    meetingID int NOT NULL,
    CONSTRAINT SubjectsMeetings_pk PRIMARY KEY (subjectMeetingID)
);

-- 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: Terms
CREATE TABLE Terms (
    termID char(4) NOT NULL,
    studiesID char(4) NOT NULL,
    startDate date NOT NULL,
    endDate date NOT NULL,
    CONSTRAINT Terms_pk PRIMARY KEY (termID)
);

-- 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 Webinars (
    serviceID char(4) NOT NULL,
    maxParticipants int NOT NULL,
    meetingID int NOT NULL,
    CONSTRAINT Webinars_pk PRIMARY KEY (serviceID)
);

-- 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.

```

SELECT 'cm' + CAST(CM.meetingID AS char(4)) AS eventID, S.serviceName as eventName, COUNT(A.attendanceID) AS enrolled, 'CMeeting' as type
FROM
    Attendees AS A INNER JOIN
        Meetings AS M ON M.meetingID = A.meetingID INNER JOIN
        CoursesMeetings AS CM ON CM.meetingID = M.meetingID INNER JOIN
        Courses AS C ON C.serviceID = CM.serviceID INNER JOIN
        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.

```
SELECT 'sm' + CAST(SM.meetingID AS char(4)) AS eventID, S.serviceName AS eventName, COUNT(A.attendanceID) AS enrolled, 'SMeeting' as type
FROM      Attendees AS A INNER JOIN
          Meetings AS M ON M.meetingID = A.meetingID INNER JOIN
          SubjectsMeetings AS SM ON SM.meetingID = M.meetingID INNER JOIN
          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.

```
SELECT S.serviceID as eventID, S.serviceName as eventName, COUNT(O.orderID) AS enrolled, S.type
FROM      Orders AS O INNER JOIN
          OrderDetails AS OD ON OD.orderID = O.orderID RIGHT OUTER JOIN
          Services AS S ON S.serviceID = OD.serviceID
WHERE S.startDate > GETDATE() AND S.type <> 'pSpotkanie'
GROUP BY S.serviceID, S.serviceName
```

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

```
CREATE VIEW ListOfDebtors AS

with paid 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),

termMeetings as
(select t.termID, m.meetingID
from Terms t
```

```

        join Subjects s on t.termID = s.termID
        join SubjectsMeetings m on s.subjectID = m.subjectID),

x as
(select a.clientID, a.firstName, a.lastName, b.meetingID
from payed a
        join termMeetings b on a.serviceID = b.termID),

recC as
(select a.clientID, p.firstName, p.lastName
from Access a
        join CourseRecords r on a.recordID = r.recordID
        left join payed p on a.clientID = p.clientID and r.serviceID = p.serviceID
where p.clientID is null and p.serviceID is null),

recS as
(select a.clientID, p.firstName, p.lastName
from Access a
        join SubjectsRecordings r on a.recordID = r.recordID
        join Subjects s on r.subjectID = s.subjectID
        left join payed p on a.clientID = p.clientID and s.termID = p.serviceID
where p.clientID is null and p.serviceID is null),

meetC as
(select a.clientID, p.firstName, p.lastName
from Attendees a
        join CoursesMeetings c on a.meetingID = c.meetingID
        left join payed p on a.clientID = p.clientID and c.serviceID = p.serviceID
where p.clientID is null and p.serviceID is null),

meetW as
(select a.clientID, p.firstName, p.lastName
from Attendees a
        join Webinars w on a.meetingID = w.meetingID
        left join payed p on a.clientID = p.clientID and w.serviceID = p.serviceID
where p.clientID is null and p.serviceID is null),

meets as
(select a.clientID, x.firstName, x.lastName
from Attendees a
        join SubjectsMeetings s on a.meetingID = s.meetingID
        left join x on a.clientID = x.clientID and s.meetingID = x.meetingID
where x.clientID is null and x.meetingID is null)

select * from recC
union
select * from recS
union
select * from meetC

```

```
union
select * from meetW
union
select * from meetS
```

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 MeetingsDuringTerm(@serviceID) WHERE (SELECT type FROM Services WHERE serviceID = @serviceID) = 'term'
```

```

        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
);
```

Procedurey

Przykładowa implementacja procedury dodającej nowego 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
END;

```

```

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 createRecording
    @MeetingID INTEGER
as
begin
    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;
end;

```

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;
        PRINT 'Meeting date updated successfully.';
    END
    ELSE
    BEGIN
        PRINT 'Meeting with the specified meetingID does not exist.';
    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
        PRINT 'Service with the specified serviceID does not exist.';
    END
END;

```

```
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 aktywujący dostępność usługi jeżeli zostały wprowadzone wszystkie zaplanowane spotkania.

```

CREATE TRIGGER UpdateAvailability
ON CoursesMeetings
AFTER INSERT
AS
BEGIN
    DECLARE @serviceID VARCHAR(4);

    SELECT TOP 1 @serviceID = serviceID
    FROM inserted;

    IF (
        (SELECT COUNT(*) FROM CoursesMeetings WHERE serviceID = @serviceID) =
        (SELECT totalMeetings FROM Courses WHERE serviceID = @serviceID)
    )
    BEGIN
        UPDATE Services
        SET availability = 1
        WHERE serviceID = @serviceID;
    END;
END;

```

Trigger wstawiający nowe spotkania jeżeli nie przekraczają limitu spotkań.

```

CREATE TRIGGER PreventExceedingTotalMeetings
ON CoursesMeetings
INSTEAD OF INSERT
AS
BEGIN

```

```

DECLARE @TotalMeetings INT;

SELECT @TotalMeetings = c.totalMeetings
FROM inserted i
JOIN Courses c ON i.serviceID = c.serviceID;

DECLARE @serviceID VARCHAR(4);

SELECT TOP 1 @serviceID = serviceID
FROM inserted;

IF ((SELECT COUNT(*) FROM CoursesMeetings WHERE (serviceID = @serviceID))
+ (SELECT COUNT(*) FROM inserted i WHERE (serviceID = @serviceID)) <= @TotalMeetings)
BEGIN
    INSERT INTO CoursesMeetings (courseMeetingID, meetingID, serviceID)
    SELECT courseMeetingID, meetingID, serviceID
    FROM inserted;
END
ELSE
BEGIN
    PRINT 'Exceeded the TotalMeetings limit in the Courses table.';
END
END

```