Podstawy baz danych

dzień i godz zajęć: śr 15:00

nr zespołu: 3

Autorzy: Bartosz Wójcik, Mikołaj Kaleta, Piotr Świerzy

1. Wymagania i funkcje systemu

Funkcje systemu

Zarządzanie webinarami:

- Przechowywanie informacji o nagraniach (np. linki).
- Udostępnianie uczestnikom dostępu na 30 dni (po płatności).

Zarządzanie kursami:

- Automatyczna weryfikacja zaliczenia kursów asynchronicznych
- Monitorowanie postępów uczestników i zaliczanie kursów na podstawie ukończenia modułów.
- Nie pozwala na dodanie nowych uczestników po przekroczeniu limitu miejsc (dla kursów hybrydowych i stacjonarnych)

Zarządzanie studiami:

- Ustalanie harmonogramów zjazdów z możliwością wprowadzania zmian losowych.
- Monitorowanie frekwencji i zarządzanie praktykami.
- Ustalanie cen dla uczestników wewnętrznych i zewnętrznych.
- Nie pozwala na dodanie nowych uczestników po przekroczeniu limitu miejsc

Obsługa płatności:

- Generowanie linków płatności.
- Obsługa informacji o statusie płatności.

- Zarządzanie zaliczkami i końcowymi płatnościami za kursy/studia.
- Blokuje dostęp do kursu/studiów, jeśli użytkownik nie zapłaci 3 dni przed ich rozpoczęciem

Raportowanie:

- Generowanie raportów finansowych, list dłużników, frekwencji i zapisów.
- Raportowanie kolizji czasowych zapisów uczestników.

Inne:

- Automatyczna generacja i wysyłka dyplomów ukończenia kursów i studiów
- Informowanie użytkowników o nadchodzących terminach płatności i zajęć
- Przechowuje produkty w koszyku użytkownika
- Nie pozwala na zapisanie się 2 razy na te same zajęcia

Użytkownicy i ich uprawnienia

Użytkownik bez konta:

- Stworzenie konta
- Wgląd na dostępne oferty

Użytkownik z kontem:

- Dokonanie płatności
- Zarządzanie koszykiem
- Dostęp do wykupionych kursów, nagrań i dyplomów
- Dostęp do harmonogramu zajęć na studiach i sylabusa
- Sprawdzenie aktualnej frekwencji
- Zaliczenie kursu on-line asynchronicznego poprzez obejrzenie nagrania
- Odrabianie nieobecności
- Wgląd na swoje oceny
- Kontakt z administratorem
- Rezygnacja ze studiów i kursów
- Edytowanie informacji o sobie
- Zgoda na przesyłanie treści marketingowych (mailowo/SMS o nowych wydarzeniach, kursach…)

Wykładowca

- Przegląd harmonogramu przypisanych webinarów, kursów i studiów
- Dodawanie materiałów edukacyjnych

- Rejestrowanie obecności użytkownika na zajęciach
- Generowanie raportu o frekwencji
- Generowanie raportu o osobach zapisanych na jego przyszłe wydarzenia
- Wystawia zaliczenia z kursów i studiów użytkownikom
- Potwierdzenie odrabiania kursu

Tłumacz

- Sprawdzanie harmonogramu wydarzeń, które wymagają ich obsługi
- Uczestnictwo w spotkaniach online i prowadzenie tłumaczenia w czasie rzeczywistym

Dyrektor Szkoły

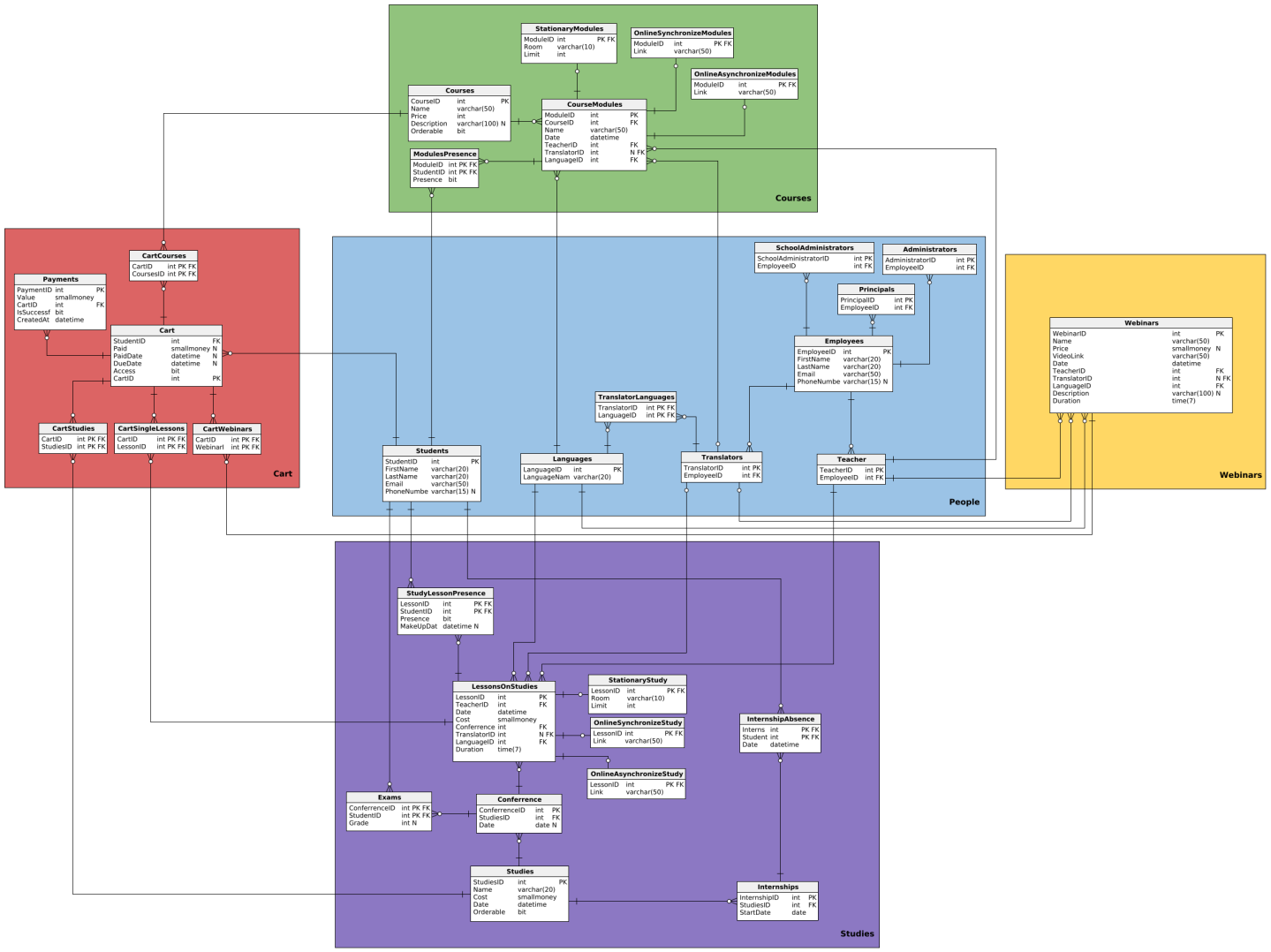
- Generowanie raportu o liczbie osób zapisanych na kursy i o frekwencji
- Może przedłużyć termin zapłaty za dany kurs, dać dostęp do webinaru przed zapłatą (w szczególnych przypadkach)
- Tworzenie sylabusu przedmiotów
- Dodawanie osób “z zewnątrz” jeśli pozwala na to limit miejsc

Administrator

- Ma dostęp do wszystkich informacji w systemie
- Może usuwać nagrania
- Może dodawać i usuwać oferty ze strony

2. Baza danych

Schemat bazy danych



Vertabelo

Opis poszczególnych tabel

Kategoria People

Tabela Students

- kod DDL

```
CREATE TABLE Students (
    StudentID int  NOT NULL,
    FirstName varchar(20)  NOT NULL,
    LastName varchar(20)  NOT NULL,
    Email varchar(50)  NOT NULL CHECK (Email LIKE '%_@__%.__%'),
    PhoneNumber varchar(15)  NULL CHECK (ISNUMERIC(PhoneNumber) = 1),
    CONSTRAINT Students_pk PRIMARY KEY  (StudentID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
StudentID	int	Klucz główny. Unikalny identyfikator studenta.
FirstName	varchar(20)	Imię studenta.
LastName	varchar(20)	Nazwisko studenta.
Email	varchar(50)	Adres e-mail studenta. UWAGA, musi spełniać format adresu e-mail (`%_@__%.__%`).
PhoneNumber	varchar(15)	Numer telefonu studenta (opcjonalny). UWAGA, musi składać się wyłącznie z cyfr.

Tabela Employees

- kod DDL

```
CREATE TABLE Employees (
    EmployeeID int  NOT NULL,
    FirstName varchar(20)  NOT NULL,
    LastName varchar(20)  NOT NULL,
    Email varchar(50)  NOT NULL CHECK (Email LIKE '%_@__%.__%'),
    PhoneNumber varchar(15)  NULL CHECK (ISNUMERIC(PhoneNumber) = 1),
    CONSTRAINT Employees_pk PRIMARY KEY  (EmployeeID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
EmployeeID	int	Klucz główny. Unikalny identyfikator pracownika.
FirstName	varchar(20)	Imię pracownika.
LastName	varchar(20)	Nazwisko pracownika.
Email	varchar(50)	Adres e-mail pracownika. UWAGA, musi spełniać format adresu e-mail (`%_@__%.__%`).
PhoneNumber	varchar(15)	Numer telefonu pracownika (opcjonalny). UWAGA, musi składać się wyłącznie z cyfr.

Tabela Administrators

- kod DDL

```
CREATE TABLE Administrators (
    AdministratorID int  NOT NULL,
    EmployeeID int  NOT NULL,
    CONSTRAINT Administrators_pk PRIMARY KEY  (AdministratorID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
AdministratorID	int	Klucz główny. Unikalny identyfikator administratora.

Nazwa atrybutu	Typ	Opis/Uwagi
EmployeeID	int	Identyfikator pracownika przypisanego do roli administratora.

Tabela Principals

- kod DDL

```
CREATE TABLE Principals (
    PrincipalID int  NOT NULL,
    EmployeeID int  NOT NULL,
    CONSTRAINT Principals_pk PRIMARY KEY  (PrincipalID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
PrincipalID	int	Klucz główny. Unikalny identyfikator dyrektora.
EmployeeID	int	Identyfikator pracownika przypisanego do roli dyrektora.

Tabela SchoolAdministrators

- kod DDL

```
CREATE TABLE SchoolAdministrators (
    SchoolAdministratorID int  NOT NULL,
    EmployeeID int  NOT NULL,
    CONSTRAINT SchoolAdministrators_pk PRIMARY KEY  (SchoolAdministratorID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
SchoolAdministratorID	int	Klucz główny. Unikalny identyfikator administratora szkoły.
EmployeeID	int	Identyfikator pracownika przypisanego do roli administratora szkoły.

Tabela Translators

- kod DDL

```
CREATE TABLE Translators (
    TranslatorID int  NOT NULL,
    EmployeeID int  NOT NULL,
    CONSTRAINT Translators_pk PRIMARY KEY  (TranslatorID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
TranslatorID	int	Klucz główny. Unikalny identyfikator tłumacza.
EmployeeID	int	Identyfikator pracownika przypisanego do roli tłumacza.

Tabela Teacher

- kod DDL

```
CREATE TABLE Teacher (
    TeacherID int  NOT NULL,
    EmployeeID int  NOT NULL,
    CONSTRAINT Teacher_pk PRIMARY KEY  (TeacherID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
TeacherID	int	Klucz główny. Unikalny identyfikator nauczyciela.
EmployeeID	int	Identyfikator pracownika przypisanego do roli nauczyciela.

Tabela Languages

- kod DDL

```
CREATE TABLE Languages (
    LanguageID int  NOT NULL,
    LanguageName varchar(20)  NOT NULL,
    CONSTRAINT Languages_pk PRIMARY KEY  (LanguageID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
LanguageID	int	Klucz główny. Unikalny identyfikator języka.
LanguageName	varchar(20)	Nazwa języka.

Tabela TranslatorLanguages

- kod DDL

```
CREATE TABLE TranslatorLanguages (
    TranslatorID int  NOT NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT TranslatorLanguages_pk PRIMARY KEY  (TranslatorID,LanguageID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
TranslatorID	int	Część klucza głównego. Identyfikator tłumacza.
LanguageID	int	Część klucza głównego. Identyfikator języka, który tłumacz obsługuje.

Kategoria Studies

Tabela LessonsOnStudies

- kod DDL

```
CREATE TABLE LessonsOnStudies (
    LessonID int  NOT NULL,
    TeacherID int  NOT NULL,
    Date datetime  NOT NULL,
    Cost smallmoney  NOT NULL DEFAULT 1000,
    ConferrenceID int  NOT NULL,
    TranslatorID int NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT NonNegativeCost NOT NULL CHECK (Cost>0),
    CONSTRAINT LessonsOnStudies_pk PRIMARY KEY  (LessonID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
LessonID	int	Klucz główny. Unikalny identyfikator lekcji.
TeacherID	int	Identyfikator nauczyciela prowadzącego lekcję.
Date	datetime	Data i godzina przeprowadzenia lekcji.
Cost	smallmoney	Koszt lekcji. UWAGA, wartość musi być większa niż 0. Domyślnie 1000.
ConferrenceID	int	Identyfikator zjazdu.
TranslatorID	int	Identyfikator translatora, może być NULL.
LanguageID	int	Identyfikator języka, w którym prowadzona jest lekcja.

Tabela Conferrence

- kod DDL

```
CREATE TABLE Conferrence (
    ConferrenceID int  NOT NULL,
    StudiesID int  NOT NULL,
    Date datetime  NOT NULL,
    CONSTRAINT Subjects_pk PRIMARY KEY  (ConferrenceID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
ConferrenceID	int	Klucz główny. Unikalny identyfikator zjazdu.
StudiesID	int	Identyfikator programu studiów, do którego należy zjazd.
Date	datetime	Data zjazdu.

Tabela Studies

- kod DDL

```
CREATE TABLE Studies (
    StudiesID int  NOT NULL,
    Name varchar(20)  NOT NULL,
    Cost smallmoney  NOT NULL,
    Date datetime  NOT NULL,
    Orderable bit  NOT NULL,
    CONSTRAINT Studies_pk PRIMARY KEY  (StudiesID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
StudiesID	int	Klucz główny. Unikalny identyfikator programu studiów.
Name	varchar(20)	Nazwa programu studiów.
Cost	smallmoney	Koszt programu studiów.
Date	datetime	Data rozpoczęcia programu studiów.
Orderable	bit	Informacja o możliwości zakupu (1 - dostępny, 0 - niedostępny)

Tabela OnlineSynchronizeStudy

- kod DDL

```
CREATE TABLE OnlineSynchronizeStudy (
    LessonID int  NOT NULL,
    Link varchar(50)  NOT NULL,
    CONSTRAINT OnlineSynchronizeStudy_pk PRIMARY KEY  (LessonID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
LessonID	int	Klucz główny. Unikalny identyfikator lekcji online.
Link	varchar(50)	Link do zajęć online.

Tabela OnlineAsynchronizeStudy

- kod DDL

```
CREATE TABLE OnlineAsynchronizeStudy (
    LessonID int  NOT NULL,
    Link varchar(50)  NOT NULL,
    CONSTRAINT OnlineAsynchronizeStudy_pk PRIMARY KEY  (LessonID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
LessonID	int	Klucz główny. Unikalny identyfikator lekcji online.
Link	varchar(50)	Link do materiałów.

Tabela StationaryStudy

- kod DDL

```
CREATE TABLE StationaryStudy (
    LessonID int  NOT NULL,
    Room varchar(10)  NOT NULL,
    Limit int  NOT NULL,
    CONSTRAINT StationaryStudy_pk PRIMARY KEY  (LessonID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
LessonID	int	Klucz główny. Unikalny identyfikator lekcji.
Room	varchar(10)	Numer sali.
Limit	int	Maksymalna liczba uczestników.

Tabela Internships

- kod DDL

```
CREATE TABLE Internships (
    InternshipID int  NOT NULL,
    StudiesID int  NOT NULL,
    CONSTRAINT Internships_pk PRIMARY KEY  (InternshipID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
InternshipID	int	Klucz główny. Unikalny identyfikator praktyki.
StudiesID	int	Identyfikator powiązanych studiów.

Tabela InternshipAbsence

- kod DDL

```
CREATE TABLE InternshipAbsence (
    InternshipID int  NOT NULL,
    StudentID int  NOT NULL,
    Date datetime  NOT NULL,
    CONSTRAINT InternshipAbsence_pk PRIMARY KEY  (InternshipID,StudentID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
InternshipID	int	Część klucza głównego. Identyfikator praktyki.
StudentID	int	Część klucza głównego. Identyfikator studenta biorącego udział w praktyce.
Date	datetime	Data nieobecności.

Tabela StudyLessonPresence

- kod DDL

```
CREATE TABLE StudyLessonPresence (
    LessonID int  NOT NULL,
    StudentID int  NOT NULL,
    Presence bit  NOT NULL,
    MakeUpDate datetime  NULL,
    CONSTRAINT StudyLessonPresence_pk PRIMARY KEY  (StudentID,LessonID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
LessonID	int	Część klucza głównego. Identyfikator lekcji.
StudentID	int	Część klucza głównego. Identyfikator studenta biorącego udział w lekcji.
Presence	bit	Informacja o obecności (1 - obecny, 0 - nieobecny).

Nazwa atrybutu	Typ	Opis/Uwagi
MakeUpDate	datetime	Data

Tabela Exams

- kod DDL

```
CREATE TABLE Exams (
    ConferrenceID int  NOT NULL,
    StudentID int  NOT NULL,
    Grade int  NULL,
    CONSTRAINT Exams_pk PRIMARY KEY  (ConferrenceID,StudentID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
ConferrenceID	int	Część klucza głównego. Identyfikator zjazdu.
StudentID	int	Część klucza głównego. Identyfikator studenta biorącego udział w lekcji.
Grade	int	Ocena z egzaminu.

Kategoria Courses

Tabela Courses

- kod DDL

```
CREATE TABLE Courses (
    CourseID int  NOT NULL,
    Name varchar(50)  NOT NULL,
    Price int  NOT NULL,
    Description varchar(100)  NULL,
    Orderable bit  NOT NULL,
    CONSTRAINT Courses_pk PRIMARY KEY  (CourseID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
CourseID	int	Klucz główny. Unikalny identyfikator kursu.
Name	int	Nazwa kursu.
Price	int	Cena kursu.
Description	int	Opcjonalny opis kursu.
Orderable	bit	Informacja o możliwości zakupu (1 - dostępny, 0 - niedostępny)

Tabela CourseModules

- kod DDL

```
CREATE TABLE CourseModules (
    ModuleID int  NOT NULL,
    CourseID int  NOT NULL,
    Name varchar(50)  NOT NULL,
    Date datetime  NOT NULL,
    TeacherID int  NOT NULL,
    TranslatorID int  NULL,
    LanguageID int  NOT NULL,
    CONSTRAINT CourseModules_pk PRIMARY KEY  (ModuleID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
ModuleID	int	Klucz główny. Unikalny identyfikator modułu kursu.
CourseID	int	Identyfikator kursu, do którego należy moduł.
Name	varchar(50)	Nazwa modułu kursu.
Date	datetime	Data przeprowadzenia modułu kursu.
TeacherID	int	Identyfikator nauczyciela prowadzącego moduł.
TranslatorID	int	Identyfikator tłumacza. Może być NULL.
LanguageID	int	Identyfikator języka, w którym prowadzony jest moduł.

Tabela ModulesPresence

- kod DDL

```
CREATE TABLE ModulesPresence (
    ModuleID int  NOT NULL,
    StudentID int  NOT NULL,
    Presence bit  NOT NULL,
    CONSTRAINT ModulesPresence_pk PRIMARY KEY  (ModuleID,StudentID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
ModuleID	int	Część klucza głównego. Identyfikator modułu.
StudentID	int	Część klucza głównego. Identyfikator studenta.
Presence	bit	Informacja o obecności na zajęciach (0 - nieobecny, 1 - obecny).

Tabela StationaryModules

- kod DDL

```
CREATE TABLE StationaryModules (
    ModuleID int  NOT NULL,
    Room varchar(10)  NOT NULL,
    Limit int  NOT NULL,
    CONSTRAINT StationaryModules_pk PRIMARY KEY  (ModuleID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
ModuleID	int	Klucz główny. Identyfikator modułu.
Room	varchar(10)	Numer pokoju.
Limit	int	Maksymalna liczba uczestników.

Tabela OnlineAsynchronizeModules

- kod DDL

```
CREATE TABLE OnlineAsynchronizeModules (
    ModuleID int  NOT NULL,
    Link varchar(50)  NOT NULL,
    CONSTRAINT OnlineAsynchronizeModules_pk PRIMARY KEY  (ModuleID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
ModuleID	int	Klucz główny. Identyfikator modułu.
Link	varchar(50)	Link do zasobów modułu online.

Tabela OnlineSynchronizeModules

- kod DDL

```
CREATE TABLE OnlineSynchronizeModules (
    ModuleID int  NOT NULL,
    Link varchar(50)  NOT NULL,
    CONSTRAINT OnlineSynchronizeModules_pk PRIMARY KEY  (ModuleID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
ModuleID	int	Klucz główny. Identyfikator modułu.
Link	varchar(50)	Link do zasobów modułu.

Kategoria Webinars

Tabela Webinars

- kod DDL

```
CREATE TABLE Webinars (
    WebinarID int  NOT NULL,
    Name varchar(50)  NOT NULL,
    Price smallmoney  NULL,
    VideoLink varchar(50)  NOT NULL,
    Date datetime  NOT NULL,
    TeacherID int  NOT NULL,
    TranslatorID int  NULL,
    LanguageID int  NOT NULL,
    Description varchar(100)  NULL,
    Duration time(7)  NOT NULL,
    CONSTRAINT Webinars_pk PRIMARY KEY  (WebinarID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
WebinarID	int	Klucz główny. Identyfikator webinaru.
Name	varchar(50)	Nazwa webinaru.
Price	smallmoney	Cena webinaru. Może być pusta (NULL).
VideoLink	varchar(50)	Link do nagrania webinaru.
Date	datetime	Data przeprowadzenia webinaru.
TeacherID	int	Identyfikator nauczyciela prowadzącego webinar.
TranslatorID	int	Identyfikator tłumacza. Może być NULL.
LanguageID	int	Identyfikator języka, w którym prowadzony jest webinar.
Description	varchar(100)	Opis webinaru. Może być pusty (NULL).
Duration	time(7)	Czas trwania webinaru.

Kategoria Cart

Tabela Cart

- kod DDL

```
CREATE TABLE Cart (
    CartID int IDENTITY(1,1) NOT NULL,
    StudentID int  NOT NULL,
    Paid smallmoney  NOT NULL,
    PaidDate datetime  NOT NULL,
    DueDate datetime  NOT NULL,
    Access bit  NOT NULL,
    CONSTRAINT Cart_pk PRIMARY KEY  (CartID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
CartID	int	Klucz główny. Identyfikator koszyka.
StudentID	int	Identyfikator studenta, do którego należy koszyk.
Paid	smallmoney	Kwota zapłacona za koszyk.
PaidDate	datetime	Data dokonania płatności za koszyk.
DueDate	datetime	Data do kiedy należy dokonać płatności.
Access	bit	Informacja o tym, czy student ma już dostęp do przedmiotów (1 - dostęp, 0 - brak dostępu).

Tabela CartCourses

- kod DDL

```
CREATE TABLE CartCourses (
    CartID int  NOT NULL,
    CoursesID int  NOT NULL,
    CONSTRAINT CartCourses_pk PRIMARY KEY  (CoursesID,CartID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
CartID	int	Część klucza głównego. Identyfikator koszyka.
CoursesID	int	Część klucza głównego. Identyfikator kursu.

Tabela CartWebinars

- kod DDL

```
CREATE TABLE CartWebinars (
    CartID int  NOT NULL,
    WebinarID int  NOT NULL,
    CONSTRAINT CartWebinars_pk PRIMARY KEY  (CartID,WebinarID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
CartID	int	Część klucza głównego. Identyfikator koszyka.
WebinarID	int	Część klucza głównego. Identyfikator webinaru.

Tabela CartStudies

- kod DDL

```
CREATE TABLE CartStudies (
    CartID int  NOT NULL,
    StudiesID int  NOT NULL,
    CONSTRAINT CartStudies_pk PRIMARY KEY  (CartID,StudiesID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
CartID	int	Część klucza głównego. Identyfikator koszyka.
StudiesID	int	Część klucza głównego. Identyfikator studiów.

Tabela CartSingleLessons

- kod DDL

```
CREATE TABLE CartSingleLessons (
    CartID int  NOT NULL,
    LessonID int  NOT NULL,
    CONSTRAINT CartSingleLessons_pk PRIMARY KEY  (CartID,LessonID)
);
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
CartID	int	Część klucza głównego. Identyfikator koszyka.
LessonID	int	Część klucza głównego. Identyfikator lekcji.

Tabela Payments

- kod DDL

```
CREATE TABLE dbo.Payments (
    PaymentID int  NOT NULL IDENTITY(1, 1),
    Value smallmoney  NOT NULL,
    CartID int  NOT NULL,
    IsSuccessful bit  NOT NULL,
    CreatedAt datetime  NOT NULL,
    CONSTRAINT CHK_Payments_Value CHECK (( ( [ Value ] > ( 0 ) ) )),
    CONSTRAINT Payments_pk PRIMARY KEY (PaymentID)
)
```

- Opis

Nazwa atrybutu	Typ	Opis/Uwagi
PaymentID	int	Klucz główny. Identyfikator płatności.
Value	smallmoney	Wartość płatności.
CartID	int	Identyfikator koszyka.
IsSuccessful	bit	Informacja o pomyślności transakcji.

Nazwa atrybutu	Typ	Opis/Uwagi
CreatedAt	datetime	Data utworzenia płatności.

3. Widoki, procedury/funkcje, triggery

Widoki

Zestawienie przychodów każdego szkolenia (FINANCIAL_REPORT)

```
CREATE VIEW FINANCIAL_REPORT AS
SELECT *, 'Webinar' AS Type
FROM [FINANCIAL_REPORT_WEBINARS]

UNION

SELECT *, 'Course' AS Type
FROM [FINANCIAL_REPORT_COURSES]

UNION

SELECT *, 'Study' AS Type
FROM [FINANCIAL_REPORT_STUDIES]
```

Zestawienie przychodów z webinarów (FINANCIAL_REPORT_WEBINARS)

```
CREATE VIEW FINANCIAL_REPORT_WEBINARS AS
SELECT w.WebinarID AS ID, w.Name, w.Price *
    (SELECT count(*)
    FROM CartWebinars cw JOIN
    Cart c ON c.CartID = cw.CartID
    WHERE cw.WebinarID = w.WebinarID) AS TotalIncome
FROM Webinars w
```

Roczne zestawienie przychodów z webinarów (FINANCIAL_REPORT_WEBINARS_BY_YEAR)

```sql
CREATE VIEW FINANCIAL_REPORT_WEBINARS_BY_YEAR AS
SELECT
    w.WebinarID AS ID,
    w.Name,
    YEAR(c.PaidDate) AS IncomeYear,
    SUM(c.Paid) AS TotalIncome
FROM
    Webinars w
    JOIN CartWebinars cw ON cw.WebinarID = w.WebinarID
    JOIN Cart c ON c.CartID = cw.CartID
WHERE
    c.PaidDate IS NOT NULL -- Uwzględniamy tylko zamówienia, które zostały opłacone
GROUP BY
    w.WebinarID,
    w.Name,
    YEAR(c.PaidDate);
```

Zestawienie przychodów z kursów (FINANCIAL_REPORT_COURSES)

```sql
CREATE VIEW FINANCIAL_REPORT_COURSES AS
SELECT cs.CourseID AS ID, cs.Name, cs.Price *
    (SELECT count(*)
    FROM CartCourses cc JOIN
    Cart c ON c.CartID = cc.CartID
    WHERE cc.CoursesID = cs.CourseID) AS TotalIncome
FROM Courses cs
```

Roczne zestawienie przychodów z kursów (FINANCIAL_REPORT_COURSES_BY_YEAR)

```sql
CREATE VIEW FINANCIAL_REPORT_COURSES_BY_YEAR AS
SELECT
    cs.CourseID AS ID,
    cs.Name,
    YEAR(c.PaidDate) AS IncomeYear,
    sum(c.Paid) AS TotalIncome
FROM
    Courses cs
    JOIN CartCourses cc ON cc.CoursesID = cs.CourseID
    JOIN Cart c ON c.CartID = cc.CartID
WHERE
    c.PaidDate IS NOT NULL -- Uwzględniamy tylko zamówienia, które zostały opłacone
GROUP BY
    cs.CourseID,
    cs.Name,
    YEAR(c.PaidDate)
```

Zestawienie przychodów ze studiów (FINANCIAL_REPORT_STUDIES)

```sql
CREATE VIEW FINANCIAL_REPORT_STUDIES AS
SELECT s.StudiesID AS ID, s.Name, s.Cost *
    (SELECT count(*)
    FROM CartStudies cs JOIN
    Cart c ON c.CartID = cs.CartID
    WHERE cs.StudiesID = s.StudiesID) AS TotalIncome
FROM Studies s
```

Roczne zestawienie przychodów ze studiów (FINANCIAL_REPORT_STUDIES_BY_YEAR)

```sql
CREATE VIEW FINANCIAL_REPORT_STUDIES_BY_YEAR AS
SELECT
    s.StudiesID AS ID,
    s.Name,
    YEAR(c.PaidDate) AS IncomeYear,
    SUM(c.Paid) AS TotalIncome
FROM
    Studies s
    JOIN CartStudies cs ON cs.StudiesID = s.StudiesID
    JOIN Cart c ON c.CartID = cs.CartID
WHERE
    c.PaidDate IS NOT NULL -- Uwzględniamy tylko zamówienia, które zostały opłacone
GROUP BY
    s.StudiesID,
    s.Name,
    YEAR(c.PaidDate)
```

Zestawienie przychodów z lekcji studyjnych (FINANCIAL_REPORT_STUDIES_LESSON)

```sql
CREATE VIEW FINANCIAL_REPORT_STUDIES_LESSON AS
SELECT s.LessonID AS ID, 'SingleLesson' as Name, s.Cost *
    (SELECT count(*)
    FROM CartSingleLessons cs JOIN
    Cart c ON c.CartID = cs.CartID
    WHERE cs.LessonID = s.LessonID) AS TotalIncome
FROM LessonsOnStudies s
```

Zestawienie obecności studentów na wszystkich szkoleniach (PRESENCE_ALL)

```
CREATE VIEW PRESENCE_ALL AS
SELECT *, 'Course Module' AS EventType
FROM [PRESENCE_COURSE_MODULES]

UNION ALL

SELECT *, 'Study Lesson' AS EventType
FROM [PRESENCE_STUDY_LESSONS]
```

Zestawienie obecności na spotkaniach studyjnych (PRESENCE_STUDY_LESSONS)

```
CREATE VIEW PRESENCE_STUDY_LESSONS AS
SELECT slp.LessonID AS ID, los.Date, s.FirstName,
s.LastName, (CASE WHEN Presence = 1 THEN 'Present' ELSE 'Absent' END) AS Presence
FROM LessonsOnStudies AS los INNER JOIN
StudyLessonPresence AS slp ON los.LessonID = slp.LessonID INNER JOIN
Students AS s ON slp.StudentID = s.StudentID
WHERE (los.Date < GETDATE())
```

Zestawienie obecności na modułach (PRESENCE_COURSE_MODULES)

```
CREATE VIEW PRESENCE_COURSE_MODULES AS
SELECT cm.ModuleID AS ID, cm.Date, s.FirstName,
s.LastName, (CASE WHEN Presence = 1 THEN 'Present' ELSE 'Absent' END) AS Presence
FROM CourseModules AS cm INNER JOIN
ModulesPresence AS mp ON cm.ModuleID = mp.ModuleID INNER JOIN
Students AS s ON mp.StudentID = s.StudentID
WHERE (cm.Date < GETDATE())
```

Zestawienie wydarzeń w nadchodzącym miesiącu (UPCOMING_EVENTS_MONTH)

- Komentarz
  Jest to zestawienie, które pokazuje wszystkie zaplanowane wydarzenia na miesiąc do przodu. Dodadkowo zawiera imię i nazwisko nauczyciela prowadzącego i typ spotkania (Online synch., asynch., stacjonarne)

```
CREATE VIEW UPCOMING_EVENTS_MONTH AS
SELECT *, 'Webinar' as EventType
FROM [UPCOMING_WEBINARS_MONTH]

UNION ALL
SELECT *, 'Study Lesson' as EventType
FROM [UPCOMING_STUDY_LESSONS_MONTH]

UNION ALL
SELECT *, 'Course Module' as EventType
FROM [UPCOMING_COURSE_MODULES_MONTH]
```

Zestawienie spotkań studyjnych w następnym miesiącu (UPCOMING_STUDY_LESSONS_MONTH)

```
CREATE VIEW UPCOMING_STUDY_LESSONS_MONTH AS
SELECT
    l.Date,
    s.ConferrenceID as EventID,
    e.FirstName + ' ' + e.LastName AS Teacher,
        CASE
        WHEN SL.LessonID IS NOT NULL THEN 'Stationary'
        WHEN OAS.LessonID IS NOT NULL THEN 'Online Asynchronous'
        WHEN OS.LessonID IS NOT NULL THEN 'Online Synchronous'
        ELSE 'Unknown'
    END AS MeetingType
FROM LessonsOnStudies l
JOIN Conferrence s ON l.ConferrenceID = s.ConferrenceID
JOIN Teacher t ON l.TeacherID = t.TeacherID
JOIN Employees e ON t.EmployeeID = e.EmployeeID
LEFT JOIN StationaryStudy sl ON l.LessonID = sl.LessonID
LEFT JOIN OnlineAsynchronizeStudy oas ON l.LessonID = oas.LessonID
LEFT JOIN OnlineSynchronizeStudy os ON l.LessonID = os.LessonID
WHERE l.Date BETWEEN GETDATE() AND DATEADD(MONTH, 1, GETDATE())
```

Zestawienie modułów w następnym miesiącu (UPCOMING_COURSE_MODULES_MONTH)

```sql
CREATE VIEW UPCOMING_COURSE_MODULES_MONTH AS
SELECT
    cm.Date,
    cm.ModuleID as EventID,
    e.FirstName + ' ' + e.LastName AS Teacher,
        CASE
        WHEN SM.ModuleID IS NOT NULL THEN 'Stationary'
        WHEN OAM.ModuleID IS NOT NULL THEN 'Online Asynchronous'
        WHEN OSM.ModuleID IS NOT NULL THEN 'Online Synchronous'
        ELSE 'Unknown'
    END AS MeetingType
FROM CourseModules cm
JOIN Courses c ON cm.CourseID = c.CourseID
JOIN Teacher t ON cm.TeacherID = t.TeacherID
JOIN Employees e ON t.EmployeeID = e.EmployeeID
LEFT JOIN StationaryModules sm ON cm.ModuleID = sm.ModuleID
LEFT JOIN OnlineAsynchronizeModules oam ON cm.ModuleID = oam.ModuleID
LEFT JOIN OnlineSynchronizeModules osm ON cm.ModuleID = osm.ModuleID
WHERE cm.Date BETWEEN GETDATE() AND DATEADD(MONTH, 1, GETDATE())
```

Zestawienie webinarów w następnym miesiącu (UPCOMING_WEBINARS_MONTH)

```sql
CREATE VIEW UPCOMING_WEBINARS_MONTH AS
SELECT
    w.Date,
    w.WebinarID as EventID,
    e.FirstName + ' ' + e.LastName AS Teacher,
        'Online' AS MeetingType
FROM Webinars w
JOIN Teacher t ON w.TeacherID = t.TeacherID
JOIN Employees e ON t.EmployeeID = e.EmployeeID
WHERE w.Date BETWEEN GETDATE() AND DATEADD(MONTH, 1, GETDATE());
```

Zestawienie dłużników (DEBTORS)

```
CREATE VIEW DEBTORS AS
SELECT c.CartID, fp.Name, fp.[Full Price], CASE WHEN c.Paid IS NULL THEN 0 ELSE c.Paid END AS Pa
s.FirstName + ' ' + s.LastName AS [Full Name], s.Email, s.PhoneNumber
FROM

(SELECT cc.CartID, c.Price AS [Full Price], c.Name FROM CartCourses AS cc INNER JOIN
Courses AS c ON cc.CoursesID = c.CourseID

UNION

SELECT cw.CartID, CASE WHEN w.Price IS NULL THEN 0 ELSE w.Price END AS [Full Price], w.Name FRO
Webinars AS w ON cw.WebinarID = w.WebinarID

UNION

SELECT cs.CartID, s.Cost AS [Full Price], s.Name FROM CartStudies AS cs INNER JOIN
Studies AS s ON cs.StudiesID = s.StudiesID

UNION

SELECT csl.CartID, los.Cost AS [Full Price], st.Name + ' - ' + su.ConferrenceID AS ConferrenceI
LessonsOnStudies AS los ON csl.LessonID = los.LessonID INNER JOIN
Conferrence AS su ON los.ConferrenceID = su.ConferrenceID INNER JOIN
Studies AS st ON su.StudiesID = st.StudiesID) AS fp JOIN
Cart AS c ON fp.CartID = c.CartID INNER JOIN
Students AS s ON c.StudentID = s.StudentID
WHERE (fp.[Full Price] > c.Paid OR c.Paid IS NULL) AND c.Access = 1;
```

Zestawienie liczby osób zapisanych na przyszłe wydarzenia (PEOPLE_SIGNED_FOR_FUTURE_EVENTS)

```
CREATE VIEW PEOPLE_SIGNED_FOR_FUTURE_EVENTS AS
SELECT *, 'Webinar' as eventType
FROM [PEOPLE_SIGNED_FOR_FUTURE_WEBINARS]

UNION ALL

SELECT *, 'Study Lesson' as eventType
FROM [PEOPLE_SIGNED_FOR_FUTURE_STUDY_LESSONS]

UNION ALL

SELECT *, 'Course Module' as eventType
FROM [PEOPLE_SIGNED_FOR_FUTURE_COURSE_MODULES]
```

Zestawienie liczby osób zapisanych na przyszłe spotkania studyjne (PEOPLE_SIGNED_FOR_FUTURE_STUDY_LESSONS)

```
CREATE VIEW PEOPLE_SIGNED_FOR_FUTURE_STUDY_LESSONS AS
SELECT
    l.Date,
    s.ConferrenceID as eventID,
    e.FirstName + ' ' + e.LastName AS Teacher,
        CASE
        WHEN SL.LessonID IS NOT NULL THEN 'Stationary'
        WHEN OAS.LessonID IS NOT NULL THEN 'Online Asynchronous'
        WHEN OS.LessonID IS NOT NULL THEN 'Online Synchronous'
        ELSE 'Unknown'
    END AS Meeting_Type,
        COUNT(*) AS Signed_Users
FROM LessonsOnStudies l
JOIN Conferrence s ON l.ConferrenceID = s.ConferrenceID
JOIN Teacher t ON l.TeacherID = t.TeacherID
JOIN Employees e ON t.EmployeeID = e.EmployeeID
LEFT JOIN StationaryStudy sl ON l.LessonID = sl.LessonID
LEFT JOIN OnlineAsynchronizeStudy oas ON l.LessonID = oas.LessonID
LEFT JOIN OnlineSynchronizeStudy os ON l.LessonID = os.LessonID
INNER JOIN CartSingleLessons AS csl ON l.LessonID = csl.LessonID
INNER JOIN Cart AS c ON csl.CartID = c.CartID
WHERE l.Date > GETDATE() AND c.Access = 1
GROUP BY CASE WHEN SL.LessonID IS NOT NULL THEN 'Stationary' WHEN OAS.LessonID IS NOT NULL THEN
                e.FirstName + ' ' + e.LastName, s.ConferrenceID, l.Date
```

Zestawienie liczby osób zapisanych na przyszłe moduły (PEOPLE_SIGNED_FOR_FUTURE_COURSE_MODULES)

```
CREATE VIEW PEOPLE_SIGNED_FOR_FUTURE_COURSE_MODULES AS
SELECT
    cm.Date,
    cm.ModuleID AS eventID,
    e.FirstName + ' ' + e.LastName AS Teacher,
        CASE
        WHEN SM.ModuleID IS NOT NULL THEN 'Stationary'
        WHEN OAM.ModuleID IS NOT NULL THEN 'Online Asynchronous'
        WHEN OSM.ModuleID IS NOT NULL THEN 'Online Synchronous'
        ELSE 'Unknown'
    END AS Meeting_Type,
        COUNT(*) AS Signed_Users
FROM CourseModules cm
JOIN Courses c ON cm.CourseID = c.CourseID
JOIN Teacher t ON cm.TeacherID = t.TeacherID
JOIN Employees e ON t.EmployeeID = e.EmployeeID
LEFT JOIN StationaryModules sm ON cm.ModuleID = sm.ModuleID
LEFT JOIN OnlineAsynchronizeModules oam ON cm.ModuleID = oam.ModuleID
LEFT JOIN OnlineSynchronizeModules osm ON cm.ModuleID = osm.ModuleID
INNER JOIN CartCourses AS cc ON c.CourseID = cc.CoursesID
INNER JOIN Cart AS ca ON cc.CartID = ca.CartID
WHERE cm.Date > GETDATE() AND ca.Access = 1
GROUP BY cm.Date, cm.ModuleID, e.FirstName + ' ' + e.LastName, CASE WHEN SM.ModuleID IS NOT NULL
                THEN 'Online Asynchronous' WHEN OSM.ModuleID IS NOT NULL THEN 'Online Synchron
```

Zestawienie liczby osób zapisanych na przyszłe webinary (PEOPLE_SIGNED_FOR_FUTURE_WEBINARS)

```
CREATE VIEW PEOPLE_SIGNED_FOR_FUTURE_WEBINARS AS
SELECT
    w.Date,
    w.WebinarID as eventID,
    e.FirstName + ' ' + e.LastName AS Teacher,
        'Online' AS Meeting_Type,
        COUNT(*) AS Signed_Users
FROM Webinars w
JOIN Teacher t ON w.TeacherID = t.TeacherID
JOIN Employees e ON t.EmployeeID = e.EmployeeID
INNER JOIN CartWebinars AS cw ON w.WebinarID = cw.WebinarID
INNER JOIN Cart AS c ON cw.CartID = c.CartID
WHERE w.Date > GETDATE() AND c.Access = 1
GROUP BY w.Date, w.WebinarID, e.FirstName + ' ' + e.LastName
```

Zestawienie obecności na poszczególnych wydarzeniach (ATTENDANCE)

```
CREATE VIEW ATTENDANCE AS
SELECT *, 'Study Lesson' as type FROM dbo.ATTENDANCE_ON_STUDY_LESSONS

UNION ALL

SELECT *, 'Course Module' as type FROM dbo.ATTENDANCE_ON_COURSE_MODULES
```

Zestawienie obecności na poszczególnych spotkaniach studyjnych (ATTENDANCE_ON_STUDY_LESSONS)

```
CREATE VIEW ATTENDANCE_ON_STUDY_LESSONS AS
SELECT slp.LessonID AS 'ID', los.Date, ROUND(100*(CAST(SUM(CAST(slp.Presence AS Int)) AS FLOAT)
FROM LessonsOnStudies AS los INNER JOIN
StudyLessonPresence AS slp ON los.LessonID = slp.LessonID INNER JOIN
Students AS s ON slp.StudentID = s.StudentID
WHERE (los.Date < GETDATE())
GROUP BY slp.LessonID, los.Date
```

Zestawienie obecności na poszczególnych modułach (ATTENDANCE_ON_COURSE_MODULES)

```
CREATE VIEW ATTENDANCE_ON_COURSE_MODULES AS
SELECT cm.ModuleID AS 'ID', cm.Date, ROUND(100*CAST(SUM(CAST(mp.Presence AS Int)) AS FLOAT) / C(
FROM CourseModules AS cm INNER JOIN
ModulesPresence AS mp ON cm.ModuleID = mp.ModuleID INNER JOIN
Students AS s ON mp.StudentID = s.StudentID
WHERE (cm.Date < GETDATE())
GROUP BY cm.ModuleID, cm.Date;
```

Procedury

Pomyślna opłata części koszyka (AcceptedPartialPaymentForTheCart)

```
CREATE PROCEDURE AcceptedPartialPaymentForTheCart
    @CartID INT,
        @AmountPaid SMALLMONEY
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

                IF NOT EXISTS(SELECT CartID FROM Cart WHERE CartID=@CartID)
                BEGIN
            THROW 51000, 'Cart does not exist.', 1;
        END

        -- Create payment record
        INSERT INTO Payments (Value, CartID, IsSuccessful, CreatedAt)
        VALUES (@AmountPaid, @CartID, 1, GETDATE());

        -- Execute additional payment logic
        EXEC dbo.PayPartialForCart @CartID = @CartID, @AmountPaid = @AmountPaid;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Pomyślna opłata koszyka (AcceptedPaymentForTheCart)

```
CREATE PROCEDURE AcceptedPaymentForTheCart
    @CartID INT
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON;

    DECLARE @PaymentValue SMALLMONEY;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Verify cart exists and lock it
        SELECT @PaymentValue = dbo.GetRemainingPaymentForCart(@CartID)  -- Assuming Paid column
        FROM Cart WITH (UPDLOCK)
        WHERE CartID = @CartID;

                IF NOT EXISTS(SELECT CartID FROM Cart WHERE CartID=@CartID)
                BEGIN
            THROW 51000, 'Cart does not exist.', 1;
        END

        -- Create payment record
        INSERT INTO Payments (Value, CartID, IsSuccessful, CreatedAt)
        VALUES (@PaymentValue, @CartID, 1, GETDATE());

        -- Execute additional payment logic
        EXEC dbo.PayForWholeCart @CartID = @CartID;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Dodaj kurs do koszyka (AddCourseToCart)

```
CREATE PROCEDURE AddCourseToCart
    @StudentID INT,
    @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON; -- Auto-rollback on errors

    DECLARE @CartID INT,
            @CourseStartDate DATETIME,
            @Vacancies INT;

    BEGIN TRANSACTION;

    -- Check course existence and get start date with lock
    SELECT @CourseStartDate = DATEADD(DAY, -3, dbo.GetEarliestModuleDate(@CourseID))
    FROM Courses WITH (UPDLOCK)
    WHERE CourseID = @CourseID;

    IF @CourseStartDate IS NULL
    BEGIN
        ROLLBACK;
        THROW 51000, 'Invalid CourseID provided.', 1;
    END;

    -- Check course vacancies
    SET @Vacancies = dbo.HowManyCourseVacancies(@CourseID);

    IF @Vacancies <= 0
    BEGIN
        ROLLBACK;
        THROW 51001, 'No available vacancies for this course.', 1;
    END;

    -- Insert new cart (CartID is auto-generated)
    INSERT INTO Cart (StudentID, Paid, PaidDate, DueDate, Access)
    VALUES (@StudentID, 0, NULL, @CourseStartDate, 0);

    -- Get auto-generated CartID
    SET @CartID = SCOPE_IDENTITY();
```

```
    -- Add course to cart
    INSERT INTO CartCourses (CartID, CoursesID)
    VALUES (@CartID, @CourseID);

    COMMIT TRANSACTION;
END;
```

Dodaj nieobecność na stażu (AddInternshipAbsence)

```
CREATE PROCEDURE AddInternshipAbsence
    @InternshipID INT,
    @StudentIDList NVARCHAR(MAX), -- Lista ID studentów w formacie np. '1,2,3'
    @Date DATETIME
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        -- Rozdzielenie listy StudentID na tabelę
        DECLARE @StudentIDs TABLE (StudentID INT);
        DECLARE @XML XML = CAST('<i>' + REPLACE(@StudentIDList, ',', '</i><i>') + '</i>' AS XML)

        INSERT INTO @StudentIDs (StudentID)
        SELECT T.c.value('.', 'INT')
        FROM @XML.nodes('/i') T(c);

        -- Sprawdzenie, czy staż istnieje
        IF NOT EXISTS (
            SELECT 1
            FROM [dbo].[Internships]
            WHERE [InternshipID] = @InternshipID
        )
        BEGIN
            THROW 60001, 'Nie znaleziono stażu o podanym ID.', 1;
        END

        -- Iteracja przez studentów i dodawanie indywidualnie
        DECLARE @StudentID INT;

        DECLARE student_cursor CURSOR FOR
        SELECT StudentID FROM @StudentIDs;

        OPEN student_cursor;

        FETCH NEXT FROM student_cursor INTO @StudentID;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            -- Sprawdzenie dostępu studenta do kursu
            IF NOT EXISTS (
```

```
                SELECT 1
                FROM [dbo].[Cart] c
                INNER JOIN CartStudies cs on c.CartID = cs.CartID
                INNER JOIN Studies s on s.StudiesID = cs.StudiesID
                INNER JOIN Internships i on i.StudiesID = s.StudiesID
                WHERE StudentID = @StudentID AND InternshipID = @InternshipID AND Access = 1
            )
            BEGIN
                -- Rzucenie błędu, jeśli student nie jest przypisany do kursu
                DECLARE @ErrorMessage NVARCHAR(200);
                SET @ErrorMessage = 'StudentID ' + CAST(@StudentID AS NVARCHAR) + ' nie jest pr
                THROW 60002, @ErrorMessage, 1;
            END

            -- Dodanie nieobecności, jeśli nie istnieje
            IF NOT EXISTS (
                SELECT 1
                FROM [dbo].[InternshipAbsence]
                WHERE InternshipID = @InternshipID AND StudentID = @StudentID AND Date = @Date
            )
            BEGIN
                INSERT INTO [dbo].[InternshipAbsence] (InternshipID, StudentID, Date)
                VALUES (@InternshipID, @StudentID, @Date);
            END

            FETCH NEXT FROM student_cursor INTO @StudentID;
        END

        CLOSE student_cursor;
        DEALLOCATE student_cursor;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
                IF @@TRANCOUNT > 0
                BEGIN
                        ROLLBACK TRANSACTION
                END;
        THROW;
    END CATCH
END;
```

Dodaj obecność na zajęciach (AddLessonPresence)

```
CREATE PROCEDURE AddLessonPresence
    @LessonID INT,
    @StudentIDList NVARCHAR(MAX) -- Lista ID studentów w formacie np. '1,2,3'
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        -- Rozdzielenie listy StudentID na tabelę
        DECLARE @StudentIDs TABLE (StudentID INT);
        DECLARE @XML XML = CAST('<i>' + REPLACE(@StudentIDList, ',', '</i><i>') + '</i>' AS XML)

        INSERT INTO @StudentIDs (StudentID)
        SELECT T.c.value('.', 'INT')
        FROM @XML.nodes('/i') T(c);

        -- Sprawdzenie, czy lekcja istnieje
        IF NOT EXISTS (
            SELECT 1
            FROM LessonsOnStudies
            WHERE LessonID = @LessonID
        )
        BEGIN
            THROW 60001, 'Nie znaleziono lekcji o podanym ID.', 1;
        END

                -- Iteracja przez studentów i dodawanie indywidualnie
        DECLARE @StudentID INT;

        DECLARE student_cursor CURSOR FOR
        SELECT StudentID FROM @StudentIDs;

        OPEN student_cursor;

        FETCH NEXT FROM student_cursor INTO @StudentID;

        WHILE @@FETCH_STATUS = 0
        BEGIN
                        -- Sprawdzenie dostępu studenta do kursu
            IF NOT EXISTS (
                SELECT 1
```

```
        FROM [dbo].[Cart] c
        INNER JOIN CartStudies cs on c.CartID = cs.CartID
        INNER JOIN Studies s on s.StudiesID = cs.StudiesID
        INNER JOIN Conferrence i on i.StudiesID = s.StudiesID
                        INNER JOIN LessonsOnStudies l on l.ConferrenceID = i.Conferrence
        WHERE StudentID = @StudentID AND LessonID = @LessonID AND Access = 1

                        UNION

                        SELECT 1
        FROM [dbo].[Cart] c
        INNER JOIN CartSingleLessons cs on c.CartID = cs.CartID
        INNER JOIN LessonsOnStudies s on s.LessonID = cs.LessonID
        WHERE StudentID = @StudentID AND s.LessonID = @LessonID AND Access = 1
    )
    BEGIN
        -- Rzucenie błędu, jeśli student nie jest przypisany do kursu
        DECLARE @ErrorMessage NVARCHAR(200);
        SET @ErrorMessage = 'StudentID ' + CAST(@StudentID AS NVARCHAR) + ' nie jest pr
        THROW 60002, @ErrorMessage, 1;
    END

                -- Dodanie obecności, jeśli nie istnieje
    IF NOT EXISTS (
        SELECT 1
        FROM [dbo].StudyLessonPresence
        WHERE StudentID = @StudentID AND LessonID = @LessonID
    )
    BEGIN
        INSERT INTO [dbo].StudyLessonPresence (LessonID, StudentID, Presence)
        VALUES (@LessonID, @StudentID, 1);
    END
                ELSE
                BEGIN
                        DECLARE @ErrorMessage2 NVARCHAR(200);
        SET @ErrorMessage2 = 'StudentID ' + CAST(@StudentID AS NVARCHAR) + ' ma już obe
        THROW 60002, @ErrorMessage2, 1;
                END

                FETCH NEXT FROM student_cursor INTO @StudentID;
END

        CLOSE student_cursor;
```

```
    DEALLOCATE student_cursor;

                -- Ustaw obecność (Presence = 0) dla studentów, którzy mają kurs w koszy
    INSERT INTO StudyLessonPresence (LessonID, StudentID, Presence)
    SELECT DISTINCT @LessonID, st.StudentID, 0
            FROM [dbo].Students st
                            INNER JOIN Cart c on st.StudentID = c.StudentID
            INNER JOIN CartStudies cs on c.CartID = cs.CartID
            INNER JOIN Studies s on s.StudiesID = cs.StudiesID
            INNER JOIN Conferrence i on i.StudiesID = s.StudiesID
                            INNER JOIN LessonsOnStudies l on l.ConferrenceID = i.Conferrence
            WHERE st.StudentID NOT IN (SELECT StudentID FROM @StudentIDs) AND LessonID = @Le
                            AND NOT EXISTS (
          SELECT 1
          FROM  StudyLessonPresence slp
          WHERE slp.LessonID = @LessonID AND slp.StudentID = st.StudentID
      );



            INSERT INTO StudyLessonPresence (LessonID, StudentID, Presence)
    SELECT DISTINCT @LessonID, st.StudentID, 0
                            FROM Students st
            INNER JOIN Cart c on c.StudentID = st.StudentID
            INNER JOIN CartSingleLessons cs on c.CartID = cs.CartID
            INNER JOIN LessonsOnStudies s on s.LessonID = cs.LessonID
                            WHERE st.StudentID NOT IN (SELECT StudentID FROM @StudentIDs) AN
      AND NOT EXISTS (
          SELECT 1
          FROM StudyLessonPresence slp
          WHERE slp.LessonID = @LessonID AND slp.StudentID = st.StudentID
      );

    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
            IF @@TRANCOUNT > 0
            BEGIN
                    ROLLBACK TRANSACTION;
            END;
    -- Rzucenie błędu z informacją
```

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

Dodaj lekcję do koszyka (AddLessonToCart)

```
CREATE PROCEDURE AddLessonToCart
    @StudentID INT,
    @LessonID INT
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON; -- Auto-rollback on errors

    DECLARE @CartID INT,
            @LessonDate DATETIME,
            @Vacancies INT;

    BEGIN TRANSACTION;

    -- Check lesson existence with lock
    SELECT @LessonDate = Date
    FROM LessonsOnStudies WITH (UPDLOCK)
    WHERE LessonID = @LessonID;

    IF @LessonDate IS NULL
    BEGIN
        ROLLBACK;
        THROW 51000, 'Invalid LessonID provided.', 1;
    END;

    -- Check vacancies using the lesson vacancy function
    SET @Vacancies = dbo.HowManyLessonVacancies(@LessonID);

    IF @Vacancies <= 0
    BEGIN
        ROLLBACK;
        THROW 51001, 'No available vacancies for this lesson.', 1;
    END;

    -- Insert new cart (CartID is auto-generated)
    INSERT INTO Cart (StudentID, Paid, PaidDate, DueDate, Access)
    VALUES (@StudentID, 0, NULL, @LessonDate, 0);

    -- Get the auto-generated CartID
    SET @CartID = SCOPE_IDENTITY();
```

```
    -- Add lesson to cart
    INSERT INTO CartSingleLessons (CartID, LessonID)
    VALUES (@CartID, @LessonID);

    COMMIT TRANSACTION;
END;
```

Dodaj obecność na module (AddModulePresence)

```
CREATE PROCEDURE AddModulePresence
    @ModuleID INT,
    @StudentIDList NVARCHAR(MAX) -- Lista ID studentów w formacie np. '1,2,3'
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        -- Rozdzielenie listy StudentID na tabelę
        DECLARE @StudentIDs TABLE (StudentID INT);
        DECLARE @XML XML = CAST('<i>' + REPLACE(@StudentIDList, ',', '</i><i>') + '</i>' AS XML)

        INSERT INTO @StudentIDs (StudentID)
        SELECT T.c.value('.', 'INT')
        FROM @XML.nodes('/i') T(c);

        -- Pobierz CourseID dla danego ModuleID
        DECLARE @CourseID INT;
        SELECT @CourseID = CourseID
        FROM CourseModules
        WHERE ModuleID = @ModuleID;

        IF @CourseID IS NULL
        BEGIN
            THROW 50001, 'Nie znaleziono kursu dla podanego ModuleID.', 1;
        END


                -- Iteracja przez studentów i dodawanie indywidualnie
        DECLARE @StudentID INT;

        DECLARE student_cursor CURSOR FOR
        SELECT StudentID FROM @StudentIDs;

        OPEN student_cursor;

        FETCH NEXT FROM student_cursor INTO @StudentID;

        WHILE @@FETCH_STATUS = 0
        BEGIN
                        -- Sprawdzenie dostępu studenta do kursu
```

```
    IF NOT EXISTS (
        SELECT 1
        FROM [dbo].[Cart] c
        INNER JOIN CartCourses cs on c.CartID = cs.CartID
        INNER JOIN Courses s on s.CourseID = cs.CoursesID
        INNER JOIN CourseModules i on i.CourseID = s.CourseID
        WHERE StudentID = @StudentID AND ModuleID = @ModuleID AND Access = 1
    )
    BEGIN
        -- Rzucenie błędu, jeśli student nie jest przypisany do kursu
        DECLARE @ErrorMessage NVARCHAR(200);
        SET @ErrorMessage = 'StudentID ' + CAST(@StudentID AS NVARCHAR) + ' nie jest pr
        THROW 60002, @ErrorMessage, 1;
    END

            -- Dodanie nieobecności, jeśli nie istnieje
    IF NOT EXISTS (
        SELECT 1
        FROM [dbo].ModulesPresence
        WHERE ModuleID = @ModuleID AND StudentID = @StudentID
    )
    BEGIN
        INSERT INTO [dbo].ModulesPresence (ModuleID, StudentID, Presence)
        VALUES (@ModuleID, @StudentID, 1);
    END
            ELSE
            BEGIN
                    DECLARE @ErrorMessage2 NVARCHAR(200);
        SET @ErrorMessage2 = 'StudentID ' + CAST(@StudentID AS NVARCHAR) + ' ma już obe
        THROW 60002, @ErrorMessage2, 1;
            END

    FETCH NEXT FROM student_cursor INTO @StudentID;
END

        CLOSE student_cursor;
DEALLOCATE student_cursor;

-- Ustaw obecność (Presence = 0) dla studentów, którzy mają kurs w koszyku i Access = 1
INSERT INTO ModulesPresence (ModuleID, StudentID, Presence)
SELECT DISTINCT @ModuleID, s.StudentID, 0
FROM Students s
INNER JOIN Cart c ON c.StudentID = s.StudentID
```

```
        INNER JOIN CartCourses cc ON cc.CartID = c.CartID AND cc.CoursesID = @CourseID
        WHERE s.StudentID NOT IN (SELECT StudentID FROM @StudentIDs)
          AND c.Access = 1
          AND NOT EXISTS (
              SELECT 1
              FROM ModulesPresence mp
              WHERE mp.ModuleID = @ModuleID AND mp.StudentID = s.StudentID
          );

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
                BEGIN
                        ROLLBACK TRANSACTION;
                END;
        -- Rzucenie błędu z informacją
        THROW;
    END CATCH
END;
```

Dodaj studia do koszyka (AddStudiesToCart)

```sql
CREATE PROCEDURE AddStudiesToCart
    @StudentID INT,
    @StudiesID INT
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON; -- Auto-rollback on errors

    DECLARE @Vacancies INT,
            @CartID INT,
            @StudiesStartDate DATETIME;

    BEGIN TRANSACTION;

    -- Check study existence with lock
    SELECT @StudiesStartDate = DATEADD(DAY, -3, Date)
    FROM Studies WITH (UPDLOCK)
    WHERE StudiesID = @StudiesID;

    IF @StudiesStartDate IS NULL
    BEGIN
        ROLLBACK;
        THROW 51000, 'Invalid StudiesID provided.', 1;
    END;

    -- Check vacancies (uses UPDLOCK in function)
    SET @Vacancies = dbo.HowManyStudyVacancies(@StudiesID);

    IF @Vacancies <= 0
    BEGIN
        ROLLBACK;
        THROW 51001, 'No available vacancies for these studies.', 1;
    END;

    -- Insert new cart (CartID is auto-generated)
    INSERT INTO Cart (StudentID, Paid, PaidDate, DueDate, Access)
    VALUES (@StudentID, 0, NULL, @StudiesStartDate, 0);

    -- Get the auto-generated CartID
    SET @CartID = SCOPE_IDENTITY();
```

```
    -- Link studies to cart
    INSERT INTO CartStudies (CartID, StudiesID)
    VALUES (@CartID, @StudiesID);

    COMMIT TRANSACTION;
END;
```

Dodaj nowego tłumacza wraz z jego władanymi językami

(AddTranslatorWithLanguages)

```
CREATE PROCEDURE AddTranslatorWithLanguages
    @EmployeeID INT,
    @LanguageIDs NVARCHAR(MAX) -- Lista języków w formacie: '1,2,3'
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;

        -- Znalezienie pierwszego wolnego TranslatorID
        DECLARE @TranslatorID INT;
        SELECT @TranslatorID = ISNULL(MAX(TranslatorID), 0) + 1 FROM Translators;

        -- Dodanie nowego tłumacza do tabeli Translators
        INSERT INTO Translators (TranslatorID, EmployeeID)
        VALUES (@TranslatorID, @EmployeeID);

        -- Iteracja po ID języków i dodanie wpisów do TranslatorLanguages
        DECLARE @LanguageID INT;
        DECLARE @LanguageTable TABLE (ID INT);

        -- Rozdzielenie wartości z listy języków (np. '1,2,3') i zapisanie w tabeli tymczasowej
        INSERT INTO @LanguageTable (ID)
        SELECT value
        FROM STRING_SPLIT(@LanguageIDs, ',');

        -- Iteracja po językach i dodanie do TranslatorLanguages
        DECLARE cur CURSOR FOR
        SELECT ID FROM @LanguageTable;

        OPEN cur;
        FETCH NEXT FROM cur INTO @LanguageID;

        WHILE @@FETCH_STATUS = 0
        BEGIN
            INSERT INTO TranslatorLanguages (TranslatorID, LanguageID)
            VALUES (@TranslatorID, @LanguageID);

            FETCH NEXT FROM cur INTO @LanguageID;
        END;

        CLOSE cur;
```

```
        DEALLOCATE cur;

        -- Zatwierdzenie transakcji
        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        -- W przypadku błędu cofnięcie transakcji
        ROLLBACK TRANSACTION;

        -- Rzucenie komunikatu błędu
        THROW;
    END CATCH
END;
```

Dodaj webinar do koszyka (AddWebinarToCart)

```sql
CREATE PROCEDURE AddWebinarToCart
    @StudentID INT,
    @WebinarID INT
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON; -- Auto-rollback on errors

    DECLARE @CartID INT,
            @WebinarPrice SMALLMONEY,
            @Vacancies INT;

    BEGIN TRANSACTION;

    -- Check webinar existence and get price with lock
    SELECT @WebinarPrice = Price
    FROM Webinars WITH (UPDLOCK)
    WHERE WebinarID = @WebinarID;

    IF @WebinarPrice IS NULL
    BEGIN
        ROLLBACK;
        THROW 51000, 'Invalid WebinarID provided.', 1;
    END;

    -- Insert new cart (CartID is auto-generated)
    INSERT INTO Cart (StudentID, Paid, PaidDate, DueDate, Access)
    VALUES (
        @StudentID,
        0,
        NULL,  -- Changed from 0 to NULL for proper datetime handling
        DATEADD(DAY, 7, GETDATE()),
        CASE WHEN @WebinarPrice = 0 THEN 1 ELSE 0 END
    );

    -- Get auto-generated CartID
    SET @CartID = SCOPE_IDENTITY();

    -- Add webinar to cart
    INSERT INTO CartWebinars (CartID, WebinarID)
    VALUES (@CartID, @WebinarID);
```

```
    COMMIT TRANSACTION;
END;
```

Niepomyślna opłata części koszyka (FailedPartialPaymentForTheCart)

```
CREATE PROCEDURE FailedPartialPaymentForTheCart
    @CartID INT,
        @AmountPaid SMALLMONEY
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

                IF NOT EXISTS(SELECT CartID FROM Cart WHERE CartID=@CartID)
                BEGIN
            THROW 51000, 'Cart does not exist.', 1;
        END

        -- Create payment record
        INSERT INTO Payments (Value, CartID, IsSuccessful, CreatedAt)
        VALUES (@AmountPaid, @CartID, 0, GETDATE());


        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Niepomyślna opłata koszyka (FailedPaymentForTheCart)

```
CREATE PROCEDURE FailedPaymentForTheCart
    @CartID INT
AS
BEGIN
    SET NOCOUNT ON;
    SET XACT_ABORT ON;

    DECLARE @PaymentValue SMALLMONEY;

    BEGIN TRY
        BEGIN TRANSACTION;

        -- Verify cart exists and lock it
        SELECT @PaymentValue = dbo.GetRemainingPaymentForCart(@CartID)  -- Assuming Paid column
        FROM Cart WITH (UPDLOCK)
        WHERE CartID = @CartID;

                IF NOT EXISTS(SELECT CartID FROM Cart WHERE CartID=@CartID)
                BEGIN
            THROW 51000, 'Cart does not exist.', 1;
        END

        -- Create payment record
        INSERT INTO Payments (Value, CartID, IsSuccessful, CreatedAt)
        VALUES (@PaymentValue, @CartID, 0, GETDATE());

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0 ROLLBACK TRANSACTION;
        THROW;
    END CATCH
END;
```

Zapłać za cały koszyk (PayForWholeCart)

```
CREATE PROCEDURE PayForWholeCart
    @CartID INT
AS
BEGIN
    DECLARE @TotalValue SMALLMONEY;

    -- Pobierz pełną wartość koszyka
    SELECT @TotalValue = dbo.GetRemainingPaymentForCart(@CartID);

    -- Aktualizuj tabelę Cart: oznacz jako opłacone
    UPDATE Cart
    SET Paid = Paid + @TotalValue,
        PaidDate = GETDATE()
    WHERE CartID = @CartID;
END;
```

Zapłać częściowo za koszyk (PayPartialForCart)

```
CREATE PROCEDURE PayPartialForCart
    @CartID INT,
    @AmountPaid SMALLMONEY
AS
BEGIN
    DECLARE @RemainingPayment SMALLMONEY;

    -- Pobierz pozostałą kwotę do zapłaty
    SELECT @RemainingPayment = dbo.GetRemainingPaymentForCart(@CartID);

    -- Sprawdź, czy podana kwota nie przekracza pozostałej
    IF @AmountPaid > @RemainingPayment
    BEGIN
        SELECT @AmountPaid = @RemainingPayment
    END

    -- Dodaj do Paid nową płatność
    UPDATE Cart
    SET Paid = ISNULL(Paid, 0) + @AmountPaid,
        PaidDate = CASE
                     WHEN ISNULL(Paid, 0) + @AmountPaid = dbo.GetCartValue(@CartID) THEN GETDATI
                     ELSE PaidDate
                   END
    WHERE CartID = @CartID;
END;
```

Funkcje

Szczegóły koszyka dla danego studenta

(GetCartDetailsForStudent)

```
CREATE FUNCTION GetCartDetailsForStudent (@StudentID INT)
RETURNS TABLE
AS
RETURN
(
    SELECT
        c.CartID,
        c.Paid,
        c.PaidDate,
        c.DueDate,
        c.Access,
        cc.CountCourses AS CourseID,
        cs.CountStudies AS StudiesID,
        cw.CountWebinars AS WebinarID,
                cls.CountLessons AS SingleLessonID,
                dbo.GetCartValue(c.CartID) as CartValue,
                ISNULL(ISNULL(ISNULL(cc.Name, cs.Name), cw.Name), cls.Name) AS NameOfEvent,
                ISNULL(ISNULL(ISNULL(cc.Date, cs.Date), cw.Date), cls.Date) AS DateOfEvent
    FROM
        Cart c
    LEFT JOIN
        (SELECT CartID, CoursesID AS CountCourses, Name, dbo.GetEarliestModuleDate(CoursesID) AS
                INNER JOIN Courses c ON cc.CoursesID = c.CourseID
                GROUP BY CartID, CoursesID, Name) cc
        ON c.CartID = cc.CartID
    LEFT JOIN
        (SELECT CartID, s.StudiesID AS CountStudies, Name, Date FROM CartStudies cs
                INNER JOIN Studies s ON cs.StudiesID = s.StudiesID
                GROUP BY CartID, s.StudiesID, Name, Date) cs
        ON c.CartID = cs.CartID
    LEFT JOIN
        (SELECT CartID, w.WebinarID AS CountWebinars, Name, Date FROM CartWebinars cw
                INNER JOIN Webinars w ON w.WebinarID = cw.WebinarID
                GROUP BY CartID, w.WebinarID, Name, Date) cw
        ON c.CartID = cw.CartID
        LEFT JOIN
                (SELECT CartID, csl.LessonID AS CountLessons, CONCAT( 'Single lesson in ', s.Nar
                INNER JOIN LessonsOnStudies los ON csl.LessonID = los.LessonID
                INNER JOIN Conferrence cf ON los.ConferrenceID = cf.ConferrenceID
                INNER JOIN Studies s ON cf.StudiesID = s.StudiesID
                GROUP BY CartID, csl.LessonID, s.Name, los.Date) cls
```

```
                ON c.CartID = cls.CartID
    WHERE
        c.StudentID = @StudentID
);
```

Sprawdzanie czy student zdał egzaminy (CheckIfPassedExams)

```
CREATE FUNCTION CheckIfPassedExams(
    @StudentID INT,
    @StudiesID INT
)
RETURNS bit
AS
BEGIN

        IF EXISTS (
                SELECT * FROM Studies s
                INNER JOIN Conferrence c ON s.StudiesID = c.StudiesID AND s.StudiesID = @Studies
                LEFT OUTER JOIN Exams e ON c.ConferrenceID = e.ConferrenceID AND e.StudentID = @
                WHERE ISNULL(e.Grade, 0) <= 2
        ) BEGIN
                RETURN 0;
        END;
        RETURN 1;
END;
```

Sprawdzanie czy student zdaje kurs (CheckPassingForCourse)

```
CREATE FUNCTION CheckPassingForCourse(
    @StudentID INT,
    @CourseID INT
)
RETURNS bit
AS
BEGIN
    -- Sprawdzenie, czy student ma dostęp do kursu
    IF NOT EXISTS (
        SELECT 1
        FROM CartCourses AS cc
        JOIN Cart AS c ON cc.CartID = c.CartID
        WHERE c.StudentID = @StudentID AND cc.CoursesID = @CourseID AND c.Access = 1
    )
    BEGIN
        RETURN 0; -- Brak dostępu oznacza brak możliwości zaliczenia
    END;

    -- Sprawdzenie, czy frekwencja >= 80%
    IF [dbo].[GetCourseAttendanceForStudent](@StudentID, @CourseID) >= 0.80
    BEGIN
        RETURN 1; -- Zaliczenie
    END;

    RETURN 0; -- Brak zaliczenia
END;
```

Sprawdzanie czy student zdaje studia (CheckPassingForStudies)

```
CREATE FUNCTION CheckPassingForStudies(
    @StudentID INT,
    @StudiesID INT
)
RETURNS bit
AS
BEGIN
    -- Sprawdzenie, czy student ma dostęp do kursu
    IF NOT EXISTS (
        SELECT 1
        FROM CartStudies AS cs
        JOIN Cart AS c ON cs.CartID = c.CartID
        WHERE c.StudentID = @StudentID AND cs.StudiesID = @StudiesID AND c.Access = 1
    )
    BEGIN
        RETURN 0; -- Brak dostępu oznacza brak możliwości zaliczenia
    END;

    -- Sprawdzenie, czy zdane praktyki i egzaminy oraz czy frekwencja dla każdego zjazdu >= 80%
    IF ([dbo].[CheckStudentInternships](@StudentID, @StudiesID) = 1)
        AND ([dbo].[CheckIfPassedExams](@StudentID, @StudiesID) = 1)
        AND NOT EXISTS(
                SELECT *
                FROM Conferrence
                WHERE StudiesID = @StudiesID AND [dbo].[GetConferrenceAttendanceForStudent](@Stu
        )
    BEGIN
        RETURN 1; -- Zaliczenie
    END;

    RETURN 0; -- Brak zaliczenia
END;
```

Sprawdzanie staży studenta (CheckStudentInternships)

```
CREATE FUNCTION CheckStudentInternships(@StudentID int, @StudiesID int)
RETURNS bit
AS
BEGIN
    DECLARE @Result bit = 1
-- Sprawdzamy, czy student o danym ID ma nieobecność na jakichkolwiek zakończonych praktykach
    IF EXISTS (
                SELECT 1 FROM InternshipAbsence ia
                INNER JOIN Internships i ON ia.InternshipID = i.InternshipID AND i.StudiesID = @
                WHERE StudentID = @StudentID AND DATEADD(Day, 14, DATE) < GETDATE())
    BEGIN
        SET @Result = 0
    END

    RETURN @Result
END
```

Sprawdzanie jakimi językami włada tłumacz (CheckTranslatorLanguage)

```
CREATE FUNCTION CheckTranslatorLanguage
(@TranslatorID int null, @LanguageID int null)
RETURNS bit AS
BEGIN
    IF @TranslatorID IS NOT NULL AND NOT EXISTS (SELECT * FROM Translators WHERE TranslatorID =
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @LanguageID IS NOT NULL AND NOT EXISTS (SELECT * FROM Languages WHERE LanguageID = @Langu
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @TranslatorID IS NULL AND @LanguageID IS NOT NULL
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @TranslatorID IS NOT NULL AND @LanguageID IS NULL
    BEGIN
        RETURN CAST(0 AS bit)
    END

    IF @TranslatorID IS NOT NULL AND @LanguageID IS NOT NULL AND NOT EXISTS (
        SELECT * FROM TranslatorLanguages WHERE TranslatorID = @TranslatorID AND LanguageID = @L
    BEGIN
        RETURN CAST(0 AS bit)
    END

    RETURN CAST(1 AS bit)
END
```

Sprawdzanie wartości danego koszyka (GetCartValue)

```
CREATE FUNCTION GetCartValue(@CartID int)
RETURNS money
AS
BEGIN
    DECLARE @StudiesSum money
    DECLARE @StudyMeetingsSum money
    DECLARE @CoursesSum money
    DECLARE @WebinarsSum money

    SELECT @StudiesSum = ISNULL(SUM(s.Cost), 0)
    FROM Studies AS s
    JOIN CartStudies AS cs ON s.StudiesID = cs.StudiesID
    WHERE cs.CartID = @CartID

    SELECT @StudyMeetingsSum = ISNULL(SUM(ls.Cost), 0)
    FROM LessonsOnStudies AS ls
        JOIN CartSingleLessons as cl ON ls.LessonID = cl.LessonID
    WHERE cl.CartID = @CartID

    SELECT @CoursesSum = ISNULL(SUM(c.Price), 0)
    FROM Courses AS c
    JOIN CartCourses AS cc ON cc.CoursesID = c.CourseID
    WHERE cc.CartID = @CartID

    SELECT @WebinarsSum = ISNULL(SUM(w.Price), 0)
    FROM Webinars AS w
    JOIN CartWebinars AS cw ON w.WebinarID = cw.WebinarID
    WHERE cw.CartID = @CartID

    RETURN @StudiesSum + @CoursesSum + @WebinarsSum + @StudyMeetingsSum
END
```

Zliczanie frekwencji na danym przedmiocie (GetConferrenceAttendanceForStudent)

```
CREATE FUNCTION GetConferrenceAttendanceForStudent(@StudentID int, @ConferrenceID int)
RETURNS FLOAT
AS
BEGIN
    -- Sprawdzenie, czy student istnieje czy kurs istnieje
    IF NOT EXISTS (
        SELECT 1
        FROM Students
        WHERE StudentID = @StudentID
    ) OR NOT EXISTS (
        SELECT 1
        FROM Conferrence
        WHERE ConferrenceID = @ConferrenceID
    )
    BEGIN
        RETURN 0;
    END;

    RETURN ISNULL((
                SELECT AVG(CAST(Presence AS float))
                FROM LessonsOnStudies los
                INNER JOIN StudyLessonPresence slp ON los.LessonID = slp.LessonID AND los.Confe
                WHERE StudentID = @StudentID
        ), 0)
END;
```

Zliczanie frekwencji na danym kursie (GetCourseAttendanceForStudent)

```
CREATE FUNCTION GetCourseAttendanceForStudent(
    @StudentID INT,
    @CourseID INT
)
RETURNS FLOAT
AS
BEGIN

    -- Sprawdzenie, czy student istnieje, czy kurs istnieje, i czy student ma dostęp
    IF NOT EXISTS (
        SELECT 1
        FROM Students
        WHERE StudentID = @StudentID
    ) OR NOT EXISTS (
        SELECT 1
        FROM Courses
        WHERE CourseID = @CourseID
    )
    BEGIN
        RETURN 0;
    END;

        RETURN ISNULL((
                SELECT AVG(CAST(Presence AS float))
                FROM ModulesPresence mp
                JOIN CourseModules cm ON mp.ModuleID = cm.ModuleID AND CourseID = @CourseID
                WHERE mp.StudentID = @StudentID
        ), 0)
END;
```

Sprawdzanie daty najbliższego modułu w kursie (GetEarliestModuleDate)

```
CREATE FUNCTION GetEarliestModuleDate(@CourseID INT)
RETURNS DATETIME
AS
BEGIN
    DECLARE @EarliestDate DATETIME;

    SELECT @EarliestDate = MIN(Date)
    FROM CourseModules
    WHERE CourseID = @CourseID;

    RETURN @EarliestDate;
END;
```

Sprawdź maksymalną pojemność kursu (GetMaxCourseCapacity)

```
CREATE FUNCTION GetMaxCourseCapacity(@CourseID int)
RETURNS int
AS
BEGIN
    DECLARE @MaxCapacity int;

    SELECT @MaxCapacity = MIN(sm.Limit)
    FROM StationaryModules sm
    INNER JOIN CourseModules cm ON sm.ModuleID = cm.ModuleID
    WHERE cm.CourseID = @CourseID;

    RETURN @MaxCapacity -- if there are no stationary meetings there is no limit so function ret
END;
```

Sprawdź maksymalną pojemność studiów (GetMaxStudyCapacity)

```
CREATE FUNCTION GetMaxStudyCapacity(@StudiesID int)
RETURNS int
AS
BEGIN
    DECLARE @MaxCapacity int;

    SELECT @MaxCapacity = MIN(Limit)
    FROM StationaryStudy ss
    INNER JOIN LessonsOnStudies ls ON ss.LessonID = ls.LessonID
    INNER JOIN Conferrence c ON ls.ConferrenceID = c.ConferrenceID
    WHERE c.StudiesID = @StudiesID;

    RETURN @MaxCapacity -- if there are no stationary meetings there is no limit so function ret
END;
```

Sprawdzanie ile należy dopłacić do koszyka (GetRemainingPaymentForCart)

```
CREATE FUNCTION GetRemainingPaymentForCart(@CartID int)
RETURNS money
AS
BEGIN



    RETURN dbo.GetCartValue(@CartID) - (
        SELECT ISNULL(Paid, 0)
        FROM Cart
        WHERE CartID = @CartID
        );
END
```

Sprawdzanie ilości wolnych miejsc na kursie (HowManyCourseVacancies)

```sql
CREATE FUNCTION HowManyCourseVacancies(@CourseID int)
RETURNS int
AS
BEGIN
    DECLARE @MaximumCapacity int;
    SELECT @MaximumCapacity = dbo.GetMaxCourseCapacity(@CourseID);

    IF @MaximumCapacity IS NULL
    BEGIN
        RETURN @MaximumCapacity
    END

    DECLARE @CurrentCapacity int;
    SELECT @CurrentCapacity = COUNT(*)
    FROM Students
    WHERE StudentID IN (
        SELECT s.StudentID
        FROM Students s
        JOIN Cart c WITH (UPDLOCK) ON s.StudentID = c.StudentID  -- Lock cart entries
        JOIN CartCourses cs WITH (UPDLOCK) ON c.CartID = cs.CartID  -- Lock course assignments
        WHERE cs.CoursesID = @CourseID
    );

    RETURN @MaximumCapacity - @CurrentCapacity
END
```

Sprawdzanie ilości wolnych miejsc na lekcji

(HowManyLessonVacancies)

```sql
CREATE FUNCTION HowManyLessonVacancies (@LessonID int)
RETURNS int
AS
BEGIN
    IF NOT EXISTS(
        SELECT *
        FROM StationaryStudy WITH (UPDLOCK) -- Lock lesson metadata
        WHERE LessonID = @LessonID
    )
    BEGIN
        RETURN NULL;
    END

    DECLARE @Limit int;
    SELECT @Limit = MIN(Limit)
    FROM StationaryStudy WITH (UPDLOCK) -- Maintain lock
    WHERE LessonID = @LessonID;

    DECLARE @StudiesID int;
    SELECT @StudiesID = MIN(StudiesID)
    FROM LessonsOnStudies ls WITH (UPDLOCK) -- Lock lesson-study association
    JOIN Conferrence c WITH (UPDLOCK) ON ls.ConferrenceID = c.ConferrenceID -- Lock conference
    WHERE ls.LessonID = @LessonID;

    DECLARE @CurrentCapacity int;
    SELECT @CurrentCapacity = COUNT(*)
    FROM Students
    WHERE StudentID IN (
        SELECT s.StudentID
        FROM Students s
        JOIN Cart AS c WITH (UPDLOCK) ON s.StudentID = c.StudentID -- Lock cart
        JOIN CartStudies cs WITH (UPDLOCK) ON c.CartID = cs.CartID -- Lock study assignments
        WHERE cs.StudiesID = @StudiesID
    );

    SELECT @CurrentCapacity += ISNULL(COUNT(*), 0)
    FROM CartSingleLessons cl WITH (UPDLOCK) -- Lock single-lesson entries
    JOIN Cart c WITH (UPDLOCK) ON cl.CartID = c.CartID -- Lock cart
    WHERE cl.LessonID = @LessonID AND c.Access = 1;
```

```
    RETURN @Limit - @CurrentCapacity;
END;
```

Sprawdzanie ilości wolnych miejsc na studiach (HowManyStudyVacancies)

```
CREATE FUNCTION HowManyStudyVacancies(@StudiesID int)
RETURNS int
AS
BEGIN
    DECLARE @MaximumCapacity int;
    SELECT @MaximumCapacity = dbo.GetMaxStudyCapacity(@StudiesID);

    IF @MaximumCapacity IS NULL
    BEGIN
        RETURN @MaximumCapacity
    END

    DECLARE @CurrentCapacity int;
    SELECT @CurrentCapacity = COUNT(*)
        FROM Students
        WHERE StudentID IN (
            SELECT s.StudentID
            FROM Students s
            JOIN Cart AS c WITH (UPDLOCK) ON s.StudentID = c.StudentID
            JOIN CartStudies cs WITH (UPDLOCK) ON c.CartID = cs.CartID
            WHERE cs.StudiesID = @StudiesID
        );

    RETURN @MaximumCapacity - @CurrentCapacity
END
```

Triggery

Dawanie dostępu studentom, którzy opłacili zamówienie (trg_SetAccessAfterPayment)

```
CREATE TRIGGER trg_SetAccessAfterPayment
ON Cart
AFTER INSERT, UPDATE
AS
BEGIN
  UPDATE Cart
  SET Access = 1
  WHERE CartID IN (
    SELECT CartID
    FROM inserted
    WHERE Paid = dbo.GetCartValue(CartID)
  ) AND Access = 0;
END;
```

Aktualizacja obecności (trg_UpdatePresence)

```
CREATE TRIGGER trg_UpdatePresence
ON StudyLessonPresence
AFTER INSERT, UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    -- Aktualizacja wartości Presence na 1, gdy MakeUpDate nie jest NULL
    UPDATE SLP
    SET Presence = 1
    FROM StudyLessonPresence SLP
    INNER JOIN inserted i
        ON SLP.StudentID = i.StudentID
        AND SLP.LessonID = i.LessonID
    WHERE i.MakeUpDate IS NOT NULL;
END;
```

4. Inne

Indeksy

Tabela Students

```
CREATE UNIQUE INDEX idx_students_student_id ON Students(StudentID);
CREATE INDEX idx_students_lastname ON Students(LastName);
```

Tabela Cart

```
CREATE UNIQUE INDEX idx_cart_cart_id ON Cart(CartID);
CREATE INDEX idx_cart_student_id ON Cart(StudentID);
```

Tabela Courses

```
CREATE UNIQUE INDEX idx_courses_course_id ON Courses(CourseID);
CREATE INDEX idx_courses_name ON Courses(Name);
CREATE INDEX idx_courses_price ON Courses(Price);
```

Tabela Webinars

```
CREATE UNIQUE INDEX idx_webinars_webinar_id ON Webinars(WebinarID);
CREATE INDEX idx_webinars_teacher_id ON Webinars(TeacherID);
CREATE INDEX idx_webinars_date ON Webinars(Date);
CREATE INDEX idx_webinars_duration ON Webinars(Duration);
CREATE INDEX idx_webinars_price ON Webinars(Price);
```

Tabela LessonsOnStudies

```
CREATE UNIQUE INDEX idx_lessons_on_studies_lesson_id ON LessonsOnStudies(LessonID);
CREATE INDEX idx_lessons_on_studies_duration ON LessonsOnStudies(Duration);
```

Tabela CartCourses

```
CREATE UNIQUE INDEX idx_cart_courses_cart_id_course_id ON CartCourses(CartID, CoursesID);
```

Tabela Employees

```
CREATE UNIQUE INDEX idx_employees_employee_id ON Employees(EmployeeID);
CREATE INDEX idx_employees_lastname ON Employees(LastName);
```

Tabela Payments

```
CREATE UNIQUE INDEX idx_payments_payment_id ON Payments(PaymentID);
CREATE INDEX idx_payments_cart_id ON Payments(CartID);
CREATE INDEX idx_payments_created_at ON Payments(CreatedAt);
```

Tabela StudyLessonsPresence

```
CREATE INDEX idx_study_lessons_presence_student_id ON StudyLessonPresence(StudentID);
CREATE INDEX idx_study_lessons_presence_lesson_id ON StudyLessonPresence(LessonID);
```

Tabela Teachers

```
CREATE UNIQUE INDEX idx_teachers_teacher_id ON Teacher(TeacherID);
```

Tabela Languages

```
CREATE UNIQUE INDEX idx_languages_language_id ON Languages(LanguageID);
CREATE INDEX idx_languages_name ON Languages(LanguageName);
```

Tabela Internships

```
CREATE INDEX idx_internships_start_date ON Internships(StartDate);
```

Uprawnienia

Rola admin

```
CREATE ROLE admin
GRANT CONTROL ON DATABASE::[u_swierzy] TO admin;
```

Rola dyrektor

```
CREATE ROLE dyrektor
GRANT SELECT ON PRESENCE_ALL TO dyrektor
GRANT SELECT ON PRESENCE_STUDY_LESSONS TO dyrektor
GRANT SELECT ON PRESENCE_COURSE_MODULES TO dyrektor
GRANT SELECT ON UPCOMING_EVENTS_MONTH TO dyrektor
GRANT SELECT ON UPCOMING_STUDY_LESSONS_MONTH TO dyrektor
GRANT SELECT ON UPCOMING_COURSE_MODULES_MONTH TO dyrektor
GRANT SELECT ON UPCOMING_WEBINARS_MONTH TO dyrektor
GRANT SELECT ON DEBTORS TO dyrektor
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_EVENTS TO dyrektor
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_STUDY_LESSONS TO dyrektor
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_COURSE_MODULES TO dyrektor
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_WEBINARS TO dyrektor
GRANT SELECT ON ATTENDANCE TO dyrektor
GRANT SELECT ON ATTENDANCE_ON_STUDY_LESSONS TO dyrektor
GRANT SELECT ON ATTENDANCE_ON_COURSE_MODULES TO dyrektor
```

Rola administrator_szkoly

```
CREATE ROLE administrator_szkoly
GRANT SELECT ON FINANCIAL_REPORT TO administrator_szkoly
GRANT SELECT ON FINANCIAL_REPORT_WEBINARS TO administrator_szkoly
GRANT SELECT ON FINANCIAL_REPORT_COURSES TO administrator_szkoly
GRANT SELECT ON FINANCIAL_REPORT_STUDIES TO administrator_szkoly
GRANT SELECT ON UPCOMING_EVENTS_MONTH TO administrator_szkoly
GRANT SELECT ON UPCOMING_STUDY_LESSONS_MONTH TO administrator_szkoly
GRANT SELECT ON UPCOMING_COURSE_MODULES_MONTH TO administrator_szkoly
GRANT SELECT ON UPCOMING_WEBINARS_MONTH TO administrator_szkoly
GRANT SELECT ON DEBTORS TO administrator_szkoly
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_EVENTS TO administrator_szkoly
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_STUDY_LESSONS TO administrator_szkoly
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_COURSE_MODULES TO administrator_szkoly
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_WEBINARS TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON Students TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON Employees TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON Administrators TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON Principals TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON SchoolAdministrators TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON Teacher TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON Translators TO administrator_szkoly
GRANT SELECT, INSERT, UPDATE, DELETE ON Languages TO administrator_szkoly
GRANT EXECUTE ON AddTranslatorWithLanguages TO administrator_szkoly
```

Rola nauczyciel

```
CREATE ROLE nauczyciel
GRANT SELECT ON ATTENDANCE TO nauczyciel
GRANT SELECT ON ATTENDANCE_ON_STUDY_LESSONS TO nauczyciel
GRANT SELECT ON ATTENDANCE_ON_COURSE_MODULES TO nauczyciel
GRANT EXECUTE ON AddLessonPresence TO nauczyciel
GRANT EXECUTE ON AddModulePresence TO nauczyciel
GRANT EXECUTE ON AddInternshipAbsence TO nauczyciel
GRANT SELECT, INSERT, UPDATE, DELETE ON Webinars to nauczyciel
```

Rola kierownik_kursu

```
CREATE ROLE kierownik_kursu
GRANT nauczyciel TO kierownik_kursu
GRANT SELECT, INSERT, UPDATE, DELETE ON Courses TO kierownik_kursu
GRANT SELECT, INSERT, UPDATE, DELETE ON CourseModules TO kierownik_kursu
GRANT SELECT, INSERT, UPDATE, DELETE ON StationaryModules TO kierownik_kursu
GRANT SELECT, INSERT, UPDATE, DELETE ON OnlineAsynchronizeModules TO kierownik_kursu
GRANT SELECT, INSERT, UPDATE, DELETE ON OnlineSynchronizeModules TO kierownik_kursu
GRANT EXECUTE ON GetMaxCourseCapacity TO kierownik_kursu
GRANT EXECUTE ON HowManyCourseVacancies TO kierownik_kursu
GRANT EXECUTE ON GetEarliestModuleDate TO kierownik_kursu
GRANT EXECUTE ON GetCourseAttendanceForStudent TO kierownik_kursu
GRANT EXECUTE ON CheckPassingForCourse TO kierownik_kursu
```

Rola kierownik_przedmiotu

```
CREATE ROLE kierownik_przedmiotu
GRANT nauczyciel TO kierownik_przedmiotu
GRANT SELECT, INSERT, UPDATE, DELETE ON LessonsOnStudies TO kierownik_przedmiotu
GRANT SELECT, INSERT, UPDATE, DELETE ON StationaryStudy TO kierownik_przedmiotu
GRANT SELECT, INSERT, UPDATE, DELETE ON OnlineAsynchronizeStudy TO kierownik_przedmiotu
GRANT SELECT, INSERT, UPDATE, DELETE ON OnlineSynchronizeStudy TO kierownik_przedmiotu
GRANT EXECUTE ON HowManyLessonVacancies TO kierownik_przedmiotu
```

Rola kierownik_studiow

```
CREATE ROLE kierownik_studiow
GRANT kierownik_przedmiotu TO kierownik_studiow
GRANT SELECT, INSERT, UPDATE, DELETE ON Studies TO kierownik_studiow
GRANT SELECT, INSERT, UPDATE, DELETE ON Exams TO kierownik_studiow
GRANT SELECT, INSERT, UPDATE, DELETE ON Internships TO kierownik_studiow
GRANT EXECUTE ON GetMaxStudyCapacity TO kierownik_studiow
GRANT EXECUTE ON HowManyStudyVacancies TO kierownik_studiow
GRANT EXECUTE ON CheckPassingForStudies TO kierownik_studiow
GRANT EXECUTE ON CheckStudentInternships TO kierownik_studiow
```

Rola kierownik_praktyk

```
CREATE ROLE kierownik_praktyk
GRANT nauczyciel TO kierownik_praktyk
GRANT SELECT, INSERT, UPDATE, DELETE ON InternshipAbsence TO kierownik_praktyk
GRANT SELECT, INSERT, UPDATE, DELETE ON Internships TO kierownik_praktyk
GRANT EXECUTE ON CheckStudentInternships TO kierownik_praktyk
```

Rola tlumacz

```
CREATE ROLE tlumacz
GRANT SELECT ON UPCOMING_EVENTS_MONTH TO tlumacz
GRANT SELECT ON Webinars TO tlumacz
GRANT SELECT ON LessonsOnStudies TO tlumacz
GRANT SELECT ON CourseModules TO tlumacz
```

Rola student

```
CREATE ROLE student
GRANT SELECT ON PEOPLE_SIGNED_FOR_FUTURE_EVENTS TO student
GRANT SELECT ON UPCOMING_EVENTS_MONTH TO student
GRANT EXECUTE ON CheckStudentInternships TO student
GRANT EXECUTE ON HowManyCourseVacancies TO student
GRANT EXECUTE ON GetMaxCourseCapacity TO student
GRANT EXECUTE ON HowManyStudyVacancies TO student
GRANT EXECUTE ON GetMaxStudyCapacity TO student
```

Komentarz końcowy

Dane do tabel w bazie zostały wygenerowane przy pomocy sztucznej inteligencji.
Dane wrażliwe typu numery telefonów, emaile itp. zostały wygenerowane przy pomocy własnoręcznie napisanych programów pomocniczych.
Wszystkie dane zostały ręcznie zweryfikowane.