Podstawy Baz Danych

Zespół nr 3 (śr. 16:45): Maciej Wiśniewski, Konrad Szymański, Kajetan Frątczak

Funkcje Systemu

- zlicza frekwencję użytkowników
- zalicza moduł prowadzony online asynchronicznie
- udostępnia nagranie użytkownikowi
    - po ukończeniu webinaru i kursu online synchronicznego udostępnia użytkownikowi nagranie na okres 30 dni
- generuje raporty i listy na żądanie uprawnionych użytkowników:
    - generowanie raportu finansowego
    - generowanie listy osób zalegających z opłatami
    - generowanie raportu o liczbie osób zapisanych na przyszłe wydarzenia
    - generowanie raportu o frekwencji
    - generowanie listy obecności
    - generowanie raportu o kolizjach użytkowników - informuje użytkownika zapisanego na dwa wydarzenia odbywające się w tym samym czasie
- przechowuje produkty w koszyku
- zapisuje wybrane produkty przez użytkownika i pozwala na płatność za wszystkie naraz
- weryfikuje terminowe dopłaty
- blokuje dostęp do kursu/studiów, jeśli użytkownik nie zapłaci 3 dni przed ich rozpoczęciem
- pilnuje limitów osób na kursach oraz studiach
- uniemożliwia dodanie użytkownika przez koordynatora przedmiotu, gdy nie ma już miejsc
- umożliwia zakładanie/edycję konta
- przechowuje dane kontaktowe
- przetrzymuje dane kursów, studiów, webinarów i użytkowników

Użytkownicy i funkcje jakie mogą realizować

Użytkownicy systemu:
- Uczestnik kursu (student, uczestnik kursu, uczestnik webinaru, uczestnik pojedynczego spotkania studyjnego)
- Administrator
- Dyrektor Szkoły
- Koordynator (studiów, przedmiotu, kursu, webinaru)
- Wykładowca
- Prowadzący praktyki
- Księgowy
- Tłumacz
- Osoba niewidniejąca w bazie (bez konta)

☐ Uczestnik kursu
- Zakłada konto
- Edytuje (dane niewrażliwe, np. adres korespondencyjny)/usuwa konto
- Zapisuje się na bezpłatne webinary
- Zapisuje się na płatne webinary/studia/kursy/spotkania studyjne
- Dodaje/usuwa zajęcia do/z koszyka
- Sprawdza swój harmonogram zajęć
- Sprawdza swoje oceny
- Generuje raport swojej obecności na zajęciach
- Ma dostęp do materiałów z zajęć prowadzonych online (na okres 30 dni)
- Może zrezygnować ze studiów

☐ Administrator
- Modyfikuje dane użytkowników
- Modyfikuje dane kursów, studiów i webinarów (dodawanie, usuwanie, zmiany)
- Zmienia role użytkowników systemu
- Sprawdza harmonogram, ogólnie i poszczególnych użytkowników
- Wprowadza zmiany do harmonogramu

☐ Dyrektor Szkoły
- Sprawdza/edytuje harmonogram
- Generuje raporty finansowe
- Generuje, przegląda, edytuje listy użytkowników
- Sprawdza frekwencję
- Generuje raporty o frekwencji
- Zarządza zaległymi płatnościami
- Ma możliwość odroczenia płatności
- Akceptuje listę uczestników płatnego kurs/webinaru/studiów

☐ Koordynator
- ❖ Studiów
  - Tworzy sylabus/program studiów
  - Wprowadza zmiany do harmonogramu
  - Może dodać/usunąć dodatkowych użytkowników do studiów
- ❖ Przedmiotu
  - Zalicza przedmioty studentom/wpisuje im oceny
  - Decyduje w sprawach odnośnie odrabiania
  - Tworzy spotkania studyjne
  - Generuje raport o liczbie osób zapisanych
  - Wybiera wykładowców i tłumaczy
- ❖ Kursu
  - Tworzy kurs
  - Generuje raport o liczbie osób zapisanych
  - Modyfikuje dane kursu
  - Wpisuje oceny z kursu

☐ Wykładowca
- Wpisuje obecności na prowadzonych zajęciach
- Generuje listy obecności i je modyfikuje
- Ma dostęp do harmonogramu prowadzonych przez niego zajęć
- Tworzy webinary i wybiera ich typ(płatny/darmowy)
- Wysyła prośby o zmianę harmonogramu (np. z powodu zdarzeń losowych)
- Wpisuje oceny z przedmiotu/kursu

☐ Prowadzący praktyki
- Zalicza praktyki studentom
- Wpisuje obecności na praktykach
- Ma dostęp do harmonogramu prowadzonych przez niego zajęć

☐ Księgowy
- Generuje raporty finansowe
- Zwraca nadpłaty
- Zbiera informacje o ilości zapisanych osób na przyszłe wydarzenia

☐ Tłumacz
- Ma dostęp do harmonogramu zajęć nie prowadzonych po polsku
- Wysyła prośby o zmianę harmonogramu (np. z powodu zdarzeń losowych)
- Przegląda zajęcia, które nie są prowadzone po polsku
- Tłumaczy zajęcia na żywo

☐ Osoba niewidniejąca w bazie
- Zakłada konto, dane zapisywane są do bazy
- Przegląda dostępną ofertę
- Ma dostęp do danych kontaktowych

Historie użytkownika dla uczestnika kursu

- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość zapisać się na zajęcia.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość dostępu do materiału z poprzednich zajęć.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość sprawdzenia swoich ocen.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość sprawdzenia harmonogramu swoich zajęć.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość generowania raportu obecności.
- Jako uczestnik webinaru/kursu/studiów chciałbym mieć możliwość rezygnacji z webinaru/kursu/studiów.

Historie użytkownika dla administratora

- Jako administrator chciałbym mieć możliwość modyfikacji danych użytkowników.
- Jako administrator chciałbym mieć możliwość dodawać/usuwać użytkowników.
- Jako administrator chciałbym mieć możliwość zmiany ról użytkowników w systemie.
- Jako administrator chciałbym mieć możliwość wprowadzenia zmian w harmonogramie.
- Jako administrator chciałbym mieć możliwość sprawdzenia harmonogramu dla poszczególnych użytkowników jak i dla ogółu.

Historie użytkownika dla Dyrektora Szkoły

- Jako Dyrektor Szkoły chciałbym mieć możliwość generowania raportów finansowych.
- Jako Dyrektor Szkoły chciałbym mieć możliwość edytowania harmonogramu zajęć.
- Jako Dyrektor Szkoły chciałbym mieć możliwość zarządzania zaległymi płatnościami (odroczenia terminu, wstrzymania blokady dostępu).
- Jako Dyrektor Szkoły chciałbym mieć możliwość przeglądania i edytowania listy użytkowników systemu.
- Jako Dyrektor Szkoły chciałbym mieć możliwość sprawdzenia raportów o frekwencji użytkowników.

Historie użytkownika dla koordynatora (studiów, przedmiotu, kursu)

- Jako koordynator chciałbym mieć możliwość tworzenia programu kursu/przedmiotu/studiów.
- Jako koordynator chciałbym mieć możliwość uprawnienia administratora dla osób pod moją koordynacją.
- Jako koordynator chciałbym mieć możliwość przypisania osób prowadzących zajęcia.
- Jako koordynator chciałbym mieć możliwość przyznania stypendium.
- Jako koordynator chciałbym mieć możliwość generować raport zapisanych osób.
- Jako koordynator chciałbym mieć możliwość wpisania ocen.

Historie użytkownika dla wykładowcy

- Jako wykładowca chciałbym mieć możliwość modyfikowania obecności na prowadzonych zajęciach.
- Jako wykładowca chciałbym mieć możliwość dostępu do harmonogramu prowadzonych zajęć.
- Jako wykładowca chciałbym mieć możliwość tworzenia webinarów i wybór jego typu.
- Jako wykładowca chciałbym mieć możliwość wysłania prośby o zmianę harmonogramu.

Historie użytkownika dla prowadzącego praktyki

- Jako prowadzący praktyki chciałbym mieć możliwość wpisywania obecności studentów na zajęciach praktycznych.
- Jako prowadzący praktyki chciałbym mieć możliwość zaliczania praktyk studentom.
- Jako prowadzący praktyki chciałbym mieć możliwość sprawdzenia harmonogramu swoich zajęć.

Historie użytkownika dla księgowego

- Jako księgowy chciałbym mieć możliwość generowania raportów finansowych.
- Jako księgowy chciałbym mieć możliwość sprawdzenia kto, a kto nie opłacił.
- Jako księgowy chciałbym mieć możliwość zwrotu nadpłat.

Historie użytkownika dla tłumacza

- Jako tłumacz chciałbym mieć możliwość przeglądania harmonogramu zajęć, które nie są prowadzone po polsku.
- Jako tłumacz chciałbym mieć możliwość wysłania prośby o zmianę harmonogramu zajęć, które mam tłumaczyć.
- Jako tłumacz chciałbym mieć możliwość otrzymywania materiałów do przygotowania się przed zajęciami.
- Jako tłumacz chciałbym mieć możliwość tłumaczenia zajęć na żywo dla uczestników.

Historie użytkownika dla osoby niewidniejącej w bazie

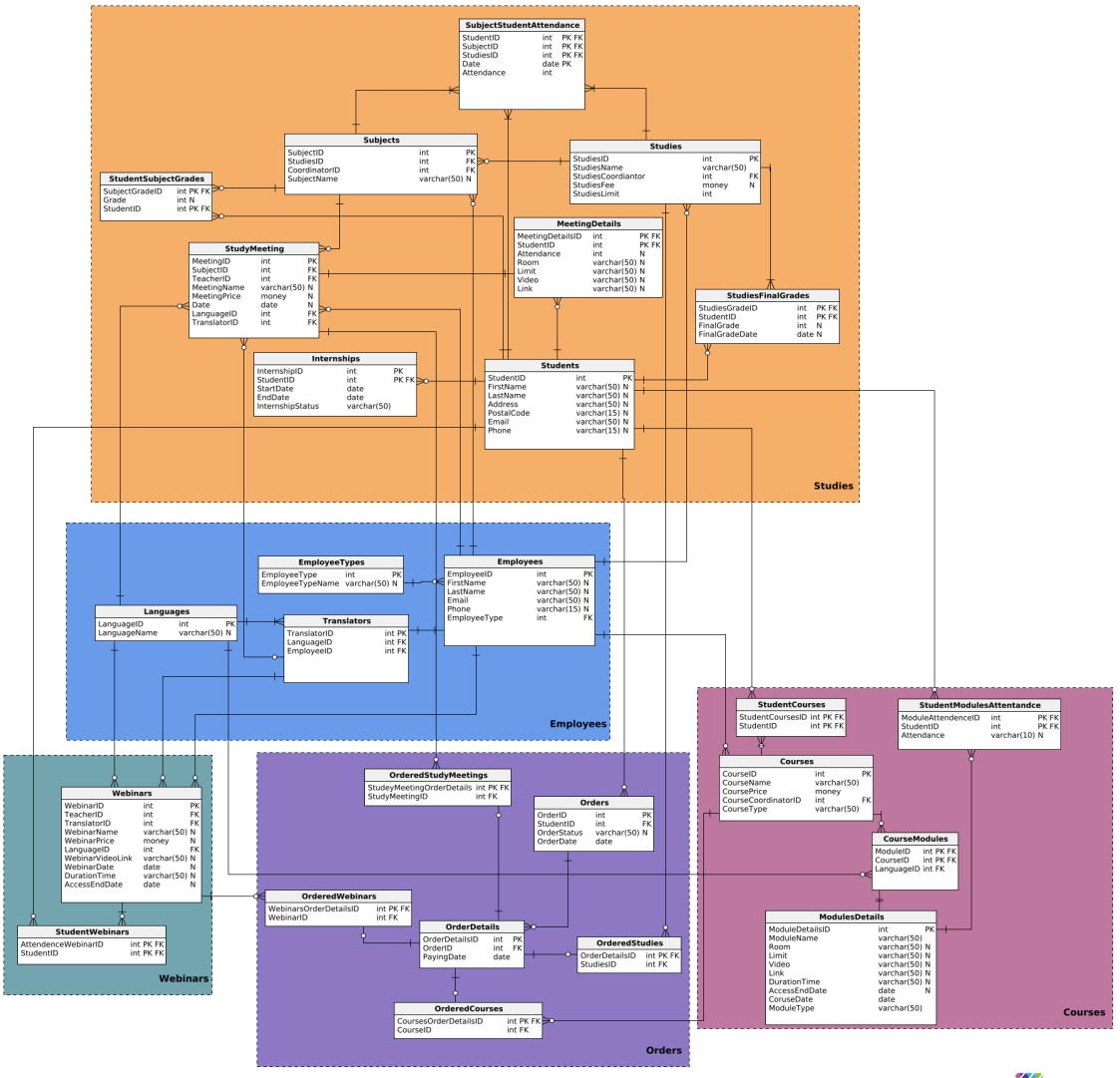
- Jako osoba niewidniejąca w bazie chciałbym mieć możliwość przeglądania dostępnej oferty kursów, webinarów i studiów.
- Jako osoba niewidniejąca w bazie chciałbym mieć możliwość przeglądania danych kontaktowych do obsługi systemu.
- Jako osoba niewidniejąca w bazie chciałbym mieć możliwość założenia konta w systemie, aby zapisać się na kurs/webinar.

Przykładowy przypadek użycia

Opłata za zajęcia.

1. Cel: Opłata za zajęcia.
2. Aktorzy systemu
    - uczestnik kursu
    - student
    - członek webinaru
3. Scenariusz główny
    - zalogowanie się na konto
    - wybór zajęć, za które chce zapłacić
    - system wpisuje użytkownika na listę
    - przejście do systemu opłat
    - opłacanie zajęć i wyświetlenie potwierdzenia
    - Dyrektor zatwierdza dołączenie użytkownika na zajęcia
4. Scenariusze alternatywne
    - odrzucenie płatności, wyświetlenie informacji o niepowodzeniu płatności
    -

Diagram bazy danych



Vertabelo

Kod DDL

```
-- tables
-- Table: CourseModules
CREATE TABLE dbo.CourseModules (
    ModuleID int  NOT NULL,
    CourseID int  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT PK_CourseModules PRIMARY KEY CLUSTERED
(ModuleID,CourseID)
)
ON PRIMARY;

-- Table: Courses
CREATE TABLE dbo.Courses (
    CourseID int  NOT NULL,
    CourseName varchar(50)  NOT NULL,
    CoursePrice money  NOT NULL,
    CourseCoordinatorID int  NOT NULL,
    CourseType varchar(50)  NOT NULL,
    CONSTRAINT PK_Courses PRIMARY KEY CLUSTERED (CourseID)
)
ON PRIMARY;

-- Table: EmployeeTypes
CREATE TABLE dbo.EmployeeTypes (
    EmployeeType int  NOT NULL,
    EmployeeTypeName varchar(50)  NULL,
    CONSTRAINT PK_EmployeeTypes PRIMARY KEY CLUSTERED
(EmployeeType)
)
ON PRIMARY;

-- Table: Employees
CREATE TABLE dbo.Employees (
    EmployeeID int  NOT NULL,
    FirstName varchar(50)  NULL,
    LastName varchar(50)  NULL,
    Email varchar(50)  NULL,
    Phone varchar(15)  NULL,
    EmployeeType int  NOT NULL,
    CONSTRAINT PK_Employees PRIMARY KEY CLUSTERED (EmployeeID)
)
ON PRIMARY;
```

```
-- Table: Internships
CREATE TABLE Internships (
    InternshipID int  NOT NULL,
    StudentID int  NOT NULL,
    StartDate date  NOT NULL,
    EndDate date  NOT NULL,
    InternshipStatus varchar(50)  NOT NULL,
    CONSTRAINT Internships_pk PRIMARY KEY
(InternshipID,StudentID)
);

-- Table: Languages
CREATE TABLE dbo.Languages (
    LanguageID int  NOT NULL,
    LanguageName varchar(50)  NULL,
    CONSTRAINT PK_Languages PRIMARY KEY CLUSTERED (LanguageID)
)
ON PRIMARY;

-- Table: MeetingDetails
CREATE TABLE dbo.MeetingDetails (
    MeetingDetailsID int  NOT NULL,
    StudentID int  NOT NULL,
    Attendance int  NULL,
    Room varchar(50)  NULL,
    Limit varchar(50)  NULL,
    Video varchar(50)  NULL,
    Link varchar(50)  NULL,
    CONSTRAINT PK_MeetingDetails PRIMARY KEY CLUSTERED
(MeetingDetailsID,StudentID)
)
ON PRIMARY;

-- Table: ModulesDetails
CREATE TABLE dbo.ModulesDetails (
    ModuleDetailsID int  NOT NULL,
    ModuleName varchar(50)  NOT NULL,
    Room varchar(50)  NULL,
    Limit varchar(50)  NULL,
    Video varchar(50)  NULL,
    Link varchar(50)  NULL,
    DurationTime varchar(50)  NULL,
    AccessEndDate date  NULL,
    CoruseDate date  NOT NULL,
    ModuleType varchar(50)  NOT NULL,
    CONSTRAINT PK_ModulesTypes PRIMARY KEY CLUSTERED
(ModuleDetailsID)
```

```
)
ON PRIMARY;
-- Table: OrderDetails
CREATE TABLE dbo.OrderDetails (
    OrderDetailsID int  NOT NULL,
    OrderID int  NOT NULL,
    PayingDate date  NOT NULL,
    CONSTRAINT PK_OrderDetails PRIMARY KEY CLUSTERED
(OrderDetailsID)
)
ON PRIMARY;

-- Table: OrderedCourses
CREATE TABLE dbo.OrderedCourses (
    CoursesOrderDetailsID int  NOT NULL,
    CourseID int  NOT NULL,
    CONSTRAINT PK_OrderedCourses PRIMARY KEY CLUSTERED
(CoursesOrderDetailsID)
)
ON PRIMARY;

-- Table: OrderedStudies
CREATE TABLE dbo.OrderedStudies (
    OrderDetailsID int  NOT NULL,
    StudiesID int  NOT NULL,
    CONSTRAINT PK_OrderedStudies PRIMARY KEY CLUSTERED
(OrderDetailsID)
)
ON PRIMARY;

-- Table: OrderedStudyMeetings
CREATE TABLE dbo.OrderedStudyMeetings (
    StudeyMeetingOrderDetailsID int  NOT NULL,
    StudyMeetingID int  NOT NULL,
    CONSTRAINT PK_OrderedStudyMeetings PRIMARY KEY CLUSTERED
(StudeyMeetingOrderDetailsID)
)
ON PRIMARY;

-- Table: OrderedWebinars
CREATE TABLE dbo.OrderedWebinars (
    WebinarsOrderDetailsID int  NOT NULL,
    WebinarID int  NOT NULL,
    CONSTRAINT PK_OrderedWebinars PRIMARY KEY CLUSTERED
(WebinarsOrderDetailsID)
)
ON PRIMARY;
```

```
-- Table: Orders
CREATE TABLE dbo.Orders (
    OrderID int  NOT NULL,
    StudentID int  NOT NULL,
    OrderStatus varchar(50)  NULL,
    OrderDate date  NOT NULL,
    CONSTRAINT PK_Orders PRIMARY KEY CLUSTERED (OrderID)
)
ON PRIMARY;

-- Table: StudentCourses
CREATE TABLE dbo.StudentCourses (
    StudentCoursesID int  NOT NULL,
    StudentID int  NOT NULL,
    CONSTRAINT PK_CourseDetails PRIMARY KEY CLUSTERED
(StudentCoursesID,StudentID)
)
ON PRIMARY;

-- Table: StudentModulesAttentandce
CREATE TABLE dbo.StudentModulesAttentandce (
    ModuleAttendenceID int  NOT NULL,
    StudentID int  NOT NULL,
    Attendance varchar(10)  NULL,
    CONSTRAINT PK_ModulesDetails PRIMARY KEY CLUSTERED
(ModuleAttendenceID,StudentID)
)
ON PRIMARY;

-- Table: StudentSubjectGrades
CREATE TABLE dbo.StudentSubjectGrades (
    SubjectGradeID int  NOT NULL,
    Grade int  NULL,
    StudentID int  NOT NULL,
    CONSTRAINT PK_SubjectDetails PRIMARY KEY CLUSTERED
(SubjectGradeID,StudentID)
)
ON PRIMARY;

-- Table: StudentWebinars
CREATE TABLE dbo.StudentWebinars (
    AttendenceWebinarID int  NOT NULL,
    StudentID int  NOT NULL,
    CONSTRAINT PK_WebinarList PRIMARY KEY CLUSTERED
(AttendenceWebinarID,StudentID)
)
```

```
ON PRIMARY;

-- Table: Students
CREATE TABLE dbo.Students (
    StudentID int  NOT NULL,
    FirstName varchar(50)  NULL,
    LastName varchar(50)  NULL,
    Address varchar(50)  NULL,
    PostalCode varchar(15)  NULL,
    Email varchar(50)  NULL,
    Phone varchar(15)  NULL,
    CONSTRAINT PK_Students PRIMARY KEY CLUSTERED (StudentID)
)
ON PRIMARY;

-- Table: Studies
CREATE TABLE dbo.Studies (
    StudiesID int  NOT NULL,
    StudiesName varchar(50)  NOT NULL,
    StudiesCoordiantor int  NOT NULL,
    StudiesFee money  NULL,
    StudiesLimit int  NOT NULL,
    CONSTRAINT PK_Studies PRIMARY KEY CLUSTERED (StudiesID)
)
ON PRIMARY;

-- Table: StudiesFinalGrades
CREATE TABLE dbo.StudiesFinalGrades (
    StudiesGradeID int  NOT NULL,
    StudentID int  NOT NULL,
    FinalGrade int  NULL,
    FinalGradeDate date  NULL,
    CONSTRAINT PK_StudiesDetails PRIMARY KEY CLUSTERED
(StudiesGradeID,StudentID)
)
ON PRIMARY;

-- Table: StudyMeeting
CREATE TABLE dbo.StudyMeeting (
    MeetingID int  NOT NULL,
    SubjectID int  NOT NULL,
    TeacherID int  NOT NULL,
    MeetingName varchar(50)  NULL,
    MeetingPrice money  NULL,
    Date date  NULL,
    LanguageID int  NOT NULL,
    TranslatorID int  NOT NULL,
    CONSTRAINT PK_StudyMeeting PRIMARY KEY CLUSTERED (MeetingID)
```

```
)
ON PRIMARY;
-- Table: SubjectStudentAttendance
CREATE TABLE SubjectStudentAttendance (
    StudentID int  NOT NULL,
    SubjectID int  NOT NULL,
    StudiesID int  NOT NULL,
    Date date  NOT NULL,
    Attendance int  NOT NULL,
    CONSTRAINT SubjectStudentAttendance_pk PRIMARY KEY
(StudentID,SubjectID,Date,StudiesID)
);

-- Table: Subjects
CREATE TABLE dbo.Subjects (
    SubjectID int  NOT NULL,
    StudiesID int  NOT NULL,
    CoordinatorID int  NOT NULL,
    SubjectName varchar(50)  NULL,
    CONSTRAINT PK_Subjects PRIMARY KEY CLUSTERED (SubjectID)
)
ON PRIMARY;

-- Table: Translators
CREATE TABLE dbo.Translators (
    TranslatorID int  NOT NULL,
    LanguageID int  NOT NULL,
    EmployeeID int  NOT NULL,
    CONSTRAINT PK_Tranlators PRIMARY KEY CLUSTERED (TranslatorID)
)
ON PRIMARY;

-- Table: Webinars
CREATE TABLE dbo.Webinars (
    WebinarID int  NOT NULL,
    TeacherID int  NOT NULL,
    TranslatorID int  NOT NULL,
    WebinarName varchar(50)  NULL,
    WebinarPrice money  NULL,
    LanguageID int  NOT NULL,
    WebinarVideoLink varchar(50)  NULL,
    WebinarDate date  NULL,
    DurationTime varchar(50)  NULL,
    AccessEndDate date  NULL,
    CONSTRAINT PK_Webinars PRIMARY KEY CLUSTERED (WebinarID)
)
ON PRIMARY;
```

```
-- foreign keys
-- Reference: Employees_Tranlators (table: Translators)
ALTER TABLE dbo.Translators ADD CONSTRAINT Employees_Tranlators
    FOREIGN KEY (EmployeeID)
    REFERENCES dbo.Employees (EmployeeID);

-- Reference: FK_CourseDetails_Courses (table: StudentCourses)
ALTER TABLE dbo.StudentCourses ADD CONSTRAINT
FK_CourseDetails_Courses
    FOREIGN KEY (StudentCoursesID)
    REFERENCES dbo.Courses (CourseID);

-- Reference: FK_CourseDetails_Students (table: StudentCourses)
ALTER TABLE dbo.StudentCourses ADD CONSTRAINT
FK_CourseDetails_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

-- Reference: FK_CourseModules_Courses (table: CourseModules)
ALTER TABLE dbo.CourseModules ADD CONSTRAINT
FK_CourseModules_Courses
    FOREIGN KEY (CourseID)
    REFERENCES dbo.Courses (CourseID);

-- Reference: FK_CourseModules_Languages (table: CourseModules)
ALTER TABLE dbo.CourseModules ADD CONSTRAINT
FK_CourseModules_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);

-- Reference: FK_CourseModules_ModulesTypes (table: CourseModules)
ALTER TABLE dbo.CourseModules ADD CONSTRAINT
FK_CourseModules_ModulesTypes
    FOREIGN KEY (ModuleID)
    REFERENCES dbo.ModulesDetails (ModuleDetailsID);

-- Reference: FK_Courses_Employees (table: Courses)
ALTER TABLE dbo.Courses ADD CONSTRAINT FK_Courses_Employees
    FOREIGN KEY (CourseCoordinatorID)
    REFERENCES dbo.Employees (EmployeeID);

-- Reference: FK_Employees_EmployeeTypes (table: Employees)
ALTER TABLE dbo.Employees ADD CONSTRAINT
FK_Employees_EmployeeTypes
    FOREIGN KEY (EmployeeType)
    REFERENCES dbo.EmployeeTypes (EmployeeType);
```

```
-- Reference: FK_OrderDetails_OrderedCourses (table:
OrderedCourses)
ALTER TABLE dbo.OrderedCourses ADD CONSTRAINT
FK_OrderDetails_OrderedCourses
    FOREIGN KEY (CoursesOrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

-- Reference: FK_OrderDetails_OrderedStudies (table:
OrderedStudies)
ALTER TABLE dbo.OrderedStudies ADD CONSTRAINT
FK_OrderDetails_OrderedStudies
    FOREIGN KEY (OrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

-- Reference: FK_OrderDetails_OrderedStudyMeetings (table:
OrderedStudyMeetings)
ALTER TABLE dbo.OrderedStudyMeetings ADD CONSTRAINT
FK_OrderDetails_OrderedStudyMeetings
    FOREIGN KEY (StudeyMeetingOrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

-- Reference: FK_OrderedCourses_Courses (table: OrderedCourses)
ALTER TABLE dbo.OrderedCourses ADD CONSTRAINT
FK_OrderedCourses_Courses
    FOREIGN KEY (CourseID)
    REFERENCES dbo.Courses (CourseID);

-- Reference: FK_OrderedWebinars_OrderDetails (table:
OrderedWebinars)
ALTER TABLE dbo.OrderedWebinars ADD CONSTRAINT
FK_OrderedWebinars_OrderDetails
    FOREIGN KEY (WebinarsOrderDetailsID)
    REFERENCES dbo.OrderDetails (OrderDetailsID);

-- Reference: FK_Orders_Students (table: Orders)
ALTER TABLE dbo.Orders ADD CONSTRAINT FK_Orders_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

-- Reference: FK_StudiesDetails_Students (table:
StudiesFinalGrades)
ALTER TABLE dbo.StudiesFinalGrades ADD CONSTRAINT
FK_StudiesDetails_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

-- Reference: FK_Studies_Employees (table: Studies)
```

```
ALTER TABLE dbo.Studies ADD CONSTRAINT FK_Studies_Employees
    FOREIGN KEY (StudiesCoordiantor)
    REFERENCES dbo.Employees (EmployeeID);


-- Reference: FK_Studies_StudiesDetails (table:
StudiesFinalGrades)
ALTER TABLE dbo.StudiesFinalGrades ADD CONSTRAINT
FK_Studies_StudiesDetails
    FOREIGN KEY (StudiesGradeID)
    REFERENCES dbo.Studies (StudiesID);

-- Reference: FK_StudyMeeting_Employees (table: StudyMeeting)
ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT
FK_StudyMeeting_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES dbo.Employees (EmployeeID);

-- Reference: FK_StudyMeeting_Languages (table: StudyMeeting)
ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT
FK_StudyMeeting_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);

-- Reference: FK_StudyMeeting_MeetingDetails (table:
MeetingDetails)
ALTER TABLE dbo.MeetingDetails ADD CONSTRAINT
FK_StudyMeeting_MeetingDetails
    FOREIGN KEY (MeetingDetailsID)
    REFERENCES dbo.StudyMeeting (MeetingID);

-- Reference: FK_StudyMeeting_Subjects (table: StudyMeeting)
ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT
FK_StudyMeeting_Subjects
    FOREIGN KEY (SubjectID)
    REFERENCES dbo.Subjects (SubjectID);

-- Reference: FK_StudyMeeting_Tranlators (table: StudyMeeting)
ALTER TABLE dbo.StudyMeeting ADD CONSTRAINT
FK_StudyMeeting_Tranlators
    FOREIGN KEY (TranslatorID)
    REFERENCES dbo.Translators (TranslatorID);

-- Reference: FK_Subjects_Employees (table: Subjects)
ALTER TABLE dbo.Subjects ADD CONSTRAINT FK_Subjects_Employees
    FOREIGN KEY (CoordinatorID)
    REFERENCES dbo.Employees (EmployeeID);
```

```
-- Reference: FK_Subjects_Studies (table: Subjects)
ALTER TABLE dbo.Subjects ADD CONSTRAINT FK_Subjects_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES dbo.Studies (StudiesID);




-- Reference: FK_Tranlators_Languages (table: Translators)
ALTER TABLE dbo.Translators ADD CONSTRAINT FK_Tranlators_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);

-- Reference: FK_WebinarList_Students (table: StudentWebinars)
ALTER TABLE dbo.StudentWebinars ADD CONSTRAINT
FK_WebinarList_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

-- Reference: FK_Webinars_Employees (table: Webinars)
ALTER TABLE dbo.Webinars ADD CONSTRAINT FK_Webinars_Employees
    FOREIGN KEY (TeacherID)
    REFERENCES dbo.Employees (EmployeeID);

-- Reference: FK_Webinars_Languages (table: Webinars)
ALTER TABLE dbo.Webinars ADD CONSTRAINT FK_Webinars_Languages
    FOREIGN KEY (LanguageID)
    REFERENCES dbo.Languages (LanguageID);

-- Reference: FK_Webinars_Tranlators (table: Webinars)
ALTER TABLE dbo.Webinars ADD CONSTRAINT FK_Webinars_Tranlators
    FOREIGN KEY (TranslatorID)
    REFERENCES dbo.Translators (TranslatorID);

-- Reference: FK_Webinars_WebinarList (table: StudentWebinars)
ALTER TABLE dbo.StudentWebinars ADD CONSTRAINT
FK_Webinars_WebinarList
    FOREIGN KEY (AttendenceWebinarID)
    REFERENCES dbo.Webinars (WebinarID);

-- Reference: ModulesDetails_StudentModulesAttentandce (table:
StudentModulesAttentandce)
ALTER TABLE dbo.StudentModulesAttentandce ADD CONSTRAINT
ModulesDetails_StudentModulesAttentandce
    FOREIGN KEY (ModuleAttendenceID)
    REFERENCES dbo.ModulesDetails (ModuleDetailsID);

-- Reference: OrderDetails_Orders (table: OrderDetails)
```

```
ALTER TABLE dbo.OrderDetails ADD CONSTRAINT OrderDetails_Orders
    FOREIGN KEY (OrderID)
    REFERENCES dbo.Orders (OrderID);

-- Reference: OrderedStudies_Studies (table: OrderedStudies)
ALTER TABLE dbo.OrderedStudies ADD CONSTRAINT
OrderedStudies_Studies
    FOREIGN KEY (StudiesID)
    REFERENCES dbo.Studies (StudiesID);
-- Reference: OrderedStudyMeetings_StudyMeeting (table:
OrderedStudyMeetings)
ALTER TABLE dbo.OrderedStudyMeetings ADD CONSTRAINT
OrderedStudyMeetings_StudyMeeting
    FOREIGN KEY (StudyMeetingID)
    REFERENCES dbo.StudyMeeting (MeetingID);

-- Reference: OrderedWebinars_Webinars (table: OrderedWebinars)
ALTER TABLE dbo.OrderedWebinars ADD CONSTRAINT
OrderedWebinars_Webinars
    FOREIGN KEY (WebinarID)
    REFERENCES dbo.Webinars (WebinarID);

-- Reference: StudentModulesAttentandce_Students (table:
StudentModulesAttentandce)
ALTER TABLE dbo.StudentModulesAttentandce ADD CONSTRAINT
StudentModulesAttentandce_Students
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

-- Reference: StudentSubjectGrades_Subjects (table:
StudentSubjectGrades)
ALTER TABLE dbo.StudentSubjectGrades ADD CONSTRAINT
StudentSubjectGrades_Subjects
    FOREIGN KEY (SubjectGradeID)
    REFERENCES dbo.Subjects (SubjectID);

-- Reference: Students_Internships (table: Internships)
ALTER TABLE Internships ADD CONSTRAINT Students_Internships
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

-- Reference: Students_MeetingDetails (table: MeetingDetails)
ALTER TABLE dbo.MeetingDetails ADD CONSTRAINT
Students_MeetingDetails
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);
```

```
-- Reference: Students_StudentSubjectGrades (table:
StudentSubjectGrades)
ALTER TABLE dbo.StudentSubjectGrades ADD CONSTRAINT
Students_StudentSubjectGrades
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);



-- Reference: Students_SubjectStudentAttendance (table:
SubjectStudentAttendance)
ALTER TABLE SubjectStudentAttendance ADD CONSTRAINT
Students_SubjectStudentAttendance
    FOREIGN KEY (StudentID)
    REFERENCES dbo.Students (StudentID);

-- Reference: Studies_SubjectStudentAttendance (table:
SubjectStudentAttendance)
ALTER TABLE SubjectStudentAttendance ADD CONSTRAINT
Studies_SubjectStudentAttendance
    FOREIGN KEY (StudiesID)
    REFERENCES dbo.Studies (StudiesID);

-- Reference: Subjects_SubjectStudentAttendance (table:
SubjectStudentAttendance)
ALTER TABLE SubjectStudentAttendance ADD CONSTRAINT
Subjects_SubjectStudentAttendance
    FOREIGN KEY (SubjectID)
    REFERENCES dbo.Subjects (SubjectID);
```

Widoki

1. FINANCIAL_REPORT

Zestawienie łącznych przychodów ze wszystkich źródeł: webinarów, kursów i studiów.

```
CREATE VIEW FINANCIAL_REPORT AS
-- Zestawienie przychodów z webinarów
SELECT w.WebinarID AS ID,
       w.WebinarName AS Name,
       'Webinar' AS Type,
       w.WebinarPrice *
       (SELECT COUNT(*)
```

```
        FROM OrderedWebinars ow
        JOIN OrderDetails od ON ow.WebinarsOrderDetailsID =
od.OrderDetailsID
        JOIN Orders o ON od.OrderID = o.OrderID
        WHERE ow.WebinarID = w.WebinarID) AS TotalIncome
FROM Webinars w

UNION

-- Zestawienie przychodów z kursów
SELECT c.CourseID AS ID,
       c.CourseName AS Name,
       'Course' AS Type,
       c.CoursePrice *
       (SELECT COUNT(*)
        FROM OrderedCourses oc
        JOIN OrderDetails od ON oc.CoursesOrderDetailsID =
od.OrderDetailsID
        JOIN Orders o ON od.OrderID = o.OrderID
        WHERE oc.CourseID = c.CourseID) AS TotalIncome
FROM Courses c

UNION

-- Zestawienie przychodów ze studiów
SELECT s.StudiesID AS ID,
       s.StudiesName AS Name,
       'Study' AS Type,
       s.StudiesFee *
       (SELECT COUNT(*)
        FROM OrderedStudies os
        JOIN OrderDetails od ON os.OrderDetailsID =
od.OrderDetailsID
        JOIN Orders o ON od.OrderID = o.OrderID
        WHERE os.StudiesID = s.StudiesID) +
       (SELECT SUM(sm.MeetingPrice)
        FROM StudyMeeting sm
        JOIN Subjects sb ON sm.SubjectID = sb.SubjectID
        WHERE sb.StudiesID = s.StudiesID) AS TotalIncome
FROM Studies s;
```

2. WEBINARS_FINANCIAL_REPORT

Raport finansowy pokazujący przychody tylko z webinarów.

```
CREATE VIEW WEBINARS_FINANCIAL_REPORT AS

SELECT ID AS 'Webinar ID', Name, TotalIncome
```

```
FROM FINANCIAL_REPORT

WHERE Type = 'Webinar';
```

3. STUDIES_FINANCIAL_REPORT

Raport finansowy pokazujący przychody tylko ze studiów.

```
CREATE VIEW STUDIES_FINANCIAL_REPORT AS

SELECT ID AS 'Study ID', Name, TotalIncome

FROM FINANCIAL_REPORT

WHERE Type = 'Study';
```

4. COURSES_FINANCIAL_REPORT

Raport finansowy pokazujący przychody tylko z kursów.

```
CREATE VIEW COURSES_FINANCIAL_REPORT AS

SELECT ID AS 'Course ID', Name, TotalIncome

FROM FINANCIAL_REPORT

WHERE Type = 'Course';
```

5. STUDENT_DEBTORS

Widok prezentujący listę wszystkich studentów z nieuregulowanymi płatnościami wraz z kwotami.

```
CREATE VIEW STUDENT_DEBTORS AS

WITH OrderTotals AS (

    SELECT

        o.OrderID,

        o.StudentID,

        s.FirstName,

        s.LastName,
```

```
        s.Email,
        o.OrderDate,
        o.OrderStatus,
        -- Calculate total order value for courses
        COALESCE(SUM(c.CoursePrice), 0) AS TotalCourseCharges,
        -- Calculate total order value for studies
        COALESCE(SUM(st.StudiesFee), 0) AS TotalStudiesFees,
        -- Calculate total order value for webinars
        COALESCE(SUM(w.WebinarPrice), 0) AS TotalWebinarCharges,
        -- Calculate total order value for study meetings
        COALESCE(SUM(sm.MeetingPrice), 0) AS TotalMeetingCharges
    FROM
        dbo.Orders o
    INNER JOIN
        dbo.Students s ON o.StudentID = s.StudentID
    LEFT JOIN
        dbo.OrderedCourses oc ON oc.CoursesOrderDetailsID =
o.OrderID
    LEFT JOIN
        dbo.Courses c ON c.CourseID = oc.CourseID
    LEFT JOIN
        dbo.OrderedStudies os ON os.OrderDetailsID = o.OrderID
    LEFT JOIN
        dbo.Studies st ON st.StudiesID = os.StudiesID
    LEFT JOIN
        dbo.OrderedWebinars ow ON ow.WebinarsOrderDetailsID =
o.OrderID
```

```
    LEFT JOIN

        dbo.Webinars w ON w.WebinarID = ow.WebinarID

    LEFT JOIN

        dbo.OrderedStudyMeetings osm ON
osm.StudeyMeetingOrderDetailsID = o.OrderID

    LEFT JOIN

        dbo.StudyMeeting sm ON sm.MeetingID = osm.StudyMeetingID

    WHERE

        o.OrderStatus IS NULL OR o.OrderStatus = 'unpaid'

    GROUP BY

        o.OrderID, o.StudentID, s.FirstName, s.LastName, s.Email,
o.OrderDate, o.OrderStatus

)

SELECT

    StudentID,

    FirstName,

    LastName,

    Email,

    OrderID,

    OrderDate,

    OrderStatus,

    (TotalCourseCharges + TotalStudiesFees + TotalWebinarCharges +
TotalMeetingCharges) AS TotalUnpaidAmount

FROM

    OrderTotals

WHERE

    (TotalCourseCharges + TotalStudiesFees + TotalWebinarCharges +
TotalMeetingCharges) > 0;
```

6. FUTURE_EVENTS_STATS

Zestawienie zbiorcze informacji o wszystkich przyszłych wydarzeniach, łączące dane ze spotkań, webinarów i modułów kursowych.

```
CREATE VIEW FUTURE_EVENTS_STATS AS

SELECT

    'Meeting' as EventType,

    MeetingName as EventName,

    MeetingDate as EventDate,

    LanguageName,

    TeacherName,

    RegisteredStudents,

    MaxCapacity,

    RemainingSpots

FROM FUTURE_MEETING_STATS

UNION ALL

SELECT

    'Webinar' as EventType,

    WebinarName as EventName,

    WebinarDate as EventDate,

    LanguageName,

    TeacherName,

    RegisteredStudents,

    MaxCapacity,

    RemainingSpots

FROM FUTURE_WEBINAR_STATS
```

```
UNION ALL

SELECT

    'Course Module' as EventType,

    ModuleName as EventName,

    ModuleDate as EventDate,

    LanguageName,

    NULL as TeacherName,

    RegisteredStudents,

    MaxCapacity,

    RemainingSpots

FROM FUTURE_MODULE_STATS;
```

7. FUTURE_MEETING_STATS

Szczegółowe statystyki dotyczące przyszłych spotkań, zawierające informacje o prowadzących i liczbie dostępnych miejsc.

```
CREATE VIEW FUTURE_MEETING_STATS AS

SELECT

    sm.MeetingID,

    sm.MeetingName,

    sm.Date as MeetingDate,

    l.LanguageName,

    e.FirstName + ' ' + e.LastName as TeacherName,

    COUNT(DISTINCT md.StudentID) as RegisteredStudents,

    md.Limit as MaxCapacity,

    CASE

        WHEN md.Limit IS NULL THEN NULL
```

```
        ELSE md.Limit - COUNT(DISTINCT md.StudentID)
    END as RemainingSpots
FROM dbo.StudyMeeting sm
LEFT JOIN dbo.MeetingDetails md ON sm.MeetingID =
md.MeetingDetailsID
INNER JOIN dbo.Languages l ON sm.LanguageID = l.LanguageID
INNER JOIN dbo.Employees e ON sm.TeacherID = e.EmployeeID
WHERE sm.Date > GETDATE()
GROUP BY
    sm.MeetingID,
    sm.MeetingName,
    sm.Date,
    l.LanguageName,
    e.FirstName + ' ' + e.LastName,
    md.Limit;
```

8. FUTURE_MODULE_STATS

Statystyki przyszłych modułów kursowych wraz z informacjami o liczbie zapisanych uczestników i dostępnych miejscach.

```
 CREATE VIEW FUTURE_MODULE_STATS AS
SELECT
    md.ModuleDetailsID,
    md.ModuleName,
    md.CoruseDate as ModuleDate,
    c.CourseName,
    l.LanguageName,
    COUNT(DISTINCT sc.StudentID) as RegisteredStudents,
```

```
    md.Limit as MaxCapacity,

    CASE

        WHEN md.Limit IS NULL THEN NULL

        ELSE md.Limit - COUNT(DISTINCT sc.StudentID)

    END as RemainingSpots

FROM dbo.ModulesDetails md

INNER JOIN dbo.CourseModules cm ON md.ModuleDetailsID =
cm.ModuleID

INNER JOIN dbo.Courses c ON cm.CourseID = c.CourseID

INNER JOIN dbo.Languages l ON cm.LanguageID = l.LanguageID

LEFT JOIN dbo.StudentCourses sc ON c.CourseID =
sc.StudentCoursesID

WHERE md.CoruseDate > GETDATE()

GROUP BY

    md.ModuleDetailsID,

    md.ModuleName,

    md.CoruseDate,

    c.CourseName,

    l.LanguageName,

    md.Limit;
```

9. FUTURE_WEBINAR_STATS

Statystyki przyszłych webinarów z informacjami o prowadzących i liczbie zarejestrowanych uczestników.

```
CREATE VIEW FUTURE_WEBINAR_STATS AS

SELECT

    w.WebinarID,

    w.WebinarName,
```

```
    w.WebinarDate,
    l.LanguageName,
    e.FirstName + ' ' + e.LastName as TeacherName,
    COUNT(DISTINCT sw.StudentID) as RegisteredStudents,
    NULL as MaxCapacity,
    NULL as RemainingSpots
FROM dbo.Webinars w
LEFT JOIN dbo.StudentWebinars sw ON w.WebinarID =
sw.AttendenceWebinarID
INNER JOIN dbo.Languages l ON w.LanguageID = l.LanguageID
INNER JOIN dbo.Employees e ON w.TeacherID = e.EmployeeID
WHERE w.WebinarDate > GETDATE()
GROUP BY
    w.WebinarID,
    w.WebinarName,
    w.WebinarDate,
    l.LanguageName,
    e.FirstName + ' ' + e.LastName;
```

10. COMPLETED_EVENTS_ATTENDANCE

Zestawienie zbiorcze frekwencji na wszystkich zakończonych wydarzeniach (spotkaniach, modułach kursowych i webinarach).

```
CREATE VIEW COMPLETED_EVENTS_ATTENDANCE AS
SELECT
    'Study Meeting' as EventType,
    sm.MeetingName as EventName,
    sm.Date as EventDate,
```

```
    COUNT(DISTINCT sa.StudentID) as TotalStudents,

    SUM(CASE WHEN sa.Attendance = 1 THEN 1 ELSE 0 END) as
PresentStudents,

    CAST(CAST(SUM(CASE WHEN sa.Attendance = 1 THEN 1 ELSE 0 END)
AS FLOAT) /

         CAST(COUNT(DISTINCT sa.StudentID) AS FLOAT) * 100 AS
DECIMAL(5,2)) as AttendancePercentage

FROM StudyMeeting sm

JOIN SubjectStudentAttendance sa ON sm.SubjectID = sa.SubjectID

WHERE sm.Date < GETDATE()

GROUP BY sm.MeetingName, sm.Date



UNION ALL



SELECT

    'Course Module' as EventType,

    md.ModuleName as EventName,

    md.CoruseDate as EventDate,

    COUNT(DISTINCT sma.StudentID) as TotalStudents,

    SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) as
PresentStudents,

    CAST(CAST(SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE
0 END) AS FLOAT) /

         CAST(COUNT(DISTINCT sma.StudentID) AS FLOAT) * 100 AS
DECIMAL(5,2)) as AttendancePercentage

FROM ModulesDetails md

JOIN StudentModulesAttentandce sma ON md.ModuleDetailsID =
sma.ModuleAttendenceID

WHERE md.CoruseDate < GETDATE()

GROUP BY md.ModuleName, md.CoruseDate
```

```
UNION ALL


SELECT

    'Webinar' as EventType,

    w.WebinarName as EventName,

    w.WebinarDate as EventDate,

    COUNT(DISTINCT sw.StudentID) as TotalStudents,

    COUNT(DISTINCT sw.StudentID) as PresentStudents,

    100.00 as AttendancePercentage

FROM Webinars w

JOIN StudentWebinars sw ON w.WebinarID = sw.AttendenceWebinarID

WHERE w.WebinarDate < GETDATE()

GROUP BY w.WebinarName, w.WebinarDate;
```

11. CompletedModulesAttendance

Szczegółowe statystyki frekwencji dla zakończonych modułów kursowych.

```
CREATE VIEW CompletedModulesAttendance AS

SELECT

    md.ModuleName,

    md.ModuleType,

    md.CoruseDate,

    COUNT(DISTINCT sma.StudentID) as TotalStudents,

    SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE 0 END) as
PresentStudents,

    CAST(CAST(SUM(CASE WHEN sma.Attendance = 'Present' THEN 1 ELSE
0 END) AS FLOAT) /
```

```
        CAST(COUNT(DISTINCT sma.StudentID) AS FLOAT) * 100 AS
DECIMAL(5,2)) as AttendancePercentage

FROM ModulesDetails md

JOIN StudentModulesAttentandce sma ON md.ModuleDetailsID =
sma.ModuleAttendenceID

WHERE md.CoruseDate < GETDATE()

GROUP BY md.ModuleName, md.ModuleType, md.CoruseDate;
```

12. CompletedStudyMeetingsAttendance

Szczegółowe statystyki frekwencji dla zakończonych spotkań studyjnych, zawierające informacje o prowadzących i przedmiotach.

```
CREATE VIEW CompletedStudyMeetingsAttendance AS

SELECT

    sm.MeetingName,

    s.SubjectName,

    sm.Date,

    e.FirstName + ' ' + e.LastName as TeacherName,

    l.LanguageName,

    COUNT(DISTINCT ssa.StudentID) as TotalStudents,

    SUM(CASE WHEN ssa.Attendance = 1 THEN 1 ELSE 0 END) as
PresentStudents,

    CAST(CAST(SUM(CASE WHEN ssa.Attendance = 1 THEN 1 ELSE 0 END)
AS FLOAT) /

        CAST(COUNT(DISTINCT ssa.StudentID) AS FLOAT) * 100 AS
DECIMAL(5,2)) as AttendancePercentage

FROM StudyMeeting sm

JOIN Subjects s ON sm.SubjectID = s.SubjectID

JOIN Employees e ON sm.TeacherID = e.EmployeeID

JOIN Languages l ON sm.LanguageID = l.LanguageID

JOIN SubjectStudentAttendance ssa ON sm.SubjectID = ssa.SubjectID
```

```
WHERE sm.Date < GETDATE()

GROUP BY sm.MeetingName, s.SubjectName, sm.Date, e.FirstName,
e.LastName, l.LanguageName;
```

13. COMPLETED_WEBINARS_ATTENDANCE

Szczegółowe statystyki uczestnictwa w zakończonych webinarach wraz z informacjami o cenach i czasie trwania.

```
CREATE VIEW COMPLETED_WEBINARS_ATTENDANCE AS
SELECT
    w.WebinarName,
    w.WebinarDate,
    e.FirstName + ' ' + e.LastName as TeacherName,
    l.LanguageName,
    COUNT(DISTINCT sw.StudentID) as RegisteredStudents,
    w.WebinarPrice,
    w.DurationTime
FROM Webinars w
JOIN StudentWebinars sw ON w.WebinarID = sw.AttendenceWebinarID
JOIN Employees e ON w.TeacherID = e.EmployeeID
JOIN Languages l ON w.LanguageID = l.LanguageID
WHERE w.WebinarDate < GETDATE()
GROUP BY w.WebinarName, w.WebinarDate, e.FirstName, e.LastName,
l.LanguageName, w.WebinarPrice, w.DurationTime;
```

14. ATTENDANCE_LIST

Pełna lista obecności dla wszystkich rodzajów wydarzeń, zawierająca dane osobowe uczestników i status ich obecności.

```
CREATE VIEW ATTENDANCE_LIST AS

SELECT

    'Study Meeting' as Type,

    sm.MeetingName as Name,

    sm.Date as Date,

    s.FirstName + ' ' + s.LastName as StudentName,

    s.Email,

    CASE WHEN ssa.Attendance = 1 THEN 'Present' ELSE 'Absent' END
as AttendanceStatus

FROM StudyMeeting sm
```

```
JOIN SubjectStudentAttendance ssa ON sm.SubjectID = ssa.SubjectID

JOIN Students s ON ssa.StudentID = s.StudentID


UNION ALL


SELECT

    'Course Module' as Type,

    md.ModuleName as Name,

    md.CoruseDate as Date,

    s.FirstName + ' ' + s.LastName as StudentName,

    s.Email,

    sma.Attendance as AttendanceStatus

FROM ModulesDetails md

JOIN StudentModulesAttentandce sma ON md.ModuleDetailsID =
sma.ModuleAttendenceID

JOIN Students s ON sma.StudentID = s.StudentID


UNION ALL


SELECT

    'Webinar' as Type,

    w.WebinarName as Name,

    w.WebinarDate as Date,

    s.FirstName + ' ' + s.LastName as StudentName,

    s.Email,

    'Registered' as AttendanceStatus

FROM Webinars w
```

```
JOIN StudentWebinars sw ON w.WebinarID = sw.AttendenceWebinarID

JOIN Students s ON sw.StudentID = s.StudentID;
```