Projekt z Przedmiotu
Podstawy Baz Danych

Przygotowali:
Klaudia Stodółkiewicz
Grzegorz Fido

Spis treści

WSTĘP

Niniejsza dokumentacja dotyczy bazy danych stworzonej w ramach projektu z przedmiotu Podstawy Baz Danych. Wykonano następujące elementy projektu:

1. **Opis funkcji systemu** – szczegółowo przedstawiono, jakie operacje mogą wykonywać poszczególni użytkownicy w systemie.
2. **Schemat bazy danych** – w postaci diagramu oraz opisu tabel. Każda tabela zawiera informacje o nazwach pól, typach danych, znaczeniu poszczególnych pól, a także warunkach integralności, które zostały zdefiniowane dla każdego pola. Do każdej tabeli dołączony jest również kod generujący tabelę.
3. **Spis widoków** – wraz z kodem, który je tworzy oraz szczegółowym opisem tego, co dany widok przedstawia.
4. **Spis procedur składowanych, triggerów i funkcji** – zawierający pełen kod oraz opis funkcji, triggerów i procedur składowanych, a także ich rolę i działanie w systemie.

Dokumentacja ta ma na celu przedstawienie stanu realizacji projektu.

FUNKCJE I UŻYTKOWNICY SYSTEMU

Użytkownicy bazy danych

- Dyrektor firmy
- Administrator bazy danych
- Pracownik sekretariatu
- Prowadzący zajęcia
- Koordynator przedmiotu
- Tłumacz
- Uczestnik (może nim być użytkownik uczestniczący w webinarze, kursant oraz student)
- Gość

Funkcje systemu

1. Konta i uprawnienia
    a. Gość:
        i. Założenie konta uczestnika
    b. Uczestnik:
        i. Usunięcie konta uczestnika
        ii. Ustawienie adresu korespondencyjnego w celu otrzymania dyplomu
    c. Pracownik sekretariatu:
        i. Założenie i dezaktywacja konta tłumacza
        ii. Założenie i dezaktywacja konta prowadzącego
        iii. Założenia i dezaktywacja konta koordynatora przedmiotu
    d. Dyrektor:
        i. Dodawanie i usuwanie pracownika sekretariatu
        ii. Dodawanie i usuwanie administratora
2. Zamówienia
    a. Uczestnik:
        i. Dodawanie webinaru, kursu lub studiów do koszyka
        ii. Usuwanie webinaru, kursu lub studiów z koszyka
        iii. Kupienie webinaru, kursu, studium
        iv. Uiszczenie opłaty za rejestrację na studia bądź za zjazd
    b. System:
        i. Wyliczanie wartości koszyka
        ii. Generowanie linku do płatności
        iii. Rejestracja udanej opłaty
        iv. Aktualizacja statusu zamówienia

3. Webinary
    a. Gość:
        i. Przeglądanie oferty webinarów
    b. Uczestnik:
        i. Dostęp do szczegółowych danych webinaru (gość po zapisie)
        ii. Dostęp do materiałów z webinaru (po uiszczeniu opłaty)
        iii. Ocena jakości webinaru przez uczestników
    c. Pracownik sekretariatu:
        i. Dodawanie, usuwanie webinarów z oferty
    d. Dyrektor:
        i. Umożliwienie dostępu bez opłaty
    e. Administrator:
        i. Usunięcie nagrania webinaru z platformy
    f. Tłumacz:
        i. Dodawanie tłumaczenia do webinaru
    g. Prowadzący:
        i. Modyfikacja danych webinaru
        ii. Przypisanie tłumacza do webinaru
        iii. Zarządzanie dostępem do nagrania webinaru
        iv. Przypisanie uczestników do konkretnych grup na webinarze
    h. System po uiszczeniu opłaty:
        i. Zapisanie uczestnika na webinar
4. Kursy
    a. Gość:
        i. Przeglądanie oferty kursów oraz jego modułów
    b. Uczestnik:
        i. Dostęp do szczegółowych danych kursu (gość po zapisie)
        ii. Dostęp do materiałów z kursu (po uiszczeniu opłaty)
        iii. Ocena jakości kursu przez uczestników
    c. Pracownik sekretariatu:
        i. Dodawanie kursu
        ii. Modyfikacja harmonogramu danego kursu
        iii. Generowanie dyplomów po zaliczeniu kursu
    d. Dyrektor:
        i. Umożliwienie dostępu bez opłaty
    e. Administrator:
        i. Usuwanie kursu
    f. Tłumacz:
        i. Dodawanie tłumaczenia
    g. Prowadzący:

i. Modyfikacja danych kursu
ii. Przypisanie tłumacza do kursu lub do jego modułu
iii. Przypisanie uczestników do konkretnych grup na kursie
iv. Dodawanie, modyfikacja i usuwanie modułów kursu
v. Sprawdzanie obecności na kursie

h. System (po uiszczeniu opłaty):
i. Zapis uczestnika na kurs
ii. Udzielenie zapisanemu uczestnikowi dostępu do kursu
iii. Wyliczanie wolnych miejsc na kursie (gdy jest hybrydowy)
iv. Weryfikacja obecności na kursie on-line asynchronicznym

5. Studia

a. Gość:
i. Przeglądanie oferty kierunków studiów oraz jego modułów

b. Uczestnik:
i. Dostęp do szczegółowych danych studiów (gość po zapisie)
ii. Dostęp do materiałów ze studiów (po uiszczeniu opłaty)
iii. Ocena jakości studiów przez uczestników

c. Pracownik sekretariatu:
i. Dodawanie kierunku studiów
ii. Modyfikacja informacji dotyczących danego kierunku studiów
iii. Modyfikacja harmonogramu spotkań
iv. Generowanie dyplomów po zaliczeniu kursu
v. Dodawanie przedmiotów
vi. Akceptacja zaliczonych przez studenta praktyk

d. Dyrektor:
i. Umożliwienie dostępu bez opłaty

e. Tłumacz:
i. Dodawanie tłumaczenia

f. Prowadzący:
i. Modyfikacja sylabusa (możliwa tylko przed rozpoczęciem danych studiów)
ii. Przypisanie tłumacza do zajęć prowadzonych w innym języku niż polski
iii. Dodawanie, modyfikacja i usuwanie modułów studiów

g. Koordynator przedmiotu:
i. Przypisanie uczestników do konkretnych grup
ii. Wstawianie zaliczeń z przedmiotów
iii. Akceptacja odrobionych nieobecności przez studenta

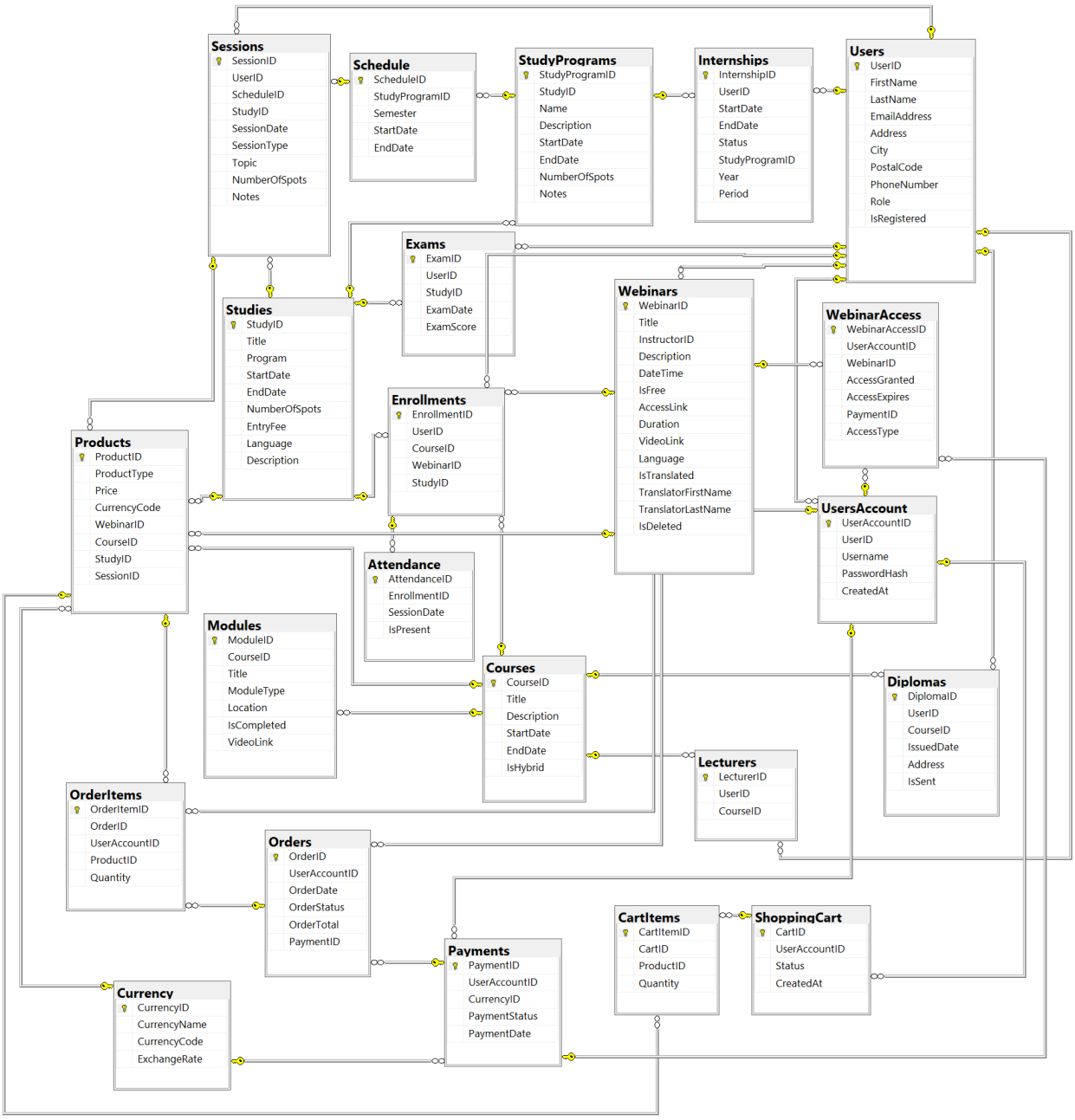
h. System po uiszczeniu opłaty:
i. Zapisanie uczestnika na studia
ii. Zapisanie na pojedyncze spotkanie nie będące częścią studium

iii. Wyliczanie maksymalnej ilości osób mogących brać udział w cyklu studiów
iv. Zapisanie uczestnika na zjazd (opłata dokonana najpóźniej 3 dni przed rozpoczęciem)

6. Raporty

a. Pracownik sekretariatu
   i. Zestawienie przychodów według webinarów, kursów i studiów – Tworzy raporty finansowe na podstawie zrealizowanych wydarzeń.
   ii. Lista osób z zaległymi płatnościami – Sporządza listę klientów, którzy nie opłacili pełnej kwoty za usługi.
   iii. Liczba zapisanych osób na przyszłe wydarzenia – Generuje raport z typem wydarzenia oraz formą (stacjonarnie/zdalnie).
   iv. Frekwencja na zakończonych wydarzeniach – Przygotowuje raporty o obecności uczestników na zakończonych spotkaniach.
   v. Lista obecności na każde wydarzenie – Zawiera daty, imiona, nazwiska i status obecności uczestników.
   vi. Lista uczestników zapisanych na kolidujące wydarzenia – Analizuje zapisy na przyszłe wydarzenia i wskazuje potencjalne konflikty.
   vii. Lista stałych klientów – Tworzy zestawienie klientów, którzy regularnie korzystają z usług.
b. Uczestnik
   i. Wykaz spotkań, na które uczestnik jest zapisany – Przegląda listę wydarzeń, w których ma wziąć udział.
c. Prowadzący:
   i. Wykaz prowadzonych wydarzeń – Otrzymuje listę wydarzeń, które prowadził w przeszłości lub będzie prowadził.
d. Tłumacz:
   i. Wykaz tłumaczonych wydarzeń – Przegląda listę wydarzeń, w których zapewniał tłumaczenie.
e. System:
   i. Automatyczne backupy – Regularnie wykonuje kopie zapasowe danych w ustalonych interwałach czasowych.
   ii. Wysyłanie przypomnień o drugiej racie – Automatycznie wysyła powiadomienia do klientów z zaległymi płatnościami.
f. Administrator:
   i. Backup na żądanie – Tworzy kopie zapasowe bazy danych na życzenie administratora.
   ii. Odtwarzanie bazy danych z backupów – Przywraca system do stanu zapisanego w kopii zapasowej.

SCHEMAT BAZY DANYCH



TABELE I WARUNKI INTEGRALNOŚCI

Users

Tabela przechowuje informacje o użytkownikach bazy danych, w tym ich dane kontaktowe i przypisane role. Każdy użytkownik posiada unikalny adres e-mail, a jego numer telefonu musi składać się z dokładnie 9 cyfr. Użytkownicy mają przypisane role w systemie, a także informacje o stanie rejestracji.

Opis pól tabeli:

1. UserID `INT`
    - Unikalny identyfikator użytkownika. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. FirstName `NVARCHAR(100)`
    - Imię użytkownika. Pole jest obowiązkowe.
    - Warunki integralności: `NOT NULL`.
3. LastName `NVARCHAR(100)`
    - Nazwisko użytkownika. Pole jest obowiązkowe.
    - Warunki integralności: `NOT NULL`.
4. EmailAddress `NVARCHAR(150)`
    - Unikalny adres e-mail użytkownika. Musi być w poprawnym formacie e-mail.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - `UNIQUE` – adres musi być unikalny.
        - Warunek `CHECK` sprawdzający format: `LIKE '%_@_%._%'`.
5. Address `NVARCHAR(100)`
    - Adres korespondencyjny użytkownika. Pole opcjonalne.
6. City `NVARCHAR(100)`
    - Miasto użytkownika. Pole opcjonalne.
7. PostalCode `NVARCHAR(8)`
    - Kod pocztowy użytkownika w formacie `XX-XXX`.
    - Warunki integralności: Warunek `CHECK` sprawdzający format: `LIKE '[0-9][0-9]-[0-9][0-9][0-9]'`.
8. PhoneNumber `VARCHAR(9)`
    - Numer telefonu użytkownika. Musi mieć dokładnie 9 cyfr.
    - Warunki integralności:
        - Warunek `CHECK` sprawdzający długość: `LEN(PhoneNumber) = 9`.

        - Warunek `CHECK` sprawdzający, czy zawiera tylko cyfry: `PhoneNumber NOT LIKE '%[^0-9]%'`.
        - Pole opcjonalne (`NULL`).
9. Role `NVARCHAR(50)`
    - Rola przypisana użytkownikowi (np. Administrator, Student, Guest).
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` ograniczający wartości do predefiniowanego zestawu ról.
10. IsRegistered `BIT`
    - Flaga oznaczająca, czy użytkownik jest zarejestrowany w systemie. Domyślna wartość to 0.
    - Warunki integralności:
        - `NOT NULL`.
        - 1, jeżeli UserID znajduje się w tabeli UserAccount, 0, jeżeli się tam nie znajduje

Kod generujący tabelę:

```
CREATE TABLE Users (
    UserID INT IDENTITY(1,1) PRIMARY KEY,
    FirstName NVARCHAR(100) NOT NULL,
    LastName NVARCHAR(100) NOT NULL,
    EmailAddress NVARCHAR(150) NOT NULL UNIQUE,
    Address NVARCHAR(100) NULL,
    City NVARCHAR(100) NULL,
    PostalCode NVARCHAR(8) NULL,
    PhoneNumber VARCHAR(9) NULL,
    Role NVARCHAR(50) NOT NULL CHECK (Role IN ('Administrator',
'Instructor', 'Coordinator', 'Student', 'Registrar', 'Translator',
'Director', 'Guest')),
    IsRegistered BIT NOT NULL DEFAULT 0,

    -- Warunki integralności
    CONSTRAINT unique_email UNIQUE (EmailAddress),
    CONSTRAINT users_valid_email CHECK (EmailAddress LIKE
'%_@_%._%'),
    CONSTRAINT check_phone_length CHECK (LEN(PhoneNumber) = 9),
    CONSTRAINT check_phone_digits CHECK (PhoneNumber IS NULL OR
PhoneNumber NOT LIKE '%[^0-9]%'),
```

```
    CONSTRAINT check_postal_code CHECK (PostalCode LIKE
'[0-9][0-9]-[0-9][0-9][0-9]'),
);
```

UsersAccount

Tabela przechowuje dane dotyczące kont użytkowników. Każde konto jest powiązane z użytkownikiem w tabeli Users poprzez klucz obcy. Nazwa użytkownika musi być unikalna i spełniać określone kryteria długości. Hasło przechowywane jest jako hash i musi spełniać wymagania dotyczące bezpieczeństwa. Data utworzenia konta nie może być w przyszłości.

Opis pól tabeli:

1. UserAccountID `INT`
    - Unikalny identyfikator konta użytkownika. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. UserID `INT`
    - Identyfikator użytkownika powiązanego z kontem.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do kolumny `UserID` w tabeli Users.
3. Username `VARCHAR(50)`
    - Unikalna nazwa użytkownika.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - `UNIQUE` – nazwa musi być unikalna.
        - Warunek `CHECK` sprawdzający, czy nazwa nie jest pusta (`<> ''`).
        - Warunek `CHECK` sprawdzający minimalną długość: `LEN(Username) >= 3`.
4. PasswordHash `VARCHAR(255)`
    - Hash hasła użytkownika. Musi spełniać wymagania dotyczące bezpieczeństwa (co najmniej jedna wielka litera, jedna cyfra, jeden znak specjalny).
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` sprawdzający format hasła z użyciem `PATINDEX`.
5. CreatedAt `DATETIME`
    - Data i czas utworzenia konta. Musi być poprawną datą i czasem, które nie są w przyszłości.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.

- Warunek `CHECK`: `CreatedAt <= GETDATE()`.

Kod generujący tabelę:

```
CREATE TABLE UsersAccount (
    UserAccountID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    UserID INT NOT NULL,
    Username VARCHAR(50) NOT NULL,
    PasswordHash VARCHAR(255) NOT NULL,
    CreatedAt DATETIME NOT NULL,

    -- Warunki integralności
    CONSTRAINT unique_username UNIQUE (Username),
    CONSTRAINT check_username_not_empty CHECK (Username <> ''),
    CONSTRAINT check_username_length CHECK (LEN(Username) >= 3),
    CONSTRAINT check_passwordhash_not_empty CHECK (PasswordHash <>
''),
    CONSTRAINT check_created_at CHECK (CreatedAt <= GETDATE()),

    CONSTRAINT fk_user_account FOREIGN KEY (UserID) REFERENCES
Users(UserID)
        ON DELETE CASCADE,

    -- Walidacja hasła
    CONSTRAINT check_password_format CHECK (
        PATINDEX('%[A-Z]%', PasswordHash) > 0 AND
        PATINDEX('%[0-9]%', PasswordHash) > 0 AND
        PATINDEX('%[^A-Za-z0-9]%', PasswordHash) > 0
    )
);
```

Webinars

Tabela przechowuje informacje dotyczące webinarów prowadzonych przez firmę. Każdy webinar ma unikalny tytuł, prowadzącego, datę i czas wydarzenia, a także informacje o tym, czy jest darmowy lub tłumaczony. Dodatkowo, dla webinarów tłumaczonych wymagane są dane tłumacza, a dla płatnych webinarów konieczne jest określenie ceny.

Opis pól tabeli:

1. WebinarID `INT`
    - Unikalny identyfikator webinaru. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. Title `NVARCHAR(255)`
    - Unikalny tytuł webinaru.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - `UNIQUE` – tytuł musi być unikalny.
3. InstructorID `INT`
    - Identyfikator prowadzącego webinar, odwołujący się do tabeli Users.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do kolumny `UserID` w tabeli Users.
4. Description `NVARCHAR(1000)`
    - Opcjonalny opis webinaru.
5. DateTime `DATETIME`
    - Data i godzina rozpoczęcia webinaru.
    - Warunki integralności: `NOT NULL`.
6. IsFree `BIT`
    - Flaga wskazująca, czy webinar jest darmowy (`1` – darmowy, `0` – płatny).
    - Warunki integralności: `NOT NULL`.
7. AccessLink `NVARCHAR(500)`
    - Opcjonalny link dostępu do webinaru.
    - Link dostępu do webinaru może być dostępny tylko wtedy, gdy obecna data i godzina (`GETDATE()`) jest wcześniejsza niż data i godzina webinaru (`DateTime`) o maksymalnie 10 minut.
8. Duration `INT`

    - Czas trwania webinaru w minutach. Musi być większy niż 0.
    - Warunki integralności: Warunek `CHECK`: `Duration > 0`.
9. VideoLink `NVARCHAR(500)`
    - Opcjonalny link do nagrania webinaru.
    - Link do nagrania webinaru może być dostępny tylko wtedy, gdy obecna data i godzina (`GETDATE()`) jest późniejsza niż data i godzina webinaru (`DateTime`).
10. Language `NVARCHAR(50)`
    - Język, w którym prowadzony jest webinar.
    - Warunki integralności: `NOT NULL`.
11. IsTranslated `BIT`
    - Flaga wskazująca, czy webinar jest tłumaczony (`1` – tłumaczony, `0` – nietłumaczony).
    - Warunki integralności: `NOT NULL`, domyślna wartość to `0`.
12. TranslatorFirstName `NVARCHAR(100)`
    - Imię tłumacza webinaru.
    - Warunki integralności:
        - Jeśli `IsTranslated = 0`, pole musi być puste (`NULL`).
        - Jeśli `IsTranslated = 1`, pole musi być uzupełnione.
13. TranslatorLastName `NVARCHAR(100)`
    - Nazwisko tłumacza webinaru.
    - Warunki integralności:
        - Jeśli `IsTranslated = 0`, pole musi być puste (`NULL`).
        - Jeśli `IsTranslated = 1`, pole musi być uzupełnione.
14. IsDeleted `BIT`
    - Flaga oznaczająca, czy webinar został usunięty. Domyślnie ustawiona na `0` (nieusunięty).
    - Warunki integralności: `NOT NULL`.

Kod generujący tabelę:

```
CREATE TABLE Webinars (
    WebinarID INT IDENTITY(1,1) PRIMARY KEY,
    Title NVARCHAR(255) NOT NULL,
    InstructorID INT NOT NULL,
    Description NVARCHAR(1000) NULL,
    DateTime DATETIME NOT NULL,
    IsFree BIT NOT NULL,
    AccessLink NVARCHAR(500) NULL,
```

```
    Duration INT NOT NULL CHECK (Duration > 0),
    VideoLink NVARCHAR(500) NULL,
    Language NVARCHAR(50) NOT NULL,
    IsTranslated BIT NOT NULL DEFAULT 0,
    TranslatorFirstName NVARCHAR(100) NULL,
    TranslatorLastName NVARCHAR(100) NULL,
    IsDeleted BIT NOT NULL DEFAULT 0,

    -- Warunki integralności
    CONSTRAINT fk_instructor FOREIGN KEY (InstructorID) REFERENCES
Users(UserID),
    CONSTRAINT unique_title UNIQUE (Title),
    CONSTRAINT chk_translator_data CHECK (
        (IsTranslated = 0 AND TranslatorFirstName IS NULL AND
TranslatorLastName IS NULL) OR
        (IsTranslated = 1 AND TranslatorFirstName IS NOT NULL AND
TranslatorLastName IS NOT NULL)
    ),
    CONSTRAINT chk_access_link_availability CHECK (
        AccessLink IS NULL OR
        GETDATE() <= DATEADD(MINUTE, -10, DateTime)
    ),
    CONSTRAINT chk_video_link_availability CHECK (
        VideoLink IS NULL OR
        GETDATE() > DateTime
    )
);
```

WebinarAccess

Tabela przechowuje informacje o dostępie użytkowników do webinarów. Każdy rekord zawiera dane dotyczące użytkownika, webinaru, dat dostępu i wygaszenia, a także szczegóły dotyczące płatności i typu dostępu.

Opis pól tabeli:

1. WebinarAccessID `INT`
    - Unikalny identyfikator dostępu do webinaru. Wartość generowana automatycznie.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. UserAccountID `INT`
    - Identyfikator konta użytkownika, który uzyskał dostęp do webinaru.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do kolumny `UserAccountID` w tabeli UsersAccount.
3. WebinarID `INT`
    - Identyfikator webinaru, do którego użytkownik ma dostęp.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do kolumny `WebinarID` w tabeli Webinars.
4. AccessGranted `DATETIME`
    - Data i czas, od którego użytkownik ma dostęp do webinaru.
    - Warunki integralności: `NOT NULL`.
5. AccessExpires `DATETIME`
    - Opis: Data i czas, po którym dostęp do webinaru wygasa. Musi być późniejsza niż data dostępu (`AccessGranted`).
    - Warunki integralności:
        - `NOT NULL`.
        - Warunek `CHECK`: `AccessGranted < AccessExpires`.
6. PaymentID `VARCHAR(20)`
    - Klucz obcy odnoszący się do tabeli Payments.
    - Warunki integralności:
        - Gdy użytkownik uzyska dostęp do webinaru, w przypadku płatnego dostępu, pole PaymentID wskazuje powiązaną płatność w tabeli Payments.
7. AccessType`VARCHAR(20)`

- Typ dostępu do webinaru. Możliwe wartości to:
  - `Free` – darmowy,
  - `Paid` – płatny.
- Warunki integralności:
  - `NOT NULL`.
  - Warunek `CHECK`: `AccessType IN ('Free', 'Paid')`.

Kod generujący tabelę:

```
CREATE TABLE WebinarAccess (
    WebinarAccessID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    UserAccountID INT NOT NULL,
    WebinarID INT NOT NULL,
    AccessGranted DATETIME NOT NULL,
    AccessExpires DATETIME NOT NULL,
    PaymentID INT,
    AccessType VARCHAR(20) NOT NULL,

    -- Warunki integralności
    CONSTRAINT fk_user_account_id FOREIGN KEY (UserAccountID)
REFERENCES UsersAccount(UserAccountID),
    CONSTRAINT fk_webinarid FOREIGN KEY (WebinarID) REFERENCES
Webinars(WebinarID),
    CONSTRAINT fk_paymentid FOREIGN KEY (PaymentID) REFERENCES
Payments(PaymentID),
    CONSTRAINT valid_access_type CHECK (AccessType IN ('Free',
'Paid')),
    CONSTRAINT valid_access_dates CHECK (AccessGranted <
AccessExpires),
    CONSTRAINT chk_webinar_payments CHECK (AccessType = 'Free' OR
PaymentID IS NOT NULL)
);
```

Courses

Tabela zawiera informacje o kursach oferowanych przez system. Kursy mają określony tytuł, opis, daty rozpoczęcia i zakończenia, cenę oraz informacje, czy są płatne i czy mają formę hybrydową.

Opis pól tabeli:

1. CourseID `INT`
    - Unikalny identyfikator kursu. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. Title `NVARCHAR(255)`
    - ytuł kursu. Pole obowiązkowe.
    - Warunki integralności: `NOT NULL`.
3. Description `NVARCHAR(MAX)`
    - Opis kursu. Pole opcjonalne.
4. StartDate `DATETIME`
    - Data rozpoczęcia kursu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` sprawdzający, czy data rozpoczęcia jest wcześniejsza niż data zakończenia: `StartDate < EndDate`.
5. EndDate `DATETIME`
    - Data zakończenia kursu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
6. IsHybrid `BIT`
    - Flaga wskazująca, czy kurs jest hybrydowy (połączenie nauki online i stacjonarnej).
    - Warunki integralności: `NOT NULL`.

Kod generujący tabelę:

```
CREATE TABLE Courses (
    CourseID INT IDENTITY(1,1) PRIMARY KEY,
    Title NVARCHAR(255) NOT NULL,
    Description NVARCHAR(MAX) NULL,
```

```
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NOT NULL,
    IsHybrid BIT NOT NULL,

    CONSTRAINT valid_course_dates CHECK (StartDate < EndDate),
    );
```

Modules

Tabela zawiera informacje o modułach przypisanych do kursów. Każdy moduł ma unikalny tytuł w obrębie kursu i jest powiązany z istniejącym kursem.

Opis pól tabeli:

1. ModuleID `INT`
    - Unikalny identyfikator modułu. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. CourseID `INT`
    - Identyfikator kursu, do którego moduł jest przypisany.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Courses i kolumny `CourseID`.
        - Jeżeli kurs zostanie usunięty, wszystkie powiązane z nim moduły również zostaną usunięte (zastosowanie `ON DELETE CASCADE`).
3. Title `NVARCHAR(255)`
    - Tytuł modułu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Każdy moduł w ramach tego samego kursu musi mieć unikalny tytuł..
4. ModuleType `NVARCHAR(50)`
    - Typ modułu.
    - Warunki integralności: `NOT NULL`.
5. Location `NVARCHAR(255)`
    - Miejsce, w którym odbywa się moduł
    - Warunki integralności: Pole opcjonalne (`NULL`).
6. IsCompleted `BIT`
    - Flaga wskazująca, czy moduł został ukończony.
    - Warunki integralności: `NOT NULL`.
7. VideoLink `NVARCHAR(500)`
    - Link do materiałów wideo związanych z modułem.
    - Warunki integralności: Pole opcjonalne (`NULL`).

Kod generujący tabelę:

```sql
CREATE TABLE Modules (
    ModuleID INT IDENTITY(1,1) PRIMARY KEY,
    CourseID INT NOT NULL,
    Title NVARCHAR(255) NOT NULL,
    ModuleType NVARCHAR(50) NOT NULL,
    Location NVARCHAR(255) NULL,
    IsCompleted BIT NOT NULL,
    VideoLink NVARCHAR(500) NULL,

    -- Warunki integralności
    CONSTRAINT fk_course FOREIGN KEY (CourseID) REFERENCES
Courses(CourseID) ON DELETE CASCADE,
    CONSTRAINT uq_course_title UNIQUE (CourseID, Title)
);
```

Lecturers

Tabela przechowuje informacje o wykładowcach przypisanych do kursów. Każdy wykładowca jest powiązany z użytkownikiem i konkretnym kursem.

Opis pól tabeli:

1. LecturerID `INT`
    - Unikalny identyfikator wykładowcy. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. UserID `INT`
    - Identyfikator użytkownika, który pełni rolę wykładowcy.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Users i kolumny `UserID`.
3. CourseID `INT`
    - Identyfikator kursu, do którego przypisany jest wykładowca.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Courses i kolumny `CourseID`.

Kod generujący tabelę:

```
CREATE TABLE Lecturers (
    LecturerID INT IDENTITY(1,1) PRIMARY KEY,
    UserID INT NOT NULL,
    CourseID INT NOT NULL,

    -- Warunki integralności
    CONSTRAINT fk_lecturers_userid FOREIGN KEY (UserID) REFERENCES
Users(UserID),
    CONSTRAINT fk_lecturers_courseid FOREIGN KEY (CourseID)
REFERENCES Courses(CourseID)
);
```

Studies

Tabela przechowuje informacje o programach studiów. Każde studium ma określony tytuł, daty rozpoczęcia i zakończenia, liczbę dostępnych miejsc, opłaty rekrutacyjne i czesne, a także domyślny język (polski).

Opis pól tabeli:

1. StudyID `INT`
    - Unikalny identyfikator studiów. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. Title `NVARCHAR(100)`
    - Kierunek studiów.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek unikalności (`UNIQUE`) zapewnia, że kierunek studiów będzie unikalny.
3. Program `TEXT`
    - Program studiów. Pole opcjonalne (`NULL`).
4. StartDate `DATETIME`
    - Data rozpoczęcia studiów.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` sprawdzający, że data rozpoczęcia jest wcześniejsza niż data zakończenia: `StartDate < EndDate`.
5. EndDate `DATETIME`
    - Data zakończenia studiów.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
6. NumberOfSpots `INT`
    - Liczba dostępnych miejsc na studia.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` zapewniający, że liczba dostępnych miejsc nie będzie mniejsza niż 0: `NumberOfSpots >= 0`.
7. EntryFee `DECIMAL(10,2)`
    - Opłata rekrutacyjna.

    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` zapewniający, że opłata rekrutacyjna nie będzie mniejsza niż 0: `EntryFee >= 0`.
8. Language `NVARCHAR(50)`
    - Język, w którym prowadzone są studia (domyślnie "polski").
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Domyślna wartość: `'polish'`.
9. Description `TEXT`
    - Opis studiów. Pole opcjonalne (`NULL`).

Kod generujący tabelę:

```
CREATE TABLE Studies (
    StudyID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    Title NVARCHAR(100) NOT NULL,
    Program TEXT,
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NOT NULL,
    NumberOfSpots INT NOT NULL CHECK (NumberOfSpots >= 0),
    EntryFee DECIMAL(10, 2) NOT NULL CHECK (EntryFee >= 0),
    Language NVARCHAR(50) NOT NULL DEFAULT 'polish',
    Description TEXT,

    -- Warunki integralności
    CONSTRAINT unique_studies_title UNIQUE (Title),
    CONSTRAINT valid_dates CHECK (StartDate < EndDate)
);
```

StudyPrograms

Tabela przechowuje informacje o programach studiów przypisanych do określonych kierunków studiów. Każdy program ma unikalną nazwę w obrębie kierunku, określoną datę rozpoczęcia i zakończenia, liczbę miejsc oraz dodatkowe uwagi.

Opis pól tabeli:

1. StudyProgramID `INT`
    - Unikalny identyfikator programu studiów. Wartość generowana automatycznie jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. StudyID `INT`
    - Identyfikator kierunku studiów, do którego przypisany jest program.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Studies i kolumny `StudyID`.
3. Name `NVARCHAR(255)`
    - Nazwa programu studiów.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Unikalność w obrębie kierunku studiów, zapewnia ją kombinacja StudyID i Name: `UNIQUE (StudyID, Name)`.
4. Description `NVARCHAR(1000)`
    - Opis programu studiów. Pole opcjonalne.
5. StartDate `DATETIME`
    - Data rozpoczęcia programu studiów.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` zapewniający, że data rozpoczęcia jest wcześniejsza niż data zakończenia: `StartDate < EndDate`.
6. EndDate `DATETIME`
    - Opis: Data zakończenia programu studiów.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
7. NumberOfSpots `INT`
    - Liczba dostępnych miejsc w programie studiów.

- Warunki integralności:
  - `NOT NULL` – pole obowiązkowe.
  - Warunek `CHECK` zapewniający, że liczba miejsc musi być większa od 0: `NumberOfSpots > 0`.

8. Notes `NVARCHAR(1000)`
   - Dodatkowe uwagi dotyczące programu studiów. Pole opcjonalne.

Kod generujący tabelę:

```
CREATE TABLE StudyPrograms (
    StudyProgramID INT IDENTITY(1, 1) PRIMARY KEY,
    StudyID INT NOT NULL,
    Name NVARCHAR(255) NOT NULL,
    Description NVARCHAR(1000) NULL,
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NOT NULL,
    NumberOfSpots INT NOT NULL CHECK (NumberOfSpots > 0),
    Notes NVARCHAR(1000) NULL,

    -- Warunki integralności
    CONSTRAINT fk_study_program FOREIGN KEY (StudyID) REFERENCES
Studies(StudyID),
    CONSTRAINT valid_program_dates CHECK (StartDate < EndDate),
    CONSTRAINT unique_program_title UNIQUE (StudyID, Name)
);
```

Schedule

Tabela przechowuje informacje o planach semestrów w ramach programów studiów. Każdy plan semestru jest powiązany z programem studiów i zawiera daty rozpoczęcia i zakończenia semestru.

Opis pól tabeli:

1. ScheduleID `INT`
    - Unikalny identyfikator planu semestru. Wartość generowana automatycznie jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. StudyProgramID `INT`
    - Identyfikator programu studiów, do którego przypisany jest plan semestru.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli StudyPrograms i kolumny `StudyProgramID`.
3. Semester `INT`
    - Numer semestru.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Brak dodatkowych warunków (np. zakresu wartości), jednak zazwyczaj warto dodać kontrolę w zależności od liczby semestrów w danym programie.
4. StartDate `DATETIME`
    - Data rozpoczęcia semestru.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
5. EndDate `DATETIME`
    - Data zakończenia semestru.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` zapewniający, że data rozpoczęcia semestru jest wcześniejsza niż data zakończenia: `StartDate < EndDate`.

Kod generujący tabelę:

```
CREATE TABLE Schedule (
```

```
    ScheduleID INT IDENTITY(1, 1) PRIMARY KEY,
    StudyProgramID INT NOT NULL,
    Semester INT NOT NULL,
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NOT NULL,

    -- Warunki integralności
    CONSTRAINT fk_programid FOREIGN KEY (StudyProgramID) REFERENCES
StudyPrograms(StudyProgramID),
    CONSTRAINT chk_valid_dates CHECK (StartDate < EndDate)
);
```

Sessions

Tabela przechowuje informacje o sesjach w ramach danego kierunku studiów. Każda sesja ma określoną datę, typ, temat, liczbę dostępnych miejsc oraz cenę zewnętrzną, wykładowcę i harmonogram.

Opis pól tabeli:

1. SessionID `INT`
    - Unikalny identyfikator sesji. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. UserID `INT`
    - Identyfikator użytkownika prowadzącego sesję.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Users i kolumny `UserID`.
3. ScheduleID `INT`
    - Identyfikator harmonogramu sesji.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Schedule i kolumny `ScheduleID`.
4. StudyID `INT`
    - Identyfikator kierunku studiów, w ramach których odbywa się sesja.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Studies i kolumny `StudyID`.
5. SessionDate `DATETIME`
    - Data sesji.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` zapewniający, że data jest większa niż `1900-01-01`: `SessionDate > '1900-01-01'`.
6. SessionType `VARCHAR(20)`
    - Typ sesji (możliwe wartości to: 'stationary', 'online', 'hybrid').

- Warunki integralności:
  - `NOT NULL` – pole obowiązkowe.
  - Warunek `CHECK` zapewniający, że wartość będzie jednym z dozwolonych typów sesji.

7. Topic `VARCHAR(255)`
   - Temat sesji.
   - Warunki integralności:
     - `NOT NULL` – pole obowiązkowe.
8. NumberOfSpots `INT`
   - Liczba dostępnych miejsc na sesję.
   - Warunki integralności:
     - `NOT NULL` – pole obowiązkowe.
     - Warunek `CHECK` zapewniający, że liczba miejsc jest większa lub równa 0 i nie przekracza liczby miejsc dostępnych w ramach studiów: `NumberOfSpots >= 0 AND NumberOfSpots <= (SELECT NumberOfSpots FROM Studies WHERE StudyID = Sessions.StudyID)`.
9. Notes `TEXT`
   - Dodatkowe uwagi dotyczące sesji. Pole opcjonalne (`NULL`).

Kod generujący tabelę:

```
CREATE TABLE Sessions (
    SessionID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    UserID INT NOT NULL,
    ScheduleID INT NOT NULL,
    StudyID INT NOT NULL,
    SessionDate DATETIME NOT NULL,
    SessionType VARCHAR(20) NOT NULL CHECK (SessionType IN
('stationary', 'online', 'hybrid')),
    Topic VARCHAR(255) NOT NULL,
    NumberOfSpots INT NOT NULL CHECK (NumberOfSpots >= 0),
    Notes TEXT NULL,


    -- Warunki integralności
    CONSTRAINT fk_session_userid FOREIGN KEY (UserID) REFERENCES
Users(UserID),
    CONSTRAINT fk_studyid FOREIGN KEY (StudyID) REFERENCES
```

```
Studies(StudyID),
    CONSTRAINT fk_session_scheduleid FOREIGN KEY (ScheduleID)
REFERENCES Schedule(ScheduleID),
    CONSTRAINT valid_sessiondate CHECK (SessionDate > '1900-01-01'),
);
```

Exams

Tabela przechowuje informacje o egzaminach, które są przypisane do użytkowników i określonych programów studiów. Każdy egzamin ma przypisaną datę, wynik oraz informacje o studiach, do których się odnosi.

Opis pól tabeli:

1. ExamID `INT`
    - Unikalny identyfikator egzaminu. Wartość generowana automatycznie jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. UserID `INT`
    - Identyfikator użytkownika, który przystępuje do egzaminu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Users i kolumny `UserID`.
3. StudyID `INT`
    - Identyfikator kierunku studiów, do którego przypisany jest egzamin.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Studies i kolumny `StudyID`.
4. ExamDate `DATETIME`
    - Data egzaminu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.

        - Warunek `CHECK` zapewniający, że data egzaminu jest po zakończeniu danego programu studiów.
5. ExamScore `DECIMAL(3, 2)`
    - Wynik egzaminu w przedziale od 0 do 5.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` zapewniający, że wynik mieści się w zakresie od 0 do 5: `ExamScore >= 0 AND ExamScore <= 5`.

Kod generujący tabelę:

```
CREATE TABLE Exams (
    ExamID INT NOT NULL IDENTITY(1, 1),
    UserID INT NOT NULL,
    StudyID INT NOT NULL,
    ExamDate DATETIME NOT NULL,
    ExamScore DECIMAL(3, 2) NOT NULL CHECK (ExamScore >= 0 AND
ExamScore <= 5),

    -- Warunki integralności
    CONSTRAINT exams_pk PRIMARY KEY (ExamID),
    CONSTRAINT fk_exams_userid FOREIGN KEY (UserID) REFERENCES
Users(UserID),
    CONSTRAINT fk_exams_studyid FOREIGN KEY (StudyID) REFERENCES
Studies(StudyID)
);
```

Internships

Tabela zawiera informacje na temat staży przypisanych do użytkowników w ramach określonych programów studiów. Każdy staż ma przypisaną datę rozpoczęcia i zakończenia, status, a także inne informacje organizacyjne.

Opis pól tabeli:

1. InternshipID `INT`
    - Unikalny identyfikator stażu. Jest generowany automatycznie
    - Warunki integralności:
        - `PRIMARY KEY` – zapewnia unikalność każdego rekordu w tabeli.
2. UserID `INT`
    - Identyfikator użytkownika, który jest przypisany do stażu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli Users, staż jest przypisany do istniejącego użytkownika.
3. StartDate `DATETIME`
    - Data rozpoczęcia stażu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - CHECK – `StartDate <= EndDate`, aby data rozpoczęcia nie była późniejsza niż data zakończenia stażu.
4. EndDate `DATETIME`
    - Data zakończenia stażu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - CHECK – `EndDate <= DATEADD(YEAR, 3, GETDATE())` – data zakończenia musi być w rozsądnych granicach (maksymalnie 3 lata w przyszłość).
5. Status `VARCHAR(20)`
    - Status stażu, który może przyjąć tylko dozwolone wartości.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Domyślnie ustawiony na `'In progress'` – staż jest tworzony z tym statusem.
        - CHECK – `Status IN ('In progress', 'Completed', 'Cancelled')` – zapewnia, że status jest jednym z dozwolonych.

6. StudyProgramID INT
    - Identyfikator programu studiów, do którego przypisany jest staż.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli StudyPrograms, co zapewnia, że staż jest przypisany do istniejącego programu studiów.
7. Year INT
    - Rok, w którym dany staż ma miejsce.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
8. Period INT
    - Okres stażu.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.

Kod generujący tabelę:

```
CREATE TABLE Internships (
    InternshipID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    UserID INT NOT NULL,
    StartDate DATETIME NOT NULL,
    EndDate DATETIME NOT NULL,
    Status VARCHAR(20) NOT NULL DEFAULT 'In progress',
    StudyProgramID INT NOT NULL,
    Year INT NOT NULL,
    Period INT NOT NULL,

    -- Warunki integralności
    CONSTRAINT fk_internship_userid FOREIGN KEY (UserID) REFERENCES
Users(UserID),
    CONSTRAINT fk_internship_studyprogramid FOREIGN KEY
(StudyProgramID) REFERENCES StudyPrograms(StudyProgramID),
    CONSTRAINT valid_internship_dates CHECK (StartDate <= EndDate),
    CONSTRAINT valid_end_date CHECK (EndDate <= DATEADD(YEAR, 3,
GETDATE())),
    CONSTRAINT valid_status CHECK (Status IN ('In progress',
'Completed', 'Cancelled')),
    CONSTRAINT unique_internship UNIQUE (StudyProgramID, Year,
Period)
```

```
);
```

Enrollments

Tabela przechowuje zapisy użytkowników na kursy, webinaria i studia. Każdy użytkownik może zapisać się tylko raz na ten sam kurs lub webinar lub studia. Usunięcie kursu lub webinaru lub kierunku studiów automatycznie usuwa powiązane zapisy użytkowników w tej tabeli. Dodatkowo, zapisanie użytkownika jest możliwe tylko, jeśli jest on zarejestrowany w systemie.

Opis pól tabeli:

1. EnrollmentID `INT`
    - Unikalny identyfikator zapisu. Wartość generowana automatycznie.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. UserID `INT`
    - Identyfikator użytkownika zapisującego się na kurs lub webinar.
    - Warunki integralności:
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Users.
        - Warunek `CHECK`: zapis może dotyczyć tylko użytkowników, którzy są zarejestrowani (`IsRegistered = 1`).
3. CourseID `INT`
    - Identyfikator kursu, na który użytkownik się zapisał.
    - Warunki integralności:
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Courses
        - Zapewnia, że zapisy będą dotyczyć tylko jednego kursu lub webinaru lub studiów.
4. WebinarID `INT`
    - Identyfikator webinaru, na który użytkownik się zapisał.
    - Warunki integralności:
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Webinars.
        - Zapewnia, że zapisy będą dotyczyć tylko jednego kursu lub webinaru lub studiów.
5. StudyID `INT`
    - Identyfikator kierunku studiów, na który użytkownik się zapisał.
    - Warunki integralności:
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do tabeli Studies.
        - Zapewnia, że zapisy będą dotyczyć tylko jednego kursu lub webinaru lub studiów.

Kod generujący tabelę:

```
CREATE TABLE Enrollments (
    EnrollmentID INT IDENTITY(1,1) PRIMARY KEY,
    UserID INT NOT NULL,
    CourseID INT NULL,
    WebinarID INT NULL,
    StudyID INT NULL,

    -- Warunki integralności
    CONSTRAINT fk_enrollments_users FOREIGN KEY (UserID) REFERENCES
Users(UserID) ON DELETE CASCADE,
    CONSTRAINT fk_enrollments_courses FOREIGN KEY (CourseID)
REFERENCES Courses(CourseID) ON DELETE CASCADE,
    CONSTRAINT fk_enrollments_webinars FOREIGN KEY (WebinarID)
REFERENCES Webinars(WebinarID) ON DELETE CASCADE,
    CONSTRAINT fk_enrollments_studies FOREIGN KEY (StudyID)
REFERENCES Studies(StudyID) ON DELETE CASCADE,
    CONSTRAINT unique_user_course UNIQUE (UserID, CourseID),
    CONSTRAINT unique_user_webinar UNIQUE (UserID, WebinarID),
);
```

Attendance

Tabela przechowuje informacje o obecności użytkowników na sesjach. Każdy wpis wskazuje, czy dany użytkownik był obecny na danej sesji. Sesje, które mogą mieć wpisy w tabeli, muszą już się odbyć, a każdy użytkownik może mieć tylko jeden wpis na jedną sesję.

Opis pól tabeli:

1. AttendanceID `INT`
    - Unikalny identyfikator wpisu w tabeli. Wartość automatycznie generowana jako numeracja sekwencyjna.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. EnrollmentID `INT`
    - Identyfikator zapisu użytkownika na sesję.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do kolumny `EnrollmentID` w tabeli Enrollments.
        - Zasada `ON DELETE CASCADE` oznacza, że usunięcie zapisu w tabeli Enrollments spowoduje usunięcie powiązanych wpisów w tabeli Attendance.
3. SessionDate `DATETIME`
    - Data i czas sesji. Sesja musi mieć datę, która jest przeszłością (już się odbyła).
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` sprawdzający, czy data sesji nie jest w przyszłości: `SessionDate <= SYSDATETIME()`.
        - Kombinacja `EnrollmentID` i `SessionDate` musi być unikalna, co oznacza, że użytkownik może mieć tylko jeden wpis na daną sesję.
4. IsPresent `BIT`
    - Flaga oznaczająca, czy użytkownik był obecny na sesji. Może mieć tylko dwie wartości: 0 (nieobecny) lub 1 (obecny).
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - Warunek `CHECK` ograniczający wartości do 0 lub 1: `IsPresent IN (0, 1)`.

Kod generujący tabelę:

```sql
CREATE TABLE Attendance (
    AttendanceID INT IDENTITY(1,1) PRIMARY KEY,
    EnrollmentID INT NOT NULL,
    SessionDate DATETIME NOT NULL,
    IsPresent BIT NOT NULL DEFAULT 0,

    -- Warunki integralności
    CONSTRAINT fk_enrollment FOREIGN KEY (EnrollmentID) REFERENCES
Enrollments(EnrollmentID) ON DELETE CASCADE,
    CONSTRAINT chk_session_date CHECK (SessionDate <= SYSDATETIME()),
    CONSTRAINT chk_is_present CHECK (IsPresent IN (0, 1)),
    CONSTRAINT unique_user_session UNIQUE (EnrollmentID, SessionDate)
);
```

Diplomas

Tabela przechowuje informacje na temat dyplomów wydanych użytkownikom po ukończeniu kursów. Zawiera dane o użytkowniku, kursie, dacie wydania dyplomu, adresie wysyłki oraz statusie wysyłki dyplomu.

Opis pól tabeli:

1. DiplomaID `INT`
    - Unikalny identyfikator dyplomu. Jest generowany automatycznie przy każdym dodaniu nowego rekordu.
    - Warunki integralności:
        - `PRIMARY KEY` – zapewnia unikalność każdego rekordu w tabeli.
2. UserID `INT`
    - Identyfikator użytkownika, który otrzymał dyplom.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli Users, zapewniając, że dyplom jest przypisany do istniejącego użytkownika.
3. CourseID `INT`
    - Identyfikator kursu, którego dyplom dotyczy.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli Courses, zapewniając, że dyplom jest przypisany do istniejącego kursu.
        - Dyplom może być wydany tylko po ukończeniu kursu.
4. IssuedDate `DATETIME`
    - Data wydania dyplomu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - CHECK – `IssuedDate <= GETDATE()` – zapewnia, że dyplom nie może zostać wydany w przyszłości, czyli data wydania musi być równa lub wcześniejsza od dzisiejszej daty.
5. Address `VARCHAR(50)`
    - Adres, na który dyplom ma zostać wysłany.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
6. IsSent `BIT`
    - Status wysyłki dyplomu.

- Warunki integralności:
  - `NOT NULL` – pole obowiązkowe.
  - CHECK – `IsSent IN (0, 1)` – zapewnia, że pole przyjmuje tylko dwie możliwe wartości: 0 (nie wysłano) lub 1 (wysłano).

Kod generujący tabelę:

```
CREATE TABLE Diplomas (
    DiplomaID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    UserID INT NOT NULL,
    CourseID INT NOT NULL,
    IssuedDate DATETIME NOT NULL,
    Address VARCHAR(50) NOT NULL,
    IsSent BIT NOT NULL,

    -- Warunki integralności
    CONSTRAINT fk_diplomas_userid FOREIGN KEY (UserID) REFERENCES
Users(UserID),
    CONSTRAINT fk_courseid FOREIGN KEY (CourseID) REFERENCES
Courses(CourseID),
    CONSTRAINT valid_issued_date CHECK (IssuedDate <= GETDATE()),
    CONSTRAINT valid_issent CHECK (IsSent IN (0, 1))
);
```

Products

Tabela Products przechowuje informacje o dostępnych produktach w systemie, takich jak webinary, kursy, studia i inne typy produktów. W tej tabeli znajdują się dane związane z ceną, typem produktu oraz powiązaniami z innymi tabelami (takimi jak Webinars, Courses, Studies - opłata za czesne, Sessions - cena dla osób z zewnątrz).

Opis pól tabeli:

1. ProductID `INT`
    - Unikalny identyfikator produktu w systemie. Jest generowany automatycznie.
    - Warunki integralności:
        - `PRIMARY KEY` – zapewnia unikalność każdego rekordu w tabeli, nie dopuszczając powtórzeń identyfikatora produktu.
2. ProductType `VARCHAR(20)`
    - Typ produktu ('Webinar', 'Course', 'Study'). Określa, do jakiego rodzaju produktu należy dany rekord.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - CHECK – `ProductType IN ('Webinar', 'Course', 'Study')` – zapewnia, że typ produktu będzie jednym z dozwolonych.
3. Price `DECIMAL(10,2)`
    - Cena produktu.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - `CHECK` – `Price >= 0.00` – cena musi być wartością nieujemną:
            1. Jeśli produktem jest Webinar, należy sprawdzić w tabeli Webinars, czy IsFree = 0, wtedy `Price > 0.00`.
            2. Jeśli produktem jest kurs, to cena zawsze jest dodatnia: `Price > 0.00`.
4. CurrencyCode `VARCHAR(10)`
    - Kod waluty powiązany z tabelą Currency.
    - Warunki integralności:
        - Klucz obcy (`FOREIGN KEY`) odnoszący się do kolumny `CurrencyCode` w tabeli Currency.
        - Warunek `CHECK`: `Price > 0`.
        - Wartość domyślna to PLN
5. WebinarID `INT`

- Identyfikator powiązanego webinaru, jeśli produkt jest webinar. To pole może zawierać wartość NULL, jeśli produkt nie jest webinar.
- Warunki integralności:
  - NULL – pole może być puste, jeśli produkt nie jest webinar.
  - FOREIGN KEY – odnosi się do tabeli Webinars.

6. CourseID INT
   - Identyfikator powiązanego kursu, jeśli produkt jest kursem. To pole może zawierać wartość NULL, jeśli produkt nie jest kursem.
   - Warunki integralności:
     - NULL – pole może być puste, jeśli produkt nie jest kursem.
     - FOREIGN KEY – odnosi się do tabeli Courses.
7. StudyID INT
   - Identyfikator powiązanego kierunku studiów, jeśli produkt jest kierunkiem studiów. To pole może zawierać wartość NULL, jeśli produkt nim nie jest.
   - Warunki integralności:
     - NULL – pole może być puste, jeśli produkt nie jest programem studiów.
     - FOREIGN KEY – odnosi się do tabeli Studies.
8. SessionID INT
   - Identyfikator powiązanego studium, jeśli produkt nim jest. To pole może zawierać wartość NULL, jeśli produkt nim nie jest.
   - Warunki integralności:
     - NULL – pole może być puste, jeśli produkt nie jest programem studiów.
     - FOREIGN KEY – odnosi się do tabeli Sessions.

Kod generujący tabelę:

```
CREATE TABLE Products (
    ProductID INT IDENTITY(1,1) PRIMARY KEY,
    ProductType VARCHAR(20) NOT NULL,
    Price DECIMAL(10, 2) NOT NULL CHECK (Price >= 0.00),
    CurrencyCode VARCHAR(10) NOT NULL DEFAULT ‘PLN’,
    WebinarID INT NULL,
    CourseID INT NULL,
    StudyID INT NULL,
    SessionID INT NULL,

    -- Warunki integralności
    CONSTRAINT chk_product_type CHECK (ProductType IN ('Webinar',
'Course', 'Study')),
```

```
    CONSTRAINT chk_price_positive CHECK (Price > 0.00),
    FOREIGN KEY (CurrencyCode) REFERENCES Currency(CurrencyCode),
    FOREIGN KEY (WebinarID) REFERENCES Webinars(WebinarID),
    FOREIGN KEY (CourseID) REFERENCES Courses(CourseID),
    FOREIGN KEY (StudyID) REFERENCES Studies(StudyID),
    FOREIGN KEY (SessionID) REFERENCES Sessions(SessionID)
);
```

Currency

Tabela przechowuje informacje o walutach, takich jak unikalny kod waluty, kurs wymiany oraz wskazanie, czy dana waluta jest domyślna.

Opis pól tabeli:

1. CurrencyID `INT`
    - Unikalny identyfikator waluty.
    - Warunki integralności: Klucz główny (`PRIMARY KEY`).
2. CurrencyName `VARCHAR(50)`
    - Nazwa waluty.
3. CurrencyCode `VARCHAR(10)`
    - Unikalny kod waluty (np. "USD", "EUR").
    - Warunki integralności:
        - `NOT NULL`.
        - Musi być unikalne (`UNIQUE`).
4. ExchangeRate `DECIMAL(10, 4)`
    - Kurs wymiany waluty względem bazowej. Musi być większy od 0.
    - Warunki integralności:
        - `NOT NULL`.
        - Warunek `CHECK`: `ExchangeRate > 0`.

Kod generujący tabelę:

```
CREATE TABLE Currency (
    CurrencyID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    CurrencyName VARCHAR(50),
    CurrencyCode VARCHAR(10) NOT NULL,
    ExchangeRate DECIMAL(10, 4) NOT NULL CHECK (ExchangeRate > 0),

    CONSTRAINT unique_currencycode UNIQUE (CurrencyCode)
);
```

Payments

Tabela Payments przechowuje informacje dotyczące płatności dokonanych przez użytkowników za zamówienia w systemie. Zawiera dane o kwocie płatności, statusie, dacie oraz powiązaniach z użytkownikami.

Opis pól tabeli:

1. PaymentID `INT`
    - Unikalny identyfikator płatności. Jest generowany automatycznie.
    - Warunki integralności:
        - `PRIMARY KEY` – zapewnia unikalność każdego rekordu w tabeli, nie dopuszczając powtórzeń identyfikatora płatności.
2. UserAccountID `INT`
    - Identyfikator konta użytkownika, który dokonał płatności.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli UsersAccount, zapewniając, że każda płatność jest przypisana do istniejącego konta użytkownika.
3. CurrencyID `INT`
    - Identyfikator waluty, w jakiej dokonano płatności.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli Currency, zapewniając, że każda płatność jest związana z istniejącą walutą.
4. PaymentStatus `VARCHAR(20)`
    - Status płatności (np. "Pending", "Completed", "Failed").
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
5. PaymentDate `DATETIME`
    - Data i godzina dokonania płatności.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - CHECK – zapewnia, że PaymentDate nie jest wcześniejsza niż data zamówienia.
          Należy upewnić się, że PaymentDate jest późniejsza lub równa OrderDate z tabeli Orders.

Kod generujący tabelę:

```sql
CREATE TABLE Payments (
    PaymentID INT PRIMARY KEY,
    UserAccountID INT NOT NULL,
    CurrencyID INT NOT NULL,
    PaymentStatus VARCHAR(20) NOT NULL CHECK (PaymentStatus IN
('Pending', 'Completed', 'Failed')),
    PaymentDate DATETIME NOT NULL,

    FOREIGN KEY (UserAccountID) REFERENCES
UsersAccount(UserAccountID),
    FOREIGN KEY (CurrencyID) REFERENCES Currency(CurrencyID),
);
```

Orders

Tabela Orders przechowuje informacje o zamówieniach złożonych przez użytkowników. Zawiera dane dotyczące statusu zamówienia, płatności, daty zamówienia oraz całkowitej kwoty. Dodatkowo, jest powiązana z tabelą UsersAccount poprzez UserAccountID, wskazując na konto użytkownika, który złożył zamówienie.

Opis pól tabeli:

1. OrderID `INT`
    - Unikalny identyfikator zamówienia. Jest generowany automatycznie.
    - Warunki integralności:
        - `PRIMARY KEY` – zapewnia unikalność każdego rekordu w tabeli.
2. UserAccountID `INT`
    - Identyfikator konta użytkownika, który złożył zamówienie.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli UsersAccount, zapewniając, że zamówienie jest przypisane do istniejącego konta użytkownika.
3. OrderDate `DATETIME`
    - Data złożenia zamówienia.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - CHECK – `OrderDate <= GETDATE()` – zapewnia, że data zamówienia nie może być późniejsza niż bieżąca data (zamówienie nie może być zapisane w przyszłości).
4. OrderStatus `VARCHAR(20)`
    - Status zamówienia (np. w trakcie realizacji, wysłane, anulowane).
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - CHECK – `OrderStatus IN ('In progress', 'Sent', 'Cancelled')` – zapewnia, że status zamówienia może przyjąć tylko jedną z trzech wartości.
5. OrderTotal `DECIMAL(10, 2)`
    - Całkowita kwota zamówienia.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.

        - CHECK – `OrderTotal >= 0` – zapewnia, że całkowita kwota zamówienia nie może być mniejsza niż 0.
6. PaymentID `INT`
    - Identyfikator płatności powiązanej z zamówieniem.
    - Warunki integralności:
        - Jest kluczem obcym odnoszącym się do tabeli Payments. Zapewnia to, że zamówienie może być powiązane tylko z istniejącą płatnością

```
CREATE TABLE Orders (
    OrderID INT IDENTITY(1,1) PRIMARY KEY,
    UserAccountID INT NOT NULL,
    OrderDate DATETIME NOT NULL,
    OrderStatus VARCHAR(20) NOT NULL,
    OrderTotal DECIMAL(10, 2) NOT NULL,
    PaymentID INT,

    -- Warunki integralności
    CONSTRAINT FK_UserAccount FOREIGN KEY (UserAccountID)
        REFERENCES UsersAccount(UserAccountID) ON DELETE CASCADE ON
UPDATE CASCADE,
    CONSTRAINT FK_Payment FOREIGN KEY (PaymentID)
        REFERENCES Payments(PaymentID) ON DELETE SET NULL ON UPDATE
CASCADE,
    CONSTRAINT CHK_OrderDate CHECK (OrderDate <= GETDATE()),
    CONSTRAINT CHK_OrderStatus CHECK (OrderStatus IN ('In progress',
'Sent', 'Cancelled')),
    CONSTRAINT CHK_OrderTotal CHECK (OrderTotal >= 0)
);
```

OrderItems

Tabela OrderItems przechowuje szczegóły dotyczące poszczególnych produktów w ramach zamówienia. Każdy rekord w tej tabeli odpowiada jednej pozycji zamówienia, która może zawierać informacje o produkcie (np. kurs, webinar, studia), ilości zamówionych sztuk, oraz cenie jednostkowej produktu.

Opis pól tabeli:

1. OrderItemID `INT`
    - Unikalny identyfikator pozycji zamówienia. Jest generowany automatycznie przy każdym dodaniu nowego rekordu, dzięki użyciu właściwości `IDENTITY(1,1)`.
    - Warunki integralności:
        - PRIMARY KEY – zapewnia unikalność każdej pozycji w tabeli OrderItems.
2. OrderID `INT`
    - Identyfikator zamówienia, do którego należy dana pozycja.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli Orders, zapewniając, że pozycja zamówienia jest przypisana do istniejącego zamówienia.
        - ON DELETE CASCADE – usunięcie zamówienia spowoduje usunięcie wszystkich pozycji zamówienia, które są z nim powiązane.
3. UserAccountID `INT`
    - Identyfikator konta użytkownika, który złożył zamówienie.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli UsersAccount, zapewniając, że pozycja zamówienia jest przypisana do istniejącego konta użytkownika.
        - ON DELETE CASCADE – usunięcie konta użytkownika spowoduje usunięcie wszystkich pozycji zamówienia przypisanych do tego konta.
4. ProductID `VARCHAR(20)`
    - Identyfikator produktu, np. Webinar, Kurs, Studia.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
5. Quantity `INT`
    - Ilość zamówionych sztuk danego produktu.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.

- CHECK – `Quantity > 0` – zapewnia, że ilość zamówionych sztuk musi być większa od zera. Nie można zamówić ułamkowych lub zerowych ilości.

Kod SQL generujący tabelę:

```
CREATE TABLE OrderItems (
    OrderItemID INT IDENTITY(1,1) PRIMARY KEY,
    OrderID INT NOT NULL,
    UserAccountID INT NOT NULL,
    ProductID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity > 0),

    -- Warunki integralności
    FOREIGN KEY (OrderID) REFERENCES Orders(OrderID) ON DELETE NO
ACTION,
    FOREIGN KEY (UserAccountID) REFERENCES
UsersAccount(UserAccountID) ON DELETE CASCADE,
    FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);
```

ShoppingCart

Tabela ShoppingCart przechowuje informacje dotyczące koszyków użytkowników, takie jak status koszyka, powiązanie z kontem użytkownika oraz data i godzina jego utworzenia. Tabela ta umożliwia śledzenie, w jakim stanie znajduje się koszyk oraz kiedy został stworzony.

Opis pól tabeli:

1. CartID `INT`
    - Unikalny identyfikator koszyka użytkownika w systemie. Jest generowany automatycznie, poczynając od 1 i inkrementując o 1 dla każdego nowego koszyka.
    - Warunki integralności:
        - `PRIMARY KEY` – zapewnia unikalność każdego koszyka w tabeli.
2. UserAccountID `INT`
    - Identyfikator konta użytkownika, do którego należy koszyk.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli UsersAccount, zapewniając, że koszyk jest przypisany do istniejącego konta użytkownika.
        - ON DELETE CASCADE – gdy konto użytkownika zostanie usunięte, powiązany koszyk także zostanie usunięty.
3. Status `VARCHAR(20)`
    - Określa bieżący status koszyka. Może przyjąć jedną z trzech wartości: 'Active' (koszyk aktywny), 'Completed' (koszyk zakończony) lub 'Abandoned' (koszyk porzucony).
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.
4. CreatedAt `DATETIME`
    - Data i godzina utworzenia koszyka.
    - Warunki integralności:
        - `NOT NULL` – pole obowiązkowe.

Kod generujący tabelę:

```
CREATE TABLE ShoppingCart (
    CartID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    UserAccountID INT NOT NULL,
    Status VARCHAR(20) NOT NULL CHECK (Status IN ('Active',
'Completed', 'Abandoned')),
    CreatedAt DATETIME NOT NULL CHECK (CreatedAt <= GETDATE()),
```

```
    FOREIGN KEY (UserAccountID) REFERENCES
UsersAccount(UserAccountID) ON DELETE CASCADE
);
```

CartItems

Tabela CartItems przechowuje szczegóły dotyczące poszczególnych produktów dodanych do koszyka zakupowego. Każdy rekord w tej tabeli reprezentuje jeden produkt w koszyku, w tym ilość zamówionych sztuk, cenę jednostkową i typ produktu (np. webinar, kurs, studia).

Opis pól tabeli:

1. CartItemID `INT`
    - Unikalny identyfikator pozycji w koszyku. Jest generowany automatycznie przy każdym dodaniu nowego rekordu, dzięki właściwości `IDENTITY(1,1)`.
    - Warunki integralności:
        - PRIMARY KEY – zapewnia unikalność każdej pozycji w tabeli CartItems.
2. CartID `INT`
    - Identyfikator koszyka, do którego należy dana pozycja.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli ShoppingCart, zapewniając, że każda pozycja koszyka jest przypisana do istniejącego koszyka.
        - ON DELETE CASCADE – usunięcie koszyka spowoduje usunięcie wszystkich pozycji koszyka, które są z nim powiązane.
3. ProductID `INT`
    - Identyfikator produktu, który został dodany do koszyka.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
        - FOREIGN KEY – odnosi się do tabeli Products (lub innej tabeli zawierającej produkty), zapewniając, że pozycja koszyka jest przypisana do istniejącego produktu.
4. Quantity `INT`
    - Ilość zamówionych sztuk danego produktu w koszyku.
    - Warunki integralności:
        - NOT NULL – pole obowiązkowe.
        - CHECK – `Quantity >= 1` – zapewnia, że ilość zamówionych sztuk musi być większa lub równa 1. Nie można dodać do koszyka mniej niż jednego przedmiotu.

Kod SQL generujący tabelę:

```
CREATE TABLE CartItems (
```

```
    CartItemID INT NOT NULL IDENTITY(1, 1) PRIMARY KEY,
    CartID INT NOT NULL,
    ProductID INT NOT NULL,
    Quantity INT NOT NULL CHECK (Quantity >= 1),

    -- Warunki integralności
    CONSTRAINT fk_cartid FOREIGN KEY (CartID) REFERENCES
ShoppingCart(CartID) ON DELETE CASCADE,
    CONSTRAINT fk_productid FOREIGN KEY (ProductID) REFERENCES
Products(ProductID)
);
```

WIDOKI

1. Raporty finansowe – zestawienie przychodów dla każdego webinaru/kursu/studium.

Ten widok generuje raport finansowy przedstawiający zestawienie przychodów dla każdego webinaru, kursu lub studium. Grupuje dane według kategorii produktu (`Webinar`, `Course`, `Study`) oraz ich unikalnych identyfikatorów. Wylicza całkowity przychód, liczbę zamówień oraz daty pierwszego i ostatniego zamówienia dla każdej grupy. Uwzględnia tylko zamówienia, które zostały opłacone (`PaymentStatus = 'Completed'`), dzięki czemu raport jest dokładny pod względem przychodów.

```
CREATE VIEW FinancialReports AS
SELECT
    CASE
        WHEN p.WebinarID IS NOT NULL THEN 'Webinar'
        WHEN p.CourseID IS NOT NULL THEN 'Course'
        WHEN p.StudyID IS NOT NULL THEN 'Study'
        ELSE 'Other'
    END AS ProductCategory,
    COALESCE(p.WebinarID, p.CourseID, p.StudyID) AS
ProductReferenceID,
    SUM(o.OrderTotal) AS TotalRevenue,
    COUNT(o.OrderID) AS TotalOrders,
    MIN(o.OrderDate) AS FirstOrderDate,
    MAX(o.OrderDate) AS LastOrderDate
FROM
    Orders o
JOIN Products p ON o.OrderID = p.ProductID
JOIN Payments pay ON o.PaymentID = pay.PaymentID
WHERE
    pay.PaymentStatus = 'Completed'
GROUP BY
    CASE
        WHEN p.WebinarID IS NOT NULL THEN 'Webinar'
        WHEN p.CourseID IS NOT NULL THEN 'Course'
        WHEN p.StudyID IS NOT NULL THEN 'Study'
```

```
        ELSE 'Other'
    END,
    COALESCE(p.WebinarID, p.CourseID, p.StudyID);
```

2. Lista „dłużników” – osoby, które skorzystały z usług, ale nie uiściły opłat.

Ten widok prezentuje listę użytkowników, którzy złożyli zamówienia, ale nie uregulowali płatności. Łączy informacje o użytkownikach, zamówieniach i statusie płatności, aby zidentyfikować osoby, których zamówienia mają status płatności inny niż `Completed` (lub brak przypisanej płatności). Zawiera dane takie jak nazwa użytkownika, data utworzenia konta, szczegóły zamówienia oraz aktualny status płatności. Widok pozwala na szybkie zidentyfikowanie „dłużników” i podjęcie odpowiednich działań.

```
CREATE VIEW DebtorsList AS
SELECT
    ua.UserAccountID,
    ua.Username,
    ua.CreatedAt AS AccountCreationDate,
    o.OrderID,
    o.OrderDate,
    o.OrderTotal,
    p.PaymentStatus
FROM
    UsersAccount ua
JOIN Orders o ON ua.UserAccountID = o.UserAccountID
LEFT JOIN Payments p ON o.PaymentID = p.PaymentID
WHERE
    (p.PaymentStatus IS NULL OR p.PaymentStatus != 'Completed')
    AND o.OrderStatus != 'Cancelled';
```

3. Ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia (z informacją, czy wydarzenie jest stacjonarnie, czy zdalnie).

Ten widok prezentuje ogólny raport dotyczący liczby zapisanych osób na przyszłe wydarzenia, w tym webinary, kursy i studia. Dla każdego wydarzenia podawany jest jego tytuł, data rozpoczęcia, tryb (stacjonarny, zdalny lub hybrydowy) oraz łączna liczba zapisanych uczestników.

- **Webinary** są zawsze oznaczane jako zdalne (online), a ich status uwzględnia tylko te wydarzenia, które nie zostały usunięte i odbędą się w przyszłości.
- **Kursy** mogą być stacjonarne lub hybrydowe, w zależności od wartości w kolumnie `IsHybrid`.
- **Studia** mają tryb ustalany na podstawie sesji (`Sessions`): jeśli wszystkie sesje są stacjonarne, studia są stacjonarne; jeśli wszystkie sesje są zdalne, studia są zdalne; w pozostałych przypadkach studia są hybrydowe.

```
CREATE VIEW EventEnrollmentReport AS
SELECT
    'Webinar' AS EventType,
    w.WebinarID AS EventID,
    w.Title AS EventTitle,
    w.DateTime AS EventDate,
    'Online' AS EventMode,
    COUNT(e.EnrollmentID) AS TotalEnrollments
FROM
    Webinars w
LEFT JOIN Enrollments e ON w.WebinarID = e.WebinarID
WHERE
    w.IsDeleted = 0 AND w.DateTime > GETDATE()
GROUP BY
    w.WebinarID, w.Title, w.DateTime

UNION ALL

SELECT
    'Course' AS EventType,
    c.CourseID AS EventID,
    c.Title AS EventTitle,
```

```
    c.StartDate AS EventDate,
    CASE
        WHEN c.IsHybrid = 1 THEN 'Hybrid'
        ELSE 'Stationary'
    END AS EventMode,
    COUNT(e.EnrollmentID) AS TotalEnrollments
FROM
    Courses c
LEFT JOIN Enrollments e ON c.CourseID = e.CourseID
WHERE
    c.StartDate > GETDATE()
GROUP BY
    c.CourseID, c.Title, c.StartDate, c.IsHybrid

UNION ALL

SELECT
    'Study' AS EventType,
    s.StudyID AS EventID,
    s.Title AS EventTitle,
    s.StartDate AS EventDate,
    CASE
        WHEN COUNT(DISTINCT sess.SessionType) = 1 AND
MIN(sess.SessionType) = 'stationary' THEN 'Stationary'
        WHEN COUNT(DISTINCT sess.SessionType) = 1 AND
MIN(sess.SessionType) = 'online' THEN 'Online'
        ELSE 'Hybrid'
    END AS EventMode,
    COUNT(e.EnrollmentID) AS TotalEnrollments
FROM
    Studies s
LEFT JOIN Sessions sess ON s.StudyID = sess.StudyID
LEFT JOIN Enrollments e ON s.StudyID = e.StudyID
WHERE
    s.StartDate > GETDATE()
GROUP BY
    s.StudyID, s.Title, s.StartDate;
```

4. Ogólny raport dotyczący frekwencji na zakończonych już wydarzeniach.

Widok służy do generowania raportu dotyczącego frekwencji na zakończonych wydarzeniach, w tym webinarach, kursach i studiach. Agreguje kluczowe dane, takie jak liczba uczestników dla każdego rodzaju wydarzenia, tytuły wydarzeń oraz daty ich zakończenia. Raport umożliwia analizę frekwencji, identyfikację najbardziej popularnych wydarzeń oraz monitorowanie trendów uczestnictwa w czasie.

```
CREATE OR REPLACE VIEW AttendanceReport AS
SELECT
    'Webinar' AS EventType,
    W.Title AS EventTitle,
    COUNT(E.EnrollmentID) AS AttendanceCount,
    W.DateTime AS EventDate
FROM
    Webinars W
LEFT JOIN
    Enrollments E
    ON W.WebinarID = E.WebinarID
WHERE
    W.IsDeleted = 0
    AND W.DateTime < GETDATE()
GROUP BY
    W.Title, W.DateTime

UNION ALL

SELECT
    'Course' AS EventType,
    C.Title AS EventTitle,
    COUNT(E.EnrollmentID) AS AttendanceCount,
    C.EndDate AS EventDate
FROM
    Courses C
LEFT JOIN
```

```
    Enrollments E
    ON C.CourseID = E.CourseID
WHERE
    C.EndDate < GETDATE()
GROUP BY
    C.Title, C.EndDate

UNION ALL

SELECT
    'Study' AS EventType,
    S.Title AS EventTitle,
    COUNT(E.EnrollmentID) AS AttendanceCount,
    S.EndDate AS EventDate
FROM
    Studies S
LEFT JOIN
    Enrollments E
    ON S.StudyID = E.StudyID
WHERE
    S.EndDate < GETDATE()
GROUP BY
    S.Title, S.EndDate;
```

5. Lista obecności dla każdego szkolenia z datą, imieniem, nazwiskiem i informacją czy uczestnik był obecny, czy nie.

Ten widok przedstawia szczegółową listę obecności dla każdego rodzaju szkolenia (webinary, kursy, studia), uwzględniając datę sesji, imię i nazwisko uczestnika oraz informację o jego obecności. Widok ułatwia monitorowanie frekwencji uczestników na wszystkich szkoleniach.

- **Webinary** są identyfikowane na podstawie zapisów w tabeli `Enrollments` oraz `Webinars`. Widok zawiera tylko te wydarzenia, które nie zostały usunięte.
- **Kursy** są rozpoznawane dzięki zapisom w tabelach `Courses` i `Enrollments`.

- **Studia** uwzględniają sesje (`Sessions`) przypisane do kierunków (`Studies`) oraz daty tych sesji w tabeli `Attendance`.

```
CREATE VIEW AttendanceList AS
SELECT
    'Webinar' AS TrainingType,
    w.Title AS TrainingTitle,
    a.SessionDate,
    CONCAT(u.FirstName, ' ', u.LastName) AS ParticipantName,
    a.IsPresent
FROM
    Attendance a
JOIN Enrollments e ON a.EnrollmentID = e.EnrollmentID
JOIN Webinars w ON e.WebinarID = w.WebinarID
JOIN UsersAccount ua ON e.UserID = ua.UserAccountID
JOIN Users u ON ua.UserID = u.UserID
WHERE
    w.IsDeleted = 0

UNION ALL

SELECT
    'Course' AS TrainingType,
    c.Title AS TrainingTitle,
    a.SessionDate,
    CONCAT(u.FirstName, ' ', u.LastName) AS ParticipantName,
    a.IsPresent
FROM
    Attendance a
JOIN Enrollments e ON a.EnrollmentID = e.EnrollmentID
JOIN Courses c ON e.CourseID = c.CourseID
JOIN UsersAccount ua ON e.UserID = ua.UserAccountID
JOIN Users u ON ua.UserID = u.UserID

UNION ALL

SELECT
    'Study' AS TrainingType,
    s.Title AS TrainingTitle,
```

```
    a.SessionDate,
    CONCAT(u.FirstName, ' ', u.LastName) AS ParticipantName,
    a.IsPresent
FROM
    Attendance a
JOIN Enrollments e ON a.EnrollmentID = e.EnrollmentID
JOIN Sessions sess ON sess.StudyID = e.StudyID AND sess.SessionDate =
a.SessionDate
JOIN Studies s ON sess.StudyID = s.StudyID
JOIN UsersAccount ua ON e.UserID = ua.UserAccountID
JOIN Users u ON ua.UserID = u.UserID;
```

6. Raport bilokacji: lista osób, które są zapisane na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo.

Widok "BilocationReport" identyfikuje użytkowników, którzy są zapisani na co najmniej dwa przyszłe szkolenia, które ze sobą kolidują czasowo. Taki raport jest użyteczny do wykrywania przypadków, w których uczestnicy mogą mieć trudności z wzięciem udziału w dwóch wydarzeniach, które odbywają się w tym samym czasie lub mają nakładające się zakresy dat.

```
CREATE VIEW BilocationReport AS
WITH FutureTrainings AS (
    SELECT
        e.UserID,
        'Webinar' AS TrainingType,
        w.Title AS TrainingTitle,
        w.DateTime AS StartTime,
        DATEADD(MINUTE, w.Duration, w.DateTime) AS EndTime
    FROM Enrollments e
    JOIN Webinars w ON e.WebinarID = w.WebinarID
    WHERE w.DateTime > GETDATE() AND w.IsDeleted = 0

    UNION ALL

    SELECT
```

```
        e.UserID,
        'Course' AS TrainingType,
        c.Title AS TrainingTitle,
        c.StartDate AS StartTime,
        c.EndDate AS EndTime
    FROM Enrollments e
    JOIN Courses c ON e.CourseID = c.CourseID
    WHERE c.StartDate > GETDATE()

    UNION ALL

    SELECT
        e.UserID,
        'Study' AS TrainingType,
        s.Title AS TrainingTitle,
        s.StartDate AS StartTime,
        s.EndDate AS EndTime
    FROM Enrollments e
    JOIN Studies s ON e.StudyID = s.StudyID
    WHERE s.StartDate > GETDATE()
)
SELECT
    u.FirstName,
    u.LastName,
    ft1.TrainingTitle AS Training1,
    ft1.TrainingType AS Training1Type,
    ft1.StartTime AS Training1Start,
    ft1.EndTime AS Training1End,
    ft2.TrainingTitle AS Training2,
    ft2.TrainingType AS Training2Type,
    ft2.StartTime AS Training2Start,
    ft2.EndTime AS Training2End
FROM
    FutureTrainings ft1
JOIN FutureTrainings ft2
    ON ft1.UserID = ft2.UserID
    AND ft1.StartTime < ft2.EndTime
    AND ft1.EndTime > ft2.StartTime
    AND ft1.TrainingTitle <> ft2.TrainingTitle
```

```
JOIN Users u ON ft1.UserID = u.UserID;
```

PROCEDURY

Konta i uprawnienia

1. Założenie konta uczestnika

Ta procedura tworzy nowe konto użytkownika, który zostaje uczestnikiem (studentem) systemu. Wykonuje operację dodania nowego użytkownika do tabeli `Users`, tworzy powiązane konto logowania w tabeli `UsersAccount` i oznacza użytkownika jako zarejestrowanego.

```
CREATE PROCEDURE AddStudentAccount
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @EmailAddress NVARCHAR(150),
    @PhoneNumber VARCHAR(9),
    @Role NVARCHAR(50),
    @Username VARCHAR(50),
    @PasswordHash VARCHAR(255)
AS
BEGIN

    INSERT INTO Users (FirstName, LastName, EmailAddress,
PhoneNumber, Role, IsRegistered)
    VALUES (@FirstName, @LastName, @EmailAddress, @PhoneNumber,
'Student', 0);

    DECLARE @UserID INT = SCOPE_IDENTITY();

    INSERT INTO UsersAccount (UserID, Username, PasswordHash,
CreatedAt)
    VALUES (@UserID, @Username, @PasswordHash, GETDATE());
```

```
    UPDATE Users SET IsRegistered = 1 WHERE UserID = @UserID;

    SELECT 'Account created successfully' AS Message;
END;
```

2. Usunięcie konta uczestnika

Procedura usuwa konto uczestnika z systemu. Usuwa użytkownika z tabel `Users` i `UsersAccount`, pod warunkiem, że użytkownik jest uczestnikiem (studentem).

```
CREATE PROCEDURE DeleteStudentAccount
    @UserID INT
AS
BEGIN
    IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID AND Role =
'Student')
    BEGIN
        DELETE FROM UsersAccount WHERE UserID = @UserID;
        DELETE FROM Users WHERE UserID = @UserID;

        SELECT 'Student account deleted successfully' AS Message;
    END
    ELSE
    BEGIN
        SELECT 'No such student account found' AS Message;
    END
END;
```

3. Ustawienie adresu korespondencyjnego w celu otrzymania dyplomu

Procedura umożliwia ustawienie lub zaktualizowanie adresu korespondencyjnego studenta, który będzie używany do wysyłki dyplomu.

```
CREATE PROCEDURE SetCorrespondenceAddress
```

```
    @UserID INT,
    @Address NVARCHAR(100),
    @City NVARCHAR(100),
    @PostalCode NVARCHAR(8)
AS
BEGIN

    UPDATE Users
    SET Address = @Address, City = @City, PostalCode = @PostalCode
    WHERE UserID = @UserID AND Role = 'Student';

    SELECT 'Correspondence address updated successfully' AS Message;
END;
```

4. Założenie i dezaktywacja konta tłumacza przez pracownika sekretariatu

Tworzy konto dla tłumacza. Procedura dodaje użytkownika do systemu z rolą 'Translator' oraz tworzy powiązane konto w systemie logowania.

```
CREATE PROCEDURE AddTranslatorAccount
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @EmailAddress NVARCHAR(150),
    @PhoneNumber VARCHAR(9),
    @Username VARCHAR(50),
    @PasswordHash VARCHAR(255)
AS
BEGIN

    INSERT INTO Users (FirstName, LastName, EmailAddress,
PhoneNumber, Role, IsRegistered)
    VALUES (@FirstName, @LastName, @EmailAddress, @PhoneNumber,
'Translator', 0);

    DECLARE @UserID INT = SCOPE_IDENTITY();
```

```
    INSERT INTO UsersAccount (UserID, Username, PasswordHash,
CreatedAt)
    VALUES (@UserID, @Username, @PasswordHash, GETDATE());


    UPDATE Users SET IsRegistered = 1 WHERE UserID = @UserID;

    SELECT 'Translator account created successfully' AS Message;
END;
```

Procedura dezaktywuje konto tłumacza, ustawiając status rejestracji na 0. Pozwala na usunięcie dostępu do systemu.

```
CREATE PROCEDURE DeactivateTranslatorAccount
    @UserID INT
AS
BEGIN
     IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID AND Role =
'Translator')
    BEGIN
        UPDATE Users SET IsRegistered = 0 WHERE UserID = @UserID;
        SELECT 'Translator account deactivated successfully' AS
Message;
    END
    ELSE
    BEGIN
        SELECT 'No such translator account found' AS Message;
    END
END;
```

5. Założenie i dezaktywacja konta prowadzącego

Tworzy konto dla prowadzącego zajęcia (instruktora). Procedura dodaje użytkownika z rolą 'Instructor' oraz tworzy konto logowania.

```
CREATE PROCEDURE AddInstructorAccount
```

```
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @EmailAddress NVARCHAR(150),
    @PhoneNumber VARCHAR(9),
    @Username VARCHAR(50),
    @PasswordHash VARCHAR(255)
AS
BEGIN

    INSERT INTO Users (FirstName, LastName, EmailAddress,
PhoneNumber, Role, IsRegistered)
    VALUES (@FirstName, @LastName, @EmailAddress, @PhoneNumber,
'Instructor', 0);


    DECLARE @UserID INT = SCOPE_IDENTITY();


    INSERT INTO UsersAccount (UserID, Username, PasswordHash,
CreatedAt)
    VALUES (@UserID, @Username, @PasswordHash, GETDATE());

    UPDATE Users SET IsRegistered = 1 WHERE UserID = @UserID;

    SELECT 'Instructor account created successfully' AS Message;
END;
```

Procedura dezaktywuje konto instruktora, zmieniając jego status rejestracji na 0.

```
CREATE PROCEDURE DeactivateInstructorAccount
    @UserID INT
AS
BEGIN
        IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID AND
Role = 'Instructor')
    BEGIN
        UPDATE Users SET IsRegistered = 0 WHERE UserID = @UserID;
        SELECT 'Instructor account deactivated successfully' AS
```

```
Message;
    END
    ELSE
    BEGIN
        SELECT 'No such instructor account found' AS Message;
    END
END;
```

6. Założenie i dezaktywacja konta koordynatora przedmiotu

Tworzy konto koordynatora przedmiotu. Koordynator zarządza kursami i zajęciami w systemie. Procedura tworzy konto użytkownika w systemie.

```
CREATE PROCEDURE AddCoordinatorAccount
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @EmailAddress NVARCHAR(150),
    @PhoneNumber VARCHAR(9),
    @Username VARCHAR(50),
    @PasswordHash VARCHAR(255)
AS
BEGIN
     INSERT INTO Users (FirstName, LastName, EmailAddress,
PhoneNumber, Role, IsRegistered)
    VALUES (@FirstName, @LastName, @EmailAddress, @PhoneNumber,
'Coordinator', 0);


    DECLARE @UserID INT = SCOPE_IDENTITY();

        INSERT INTO UsersAccount (UserID, Username, PasswordHash,
CreatedAt)
    VALUES (@UserID, @Username, @PasswordHash, GETDATE());

       UPDATE Users SET IsRegistered = 1 WHERE UserID = @UserID;

    SELECT 'Coordinator account created successfully' AS Message;
```

```
END;
```

Dezaktywuje konto koordynatora, ustawiając status rejestracji na 0. Może być używana do usunięcia konta, gdy koordynator przestaje pełnić swoją rolę.

```
CREATE PROCEDURE DeactivateCoordinatorAccount
    @UserID INT
AS
BEGIN
        IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID AND
Role = 'Coordinator')
    BEGIN
        UPDATE Users SET IsRegistered = 0 WHERE UserID = @UserID;
        SELECT 'Coordinator account deactivated successfully' AS
Message;
    END
    ELSE
    BEGIN
        SELECT 'No such coordinator account found' AS Message;
    END
END;
```

7. Dodawanie i usuwanie pracownika sekretariatu przez dyrektora

Tworzy konto pracownika sekretariatu. Pracownik sekretariatu ma uprawnienia do zarządzania kontami i administracji systemu. Procedura dodaje użytkownika z rolą 'Registrar' i tworzy konto logowania.

```
CREATE PROCEDURE AddRegistrar
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @EmailAddress NVARCHAR(150),
    @PhoneNumber VARCHAR(9),
    @Username VARCHAR(50),
    @PasswordHash VARCHAR(255)
AS
```

```
BEGIN

    INSERT INTO Users (FirstName, LastName, EmailAddress,
PhoneNumber, Role, IsRegistered)
    VALUES (@FirstName, @LastName, @EmailAddress, @PhoneNumber,
'Registrar', 0);

    DECLARE @UserID INT = SCOPE_IDENTITY();

    INSERT INTO UsersAccount (UserID, Username, PasswordHash,
CreatedAt)
    VALUES (@UserID, @Username, @PasswordHash, GETDATE());

    UPDATE Users SET IsRegistered = 1 WHERE UserID = @UserID;

    SELECT 'Registrar account added successfully' AS Message;
END;
```

Usuwa konto pracownika sekretariatu z systemu. Może być używane, gdy pracownik przestaje pełnić funkcję lub opuszcza organizację.

```
CREATE PROCEDURE DeleteRegistrar
    @UserID INT
AS
BEGIN
        IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID AND
Role = 'Registrar')
    BEGIN
        DELETE FROM UsersAccount WHERE UserID = @UserID;
        DELETE FROM Users WHERE UserID = @UserID;
        SELECT 'Registrar account deleted successfully' AS Message;
    END
    ELSE
    BEGIN
        SELECT 'No such registrar account found' AS Message;
    END
END;
```

8. Dodawanie i usuwanie administratora przez dyrektora

Tworzy konto administratora w systemie. Administrator ma pełny dostęp do wszystkich funkcji
systemu. Procedura dodaje użytkownika z rolą 'Administrator' i tworzy powiązane konto logowania.

```
CREATE PROCEDURE AddAdministrator
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @EmailAddress NVARCHAR(150),
    @PhoneNumber VARCHAR(9),
    @Username VARCHAR(50),
    @PasswordHash VARCHAR(255)
AS
BEGIN
    INSERT INTO Users (FirstName, LastName, EmailAddress,
PhoneNumber, Role, IsRegistered)
    VALUES (@FirstName, @LastName, @EmailAddress, @PhoneNumber,
'Administrator', 0);


    DECLARE @UserID INT = SCOPE_IDENTITY();


    INSERT INTO UsersAccount (UserID, Username, PasswordHash,
CreatedAt)
    VALUES (@UserID, @Username, @PasswordHash, GETDATE());

     UPDATE Users SET IsRegistered = 1 WHERE UserID = @UserID;

    SELECT 'Administrator account added successfully' AS Message;
END;
```

Usuwa konto administratora z systemu, zmieniając jego status na zarejestrowany jako 0.

```
CREATE PROCEDURE DeleteAdministrator
    @UserID INT
```

```
AS
BEGIN
        IF EXISTS (SELECT 1 FROM Users WHERE UserID = @UserID AND
Role = 'Administrator')
    BEGIN
        DELETE FROM UsersAccount WHERE UserID = @UserID;
        DELETE FROM Users WHERE UserID = @UserID;
        SELECT 'Administrator account deleted successfully' AS
Message;
    END
    ELSE
    BEGIN
        SELECT 'No such administrator account found' AS Message;
    END
END;
```

Zamówienia

1. Dodawanie webinaru, kursu lub studiów do koszyka

Procedura dodaje produkt, taki jak webinar, kurs lub studia, do koszyka użytkownika, określając ilość. Wprowadza dane koszyka i produktu do tabeli elementów koszyka.

```
CREATE PROCEDURE AddToCart
    @CartID INT,
    @ProductID INT,
    @Quantity INT
AS
BEGIN
    INSERT INTO CartItems (CartID, ProductID, Quantity)
    VALUES (@CartID, @ProductID, @Quantity);
END;
```

2. Usuwanie webinaru, kursu lub studiów z koszyka

Procedura usuwa określony produkt z koszyka użytkownika na podstawie identyfikatora koszyka i produktu. Ułatwia zarządzanie zawartością koszyka.

```
CREATE PROCEDURE RemoveFromCart
    @CartID INT,
    @ProductID INT
AS
BEGIN
    DELETE FROM CartItems
    WHERE CartID = @CartID AND ProductID = @ProductID;
END;
```

3. Kupienie webinaru, kursu, studiów

Tworzy zamówienie na podstawie zawartości koszyka, obliczając całkowitą wartość zamówienia. Przenosi produkty z koszyka do zamówienia, zmieniając status koszyka na "Zakończony".

```
CREATE PROCEDURE PlaceOrder
    @UserAccountID INT,
    @CartID INT
AS
BEGIN
    DECLARE @OrderTotal DECIMAL(10, 2) = 0;

    SELECT @OrderTotal = SUM(ci.Quantity * p.Price)
    FROM CartItems ci
    JOIN Products p ON ci.ProductID = p.ProductID
    WHERE ci.CartID = @CartID;

    INSERT INTO Orders (UserAccountID, OrderDate, OrderStatus,
OrderTotal)
    VALUES (@UserAccountID, GETDATE(), 'In progress', @OrderTotal);

    DECLARE @OrderID INT = SCOPE_IDENTITY();

    INSERT INTO OrderItems (OrderID, UserAccountID, ProductID,
Quantity)
    SELECT @OrderID, @UserAccountID, ProductID, Quantity
```

```
    FROM CartItems
    WHERE CartID = @CartID;

    UPDATE ShoppingCart
    SET Status = 'Completed'
    WHERE CartID = @CartID;
END;
```

4. Uiszczenie opłaty za rejestrację na studia bądź za zjazd

Przypisuje płatność do zamówienia na studia lub zjazd i zmienia status zamówienia na "Wysłane". Aktualizuje zamówienie o identyfikator płatności.

```
CREATE PROCEDURE PayForStudy
    @OrderID INT,
    @PaymentID INT
AS
BEGIN
    UPDATE Orders
    SET PaymentID = @PaymentID, OrderStatus = 'Sent'
    WHERE OrderID = @OrderID;
END;
```

5. Generowanie linku do płatności

Generuje URL umożliwiający dokonanie płatności dla zamówienia. Link jest dynamicznie tworzony na podstawie identyfikatora zamówienia.

```
CREATE PROCEDURE GeneratePaymentLink
    @OrderID INT,
    @PaymentURL NVARCHAR(500) OUTPUT
AS
BEGIN
    SET @PaymentURL =
CONCAT('https://paymentgateway.com/pay?orderId=', @OrderID);
END;
```

6. Rejestracja udanej opłaty

Rejestruje płatność dla zamówienia i zmienia jego status na "Wysłane". Umożliwia śledzenie opłaconych zamówień w systemie.

```
CREATE PROCEDURE RegisterPayment
    @PaymentID INT,
    @OrderID INT
AS
BEGIN
    UPDATE Orders
    SET PaymentID = @PaymentID, OrderStatus = 'Sent'
    WHERE OrderID = @OrderID;
END;
```

7. Aktualizacja statusu zamówienia

Pozwala na aktualizację statusu zamówienia do dowolnej wartości określonej przez użytkownika. Umożliwia elastyczne zarządzanie stanem zamówień.

```
CREATE PROCEDURE UpdateOrderStatus
    @OrderID INT,
    @NewStatus VARCHAR(20)
AS
BEGIN
    UPDATE Orders
    SET OrderStatus = @NewStatus
    WHERE OrderID = @OrderID;
END;
```

8. Dodanie płatności

```
CREATE PROCEDURE AddPayment
    @PaymentID INT,
    @UserAccountID INT,
    @CurrencyID INT,
```

```
    @PaymentStatus VARCHAR(20),
    @PaymentDate DATETIME
AS
BEGIN
    INSERT INTO Payments (PaymentID, UserAccountID, CurrencyID,
PaymentStatus, PaymentDate )
    VALUES (@PaymentID, @UserAccountID, @CurrencyID, @PaymentStatus,
@PaymentDate );
END;
```

9. Stworzenie koszyka

```
CREATE PROCEDURE AddShoppingCart
    @UserAccountID INT,
    @Status VARCHAR(20),
    @CreatedAt DATETIME
AS
BEGIN
    INSERT INTO ShoppingCart (UserAccountID, Status, CreatedAt )
    VALUES (@UserAccountID, @Status, @CreatedAt );
END;
```

10. Dodanie waluty

```
CREATE PROCEDURE AddCurrency
    @CurrencyName VARCHAR(50),
    @CurrencyCode VARCHAR(10),
    @ExchangeRate DECIMAL(10, 4)
AS
BEGIN
    INSERT INTO Currency (CurrencyName, CurrencyCode, ExchangeRate )
    VALUES (@CurrencyName, @CurrencyCode, @ExchangeRate );
END;
```

Webinary

1. Dostęp do szczegółowych danych webinaru

Zwraca szczegółowe dane o webinarze, takie jak opis, czas trwania czy język, jeśli użytkownik jest zapisany. Ignoruje usunięte webinary.

```
CREATE PROCEDURE GetWebinarDetails
    @WebinarID INT
AS
BEGIN
    SELECT *
    FROM Webinars
    WHERE WebinarID = @WebinarID AND IsDeleted = 0;
END;
```

2. Dostęp do materiałów z webinaru

Umożliwia dostęp do materiałów z webinaru po opłacie. W przypadku braku płatności wyświetla komunikat o braku dostępu.

```
CREATE PROCEDURE AccessWebinarMaterials
    @WebinarID INT,
    @UserID INT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM Orders o
        JOIN OrderItems oi ON o.OrderID = oi.OrderID
        WHERE o.UserAccountID = @UserID AND oi.ProductID = @WebinarID
AND o.OrderStatus = 'Sent'
    )
    BEGIN
        SELECT VideoLink
        FROM Webinars
```

```
        WHERE WebinarID = @WebinarID;
    END
    ELSE
    BEGIN
        RAISERROR ('No access to webinar’s material. Process payment
to get access.', 16, 1);
    END
END;
```

3. Dodawanie webinaru do oferty

Pracownik sekretariatu może dodać nowy webinar, określając jego szczegóły, takie jak tytuł, opis i prowadzący. Domyślnie nowy webinar jest płatny.

```
CREATE PROCEDURE AddWebinar
    @Title NVARCHAR(255),
    @InstructorID INT,
    @Description NVARCHAR(1000),
    @DateTime DATETIME,
    @Duration INT,
    @Language NVARCHAR(50)
AS
BEGIN
    INSERT INTO Webinars (Title, InstructorID, Description, DateTime,
Duration, Language, IsFree)
    VALUES (@Title, @InstructorID, @Description, @DateTime,
@Duration, @Language, 0);
END;
```

4. Usuwanie webinaru z oferty

Usuwa webinar z oferty, oznaczając go jako usunięty. Dzięki temu webinar nie jest już widoczny dla użytkowników.

```
CREATE PROCEDURE RemoveWebinar
    @WebinarID INT
```

```
AS
BEGIN
    UPDATE Webinars
    SET IsDeleted = 1
    WHERE WebinarID = @WebinarID;
END;
```

5. Umożliwienie dostępu bez opłaty

Pozwala dyrektorowi na oznaczenie webinaru jako darmowego. Ułatwia dostęp użytkownikom bez konieczności opłat.

```
CREATE PROCEDURE MakeWebinarFree
    @WebinarID INT
AS
BEGIN
    UPDATE Webinars
    SET IsFree = 1
    WHERE WebinarID = @WebinarID;
END;
```

6. Usunięcie nagrania webinaru

Administrator może usunąć nagranie z webinaru, ustawiając pusty link do wideo. Usunięcie może być wymagane ze względów prawnych lub technicznych.

```
CREATE PROCEDURE RemoveWebinarRecording
    @WebinarID INT
AS
BEGIN
    UPDATE Webinars
    SET VideoLink = NULL
    WHERE WebinarID = @WebinarID;
END;
```

7. Dodawanie tłumaczenia do webinaru

Tłumacz może dodać tłumaczenie do webinaru, wprowadzając swoje dane. Oznacza webinar jako przetłumaczony.

```
CREATE PROCEDURE AddWebinarTranslation
    @WebinarID INT,
    @TranslatorFirstName NVARCHAR(100),
    @TranslatorLastName NVARCHAR(100)
AS
BEGIN
    UPDATE Webinars
    SET IsTranslated = 1, TranslatorFirstName = @TranslatorFirstName,
TranslatorLastName = @TranslatorLastName
    WHERE WebinarID = @WebinarID;
END;
```

8. Modyfikacja danych webinaru

Prowadzący może modyfikować szczegóły webinaru, takie jak tytuł, opis, czas trwania czy data. Procedura umożliwia aktualizację istniejących wydarzeń.

```
CREATE PROCEDURE ModifyWebinar
    @WebinarID INT,
    @Title NVARCHAR(255),
    @Description NVARCHAR(1000),
    @DateTime DATETIME,
    @Duration INT
AS
BEGIN
    UPDATE Webinars
    SET Title = @Title, Description = @Description, DateTime =
@DateTime, Duration = @Duration
    WHERE WebinarID = @WebinarID;
END;
```

9. Przypisanie tłumacza do webinaru

Pozwala prowadzącemu przypisać tłumacza do webinaru, dodając jego dane do systemu. Webinar zostaje oznaczony jako przetłumaczony.

```
CREATE PROCEDURE AssignTranslatorToWebinar
    @WebinarID INT,
    @TranslatorFirstName NVARCHAR(100),
    @TranslatorLastName NVARCHAR(100)
AS
BEGIN
    UPDATE Webinars
    SET TranslatorFirstName = @TranslatorFirstName,
TranslatorLastName = @TranslatorLastName, IsTranslated = 1
    WHERE WebinarID = @WebinarID;
END;
```

10. Zarządzanie dostępem do nagrania webinaru

Prowadzący może zarządzać dostępem do nagrania webinaru, ustawiając lub modyfikując link dostępu. Procedura zwiększa kontrolę nad udostępnianiem materiałów.

```
CREATE PROCEDURE ManageWebinarRecordingAccess
    @WebinarID INT,
    @AccessLink NVARCHAR(500)
AS
BEGIN
    UPDATE Webinars
    SET AccessLink = @AccessLink
    WHERE WebinarID = @WebinarID;
END;
```

11.Zapisanie uczestnika na webinar po opłacie

System automatycznie zapisuje uczestnika na webinar po zarejestrowaniu płatności. Wprowadza dane uczestnika do listy zapisanych.

```
CREATE PROCEDURE RegisterParticipantToWebinar
    @WebinarID INT,
    @UserID INT
AS
BEGIN
    INSERT INTO WebinarParticipants (WebinarID, UserID)
    VALUES (@WebinarID, @UserID);
END;
```

Kursy

1. **Przeglądanie oferty kursów i jego modułów**

Goście mogą przeglądać dostępne kursy wraz z ich modułami. Procedura zwraca informacje o kursach oraz powiązanych częściach programu.

```
CREATE PROCEDURE ViewCoursesAndModules
AS
BEGIN
    SELECT C.CourseID, C.Title, C.Description, M.ModuleID, M.Title AS
ModuleTitle
    FROM Courses C
    LEFT JOIN Modules M ON C.CourseID = M.CourseID;
END;
```

2. **Dostęp do szczegółowych danych kursu**

Uczestnicy mają dostęp do szczegółowych danych kursu, takich jak tytuł, opis czy daty trwania. Informacje są zwracane na podstawie identyfikatora kursu.

```
CREATE PROCEDURE GetCourseDetails
```

```
    @CourseID INT
AS
BEGIN
    SELECT *
    FROM Courses
    WHERE CourseID = @CourseID;
END;
```

3. Dodawanie kursu przez pracownika sekretariatu

Pracownik sekretariatu może dodać nowy kurs, określając szczegóły, takie jak opis, daty i tryb nauczania. Procedura wspiera kursy hybrydowe i tradycyjne.

```
CREATE PROCEDURE AddCourse
    @Title NVARCHAR(255),
    @Description NVARCHAR(MAX),
    @StartDate DATETIME,
    @EndDate DATETIME,
    @IsHybrid BIT
AS
BEGIN
    INSERT INTO Courses (Title, Description, StartDate, EndDate,
IsHybrid)
    VALUES (@Title, @Description, @StartDate, @EndDate, @IsHybrid);
END;
```

4. Modyfikacja harmonogramu kursu przez pracownika sekretariatu

Umożliwia modyfikację harmonogramu kursu przez pracownika sekretariatu. Aktualizuje daty rozpoczęcia i zakończenia kursu.

```
CREATE PROCEDURE UpdateCourseSchedule
```

```
    @CourseID INT,
    @StartDate DATETIME,
    @EndDate DATETIME
AS
BEGIN
    UPDATE Courses
    SET StartDate = @StartDate, EndDate = @EndDate
    WHERE CourseID = @CourseID;
END;
```

5. **Generowanie dyplomów przez pracownika sekretariatu**

Pracownik sekretariatu może wygenerować dyplom dla uczestnika po ukończeniu kursu. Procedura rejestruje dane wydania dyplomu w systemie.

```
 CREATE PROCEDURE GenerateDiploma
    @UserID INT,
    @CourseID INT
AS
BEGIN
    INSERT INTO Diplomas (UserID, CourseID, IssuedDate, Address,
IsSent)
    VALUES (@UserID, @CourseID, GETDATE(), '', 0);
END;
```

6. **Umożliwienie dostępu bez opłaty przez dyrektora**

Dyrektor może umożliwić uczestnikowi darmowy dostęp do kursu. Procedura automatycznie zapisuje uczestnika na kurs.

```
CREATE PROCEDURE GrantFreeAccess
    @UserID INT,
    @CourseID INT
AS
```

```
BEGIN
    INSERT INTO Enrollments (UserID, CourseID)
    VALUES (@UserID, @CourseID);
END;
```

7. Usuwanie kursu przez administratora

Administrator może usunąć kurs z systemu, usuwając jego rekord z bazy danych. Procedura pozwala na trwałe usunięcie kursu.

```
 CREATE PROCEDURE DeleteCourse
    @CourseID INT
AS
BEGIN
    DELETE FROM Courses
    WHERE CourseID = @CourseID;
END;
```

8. Dodawanie tłumaczenia przez tłumacza

Tłumacz może dodać tłumaczenie do modułu kursu. Wprowadza tekst tłumaczenia do odpowiedniej tabeli.

```
CREATE PROCEDURE AddModuleTranslation
    @ModuleID INT,
    @Translation NVARCHAR(MAX)
AS
BEGIN
    INSERT INTO ModuleTranslations (ModuleID, Translation)
    VALUES (@ModuleID, @Translation);
END;
```

9. Przypisanie tłumacza do kursu przez prowadzącego

Prowadzący może przypisać tłumacza do kursu, wprowadzając dane do systemu. Procedura wspiera wielojęzyczne kursy.

```
CREATE PROCEDURE AssignTranslator
    @TranslatorID INT,
    @CourseID INT
AS
BEGIN
    INSERT INTO Translators (UserID, CourseID)
    VALUES (@TranslatorID, @CourseID);
END;
```

10. Dodawanie, modyfikacja i usuwanie modułów kursu przez prowadzącego

Prowadzący może zarządzać modułami kursu, dodając, modyfikując lub usuwając moduły. Procedura umożliwia elastyczne zarządzanie zawartością kursu.

```
CREATE PROCEDURE ManageModules
    @Action NVARCHAR(10),
    @ModuleID INT = NULL,
    @CourseID INT = NULL,
    @Title NVARCHAR(255) = NULL,
    @ModuleType NVARCHAR(50) = NULL,
    @Location NVARCHAR(255) = NULL
AS
BEGIN
    IF @Action = 'Add'
        INSERT INTO Modules (CourseID, Title, ModuleType, Location)
        VALUES (@CourseID, @Title, @ModuleType, @Location);
    ELSE IF @Action = 'Update'
        UPDATE Modules
        SET Title = @Title, ModuleType = @ModuleType, Location =
@Location
        WHERE ModuleID = @ModuleID;
    ELSE IF @Action = 'Delete'
```

```
        DELETE FROM Modules
        WHERE ModuleID = @ModuleID;
END;
```

11. Zapis uczestnika na kurs (System)

Procedura zapisuje użytkownika na wybrany kurs, dodając rekord do tabeli rejestracji. Pozwala to śledzić, które kursy zostały przypisane konkretnym uczestnikom.

```
 CREATE PROCEDURE EnrollParticipant
    @UserID INT,
    @CourseID INT
AS
BEGIN
    INSERT INTO Enrollments (UserID, CourseID)
    VALUES (@UserID, @CourseID);
END;
```

12. Udzielenie dostępu do kursu (System)

Procedura aktualizuje istniejący wpis w tabeli zapisów, przypisując użytkownika do wybranego kursu. Jest używana w przypadku zmiany dostępu do kursów dla danego uczestnika.

```
CREATE PROCEDURE GrantCourseAccess
    @UserID INT,
    @CourseID INT
AS
BEGIN
    UPDATE Enrollments
    SET CourseID = @CourseID
    WHERE UserID = @UserID;
END;
```

13. Weryfikacja obecności (System)

Procedura sprawdza obecność użytkownika na danym kursie na podstawie istniejących wpisów w tabeli obecności. Umożliwia to łatwą weryfikację uczestnictwa w zajęciach.

```
CREATE PROCEDURE VerifyAttendance
    @UserID INT,
    @CourseID INT
AS
BEGIN
    SELECT *
    FROM Attendance
    WHERE EnrollmentID = (
        SELECT EnrollmentID
        FROM Enrollments
        WHERE UserID = @UserID AND CourseID = @CourseID
    );
END;
```

14. Dodanie wykładowcy

Procedura zapisuje nowego wykładowcę w tabeli prowadzących dla wybranego kursu. Jest to kluczowe dla przypisania odpowiedniej osoby do nauczania na kursie.

```
CREATE PROCEDURE AddLecturer
    @UserID INT, @CourseID INT
AS
BEGIN
    INSERT INTO Lecturers (UserID, CourseID)
    VALUES (@UserID, @CourseID);
END;
```

Studia

1. Przeglądanie oferty kierunków studiów

Procedura wyświetla listę dostępnych kierunków studiów, które mają daty rozpoczęcia w przyszłości. Dzięki temu potencjalni kandydaci mogą zapoznać się z aktualnymi ofertami.

```
CREATE PROCEDURE ViewStudyOffers
AS
BEGIN
    SELECT StudyID, Title, Description, StartDate, EndDate,
NumberOfSpots, EntryFee, Language
    FROM Studies
    WHERE StartDate > GETDATE();
END;
GO
```

2. Przeglądanie modułów studiów

Procedura pokazuje szczegóły modułów przypisanych do wybranego kierunku studiów. Pozwala to użytkownikowi na zapoznanie się z zawartością danego programu.

```
CREATE PROCEDURE ViewStudyModules
    @StudyID INT
AS
BEGIN
    SELECT StudyProgramID, Name, Description, StartDate, EndDate,
NumberOfSpots, Notes
    FROM StudyPrograms
    WHERE StudyID = @StudyID;
END;
GO
```

3. Dostęp do szczegółowych danych studiów

Procedura pobiera wszystkie szczegóły dotyczące wybranego kierunku studiów. Uczestnik może uzyskać pełne informacje na temat programu, dat czy języka prowadzenia zajęć.

```
CREATE PROCEDURE GetStudyDetails
    @StudyID INT
AS
BEGIN
    SELECT *
    FROM Studies
    WHERE StudyID = @StudyID;
END;
GO
```

4. Dodawanie kierunku studiów

Procedura wprowadza nowy kierunek studiów do bazy danych, zapisując wszystkie niezbędne informacje, takie jak program, daty i opłaty. Jest wykorzystywana do rozbudowy oferty edukacyjnej.

```
CREATE PROCEDURE AddStudy
    @Title NVARCHAR(100),
    @Program TEXT,
    @StartDate DATETIME,
    @EndDate DATETIME,
    @NumberOfSpots INT,
    @EntryFee DECIMAL(10, 2),
    @Language NVARCHAR(50),
    @Description TEXT
AS
BEGIN
    INSERT INTO Studies (Title, Program, StartDate, EndDate,
NumberOfSpots, EntryFee, Language, Description)
```

```
    VALUES (@Title, @Program, @StartDate, @EndDate, @NumberOfSpots,
@EntryFee, @Language, @Description);
END;
GO
```

5. Modyfikacja kierunku studiów

Procedura aktualizuje szczegóły istniejącego kierunku studiów. Umożliwia wprowadzenie zmian w programie, liczbie miejsc czy terminach realizacji.

```
CREATE PROCEDURE ModifyStudy
    @StudyID INT,
    @Title NVARCHAR(100),
    @Program TEXT,
    @StartDate DATETIME,
    @EndDate DATETIME,
    @NumberOfSpots INT,
    @EntryFee DECIMAL(10, 2),
    @Language NVARCHAR(50),
    @Description TEXT
AS
BEGIN
    UPDATE Studies
    SET Title = @Title,
        Program = @Program,
        StartDate = @StartDate,
        EndDate = @EndDate,
        NumberOfSpots = @NumberOfSpots,
        EntryFee = @EntryFee,
        Language = @Language,
        Description = @Description
    WHERE StudyID = @StudyID;
END;
GO
```

6. Generowanie dyplomów

Procedura tworzy nowy wpis w tabeli dyplomów, przypisując go do uczestnika i określonego kierunku studiów. Proces ten dokumentuje zakończenie edukacji przez uczestnika.

```
CREATE PROCEDURE GenerateDiploma
    @UserID INT,
    @StudyID INT
AS
BEGIN
    INSERT INTO Diplomas (UserID, StudyID, IssueDate)
    VALUES (@UserID, @StudyID, GETDATE());
END;
GO
```

7. Zapisywanie uczestnika na studia

Procedura dodaje użytkownika do tabeli rejestracji kierunku studiów z bieżącą datą zapisu. Automatyzuje proces rejestracji i śledzenia uczestnictwa.

```
CREATE PROCEDURE EnrollParticipant
    @UserID INT,
    @StudyID INT
AS
BEGIN
    INSERT INTO Enrollments (UserID, StudyID, EnrollmentDate)
    VALUES (@UserID, @StudyID, GETDATE());
END;
GO
```

8. Przypisanie tłumacza do zajęć

Procedura przypisuje tłumacza do konkretnej sesji zajęć. Ułatwia to organizację zajęć w wielojęzycznych grupach.

```
CREATE PROCEDURE AssignTranslator
    @SessionID INT,
    @TranslatorID INT
AS
BEGIN
    UPDATE Sessions
    SET TranslatorID = @TranslatorID
    WHERE SessionID = @SessionID;
END;
GO
```

9. Dodawanie spotkania

Procedura wprowadza nowe spotkanie do harmonogramu zajęć, określając datę, temat i inne szczegóły. Pomaga w zarządzaniu i organizacji sesji edukacyjnych.

```
CREATE PROCEDURE AddSession
    @ScheduleID INT,
    @StudyID INT,
    @SessionDate DATETIME,
    @SessionType VARCHAR(20),
    @Topic VARCHAR(255),
    @NumberOfSpots INT,
    @Notes TEXT = NULL
AS
BEGIN
    INSERT INTO Sessions (ScheduleID, StudyID, SessionDate,
SessionType, Topic, NumberOfSpots, Notes )
    VALUES (@ScheduleID, @StudyID, @SessionDate, @SessionType,
@Topic, @NumberOfSpots, @Notes );
END;
GO
```

10. Dodawanie egzaminu

Procedura umożliwia dodanie nowego wpisu w tabeli egzaminów, określając użytkownika, kierunek studiów, datę egzaminu oraz jego wynik. Ułatwia to zarządzanie danymi dotyczącymi ocen i harmonogramów egzaminacyjnych.

```
CREATE PROCEDURE AddExam
    @UserID INT,
    @StudyID INT,
    @ExamDate DATETIME,
    @ExamScore DECIMAL(3, 2)
AS
BEGIN
    INSERT INTO Exams (UserID, StudyID, ExamDate, ExamScore)
VALUES (@UserID, @StudyID, @ExamDate, @ExamScore);
END;
GO
```

11. Dodawanie stażu

Procedura wprowadza dane o nowym stażu dla uczestnika, zawierające daty rozpoczęcia i zakończenia, status, przypisanie do programu studiów oraz szczegóły okresu i roku akademickiego. Umożliwia dokumentowanie przebiegu staży w ramach programu studiów.

```
CREATE PROCEDURE AddInternship
    @UserID INT,
    @StartDate DATETIME,
    @EndDate DATETIME,
    @Status VARCHAR(20) = 'In progress',
    @StudyProgramID INT,
    @Year INT,
    @Period INT
AS
BEGIN
    INSERT INTO Internships (UserID, StartDate, EndDate, Status,
StudyProgramID, Year, Period )
    VALUES (@UserID, @StartDate, @EndDate, @Status, @StudyProgramID,
@Year, @Period );
END;
GO
```

12. Dodawanie harmonogramu studiów

Procedura dodaje nowy harmonogram zajęć, określając semestr, datę początkową i końcową. Pomaga w zarządzaniu i organizacji zajęć.

```
CREATE PROCEDURE AddSchedule
    @StudyProgramID INT,
    @Semester INT,
    @StartDate DATETIME,
    @EndDate DATETIME
AS
BEGIN
    INSERT INTO Schedule (StudyProgramID, Semester, StartDate,
EndDate )
    VALUES (@StudyProgramID, @Semester, @StartDate, @EndDate );
END;
GO
```

13. Dodawanie programu studiów

Procedura dodaje nowy program studiów, określając jego nazwę, datę początkową i końcową, liczbę dostępnych miejsc i opis.

```
CREATE PROCEDURE AddStudyProgram
    @StudyID INT,
    @Name NVARCHAR(255),
    @Description NVARCHAR(1000) = NULL,
    @StartDate DATETIME,
    @EndDate DATETIME,
    @NumberOfSpots INT,
    @Notes NVARCHAR(1000) = NULL
AS
BEGIN
    INSERT INTO StudyPrograms (StudyID, Name, Description, StartDate,
EndDate, NumberOfSpots, Notes )
    VALUES (@StudyID, @Name, @Description, @StartDate, @EndDate,
@NumberOfSpots, @Notes );
END;
```

```
GO
```

TRIGGERY

Trigger aktualizuje status zamówienia w tabeli `Orders` po zmianie statusu płatności w tabeli `Payments`. Jeśli status płatności zostanie zmieniony na "Paid", "Unpaid" lub inny, status powiązanego zamówienia zostaje odpowiednio zaktualizowany na "Sent", "In progress" lub "Cancelled".

```
CREATE TRIGGER trg_update_order_status
ON dbo.Payments
AFTER INSERT
AS
BEGIN

    IF EXISTS(SELECT 1 FROM inserted WHERE PaymentStatus IS NOT
NULL)
    BEGIN
        UPDATE o
        SET o.OrderStatus = CASE
            WHEN i.PaymentStatus = 'Paid' THEN 'Sent'
            WHEN i.PaymentStatus = 'Unpaid' THEN 'In progress'
            ELSE 'Cancelled'
        END
        FROM dbo.Orders o
        INNER JOIN inserted i ON o.PaymentID = i.PaymentID
        WHERE i.PaymentID IN (SELECT PaymentID FROM inserted);
    END
END;
```

Sprawdza, czy data egzaminu (`ExamDate`) wstawiona lub zaktualizowana w tabeli `Exams` jest późniejsza niż obecna data (`GETDATE()`). Jeśli data egzaminu jest przeszła, zgłasza błąd i wycofuje transakcję.

```
CREATE TRIGGER trg_valid_examdate
ON Exams
```

```
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted
        WHERE ExamDate <= GETDATE()
    )
    BEGIN
        RAISERROR ('ExamDate must be a future date.', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Weryfikuje, czy data egzaminu (`ExamDate`) jest późniejsza niż data sesji powiązanej z egzaminem w tabeli `Sessions`. Jeśli egzamin ma datę wcześniejszą lub równą dacie sesji, zgłasza błąd i wycofuje transakcję.

```
CREATE TRIGGER trg_valid_exam_after_session
ON Exams
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Studies s ON i.StudyID = s.StudyID
        WHERE i.ExamDate <= s.EndDate
    )
    BEGIN
        RAISERROR ('ExamDate must be after the SessionDate.',
16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Sprawdza, czy data płatności (PaymentDate) wstawiana lub aktualizowana w tabeli Payments jest późniejsza lub równa dacie zamówienia (OrderDate) z tabeli Orders. Jeśli data płatności jest wcześniejsza niż data zamówienia, trigger zgłasza błąd i przerywa transakcję, zapobiegając zapisaniu nieprawidłowej płatności.

```
CREATE TRIGGER trg_validate_payment_date
ON Payments
AFTER INSERT, UPDATE
AS
BEGIN

    IF EXISTS (
        SELECT 1
        FROM INSERTED i
        JOIN Orders o ON i.PaymentID = o.PaymentID
        WHERE o.OrderDate IS NOT NULL AND i.PaymentDate <
o.OrderDate
    )
    BEGIN
        RAISERROR('End date should be later than start date.',
16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Trigger trg_set_createdat jest wywoływany po wstawieniu nowego rekordu do tabeli UsersAccount. Aktualizuje on kolumnę CreatedAt na bieżącą datę (GETDATE()), ale tylko dla tych rekordów, które zostały wstawione i mają NULL w tej kolumnie.

```
CREATE TRIGGER trg_set_createdat
ON UsersAccount
AFTER INSERT
```

```
AS
BEGIN
    UPDATE UsersAccount
    SET CreatedAt = GETDATE()
    WHERE UserAccountID IN (SELECT UserAccountID FROM INSERTED)
AND CreatedAt IS NULL;
END;
```

Sprawdza, czy data płatności (`PaymentDate`) wstawionych lub zaktualizowanych rekordów w tabeli `Payments` jest wcześniejsza niż data zamówienia (`OrderDate`) w tabeli `Orders`. Jeśli warunek jest spełniony, trigger wywołuje błąd i wycofuje transakcję, zapobiegając zapisaniu niepoprawnych danych.

```
CREATE TRIGGER trg_check_payment_date
ON Payments
AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Orders o ON i.PaymentID = o.PaymentID
        WHERE i.PaymentDate < o.OrderDate
    )
    BEGIN
        RAISERROR('PaymentDate cannot be earlier than
OrderDate', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Sprawdza, czy użytkownik (UserID) wstawiany do tabeli `Enrollments` jest zarejestrowany (IsRegistered = 1). Jeśli użytkownik nie jest zarejestrowany, operacja

wstawiania jest anulowana, a jeśli jest zarejestrowany, dane są wstawiane do tabeli `Enrollments`.

```
CREATE TRIGGER trg_check_user_registered
ON Enrollments
INSTEAD OF INSERT
AS
BEGIN
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN Users u ON i.UserID = u.UserID
        WHERE u.IsRegistered = 0
    )
    BEGIN
        RAISERROR ('User is not registered.', 16, 1);
        ROLLBACK;
        RETURN;
    END

    INSERT INTO Enrollments (UserID, CourseID, WebinarID)
    SELECT UserID, CourseID, WebinarID
    FROM inserted;
END;
GO
```

Ten trigger aktualizuje pole `IsRegistered` w tabeli `Users` po dodaniu lub aktualizacji rekordu. Jeśli użytkownik ma przypisane konto w tabeli `UsersAccount`, `IsRegistered` zostaje ustawione na `1`; w przeciwnym razie, jeśli konto jest nieprzypisane, wartość `IsRegistered` zostaje ustawiona na `0`.

```
CREATE TRIGGER trg_update_is_registered
ON Users
```

```
AFTER INSERT, UPDATE
AS
BEGIN

    UPDATE u
    SET u.IsRegistered = CASE
        WHEN ua.UserID IS NOT NULL THEN 1
        ELSE 0
    END
    FROM Users u
    LEFT JOIN UsersAccount ua ON u.UserID = ua.UserID
    WHERE u.UserID IN (SELECT UserID FROM inserted);
END;
```

Weryfikuje, czy cena produktu (webinaru lub kursu) jest odpowiednia, w zależności od jego typu i dodatkowych warunków. Jeżeli cena nie spełnia określonych zasad (np. cena webinaru, który nie jest darmowy, musi być większa niż 0), operacja zostaje anulowana za pomocą `ROLLBACK`.

```
CREATE TRIGGER trg_validate_price
ON Products
AFTER INSERT, UPDATE
AS
BEGIN
    DECLARE @ProductType VARCHAR(20),
            @Price DECIMAL(10, 2),
            @WebinarID INT,
            @CourseID INT;

    SELECT @ProductType = ProductType, @Price = Price,
@WebinarID = WebinarID, @CourseID = CourseID
    FROM INSERTED;

    -- Warunki walidacji ceny
    IF @ProductType = 'Webinar'
```

```
    BEGIN

        IF @WebinarID IS NOT NULL
        BEGIN
            DECLARE @IsFree BIT;
            SELECT @IsFree = IsFree FROM Webinars WHERE
WebinarID = @WebinarID;

            IF @IsFree = 0 AND @Price <= 0
            BEGIN
                RAISERROR('Cena produktu (Webinar) musi być
większa niż 0, gdy produkt nie jest darmowy.', 16, 1);
                ROLLBACK TRANSACTION;
            END
        END
    END
    ELSE IF @ProductType = 'Course'
    BEGIN
        IF @Price <= 0
        BEGIN
            RAISERROR('Cena produktu (Course) musi być większa
niż 0.', 16, 1);
            ROLLBACK TRANSACTION;
        END
    END

    IF @Price < 0
    BEGIN
        RAISERROR('Cena produktu musi być wartością nieujemną.',
16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

FUNKCJE

Funkcja `CanAccessCourseMaterials` sprawdza, czy użytkownik jest zapisany na określony kurs, przeszukując tabelę `Enrollments` w celu znalezienia rekordu, który pasuje do podanych identyfikatorów użytkownika i kursu. Jeśli użytkownik jest zapisany na kurs, funkcja zwraca wartość `1`, w przeciwnym razie zwraca `0`.

```
CREATE FUNCTION CanUserAccessCourseMaterials (@UserID INT,
@CourseID INT)
RETURNS BIT
AS
BEGIN
    DECLARE @HasAccess BIT = 0;
    IF EXISTS (
        SELECT 1
        FROM Enrollments
        WHERE UserID = @UserID AND CourseID = @CourseID
    )
        SET @HasAccess = 1;

    RETURN @HasAccess;
END;
```

Funkcja `ViewWebinars` zwraca tabelę, która zawiera dane o webinarach, filtrując te, które nie zostały oznaczone jako usunięte (`IsDeleted = 0`). Funkcja ta pozwala na przeglądanie dostępnych webinarów, zwracając kolumny takie jak `WebinarID`, `Title`, `DateTime` oraz `IsFree`.

```
CREATE FUNCTION ViewWebinars()
RETURNS TABLE
AS
RETURN
    SELECT WebinarID, Title, DateTime, IsFree
    FROM Webinars
    WHERE IsDeleted = 0;
```

Funkcja `CalculateCartValue` oblicza całkowitą wartość koszyka (`CartID`)

poprzez sumowanie ceny (`Price`) produktów pomnożoną przez ich ilość (`Quantity`). Jeśli w koszyku nie ma żadnych produktów, funkcja zwróci `0` jako wartość całkowitą.

```
CREATE FUNCTION CalculateCartValue (@CartID INT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @TotalValue DECIMAL(10, 2) = 0;

    SELECT @TotalValue = SUM(ci.Quantity * p.Price)
    FROM CartItems ci
    JOIN Products p ON ci.ProductID = p.ProductID
    WHERE ci.CartID = @CartID;

    IF @TotalValue IS NULL
    BEGIN
        SET @TotalValue = 0;
    END

    RETURN @TotalValue;
END;
```

Oblicza całkowitą wartość zamówienia na podstawie ceny i ilości zamówionych produktów. Zwraca sumę wartości produktów w zamówieniu.

```
CREATE FUNCTION GetTotalOrderValue (@OrderID INT)
RETURNS DECIMAL(10, 2)
AS
BEGIN
    DECLARE @TotalValue DECIMAL(10, 2);

    SELECT @TotalValue = SUM(oi.Quantity * p.Price)
    FROM OrderItems oi
    JOIN Products p ON oi.ProductID = p.ProductID
    WHERE oi.OrderID = @OrderID;
```

```
    RETURN @TotalValue;
END;
```

GENEROWANIE DANYCH

Dane użyte w bazie danych zostały wygenerowane przy pomocy skryptów w języku Python z wykorzystaniem bibliotek random, csv oraz datetime. Dane zostały zapisane do pliku csv, a następnie zaimportowane do odpowiednich tabeli w bazie danych, z ewentualnymi poprawkami zaimplementowanymi w SSMS.