AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Podstawy Baz Danych

Projekt 2024/25

Maciej Kmąk, Jakub Stachecki, Kacper Wdowiak
Informatyka WI AGH, II rok

Spis treści

1 Wprowadzenie

Niniejszy dokument przedstawia szczegółową dokumentację projektu bazy danych, którego celem było zaprojektowanie i implementacja kompleksowego systemu bazodanowego dla firmy oferującej różnego rodzaju kursy i szkolenia. Projekt obejmuje różnorodne aspekty związane z zarządzaniem procesami edukacyjnymi, uwzględniając zarówno hybrydowy model świadczenia usług, jak i specyficzne wymagania dotyczące różnych form kształcenia, takich jak webinary, kursy czy studia.
System został zaprojektowany z myślą o integracji z zewnętrznymi systemami płatności oraz możliwością generowania raportów na potrzeby analityczne i operacyjne. Implementacja projektu została zrealizowana w środowisku MS SQL Server.
Dokument zawiera szczegółowy opis elementów składowych bazy danych, w tym:

- Zdefiniowanych ról i przypisanych im uprawnień,
- Struktury tabel i ich przeznaczenia,
- Relacji pomiędzy tabelami, uwzględniających integralność danych,
- Przygotowanych mechanizmów rozszerzenia funkcjonalności bazy danych, takich jak:
  - Funkcje i procedury (`Functions`, `Stored Procedures`),
  - Triggery, które automatyzują określone operacje w systemie,
  - Widoki (`Views`), umożliwiające łatwy dostęp do przetworzonych danych,
  - Metody generowania danych testowych w celu walidacji systemu,
  - Schemat bazy danych w formie diagramów obrazujących strukturę i zależności.

Projekt został zrealizowany przez studentów kierunku Informatyka na Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie w ramach przedmiotu **Podstawy Baz Danych**.
Autorami projektu są:

- Maciej Kmąk,
- Jakub Stachecki,
- Kacper Wdowiak,

Wkład poszczególnych autorów w implementację systemu został szczegółowo omówiony w późniejszych sekcjach niniejszego dokumentu.
Niniejsza dokumentacja ma na celu nie tylko szczegółowe przedstawienie aspektów technicznych implementacji systemu bazodanowego, ale również dostarczenie pełnego obrazu procesu jego projektowania i wdrażania. Dokumentacja ta może również służyć jako punkt odniesienia przy wdrażaniu dodatkowych modułów, integracji z nowymi technologiami czy dostosowywaniu systemu.

2 Role i uprawnienia

W celu zapewnienia bezpieczeństwa, organizacji dostępu do danych oraz kontroli nad operacjami wykonywanymi w systemie, w bazie danych zostały zdefiniowane role użytkowników. Dzięki temu system umożliwia precyzyjne przypisanie dostępów w zależności od funkcji i obowiązków danej grupy użytkowników.

2.1 Możliwe role użytkowników

W projekcie bazy danych utworzono następujące role:

- **Role_Admin** – administrator systemu,
- **Role_Teacher** – nauczyciel,
- **Role_Student** – student,
- **Role_Translator** – pracownik administarcyjny,
- **Role_Employee** – tłumacz.

Każda z ról ma przypisane uprawnienia, które pozwalają jej użytkownikom na wykonywanie określonych operacji na bazie danych. W kolejnych podrozdziałach zostaną omówione szczegółowe uprawnienia dla każdej roli.

2.2 Uprawnienia przypisane do ról

Każda rola ma ściśle określone uprawnienia, które pozwalają jej użytkownikom na wykonywanie określonych operacji. Uprawnienia te obejmują:

- `SELECT` – odczyt danych z tabeli,
- `INSERT` – wstawianie nowych rekordów,
- `UPDATE` – modyfikacja istniejących danych,
- `DELETE` – usuwanie rekordów,
- `ALTER` – modyfikacja struktury tabeli,
- `REFERENCES` – definiowanie kluczy obcych i relacji między tabelami.

2.2.1 `Role_Admin` (Administrator)

```
GRANT SELECT, INSERT, UPDATE, DELETE, ALTER, REFERENCES
ON SCHEMA::dbo
TO Role_Admin;
```

Administrator ma pełny dostęp do całej bazy danych i może wykonywać wszystkie operacje. Użytkownicy tej roli mogą:

- Tworzyć, modyfikować i usuwać wszystkie obiekty bazy danych (tabele, widoki, procedury, funkcje itp.).
- Przeglądać, edytować i usuwać dane w każdej tabeli.
- Zarządzać użytkownikami i nadawać uprawnienia innym rolom.

Tym samym `Role_Admin` jest przeznaczona wyłącznie dla osób, które odpowiadają za techniczną administrację i utrzymanie bazy danych.

2.2.2 `Role_Teacher` (Nauczyciel)

```
GRANT SELECT, UPDATE ON dbo.CoursesAttendance TO Role_Teacher;
GRANT SELECT, UPDATE ON dbo.StudiesClassAttendance TO Role_Teacher;
GRANT SELECT ON dbo.Students TO Role_Teacher;
GRANT SELECT ON dbo.Courses TO Role_Teacher;
GRANT SELECT ON dbo.Studies TO Role_Teacher;
GRANT SELECT ON dbo.Translators TO Role_Teacher;
GRANT SELECT, INSERT, UPDATE ON dbo.SubjectGrades TO Role_Teacher;
```

Nauczyciele mają dostęp do danych dotyczących ich kursów oraz studentów uczestniczących w zajęciach. Uprawnienia tej roli obejmują:

- Zarządzanie obecnością studentów (`CoursesAttendance`, `StudiesClassAttendance`).
- Przeglądanie listy studentów oraz kursów.
- Wystawianie ocen (`SubjectGrades`).

Rola `Role_Teacher` nie ma praw do zarządzania krytycznymi elementami systemu, takimi jak `Employee` czy `Orders`, co pozwala ograniczyć jej zakres wyłącznie do działań dydaktycznych.

2.2.3 `Role_Student` (Student)

```
GRANT SELECT ON dbo.Activities TO Role_Student;
GRANT SELECT ON dbo.Courses TO Role_Student;
GRANT SELECT ON dbo.Webinars TO Role_Student;
GRANT SELECT ON dbo.Studies TO Role_Student;
GRANT SELECT, INSERT, DELETE ON dbo.ShoppingCart TO Role_Student;
GRANT SELECT, INSERT ON dbo.Orders TO Role_Student;
GRANT SELECT, INSERT ON dbo.OrderDetails TO Role_Student;
```

Studenci mają ograniczony dostęp do systemu i mogą jedynie przeglądać informacje o kursach, rejestrować się na nie oraz zarządzać swoimi zamówieniami. Uprawnienia studentów obejmują:

- Przeglądanie kursów, webinarów i studiów (`Activities`, `Courses`, `Webinars`, `Studies`).
- Składanie zamówień i zarządzanie koszykiem (`ShoppingCart`, `Orders`, `OrderDetails`).

2.2.4 `Role_Translator` (Tłumacz)

```
GRANT SELECT ON dbo.Webinars TO Role_Translator;
GRANT SELECT ON dbo.CourseModules TO Role_Translator;
GRANT SELECT ON dbo.Translators TO Role_Translator;
GRANT SELECT ON dbo.TranslatorsLanguages TO Role_Translator;
GRANT UPDATE ON dbo.TranslatorsLanguages TO Role_Translator;
```

Tłumacze w systemie mogą zarządzać danymi dotyczącymi języków i tłumaczeń. Uprawnienia tej roli obejmują:

- Przeglądanie webinarów i modułów kursów.
- Przeglądanie i aktualizowanie przypisanych języków (TranslatorsLanguages).

2.2.5 `Role_Employee` (Pracownik administracyjny)

```
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Orders TO Role_Employee;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.OrderDetails TO Role_Employee;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Studies TO Role_Employee;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Students TO Role_Employee;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Courses TO Role_Employee;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Teachers TO Role_Employee;
GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Translators TO Role_Employee;
GRANT SELECT, UPDATE ON dbo.OrderPaymentStatus TO Role_Employee;
GRANT SELECT, UPDATE ON dbo.RODO_Table TO Role_Employee;
```

Rola pracownika została stworzona dla użytkowników odpowiedzialnych za administracyjne aspekty funkcjonowania systemu, takie jak obsługa zamówień, studentów i kursów. Pracownik może:

- Zarządzać zamówieniami i ich szczegółami (`Orders`, `OrderDetails`).
- Dodawać, edytować i usuwać kursy, studia i studentów.
- Aktualizować status płatności zamówień (`OrderPaymentStatus`).
- Zarządzać danymi RODO (`RODO_Table`).

Pracownicy nie mają pełnych praw administracyjnych – np. nie mogą modyfikować struktury bazy czy nadawać uprawnień innym użytkownikom, co jest domeną Administratora – `Role_Admin`.

2.3 Podsumowanie zakresu ról

Jak wynika z powyższych opisów, każda rola ma jasno wydzielony obszar kompetencji.
Zdefiniowany w ten sposób **podział ról** pozwala na *precyzyjne rozdzielenie kompetencji* oraz zapewnia *bezpieczeństwo* i *porządek* w systemie. Możliwe jest także doprecyzowanie niektórych uprawnień (np. jedynie `SELECT` w konkretnej tabeli, a `INSERT` w innej), zgodnie z potrzebami organizacji.

3 Struktura wszystkich tabel

W poniższych podsekcjach opisano **wszystkie tabele** zdefiniowane w bazie danych. Dla każdej tabeli przedstawiono:

- **Kolumny** (z typami danych, NOT NULL, itp.).
- Klucz główny (PRIMARY KEY).
- Przykładowe przeznaczenie/wykorzystanie danej tabeli w systemie.

3.1 Activities

- **Nazwa tabeli**: `Activities`
- **Kolumny**:
  - `ActivityID` (INT, NOT NULL) – klucz główny (`PK_Activities`)
  - `Price` (MONEY, NOT NULL)
  - `Title` (VARCHAR(50), NOT NULL)
  - `Active` (BIT, NOT NULL)
- **Opis / przeznaczenie**: Tabela `Activities` zawiera ogólne informacje o dostępnych aktywnościach, takich jak kursy, webinary czy studia. Każda aktywność ma swoją unikalną cenę, tytuł oraz status aktywności. Tabela `Activities` jest nadrzędną encją dla różnych typów zajęć, które można wykupić lub zapisać się na nie (np.`Courses`, `Webinars`, `Studies`).

3.2 Buildings

- **Nazwa tabeli**: `Buildings`
- **Kolumny**:
  - `ClassID` (INT, NOT NULL) – klucz główny (`PK_Buildings`)
  - `BuildingName` (VARCHAR(30), NOT NULL)
  - `RoomNumber` (VARCHAR(30), NOT NULL)
- **Opis / przeznaczenie**: Tabela `Buildings` przechowuje informacje o budynkach i salach, w których odbywają się zajęcia stacjonarne.

3.3 Cities

- **Nazwa tabeli**: `Cities`
- **Kolumny**:
  - `CityID` (INT, NOT NULL) – klucz główny (`PK_Cities`)
  - `CityName` (VARCHAR(40), NOT NULL)
  - `CountryID` (INT, NOT NULL) – klucz obcy do `Countries`
- **Opis / przeznaczenie**: Spis miast. Każde miasto jest przypisane do państwa (`CountryID`). Służy do przechowywania adresów studentów (tabela `Students`).

3.4 Countries

- **Nazwa tabeli**: `Countries`
- **Kolumny**:
  - `CountryID` (INT, NOT NULL) – klucz główny (`PK_Countries`)
  - `CountryName` (VARCHAR(40), NOT NULL)
- **Opis / przeznaczenie**: Lista krajów. Z nią powiązana jest tabela `Cities`, przechowująca konkretne miasta. Wykorzystywana w tabeli `Cities`.

3.5 CourseModules

- **Nazwa tabeli**: `CourseModules`
- **Kolumny**:
  - `ModuleID` (INT, NOT NULL) – klucz główny (`PK_CourseModules`)
  - `CourseID` (INT, NOT NULL) – FK do `Courses`
  - `ModuleName` (VARCHAR(50), NOT NULL)
  - `Date` (DATETIME, NOT NULL)
  - `DurationTime` (TIME(0), NOT NULL)
  - `TeacherID` (INT, NOT NULL) – FK do `Teachers`
  - `TranslatorID` (INT, NULL) – FK do `Translators`
  - `LanguageID` (INT, NOT NULL) – FK do `Languages`
- **Opis / przeznaczenie**: Poszczególne moduły w obrębie kursu (np. lekcje, bloki tematyczne). Zawiera informacje o prowadzącym (`TeacherID`) i ewentualnym tłumaczu (`TranslatorID`).

3.6 CourseParticipants

- **Nazwa tabeli**: `CourseParticipants`
- **Klucz główny**: (`CourseID`, `StudentID`)
- **Opis / przeznaczenie**: Informuje o udziale konkretnego studenta (`StudentID`) w danym kursie (`CourseID`). Każdy student może być przypisany do wielu kursów, a każdy kurs może mieć wielu studentów.

3.7 Courses

- **Nazwa tabeli**: `Courses`
- **Kolumny** (m.in.):
  - `CourseID` (INT, NOT NULL) – PK
  - `ActivityID` (INT, NOT NULL) – FK do `Activities`
  - `CourseName` (VARCHAR(50), NOT NULL)
  - `CourseDescription` (TEXT, NULL)
  - `CoursePrice` (MONEY, NOT NULL)
  - `CourseCoordinatorID` (INT, NOT NULL) – FK do `Teachers`
- **Opis / przeznaczenie**: Tabela `Courses` przechowuje kursy (nazwa, opis, cena), które mogą być dostępne dla studentów w ramach różnych aktywności, łączy się z `Activities`.

3.8 CoursesAttendance

- **Nazwa tabeli**: `CoursesAttendance`
- **Klucz główny**: (`ModuleID`, `StudentID`)
- **Opis / przeznaczenie**: Tabela służy do monitorowania frekwencji studentów na zajęciach.

3.9 EmployeeTypes

- **Nazwa tabeli**: `EmployeeTypes`
- **Kolumny** (m.in.):
  - `EmployeeTypeID` (INT, NOT NULL) – PK
  - `EmployeeTypeName` (VARCHAR(30), NOT NULL)
- **Opis / przeznaczenie**: Typ pracownika (np. administracyjny, kadra zarządzająca, techniczny itp.). Powiązane z `Employees`.

3.10 Employees

- **Nazwa tabeli**: `Employees`
- **Kolumny** (m.in.):
  - `EmployeeID` (INT, NOT NULL) – PK
  - `FirstName`, `LastName` (VARCHAR(30), NOT NULL)
  - `HireDate` (DATE, NULL)
  - `EmployeeTypeID` (INT, NOT NULL) – FK do `EmployeeTypes`
  - `Phone` (VARCHAR(15), NULL)
  - `Email` (VARCHAR(60), NOT NULL)
- **Opis / przeznaczenie**: Tabela przechowuje listę osób zatrudnionych.

3.11 EuroExchangeRate

- **Nazwa tabeli**: `EuroExchangeRate`
- **Klucz główny**: `[Date]` (DATETIME)
- **Kolumny**:
  - `Date` (DATETIME, NOT NULL) – PK
  - `Rate` (DECIMAL(10,2), NOT NULL)
- **Opis / przeznaczenie**: Zawiera kurs wymiany waluty (EUR) w danym dniu.

3.12 Internship

- **Nazwa tabeli**: `Internship`
- **Klucz główny**: `InternshipID` (INT)
- **Kolumny** (m.in.):
  - `InternshipID` (INT, NOT NULL)
  - `StudiesID` (INT, NOT NULL) – FK do `Studies`
  - `StartDate` (DATETIME, NOT NULL)
- **Opis / przeznaczenie**: Tabela `Internship` przechowuje informacje o praktykach studenckich związanych z danym kierunkiem studiów.

3.13 InternshipAttendance

- **Nazwa tabeli**: `InternshipAttendance`
- **Klucz główny**: (`InternshipID`, `StudentID`)
- **Opis / przeznaczenie**: Tabela `InternshipAttendance` przechowuje informacje o obecności studentów na praktykach.

3.14 Languages

- **Nazwa tabeli**: `Languages`
- **Kolumny**:
  - `LanguageID` (INT, NOT NULL) – PK
  - `LanguageName` (VARCHAR(40), NOT NULL)
- **Opis / przeznaczenie**: Tabela Languages przechowuje listę dostępnych języków wykorzystywanych w systemie. Wykorzystywana w `CourseModules`, `StudiesClass`, `Webinars`, `TeacherLanguages`, `TranslatorsLanguages`.

3.15 OnlineAsyncClass

- **Nazwa tabeli**: `OnlineAsyncClass`
- **Klucz główny**: `OnlineAsyncClassID` (INT)
- **Opis / przeznaczenie**: Tabela `OnlineAsyncClass` przechowuje informacje o zajęciach asynchronicznych dostępnych online. Są to materiały edukacyjne, np. nagrane wykłady lub kursy wideo, które studenci mogą oglądać w dowolnym czasie.

3.16 OnlineAsyncModule

- **Nazwa tabeli**: `OnlineAsyncModule`
- **Klucz główny**: `OnlineAsyncModuleID` (INT)
- **Opis / przeznaczenie**: Tabela `OnlineAsyncModule` przechowuje informacje o modułach kursowych dostępnych w formie nagranych lekcji wideo, które można oglądać w dowolnym czasie.

3.17 OnlineSyncClass

- **Nazwa tabeli**: `OnlineSyncClass`
- **Klucz główny**: `OnlineSyncClassID` (INT)
- **Opis / przeznaczenie**: Tabela `OnlineSyncClass` przechowuje informacje o zajęciach online odbywających się na żywo, np. przez wideokonferencję.

3.18 OnlineSyncModule

- **Nazwa tabeli**: `OnlineSyncModule`
- **Klucz główny**: `OnlineSyncModuleID` (INT)
- **Opis / przeznaczenie**: Tabela `OnlineSyncModule` przechowuje informacje o modułach kursowych realizowanych w formie zajęć online na żywo, powiązany z `CourseModules`.

3.19 OrderDetails

- **Nazwa tabeli**: `OrderDetails`
- **Klucz główny**: (`OrderID`, `ActivityID`)
- **Opis / przeznaczenie**: Tabela `OrderDetails` przechowuje szczegóły dotyczące zakupionych aktywności w ramach zamówienia.

3.20 OrderPaymentStatus

- **Nazwa tabeli**: `OrderPaymentStatus`
- **Klucz główny**: `PaymentURL` (INT)
- **Kolumny**:
  - `PaymentURL` (INT, NOT NULL) – PK
  - `OrderPaymentStatus` (VARCHAR(20), NOT NULL)
  - `PaidDate` (DATETIME, NULL)
- **Opis / przeznaczenie**: Tabela `OrderPaymentStatus` przechowuje informacje o statusie płatności zamówienia.

3.21 Orders

- **Nazwa tabeli**: `Orders`
- **Kolumny** (m.in.):
  - `OrderID` (INT, NOT NULL) – PK
  - `StudentID` (INT, NOT NULL) – FK do `Students`
  - `OrderDate` (DATETIME, NOT NULL) – FK do `EuroExchangeRate(Date)`
  - `PaymentURL` (INT, NOT NULL) – FK do `OrderPaymentStatus`
  - `EmployeeHandling` (INT, NOT NULL) – FK do `Employees`
- **Opis / przeznaczenie**: Rejestr zamówień składanych przez studentów, wraz z informacją o płatności i obsługującym pracowniku.

3.22 RODO_Table

- **Nazwa tabeli**: `RODO_Table`
- **Klucz główny**: `StudentID` (INT)
- **Kolumny** (m.in.):
  - `StudentID` (INT, NOT NULL)
  - `Date` (DATE, NOT NULL)
  - `Withdraw` (BIT, NOT NULL)
- **Opis / przeznaczenie**: Tabela `RODO_Table` przechowuje informacje dotyczące zgód RODO studentów, w tym datę udzielenia zgody oraz jej ewentualnego wycofania.

3.23 Schedule

- **Nazwa tabeli**: `Schedule`
- **Klucz główny**: `ScheduleID` (INT)
- **Kolumny** (m.in.):
  - `ScheduleID` (INT, NOT NULL)
  - `ClassID` (INT, NOT NULL) – FK do `Buildings`

  - `CourseModuleID` (INT, NULL) – FK do `CourseModules`
  - `StudiesSubjectID` (INT, NULL) – FK do `Subject`
  - `DayOfWeek` (VARCHAR(10), NOT NULL)
  - `StartTime` (TIME, NOT NULL)
  - `EndTime` (TIME, NOT NULL)
  - `TeacherID` (INT, NOT NULL) – FK do `Teachers`
  - `TranslatorID` (INT, NULL) – FK do `Translators`
- **Opis / przeznaczenie**: Harmonogram zajęć (dzień tygodnia, godziny, sala i prowadzący).

3.24 ShoppingCart

- **Nazwa tabeli**: `ShoppingCart`
- **Klucz główny**: (`StudentID`, `ActivityID`)
- **Opis / przeznaczenie**: Tabela `ShoppingCart` przechowuje listę aktywności dodanych do koszyka przez studentów przed finalizacją zamówienia.

3.25 StationaryClass

- **Nazwa tabeli**: `StationaryClass`
- **Klucz główny**: `StationaryClassID` (INT)
- **Kolumny** (m.in.):
  - `StationaryClassID` (INT, NOT NULL)
  - `ClassID` (INT, NOT NULL) – FK do `Buildings`
  - `Limit` (INT, NOT NULL)
- **Opis / przeznaczenie**: Tabela opisująca stacjonarne zajęcia z przypisaną salą (`ClassID`). Limit uczestników definiuje maksymalną liczbę osób mogących wziąć udział.

3.26 StationaryModule

- **Nazwa tabeli**: `StationaryModule`
- **Klucz główny**: `StationaryModuleID` (INT)
- **Opis / przeznaczenie**: Tabela `StationaryModule` przechowuje informacje o modułach kursowych realizowanych w formie zajęć stacjonarnych.

3.27 Students

- **Nazwa tabeli**: `Students`
- **Klucz główny**: `StudentID` (INT)
- **Kolumny** (m.in.):
  - `StudentID` (INT, NOT NULL)
  - `FirstName`, `LastName` (VARCHAR(30), NOT NULL)
  - `Address` (VARCHAR(30), NOT NULL)

  - `CityID` (INT, NOT NULL) – FK do `Cities`
  - `PostalCode` (VARCHAR(10), NOT NULL)
  - `Phone` (VARCHAR(15), NULL)
  - `Email` (VARCHAR(60), NOT NULL)
- **Opis / przeznaczenie**: Tabela `Students` przechowuje dane osobowe studentów.

3.28 Studies

- **Nazwa tabeli**: `Studies`
- **Klucz główny**: `StudiesID` (INT)
- **Kolumny** (m.in.):
  - `StudiesID` (INT, NOT NULL)
  - `ActivityID` (INT, NOT NULL) – FK do `Activities`
  - `StudiesName` (VARCHAR(50), NOT NULL)
  - `StudiesDescription` (TEXT, NULL)
  - `StudiesEntryFeePrice` (MONEY, NOT NULL)
  - `Syllabus` (TEXT, NOT NULL)
  - `StudiesEmployee` (INT, NOT NULL) – FK do `Employees`
  - `Limit` (INT, NOT NULL)
- **Opis / przeznaczenie**: Określa kierunek studiów, jego szczegółowy opis, cenę wpisowego i pracownika odpowiedzialnego.

3.29 StudiesClass

- **Nazwa tabeli**: `StudiesClass`
- **Klucz główny**: `StudyClassID` (INT)
- **Kolumny** (m.in.):
  - `StudyClassID` (INT, NOT NULL)
  - `SubjectID` (INT, NOT NULL) – FK do `Subject`
  - `ActivityID` (INT, NOT NULL) – FK do `Activities`
  - `TeacherID` (INT, NOT NULL) – FK do `Teachers`
  - `ClassName` (VARCHAR(50), NOT NULL)
  - `ClassPrice` (MONEY, NOT NULL)
  - `Date` (DATETIME, NOT NULL)
  - `DurationTime` (TIME(0), NULL)
  - `LanguageID` (INT, NULL) – FK do `Languages`
  - `TranslatorID` (INT, NULL) – FK do `Translators`
  - `LimitClass` (INT, NOT NULL)
- **Opis / przeznaczenie**: Tabela `StudiesClass` przechowuje informacje o zajęciach (klasach) realizowanych w ramach studiów.

3.30 StudiesClassAttendance

- **Nazwa tabeli**: `StudiesClassAttendance`
- **Klucz główny**: (`StudentID`, `StudyClassID`)
- **Opis / przeznaczenie**: Tabela `StudiesClassAttendance` przechowuje informacje o obecności studentów na zajęciach w ramach studiów.

3.31 Subject

- **Nazwa tabeli**: `Subject`
- **Klucz główny**: `SubjectID` (INT)
- **Kolumny** (m.in.):
  - `SubjectID` (INT, NOT NULL)
  - `StudiesID` (INT, NOT NULL) – FK do `Studies`
  - `CoordinatorID` (INT, NOT NULL) – FK do `Teachers`
  - `SubjectName` (VARCHAR(50), NOT NULL)
  - `SubjectDescription` (TEXT, NULL)
- **Opis / przeznaczenie**: Przedmiot w ramach kierunku `Studies`. Każdy przedmiot ma koordynatora (`CoordinatorID`).

3.32 SubjectGrades

- **Nazwa tabeli**: `SubjectGrades`
- **Klucz główny**: (`StudentID`, `SubjectID`)
- **Opis / przeznaczenie**: Oceny studenta (wartość w `SubjectGrade`) z danego przedmiotu.

3.33 TeacherLanguages

- **Nazwa tabeli**: `TeacherLanguages`
- **Klucz główny**: (`TeacherID`, `LanguageID`)
- **Opis / przeznaczenie**: Informacja o językach, którymi posługuje się dany nauczyciel.

3.34 Teachers

- **Nazwa tabeli**: `Teachers`
- **Kolumny** (m.in.):
  - `TeacherID` (INT, NOT NULL) – PK
  - `FirstName`, `LastName` (VARCHAR(30), NOT NULL)
  - `HireDate` (DATE, NULL)
  - `Phone` (VARCHAR(15), NULL)
  - `Email` (VARCHAR(60), NOT NULL)
- **Opis / przeznaczenie**: Tabela `Teachers` przechowuje dane nauczycieli, którzy prowadzą zajęcia w ramach studiów oraz kursów.

3.35 Translators

- **Nazwa tabeli**: `Translators`
- **Klucz główny**: `TranslatorID` (INT)
- **Kolumny** (m.in.):
  - `TranslatorID` (INT, NOT NULL)
  - `FirstName`, `LastName` (VARCHAR(30), NOT NULL)
  - `HireDate` (DATE, NULL)
  - `Phone` (VARCHAR(15), NULL)
  - `Email` (VARCHAR(60), NOT NULL)
- **Opis / przeznaczenie**: Tabela `Translators` przechowuje dane tłumaczy, którzy obsługują kursy, wykłady i inne materiały.

3.36 TranslatorsLanguages

- **Nazwa tabeli**: `TranslatorsLanguages`
- **Klucz główny**: (`TranslatorID`, `LanguageID`)
- **Opis / przeznaczenie**: Informacja, jakie języki obsługuje dany tłumacz.

3.37 WebinarDetails

- **Nazwa tabeli**: `WebinarDetails`
- **Klucz główny**: (`StudentID`, `WebinarID`)
- **Opis / przeznaczenie**: Tabela `WebinarDetails` przechowuje informacje o uczestnictwie studentów w webinarach, w tym status ukończenia i dostępność

3.38 Webinars

- **Nazwa tabeli**: `Webinars`
- **Klucz główny**: `WebinarID` (INT)
- **Kolumny** (m.in.):
  - `WebinarID` (INT, NOT NULL)
  - `ActivityID` (INT, NOT NULL) – FK do `Activities`
  - `TeacherID` (INT, NOT NULL) – FK do `Teachers`
  - `WebinarName` (VARCHAR(50), NOT NULL)
  - `WebinarPrice` (MONEY, NOT NULL)
  - `VideoLink` (VARCHAR(50), NOT NULL)
  - `WebinarDate` (DATETIME, NOT NULL)
  - `DurationTime` (TIME(0), NOT NULL)
  - `WebinarDescription` (TEXT, NOT NULL)
  - `LanguageID` (INT, NOT NULL) – FK do `Languages`
- **Opis / przeznaczenie**: Tabela `Webinars` przechowuje informacje o dostępnych webinarach, w tym nazwę, prowadzącego, cenę i język.

4 Relacje (klucze obce)

W tej sekcji przedstawiono **kompletną listę kluczy obcych (FOREIGN KEY)**:

- **Activities →**
  - `Courses (ActivityID)`
  - `OrderDetails (ActivityID)`
  - `ShoppingCart (ActivityID)`
  - `StudiesClass (ActivityID)`
  - `Webinars (ActivityID)`
  - `Studies (ActivityID)`
- **Buildings →**
  - `StationaryClass (ClassID)`
  - `StationaryModule (ClassID)`
  - `Schedule (ClassID)`
- **Cities →**
  - `Countries (CountryID) -- w tabeli Cities istnieje kolumna CountryID`
  - `Students (CityID)`
- **Courses →**
  - `CourseModules (CourseID)`
  - `CourseParticipants (CourseID)`
  - `CoursesAttendance (ModuleID → CourseModules, ale CourseModules z kolei FK do Courses)`
- **CourseModules →**
  - `OnlineAsyncModule (OnlineAsyncModuleID = ModuleID)`
  - `OnlineSyncModule (OnlineSyncModuleID = ModuleID)`
  - `Schedule (CourseModuleID)`
  - `StationaryModule (StationaryModuleID = ModuleID)`
  - `CoursesAttendance (ModuleID)`
- **Teachers →**
  - `CourseModules (TeacherID)`
  - `StudiesClass (TeacherID)`
  - `Webinars (TeacherID)`
  - `Subject (CoordinatorID)`
  - `TeacherLanguages (TeacherID)`
  - `Courses (CourseCoordinatorID)`
  - `Schedule (TeacherID)`
- **Translators →**

  - CourseModules (TranslatorID)
  - StudiesClass (TranslatorID)
  - TranslatorsLanguages (TranslatorID)
  - Schedule (TranslatorID)

- **Languages** →
  - CourseModules (LanguageID)
  - StudiesClass (LanguageID)
  - Webinars (LanguageID)
  - TeacherLanguages (LanguageID)
  - TranslatorsLanguages (LanguageID)

- **Students** →
  - CourseParticipants (StudentID)
  - CoursesAttendance (StudentID)
  - Orders (StudentID)
  - RODO_Table (StudentID)
  - ShoppingCart (StudentID)
  - StudiesClassAttendance (StudentID)
  - SubjectGrades (StudentID)
  - WebinarDetails (StudentID)
  - InternshipAttendance (StudentID)

- **Studies** →
  - Subject (StudiesID)
  - Internship (StudiesID)

- **StudiesClass** →
  - StudiesClassAttendance (StudyClassID)
  - StationaryClass (StationaryClassID = StudyClassID)
  - OnlineAsyncClass (OnlineAsyncClassID = StudyClassID)
  - OnlineSyncClass (OnlineSyncClassID = StudyClassID)

- **Subject** →
  - StudiesClass (SubjectID)
  - StudiesClassAttendance (StudyClassID → StudiesClass, Subject → Studies???
  - SubjectGrades (SubjectID)

- **Employees →**
  - `Orders (EmployeeHandling)`
  - `Studies (StudiesEmployee)`
- **EmployeeTypes →**
  - `Employees (EmployeeTypeID)`
- **EuroExchangeRate →**
  - `Orders (OrderDate)`
- **Internship →**
  - `InternshipAttendance (InternshipID)`
- **OrderPaymentStatus →**
  - `Orders (PaymentURL)`
- **Orders →**
  - `OrderDetails (OrderID)`
- **Webinars →**
  - `WebinarDetails (WebinarID)`

5 Warunki Integralnościowe

Baza danych została zaprojektowana z myślą o zapewnieniu wysokiej spójności oraz integralności danych. Poniżej przedstawiono główne mechanizmy, które wspierają te warunki:

5.1 Klucze główne i obce

- **Klucze główne (PRIMARY KEY)** – Każda tabela posiada wyznaczony klucz główny, który zapewnia unikalność rekordów. Przykładowo:
  - Tabela `Students` ma klucz `StudentID`.
  - Tabela `Courses` ma klucz `CourseID`.
  - Tabela `Teachers` ma klucz `TeacherID`.
- **Klucze obce (FOREIGN KEY)** – Relacje pomiędzy tabelami są definiowane poprzez klucze obce, co gwarantuje, że odwołania do rekordów w tabelach powiązanych są zawsze poprawne. Przykłady:
  - `Students.CityID` odnosi się do `Cities.CityID`, co pozwala na poprawne powiązanie adresu studenta z istniejącym miastem.
  - `Courses.ActivityID` jest kluczem obcym odnoszącym się do `Activities.ActivityID`, zapewniając powiązanie kursu z jego ogólnymi danymi (cena, tytuł, status).
  - W tabelach realizujących obecność (`CoursesAttendance`, `StudiesClassAttendance`) wykorzystuje się klucze obce odnoszące się do tabel `CourseModules`, `StudiesClass` oraz `Students`.

5.2 Wartości NULL i NOT NULL – logika warunków w tabelach

Podczas definiowania tabel kluczowe kolumny, które pełnią rolę identyfikatorów, opisów czy stanów operacyjnych, zostały oznaczone jako `NOT NULL` w celu zapewnienia, że rekordy zawierają wszystkie niezbędne informacje. Natomiast kolumny zawierające dane dodatkowe lub zależne od kontekstu (np. opis, tłumacz, data opłacenia) zostały zdefiniowane jako `NULL`, co pozwala na elastyczność modelu. Poniżej przedstawiono szczegółowe przykłady z kodu:

Przykład 1: Tabela `Activities` Kod:

```
CREATE TABLE Activities (
    ActivityID      INT             NOT NULL,
    Price           MONEY           NOT NULL,
    Title           VARCHAR(50)     NOT NULL,
    Active          BIT             NOT NULL,
    CONSTRAINT PK_Activities PRIMARY KEY (ActivityID)
);
```

Opis: - `ActivityID`, `Price`, `Title` oraz `Active` są definiowane jako `NOT NULL` – każda aktywność musi mieć unikalny identyfikator, ustaloną cenę, tytuł oraz określony status (aktywna/nieaktywna). - Brak możliwości pozostawienia tych pól pustych zapobiega dodawaniu niekompletnych rekordów.

Przykład 2: Tabela `Courses` Kod:

```
CREATE TABLE Courses (
    CourseID              INT             NOT NULL,
    ActivityID            INT             NOT NULL,   -- FK do Activities
    CourseName            VARCHAR(50)     NOT NULL,
    CourseDescription     TEXT            NULL,
    CoursePrice           MONEY           NOT NULL,
```

```
    CourseCoordinatorID  INT           NOT NULL,  -- FK do Teachers
    CONSTRAINT PK_Courses PRIMARY KEY (CourseID)
);
```

Opis: - Kluczowe pola (`CourseID`, `ActivityID`, `CourseName`, `CoursePrice` oraz `CourseCoordinatorID`) są oznaczone jako `NOT NULL`, ponieważ bez nich kurs nie miałby poprawnego opisu ani nie byłby poprawnie powiązany z innymi encjami (np. z aktywnością i nauczycielem). - `CourseDescription` jest polem dodatkowym – nie zawsze musi zawierać treść, dlatego jest zdefiniowane jako `NULL`.

Przykład 3: Tabela `CourseModules` Kod:

```
CREATE TABLE CourseModules (
    ModuleID        INT           NOT NULL,
    CourseID        INT           NOT NULL,  -- FK do Courses
    ModuleName      VARCHAR(50)   NOT NULL,
    Date            DATETIME      NOT NULL,
    DurationTime    TIME(0)       NOT NULL,
    TeacherID       INT           NOT NULL,  -- FK do Teachers
    TranslatorID    INT           NULL,      -- FK do Translators
    LanguageID      INT           NOT NULL,  -- FK do Languages
    CONSTRAINT PK_CourseModules PRIMARY KEY (ModuleID)
);
```

Opis: - Wszystkie kolumny, które definiują podstawowe informacje o module (np. identyfikator modułu, kurs, nazwa, data, czas trwania, nauczyciel, język) są ustawione jako `NOT NULL`, aby zapewnić pełną informację o danym module. - Kolumna `TranslatorID` jest ustawiona jako `NULL`, ponieważ nie każdy moduł wymaga tłumacza – w niektórych przypadkach tłumaczenie może nie być potrzebne.

Przykład 4: Tabela `Employees` Kod:

```
CREATE TABLE Employees (
    EmployeeID      INT           NOT NULL,
    FirstName       VARCHAR(30)   NOT NULL,
    LastName        VARCHAR(30)   NOT NULL,
    HireDate        DATE          NULL,
    EmployeeTypeID  INT           NOT NULL,   -- FK do EmployeeTypes
    Phone           VARCHAR(15)   NULL,
    Email           VARCHAR(60)   NOT NULL,
    CONSTRAINT PK_Employees PRIMARY KEY (EmployeeID)
);
```

Opis: - Pola `EmployeeID`, `FirstName`, `LastName`, `EmployeeTypeID` oraz `Email` są krytyczne dla identyfikacji i komunikacji z pracownikiem, dlatego są oznaczone jako `NOT NULL`. - Kolumna `HireDate` jest opcjonalna – data zatrudnienia może nie być jeszcze dostępna w momencie rejestracji. - `Phone` również jest opcjonalny, ponieważ nie każdy pracownik musi od razu podać numer telefonu.

Przykład 5: Tabela `OrderPaymentStatus` Kod:

```
CREATE TABLE OrderPaymentStatus (
    PaymentURL          INT           NOT NULL,
    OrderPaymentStatus  VARCHAR(20)   NOT NULL,
    PaidDate            DATETIME      NULL,
    CONSTRAINT PK_OrderPaymentStatus PRIMARY KEY (PaymentURL)
);
```

Opis: - `PaymentURL` oraz `OrderPaymentStatus` są definiowane jako `NOT NULL` – informacje te są niezbędne do identyfikacji statusu płatności. - `PaidDate` jest ustawione jako `NULL` ponieważ data opłacenia zamówienia zostanie określona dopiero, gdy płatność zostanie zrealizowana. Do momentu potwierdzenia płatności pole to pozostaje puste.

Przykład 6: Tabela `Schedule` Kod:

```
CREATE TABLE Schedule (
    ScheduleID        INT         NOT NULL,
    ClassID           INT         NOT NULL,  -- FK do Buildings
    CourseModuleID    INT         NULL,      -- FK do CourseModules
    StudiesSubjectID  INT         NULL,      -- FK do Subject
    DayOfWeek         VARCHAR(10) NOT NULL,
    StartTime         TIME        NOT NULL,
    EndTime           TIME        NOT NULL,
    TeacherID         INT         NOT NULL,  -- FK do Teachers
    TranslatorID      INT         NULL,      -- FK do Translators
    CONSTRAINT PK_Schedule PRIMARY KEY (ScheduleID)
);
```

Opis: - Pola `ScheduleID`, `ClassID`, `DayOfWeek`, `StartTime`, `EndTime` oraz `TeacherID` są obowiązkowe, ponieważ są niezbędne do określenia harmonogramu zajęć. - Kolumny `CourseModuleID` i `StudiesSubjectID` są opcjonalne – harmonogram może dotyczyć zarówno zajęć powiązanych z modułami kursów, jak i z przedmiotami w ramach studiów. - `TranslatorID` również jest opcjonalny, gdyż nie wszystkie zajęcia wymagają tłumacza.

Ogólne zasady:

1. Wartości niezbędne dla identyfikacji encji lub dla utrzymania zależności między tabelami (np. klucze główne oraz pola kluczy obcych) zawsze są oznaczane jako `NOT NULL`.
2. Informacje opisowe lub dodatkowe, które nie są niezbędne do wykonania podstawowych operacji na danych, mogą przyjmować wartość `NULL` – pozwala to na elastyczność w modelu danych.
3. Pola, których wartość jest ustalana dynamicznie, np. data potwierdzenia płatności (`PaidDate`) lub identyfikator tłumacza (`TranslatorID`) w module, są definiowane jako opcjonalne.

Dzięki takiej strukturze oraz przemyślanej definicji kolumn, baza danych zapewnia:

- Pełną integralność danych – kluczowe informacje nie mogą zostać pominięte.
- Elastyczność – dodatkowe lub zależne dane mogą być pomijane w przypadku, gdy nie są dostępne, co jest szczególnie przydatne w dynamicznie zmieniającym się środowisku operacyjnym.
- Wsparcie mechanizmów zapewniających spójność (np. triggerów, procedur), które jeszcze bardziej wzmacniają model danych.

5.3 Triggery wspomagające integralność danych

Mechanizmy te dodatkowo zabezpieczają bazę przed niepożądanymi operacjami i utrzymują spójność danych:

- **Automatyczne ustawianie statusu płatności** – Trigger `TR_OrderPaymentStatus_AfterInsert` ustawia domyślny status zamówienia (np. `"Pending"`) oraz aktualizuje datę opłacenia, gdy status zmieni się na `"Paid"`.
- **Rejestracja uczestników po opłaceniu zamówienia** – Trigger `TR_OrderPaymentStatus_AfterUpdate_` automatycznie dodaje studenta do odpowiednich tabel (takich jak `CourseParticipants`, `WebinarDetails`, `StudiesClassAttendance`) po potwierdzeniu płatności.
- **Usuwanie powiązanych rekordów** – Trigger `TR_Courses_AfterDelete` dba o to, aby usunięcie kursu skutkowało automatycznym usunięciem powiązanych rekordów (modułów, uczestników oraz frekwencji), co zapobiega występowaniu pozostałości danych.
- **Blokowanie operacji niedozwolonych** – Trigger `TR_Teachers_InsteadOfDelete_BlockIfActive` uniemożliwia usunięcie nauczyciela, jeśli jest on przypisany do aktywnych kursów lub webinarów, chroniąc w ten sposób dane operacyjne.
- **Zapobieganie duplikacji rekordów** – Trigger `TR_Webinars_AfterInsert_UniqueTeacherWebinar` chroni przed dodaniem webinaru o tej samej nazwie przez tego samego nauczyciela.

5.4 Procedury i funkcje wspomagające integralność

Wiele procedur oraz funkcji składowanych zawiera wbudowaną logikę, która zabezpiecza przed nieprawidłowym dodawaniem lub modyfikacją danych. Przykładowo:

- Procedury takie jak `spRegisterStudentInCourse` czy `spUnregisterStudentFromCourse` sprawdzają, czy student nie jest już (lub nie jest nadal) zapisany na dany kurs, co zapobiega duplikacji lub przypadkowemu usunięciu rekordów.
- Funkcje obsługujące obliczenia (np. `ufnGetOrderTotal`, `ufnGetStationaryModuleFreeSlots`) są zaprojektowane tak, aby przy braku rekordów zwracały wartość domyślną (np. 0), co umożliwia poprawne agregowanie danych i uniknięcie błędów wynikających z wartości `NULL`.

5.5 Inicjalizacja tabel i zależności

Podczas tworzenia struktury bazy danych, w definicjach tabel zostały uwzględnione następujące aspekty:

- Każda tabela posiada dobrze zdefiniowane kolumny wraz z wymaganymi typami danych i ograniczeniami typu `NOT NULL` tam, gdzie jest to konieczne.
- W definicjach tabel określono klucze główne, a także klucze obce, które wiążą tabele ze sobą (np. `Courses` z `Activities`, `Students` z `Cities` i `Countries`).
- W trakcie inicjalizacji danych (poprzez skrypty INSERT generowane automatycznie) zachowana jest kolejność, która respektuje relacje między tabelami – najpierw wstawiane są rekordy w tabelach niezależnych, a następnie te, które odwołują się do kluczy obcych.

5.6 Podsumowanie

Warunki integralnościowe w bazie są realizowane poprzez wielopoziomowy mechanizm:

1. Definicje tabel z precyzyjnym określeniem kluczy głównych i obcych.
2. Wdrożenie triggerów, które automatyzują operacje utrzymujące spójność (np. przy aktualizacjach statusów, kasowaniu rekordów czy zapobieganiu duplikatom).
3. Procedury i funkcje, które implementują logikę biznesową oraz weryfikację danych podczas operacji wstawiania, aktualizacji i usuwania.

Dzięki tak zorganizowanej strukturze baza danych jest chroniona przed błędami użytkowników, niespójnymi danymi oraz nieautoryzowanymi zmianami, co stanowi fundament dla bezpiecznego i efektywnego działania systemu.

6 Triggery

W niniejszej sekcji opisano wszystkie triggery używane w bazie danych. Triggery służą do automatyzacji operacji, zapewnienia spójności danych oraz ochrony przed błędami użytkownika.

- **TR_OrderPaymentStatus_AfterInsert** – Automatyczne ustawianie statusu płatności.
- **TR_OrderPaymentStatus_AfterUpdate_PaymentSuccess** – Automatyczna rejestracja studentów na kursy, webinary i studia po opłaceniu zamówienia.
- **TR_Courses_AfterDelete** – Usuwanie powiązanych danych po usunięciu kursu.
- **TR_Webinars_AfterInsert_UniqueTeacherWebinar** – Zapobieganie duplikatom webinarów dla tego samego nauczyciela.
- **TR_Teachers_InsteadOfDelete_BlockIfActive** – Blokowanie usunięcia nauczyciela, jeśli prowadzi aktywne zajęcia.
- **TR_Students_AfterInsert_AddCity** – Automatyczne dodawanie miast do bazy na podstawie wpisu studenta.

6.1 TR_OrderPaymentStatus_AfterInsert

```
CREATE OR ALTER TRIGGER TR_OrderPaymentStatus_AfterInsert
ON dbo.OrderPaymentStatus
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE ops
    SET
        OrderPaymentStatus =
            CASE
                WHEN i.OrderPaymentStatus IS NULL THEN 'Pending'
                ELSE i.OrderPaymentStatus
            END,
        PaidDate =
            CASE
                WHEN i.OrderPaymentStatus = 'Paid' THEN GETDATE()
                ELSE ops.PaidDate
            END
    FROM dbo.OrderPaymentStatus ops
    JOIN inserted i ON ops.PaymentURL = i.PaymentURL;
END;
GO
```

Cel: Automatyczne ustawienie domyślnego statusu płatności oraz oznaczanie zamówienia jako opłacone.
Działanie:

- Jeśli nowa płatność nie ma określonego statusu, domyślnie ustawiany jest `"Pending"`.
- Jeśli status to `"Paid"`, ustawiana jest data zaksięgowania płatności.

6.2 TR_OrderPaymentStatus_AfterUpdate_PaymentSuccess

```
CREATE OR ALTER TRIGGER TR_OrderPaymentStatus_AfterUpdate_PaymentSuccess
ON dbo.OrderPaymentStatus
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE ops
    SET PaidDate = GETDATE()
    FROM dbo.OrderPaymentStatus ops
    JOIN inserted i ON ops.PaymentURL = i.PaymentURL
    WHERE i.OrderPaymentStatus = 'Paid';

    ;WITH Changed AS
    (
        SELECT i.PaymentURL
        FROM inserted i
        JOIN deleted d ON i.PaymentURL = d.PaymentURL
        WHERE i.OrderPaymentStatus = 'Paid'
          AND d.OrderPaymentStatus <> 'Paid'
    )
    SELECT PaymentURL
    INTO #Changed
    FROM Changed;

    INSERT INTO WebinarDetails (StudentID, WebinarID, Complete, AvailableDue)
    SELECT
       o.StudentID,
       wb.WebinarID,
       0 AS Complete,
       DATEADD(DAY, 30, GETDATE()) AS AvailableDue
    FROM #Changed ch
    JOIN dbo.Orders o ON o.PaymentURL = ch.PaymentURL
    JOIN dbo.OrderDetails od ON od.OrderID = o.OrderID
    JOIN dbo.Webinars wb ON wb.ActivityID = od.ActivityID;

    INSERT INTO CourseParticipants (CourseID, StudentID)
    SELECT
       c.CourseID,
       o.StudentID
    FROM #Changed ch
    JOIN dbo.Orders o ON o.PaymentURL = ch.PaymentURL
    JOIN dbo.OrderDetails od ON od.OrderID = o.OrderID
    JOIN dbo.Courses c ON c.ActivityID = od.ActivityID;

    INSERT INTO StudiesClassAttendance (StudyClassID, StudentID, Attendance)
    SELECT
       sc.StudyClassID,
       o.StudentID,
       0
    FROM #Changed ch
    JOIN dbo.Orders o ON o.PaymentURL = ch.PaymentURL
    JOIN dbo.OrderDetails od ON od.OrderID = o.OrderID
    JOIN dbo.Studies st ON st.ActivityID = od.ActivityID
    JOIN dbo.StudiesClass sc ON sc.ActivityID = st.ActivityID;

    DROP TABLE #Changed;
END;
GO
```

Cel: Automatyczna rejestracja studentów na kursy, webinary i studia po opłaceniu zamówienia.
Działanie:

- Po zmianie statusu płatności na `"Paid"`:
  - Student zostaje dodany do `WebinarDetails`, z 30-dniowym dostępem.
  - Student zostaje przypisany do kursów (`CourseParticipants`).
  - Student zostaje dodany do zajęć studiów (`StudiesClassAttendance`).

6.3 TR_Courses_AfterDelete

```sql
CREATE OR ALTER TRIGGER TR_Courses_AfterDelete
ON dbo.Courses
AFTER DELETE
AS
BEGIN
    SET NOCOUNT ON;

    DELETE cp
    FROM dbo.CourseParticipants cp
    JOIN deleted d ON cp.CourseID = d.CourseID;

    DELETE ca
    FROM dbo.CoursesAttendance ca
    JOIN dbo.CourseModules cm ON ca.ModuleID = cm.ModuleID
    JOIN deleted d ON cm.CourseID = d.CourseID;

    DELETE cm
    FROM dbo.CourseModules cm
    JOIN deleted d ON cm.CourseID = d.CourseID;

    PRINT 'All related participants, attendance records, and modules removed.';
END;
GO
```

Cel: Usunięcie wszystkich powiązanych danych po usunięciu kursu.
Działanie:

- Usuwa wszystkich uczestników kursu (`CourseParticipants`).
- Usuwa wpisy o frekwencji (`CoursesAttendance`).
- Usuwa moduły kursowe (`CourseModules`).

6.4 TR_Webinars_AfterInsert_UniqueTeacherWebinar

```
CREATE OR ALTER TRIGGER TR_Webinars_AfterInsert_UniqueTeacherWebinar
ON dbo.Webinars
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1
        FROM dbo.Webinars w
        JOIN inserted i ON
             w.TeacherID = i.TeacherID
             AND w.WebinarName = i.WebinarName
             AND w.WebinarID <> i.WebinarID
    )
    BEGIN
        RAISERROR('Cannot add duplicate webinar for the same teacher.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;
END;
GO
```

Cel: Zapobieganie duplikatom webinarów dla tego samego nauczyciela.
Działanie:

- Jeśli istnieje już webinar o tej samej nazwie dla danego nauczyciela, system odrzuca operację.
- Wywoływany jest błąd `RAISERROR`, a transakcja jest wycofywana.

6.5 TR_Teachers_InsteadOfDelete_BlockIfActive

```sql
CREATE OR ALTER TRIGGER TR_Teachers_InsteadOfDelete_BlockIfActive
ON dbo.Teachers
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;
    IF EXISTS (
        SELECT 1
        FROM deleted d
        JOIN CourseModules cm ON cm.TeacherID = d.TeacherID
        JOIN Courses c ON c.CourseID = cm.CourseID
        JOIN Activities a ON a.ActivityID = c.ActivityID
        WHERE a.Active = 1
    )
    OR EXISTS (
        SELECT 1
        FROM deleted d
        JOIN Webinars w ON w.TeacherID = d.TeacherID
        JOIN Activities a ON a.ActivityID = w.ActivityID
        WHERE a.Active = 1
    )
    BEGIN
        RAISERROR('Cannot delete teacher: assigned to active classes.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    DELETE t
    FROM dbo.Teachers t
    JOIN deleted d ON t.TeacherID = d.TeacherID;
END;
GO
```

Cel: Blokowanie usunięcia nauczyciela, który jest przypisany do aktywnych kursów lub webinarów.
Działanie:

- Jeśli nauczyciel prowadzi aktywne kursy lub webinary, system uniemożliwia jego usunięcie.
- Jeśli nauczyciel nie jest aktywny, usunięcie przebiega normalnie.

6.6 TR_Students_AfterInsert_AddCity

```sql
CREATE OR ALTER TRIGGER TR_Students_AfterInsert_AddCity
ON dbo.Students
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO dbo.Cities (CityID, CityName, CountryID)
    SELECT DISTINCT
        i.CityID,
        'Unknown',
        1
    FROM inserted i
    LEFT JOIN dbo.Cities c ON c.CityID = i.CityID
    WHERE c.CityID IS NULL;
END;
GO
```

Cel: Automatyczne dodawanie nowego miasta do tabeli
`Cities`, jeśli student podał miasto, które jeszcze nie istnieje.
Działanie:

- System sprawdza, czy miasto podane przez studenta istnieje w bazie.
- Jeśli nie, dodaje je jako nowy wpis w tabeli `Cities` z domyślną nazwą `"Unknown"` i `CountryID = 1`.

Podsumowanie triggerów

Triggery w systemie pełnią kluczową rolę w automatyzacji operacji, zachowaniu integralności danych oraz zabezpieczeniu przed niepożądanymi zmianami. Dzięki nim wiele procesów, które normalnie wymagałyby ręcznej interwencji administratorów bazy danych lub użytkowników, może zostać wykonanych automatycznie.
Jednym z głównych zastosowań triggerów jest automatyczne zarządzanie danymi w systemie. Przykładowo:

- Automatyczne rejestrowanie studentów na kursy i webinary po opłaceniu zamówienia eliminuje konieczność ręcznego dodawania użytkowników, przyspieszając cały proces.
- Blokowanie usunięcia nauczyciela, który jest aktywny w systemie zapobiega przypadkowym błędom, które mogłyby prowadzić do niespójności danych.
- Zapewnienie unikalności webinarów dla jednego nauczyciela dba o przejrzystość i uporządkowanie oferty edukacyjnej.

Mechanizmy takie jak automatyczne usuwanie powiązanych uczestników kursu czy czyszczenie listy frekwencji po skasowaniu kursu zapewniają, że baza pozostaje w spójnym stanie, bez zbędnych i nieużywanych wpisów.

7 Widoki (`views`)

W niniejszej sekcji przedstawiono widoki (`views`) utworzone w bazie danych. Widoki pozwalają na predefiniowane zapytania, które dostarczają uporządkowanych informacji bez konieczności wykonywania skomplikowanych operacji na wielu tabelach. Dzięki nim użytkownicy systemu mogą uzyskiwać dostęp do danych w sposób efektywny i zgodny z ich uprawnieniami.
Każdy widok odpowiada za określoną funkcjonalność w systemie, np. prezentowanie studentów zapisanych na kursy, szczegółów zamówień, czy planów zajęć. W kolejnych podsekcjach przedstawiono pełne definicje widoków oraz ich zastosowania.

7.1 Lista widoków

W bazie danych zaimplementowano pięć głównych widoków:

- **`v_StudentCourses`** – pokazuje, którzy studenci są zapisani na jakie kursy, wraz z nazwą i ceną kursu oraz danymi studenta.
- **`v_CourseModulesDetailed`** – szczegółowe informacje o modułach kursów, w tym nauczyciel, tłumacz, język oraz data modułu.
- **`v_OrdersFull`** – pełna historia zamówień, łącznie z informacją o płatnościach, sumaryczną wartością zamówienia i danymi studenta oraz pracownika obsługującego zamówienie.
- **`v_ScheduleDetailed`** – szczegółowy harmonogram zajęć, łącznie z salą, nauczycielem, tłumaczem i przypisanym modułem kursowym lub przedmiotem studiów.
- **`v_StudentGrades`** – wykaz ocen studentów z poszczególnych przedmiotów, wraz z nazwą studiów i informacją o nauczycielu prowadzącym dany przedmiot.

7.2 Widok `v_StudentCourses`

```
CREATE OR ALTER VIEW dbo.v_StudentCourses AS
SELECT
    s.StudentID,
    s.FirstName AS StudentFirstName,
    s.LastName  AS StudentLastName,
    s.Email     AS StudentEmail,
    c.CourseID,
    c.CourseName,
    c.CoursePrice
FROM dbo.Students s
JOIN dbo.CourseParticipants cp ON s.StudentID = cp.StudentID
JOIN dbo.Courses c ON cp.CourseID = c.CourseID;
GO
```

Widok `v_StudentCourses` umożliwia uzyskanie informacji na temat studentów biorących udział w kursach. Zawiera następujące dane:

- Identyfikator studenta i jego dane personalne (imię, nazwisko, e-mail).
- Informacje o kursie, w którym uczestniczy dany student (nazwa, cena).

Widok ten może być przydatny do generowania raportów dotyczących aktywnych studentów oraz analizy popularności kursów.

7.3 Widok v_CourseModulesDetailed

```
CREATE OR ALTER VIEW dbo.v_CourseModulesDetailed AS
SELECT
    cm.ModuleID,
    cm.ModuleName,
    cm.[Date] AS ModuleDate,
    cm.DurationTime,
    t.TeacherID,
    t.FirstName    AS TeacherFirstName,
    t.LastName     AS TeacherLastName,
    tr.TranslatorID,
    tr.FirstName   AS TranslatorFirstName,
    tr.LastName    AS TranslatorLastName,
    l.LanguageName AS ModuleLanguage,
    c.CourseID,
    c.CourseName
FROM dbo.CourseModules cm
JOIN dbo.Teachers t ON cm.TeacherID = t.TeacherID
LEFT JOIN dbo.Translators tr ON cm.TranslatorID = tr.TranslatorID
JOIN dbo.Languages l ON cm.LanguageID = l.LanguageID
JOIN dbo.Courses c ON cm.CourseID = c.CourseID;
GO
```

Widok ten dostarcza szczegółowych informacji na temat modułów kursowych, obejmujących:

- Nazwę modułu, datę i czas jego trwania.
- Informacje o nauczycielu prowadzącym dany moduł.
- Opcjonalne dane o tłumaczu prowadzącym zajęcia w innym języku.
- Powiązanie modułu z kursem, do którego należy.

7.4 Widok v_OrdersFull

```
CREATE OR ALTER VIEW dbo.v_OrdersFull AS
SELECT
    o.OrderID,
    o.OrderDate,
    ops.OrderPaymentStatus,
    ops.PaidDate,
    s.StudentID,
    s.FirstName AS StudentFirstName,
    s.LastName  AS StudentLastName,
    e.EmployeeID,
    e.FirstName AS EmployeeFirstName,
    e.LastName  AS EmployeeLastName,
    SUM(a.Price) AS TotalOrderPrice
FROM dbo.Orders o
JOIN dbo.OrderPaymentStatus ops ON o.PaymentURL = ops.PaymentURL
JOIN dbo.Students s ON o.StudentID = s.StudentID
JOIN dbo.Employees e ON o.EmployeeHandling = e.EmployeeID
JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
JOIN dbo.Activities a ON od.ActivityID = a.ActivityID
GROUP BY
    o.OrderID, o.OrderDate, ops.OrderPaymentStatus, ops.PaidDate,
    s.StudentID, s.FirstName, s.LastName,
    e.EmployeeID, e.FirstName, e.LastName;
GO
```

Widok `v_OrdersFull` dostarcza kompletnych informacji na temat zamówień w systemie, w tym:

- Identyfikatora zamówienia, daty oraz statusu płatności.
- Łącznej wartości zamówienia (suma cen wszystkich zakupionych aktywności).
- Danych studenta składającego zamówienie.
- Danych pracownika obsługującego zamówienie.

Widok ten jest szczególnie użyteczny do monitorowania realizowanych płatności oraz generowania raportów sprzedaży.

7.5 Widok `v_ScheduleDetailed`

```
CREATE OR ALTER VIEW dbo.v_ScheduleDetailed AS
SELECT
    sch.ScheduleID,
    sch.DayOfWeek,
    sch.StartTime,
    sch.EndTime,
    t.TeacherID,
    t.FirstName AS TeacherFirstName,
    t.LastName  AS TeacherLastName,
    tr.TranslatorID,
    tr.FirstName AS TranslatorFirstName,
    tr.LastName  AS TranslatorLastName,
    b.BuildingName,
    b.RoomNumber,
    cm.ModuleID,
    cm.ModuleName,
    sb.SubjectID,
    sb.SubjectName
FROM dbo.Schedule sch
JOIN dbo.Buildings b ON sch.ClassID = b.ClassID
JOIN dbo.Teachers t ON sch.TeacherID = t.TeacherID
LEFT JOIN dbo.Translators tr ON sch.TranslatorID = tr.TranslatorID
LEFT JOIN dbo.CourseModules cm ON sch.CourseModuleID = cm.ModuleID
LEFT JOIN dbo.Subject sb ON sch.StudiesSubjectID = sb.SubjectID;
GO
```

Widok `v_ScheduleDetailed` agreguje dane związane z harmonogramem zajęć. Pozwala na uzyskanie następujących informacji:

- Dnia tygodnia oraz godzin rozpoczęcia i zakończenia zajęć.
- Nauczyciela oraz (jeśli obecny) tłumacza prowadzącego zajęcia.
- Informacji o sali, w której odbywają się zajęcia.
- Powiązanego modułu kursowego lub przedmiotu studiów.

7.6 Widok v_StudentGrades

```
CREATE OR ALTER VIEW dbo.v_StudentGrades AS
SELECT
    sg.StudentID,
    st.FirstName AS StudentFirstName,
    st.LastName  AS StudentLastName,
    sb.SubjectID,
    sb.SubjectName,
    sb.CoordinatorID,
    tch.FirstName AS CoordinatorFirstName,
    tch.LastName  AS CoordinatorLastName,
    s.StudiesID,
    s.StudiesName,
    sg.SubjectGrade
FROM dbo.SubjectGrades sg
JOIN dbo.Students st ON sg.StudentID = st.StudentID
JOIN dbo.Subject sb ON sg.SubjectID = sb.SubjectID
JOIN dbo.Teachers tch ON sb.CoordinatorID = tch.TeacherID
JOIN dbo.Studies s ON sb.StudiesID = s.StudiesID;
GO
```

Widok ten zawiera zestawienie ocen studentów z poszczególnych przedmiotów. Pozwala uzyskać informacje:

- O studentach oraz ich ocenach.
- O przedmiotach, do których przypisane są oceny.
- O nauczycielu prowadzącym dany przedmiot.
- O studiach, w ramach których odbywa się przedmiot.

7.7 Uprawnienia do poszczególnych widoków

Dostęp do widoków został przyznany w sposób umożliwiający różnym grupom użytkowników korzystanie z odpowiednich danych zgodnie z ich rolami w systemie. Poniżej przedstawiono szczegółowe wyjaśnienie nadanych uprawnień.

```
GRANT SELECT ON OBJECT::dbo.v_StudentCourses
    TO Role_Admin, Role_Employee;
GO

GRANT SELECT ON OBJECT::dbo.v_CourseModulesDetailed
    TO Role_Admin, Role_Employee, Role_Student, Role_Teacher, Role_Translator;
GO

GRANT SELECT ON OBJECT::dbo.v_OrdersFull
    TO Role_Admin, Role_Employee, Role_Student;
GO

GRANT SELECT ON OBJECT::dbo.v_ScheduleDetailed
    TO Role_Admin, Role_Employee, Role_Student, Role_Teacher, Role_Translator;
GO

GRANT SELECT ON OBJECT::dbo.v_StudentGrades
    TO Role_Admin, Role_Employee, Role_Teacher;
GO
```

Uprawnienia do widoków przydzielono zgodnie z rolami użytkowników:

- `v_StudentCourses` – dostęp dla:
  - `Role_Admin`, `Role_Employee` – zarządzanie zapisami studentów na kursy.

  Studenci nie mają dostępu, ponieważ widok zawiera dane innych użytkowników.
- `v_CourseModulesDetailed` – dostęp dla:
  - `Role_Admin`, `Role_Employee` – kontrola administracyjna nad kursami.
  - `Role_Student` – możliwość podglądu struktury kursów.
  - `Role_Teacher` – dostęp do prowadzonych modułów.
  - `Role_Translator` – wgląd w materiały tłumaczeniowe.
- `v_OrdersFull` – dostęp dla:
  - `Role_Admin`, `Role_Employee` – zarządzanie zamówieniami.
  - `Role_Student` – wgląd we własne zamówienia.

  Nauczyciele i tłumacze nie mają dostępu, gdyż nie zarządzają płatnościami.
- `v_ScheduleDetailed` – dostęp dla:
  - `Role_Admin`, `Role_Employee` – zarządzanie harmonogramem.
  - `Role_Student` – dostęp do planu zajęć.
  - `Role_Teacher` – podgląd prowadzonych zajęć.
  - `Role_Translator` – sprawdzanie przypisanych zajęć.
- `v_StudentGrades` – dostęp dla:
  - `Role_Admin`, `Role_Employee` – monitorowanie wyników.
  - `Role_Teacher` – dostęp do ocen studentów na prowadzonych przedmiotach.

  Studenci nie mają dostępu, ponieważ powinni widzieć tylko własne oceny.

7.8 Podsumowanie

Widoki zostały zaprojektowane w celu:

- ułatwienia dostępu do często używanych zestawień danych,
- poprawy wydajności poprzez eliminację konieczności wielokrotnego wykonywania skomplikowanych zapytań,
- ograniczenia dostępu do danych wyłącznie dla uprawnionych użytkowników.

Dzięki predefiniowanym widokom system umożliwia szybkie i efektywne zarządzanie informacjami w bazie danych, zapewniając jednocześnie odpowiedni poziom bezpieczeństwa poprzez precyzyjne przypisanie uprawnień do widoków dla różnych grup użytkowników.

8 Indeksy w bazie danych

W celu optymalizacji wydajności zapytań w bazie danych zastosowano indeksy na kluczowych kolumnach tabel. Indeksy te przyspieszają wyszukiwanie danych, sortowanie oraz filtrowanie wyników. Poniżej przedstawiono szczegółowy opis utworzonych indeksów wraz z ich uzasadnieniem.

8.1 Indeksy dla studentów

Dla tabeli `Students` utworzono kilka indeksów wspierających szybkie wyszukiwanie studentów według istotnych atrybutów:

- **Indeks na e-mail** (`IX_Students_Email`) – pozwala na szybkie wyszukiwanie studentów po adresie e-mail, co jest szczególnie przydatne w systemach logowania i korespondencji.
- **Indeks na nazwisko** (`IX_Students_LastName`) – umożliwia sprawne filtrowanie i sortowanie po nazwisku.
- **Indeks na numer telefonu** (`IX_Students_Phone`) – wspomaga wyszukiwanie studentów na podstawie numeru kontaktowego.
- **Indeks na kod pocztowy** (`IX_Students_PostalCode`) – ułatwia filtrowanie studentów według lokalizacji, np. w celach analitycznych.

8.2 Indeksy dla kursów

W celu zwiększenia efektywności operacji związanych z kursami, utworzono indeksy w tabeli `Courses`:

- **Indeks na nazwę kursu** (`IX_Courses_CourseName`) – umożliwia szybkie wyszukiwanie kursów według ich nazwy.
- **Indeks na cenę kursu** (`IX_Courses_CoursePrice`) – wspiera operacje filtrowania kursów według przedziałów cenowych.
- **Indeks na koordynatora kursu** (`IX_Courses_Coordinator`) – ułatwia wyszukiwanie kursów prowadzonych przez konkretnego nauczyciela.

8.3 Indeksy dla zamówień

Optymalizacja operacji związanych z zamówieniami (`Orders`) i płatnościami (`OrderPaymentStatus`) została osiągnięta poprzez zastosowanie następujących indeksów:

- **Indeks na datę zamówienia** (`IX_Orders_OrderDate`) – pozwala na szybkie wyszukiwanie zamówień w określonych ramach czasowych.
- **Indeks na status płatności** (`IX_OrderPaymentStatus_Status`) – wspiera operacje związane z filtrowaniem zamówień według statusu (`Pending`, `Paid`).
- **Indeks na pracownika obsługującego zamówienie** (`IX_Orders_EmployeeHandling`) – ułatwia analizę zamówień obsługiwanych przez konkretnych pracowników administracyjnych.

8.4 Indeksy dla nauczycieli i tłumaczy

W celu optymalizacji operacji związanych z językami nauczycieli i tłumaczy, dodano następujące indeksy:

- **Indeks na język tłumacza** (`IX_TranslatorsLanguages_LanguageID`) – wspomaga wyszukiwanie tłumaczy obsługujących konkretne języki.

- **Indeks na język wykładowy nauczyciela** (`IX_TeacherLanguages_LanguageID`) – pozwala na szybkie filtrowanie nauczycieli według języka prowadzenia zajęć.

8.5 Indeksy dla webinarów i harmonogramu

W tabeli `Webinars` dodano indeks wspomagający operacje wyszukiwania webinarów według daty:

- **Indeks na datę webinaru** (`IX_Webinars_WebinarDate`) – przyspiesza zapytania dotyczące webinarów odbywających się w określonym terminie.

8.6 Podsumowanie

Zastosowane indeksy umożliwiają:

- Przyspieszenie wyszukiwania studentów, kursów i zamówień według kluczowych atrybutów.
- Optymalizację filtrowania według języka nauczycieli i tłumaczy.
- Usprawnienie analizy zamówień oraz statusu płatności.
- Zwiększenie efektywności operacji na webinarach i harmonogramie zajęć.

Dzięki dobrze zaprojektowanym indeksom baza danych zapewnia **wysoką wydajność** nawet przy dużej liczbie użytkowników i intensywnym wykorzystaniu systemu.

9 Funkcje (`functions`)

W poniższej sekcji przedstawiono implementację funkcji, które wspierają operacje obliczeniowe oraz analityczne w systemie bazodanowym. Każda funkcja posiada opis działania, parametrów oraz przykłady wywołań. Implementacja poniższych funkcji w systemie bazodanowym niesie ze sobą szereg korzyści:

- **Modularność i ponowne użycie kodu:** Każda funkcja realizuje jasno określone zadanie, co ułatwia późniejsze modyfikacje oraz ponowne wykorzystanie logiki biznesowej w wielu zapytaniach.
- **Poprawa wydajności:** Funkcje pozwalają na wykonywanie skomplikowanych obliczeń oraz łączenie danych w jednym wywołaniu, zmniejszając potrzebę powtarzania złożonej logiki w wielu miejscach aplikacji.
- **Łatwość utrzymania:** Zcentralizowana logika obliczeniowa umożliwia szybsze diagnozowanie błędów oraz łatwiejsze wprowadzanie zmian, co pozytywnie wpływa na utrzymanie i rozwój systemu.
- **Wsparcie dla analiz i raportowania:** Funkcje dedykowane do obliczania sum, średnich, przeliczania walut czy generowania harmonogramów umożliwiają dynamiczne generowanie raportów i analiz w czasie rzeczywistym.
- **Elastyczność:** Możliwość przekazywania parametrów (takich jak identyfikatory, daty czy języki) pozwala na dynamiczną adaptację funkcji do bieżących potrzeb biznesowych i operacyjnych.

9.1 Obliczanie całkowitej kwoty zamówienia

Opis: Funkcja `dbo.ufnGetOrderTotal` oblicza sumaryczną wartość zamówienia, sumując ceny wszystkich aktywności przypisanych do danego zamówienia (tabela `Activities`) na podstawie szczegółów zamówienia z tabeli `OrderDetails`. Jeśli zamówienie nie zawiera pozycji, zwracana jest wartość `0`.

```
CREATE OR ALTER FUNCTION dbo.ufnGetOrderTotal
(
    @OrderID INT
)
RETURNS MONEY
AS
BEGIN
    DECLARE @Total MONEY;

    SELECT @Total = SUM(A.Price)
    FROM OrderDetails OD
    JOIN Activities A ON A.ActivityID = OD.ActivityID
    WHERE OD.OrderID = @OrderID;

    IF @Total IS NULL
        SET @Total = 0;

    RETURN @Total;
END;
GO
```

Przykładowe wywołanie:

```
SELECT dbo.ufnGetOrderTotal(11) AS OrderTotal;
```

9.2 Sprawdzanie dostępności miejsca w grupie kursowej

Opis: Funkcja `dbo.ufnGetStationaryModuleFreeSlots` sprawdza liczbę wolnych miejsc w module stacjonarnym. Pobiera limit miejsc dla modułu z tabeli `StationaryModule` i odejmuje liczbę uczestników obecnych (tabela `CoursesAttendance` z warunkiem `Attendance = 1`). W przypadku braku modułu zwraca wartość `-1`.

```
CREATE OR ALTER FUNCTION dbo.ufnGetStationaryModuleFreeSlots
(
    @ModuleID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @Limit INT, @Count INT, @FreeSlots INT;

    SELECT @Limit = SM.[Limit]
    FROM StationaryModule SM
    WHERE SM.StationaryModuleID = @ModuleID;

    IF @Limit IS NULL
    BEGIN
        RETURN -1;
    END;

    SELECT @Count = COUNT(*)
    FROM CoursesAttendance CA
    WHERE CA.ModuleID = @ModuleID
      AND CA.Attendance = 1;

    SET @FreeSlots = @Limit - ISNULL(@Count,0);

    RETURN @FreeSlots;
END;
GO
```

Przykładowe wywołanie:

```
SELECT dbo.ufnGetStationaryModuleFreeSlots(10) AS FreeSlots;
```

9.3 Zliczanie aktywnych kursów w danym okresie

Opis: Funkcja `dbo.ufnCountActiveCoursesInPeriod` zlicza liczbę unikalnych kursów, które są aktywne w zadanym przedziale czasowym. Łączy tabele `Courses`, `Activities` i `CourseModules` i uwzględnia tylko kursy, dla których aktywność jest aktywna (`a.Active = 1`) oraz moduły mieszczą się w przedziale czasowym.

```
CREATE OR ALTER FUNCTION dbo.ufnCountActiveCoursesInPeriod
(
    @StartDate DATETIME,
    @EndDate   DATETIME
)
RETURNS INT
AS
BEGIN
    DECLARE @Count INT;

    WITH ActiveCourses AS
    (
      SELECT DISTINCT c.CourseID
      FROM Courses c
      JOIN Activities a ON a.ActivityID = c.ActivityID
      JOIN CourseModules cm ON cm.CourseID = c.CourseID
      WHERE a.Active = 1
        AND cm.Date >= @StartDate
        AND cm.Date <  @EndDate
    )
    SELECT @Count = COUNT(*)
    FROM ActiveCourses;

    RETURN @Count;
END;
GO
```

Przykładowe wywołanie:

```
SELECT dbo.ufnCountActiveCoursesInPeriod('2025-01-01','2025-12-31') AS
   TotalActiveCourses;
```

9.4 Pobieranie średniej ocen z przedmiotu

Opis: Funkcja `dbo.ufnGetSubjectAverageGrade` oblicza średnią ocen dla podanego przedmiotu na podstawie danych z tabeli `SubjectGrades`. Oceny są rzutowane na typ `DECIMAL(5,2)` i uśredniane. W przypadku braku ocen wynik jest ustawiany na `0`.

```
CREATE OR ALTER FUNCTION dbo.ufnGetSubjectAverageGrade
(
    @SubjectID INT
)
RETURNS DECIMAL(5,2)
AS
BEGIN
    DECLARE @Average DECIMAL(5,2);

    SELECT @Average = AVG(CAST(SubjectGrade AS DECIMAL(5,2)))
    FROM SubjectGrades
    WHERE SubjectID = @SubjectID;

    IF @Average IS NULL
        SET @Average = 0;

    RETURN @Average;
END;
GO
```

9.5 Obliczanie wolnych miejsc w budynku

Opis: Funkcja `dbo.ufnGetFreeSeatsInBuilding` analizuje zajętość sal w budynku i oblicza liczbę dostępnych miejsc. Wykorzystuje dwa CTE: pierwszy pobiera sale (z tabeli `StationaryClass`) odpowiadające danemu identyfikatorowi budynku (`ClassID`), a drugi zlicza liczbę zajętych miejsc. Wynik obliczany jest jako suma różnicy między limitem miejsc a liczbą zajętych miejsc.

```
CREATE OR ALTER FUNCTION dbo.ufnGetFreeSeatsInBuilding
(
    @ClassID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @FreeSeats INT;

    ;WITH BuildingRooms AS
    (
        SELECT SC.StationaryClassID, SC.[Limit]
        FROM StationaryClass SC
        WHERE SC.ClassID = @ClassID
    ),
    Occupancy AS
    (
        SELECT
            br.StationaryClassID,
            COUNT(*) AS Occupied
        FROM BuildingRooms br
        JOIN StudiesClass sc ON sc.StudyClassID = br.StationaryClassID
        JOIN StudiesClassAttendance sca ON sca.StudyClassID = sc.StudyClassID
        GROUP BY br.StationaryClassID
    )
    SELECT @FreeSeats = SUM(br.[Limit] - ISNULL(o.Occupied, 0))
    FROM BuildingRooms br
    LEFT JOIN Occupancy o ON o.StationaryClassID = br.StationaryClassID;

    RETURN ISNULL(@FreeSeats, 0);
END;
GO
```

9.6 Konwersja walut w cenach aktywności (EUR/PLN)

Opis: Funkcja `dbo.ufnConvertActivityPriceToEUR` przelicza cenę aktywności wyrażoną w PLN na EUR. Najpierw pobiera cenę aktywności z tabeli `Activities`, a następnie wyszukuje kurs wymiany (tabela `EuroExchangeRate`) obowiązujący na lub przed podaną datą. W razie braku ceny lub kursu, funkcja zwraca `0`.

```
CREATE OR ALTER FUNCTION dbo.ufnConvertActivityPriceToEUR
(
    @ActivityID INT,
    @RateDate   DATETIME
)
RETURNS DECIMAL(10,2)
AS
BEGIN
    DECLARE @PLN MONEY, @Rate DECIMAL(10,2), @EUR DECIMAL(10,2);

    SELECT @PLN = Price
    FROM Activities
    WHERE ActivityID = @ActivityID;

    IF @PLN IS NULL
        RETURN 0;

    SELECT TOP(1) @Rate = Rate
    FROM EuroExchangeRate
    WHERE [Date] <= @RateDate
    ORDER BY [Date] DESC;

    IF @Rate IS NULL
        RETURN 0;

    SET @EUR = CAST(@PLN AS DECIMAL(10,2)) / @Rate;

    RETURN @EUR;
END;
GO
```

Przykładowe wywołanie:

```
SELECT dbo.ufnConvertActivityPriceToEUR(101, '2025-01-10') AS PriceInEUR;
```

9.7 Pobieranie listy nauczycieli dla danego języka

Opis: Funkcja `dbo.ufnGetTeachersByLanguage` zwraca tabelaryczny zbiór danych z informacjami o nauczycielach, którzy prowadzą zajęcia w określonym języku. Łączy tabele `Teachers` i `TeacherLanguages` na podstawie `TeacherID`.

```
CREATE OR ALTER FUNCTION dbo.ufnGetTeachersByLanguage
(
    @LanguageID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT t.TeacherID,
           t.FirstName,
           t.LastName,
           t.Email
    FROM Teachers t
    JOIN TeacherLanguages tl ON tl.TeacherID = t.TeacherID
    WHERE tl.LanguageID = @LanguageID
);
GO
```

Przykładowe wywołanie:

```
SELECT * FROM dbo.ufnGetTeachersByLanguage(2);
```

9.8 Obliczanie liczby zajęć w kursie

Opis: Funkcja `dbo.ufnGetCourseTotalHours` sumuje łączny czas trwania modułów kursu (dane z tabeli `CourseModules`) wyrażony w minutach. Wynik przeliczany jest do formatu godzin dziesiętnych (np. 90 minut → 1.50 h).

```
CREATE OR ALTER FUNCTION dbo.ufnGetCourseTotalHours
(
    @CourseID INT
)
RETURNS DECIMAL(5,2)
AS
BEGIN
    DECLARE @TotalMinutes INT;

    SELECT @TotalMinutes = SUM(DATEDIFF(MINUTE, 0, DurationTime))
    FROM CourseModules
    WHERE CourseID = @CourseID;

    IF @TotalMinutes IS NULL
        SET @TotalMinutes = 0;

    RETURN CAST(@TotalMinutes AS DECIMAL(5,2)) / 60;
END;
GO
```

9.9 Wyświetlanie listy aktywności dostępnych dla danego języka

Opis: Funkcja `dbo.ufnListActivitiesByLanguage` zwraca zbiór aktywności (tabela wynikowa) odpowiadających podanemu językowi. Uwzględnia trzy typy aktywności: Webinary, Kursy, Studia.

```sql
CREATE OR ALTER FUNCTION dbo.ufnListActivitiesByLanguage
(
    @LanguageID INT
)
RETURNS @Result TABLE
(
    ActivityType VARCHAR(20),
    ActivityName VARCHAR(50),
    ActivityDate DATETIME,
    LanguageID   INT,
    Price        MONEY
)
AS
BEGIN
    INSERT INTO @Result
    SELECT
        'Webinar' AS ActivityType,
        w.WebinarName AS ActivityName,
        w.WebinarDate AS ActivityDate,
        w.LanguageID,
        a.Price
    FROM Webinars w
    JOIN Activities a ON a.ActivityID = w.ActivityID
    WHERE w.LanguageID = @LanguageID
      AND a.Active = 1;

    INSERT INTO @Result
    SELECT
        'CourseModule' AS ActivityType,
        c.CourseName + ' - ' + cm.ModuleName AS ActivityName,
        cm.Date AS ActivityDate,
        cm.LanguageID,
        a.Price
    FROM CourseModules cm
    JOIN Courses c ON c.CourseID = cm.CourseID
    JOIN Activities a ON a.ActivityID = c.ActivityID
    WHERE cm.LanguageID = @LanguageID
      AND a.Active = 1;

    INSERT INTO @Result
    SELECT
        'StudiesClass' AS ActivityType,
        sc.ClassName AS ActivityName,
        sc.[Date] AS ActivityDate,
        sc.LanguageID,
        a.Price
    FROM StudiesClass sc
    JOIN Activities a ON a.ActivityID = sc.ActivityID
    WHERE sc.LanguageID = @LanguageID
      AND a.Active = 1;

    RETURN;
END;
GO
```

Przykładowe wywołanie:

```sql
SELECT *
FROM dbo.ufnListActivitiesByLanguage(3)
ORDER BY ActivityDate;
```

9.10 Obliczanie sumarycznego czasu trwania webinaru

Opis: Funkcja `dbo.ufnGetWebinarTotalHours` oblicza łączny czas trwania webinaru na podstawie pola `DurationTime` w tabeli `Webinars`. Wynik wyrażony jest w godzinach dziesiętnych.

```sql
CREATE OR ALTER FUNCTION dbo.ufnGetWebinarTotalHours
(
    @WebinarID INT
)
RETURNS DECIMAL(5,2)
AS
BEGIN
    DECLARE @Minutes INT;

    SELECT @Minutes = DATEDIFF(MINUTE, 0, DurationTime)
    FROM Webinars
    WHERE WebinarID = @WebinarID;

    IF @Minutes IS NULL
        SET @Minutes = 0;

    RETURN CAST(@Minutes AS DECIMAL(5,2)) / 60;
END;
GO
```

9.11 Obliczanie liczby uczestników w kursie

Opis: Funkcja `dbo.ufnGetCourseTotalParticipants` zlicza liczbę uczestników w kursie na podstawie obecności (pole `Attendance = 1`) z tabeli `CoursesAttendance`.

```sql
CREATE OR ALTER FUNCTION dbo.ufnGetCourseTotalParticipants
(
    @CourseID INT
)
RETURNS INT
AS
BEGIN
    DECLARE @TotalParticipants INT;

    SELECT @TotalParticipants = COUNT(*)
    FROM CoursesAttendance
    WHERE ModuleID = @CourseID
      AND Attendance = 1;

    RETURN @TotalParticipants;
END;
GO
```

9.12 Harmonogram zajęć dla studenta

Opis: Funkcja `dbo.ufnGetStudentSchedule` zwraca tabelaryczny harmonogram zajęć dla studenta. Uwzględnia:

- Kursy (moduły) – łączone z kursami `CourseParticipants` i `CourseModules`,
- Webinary – na podstawie tabel `WebinarDetails` i `Webinars`,
- Studia – zajęcia pobierane z tabel `StudiesClassAttendance` i `StudiesClass`.

```
CREATE OR ALTER FUNCTION dbo.ufnGetStudentSchedule
(
    @StudentID INT
)
RETURNS @Schedule TABLE
(
    ActivityType  VARCHAR(20),
    ActivityName  VARCHAR(50),
    StartDate     DATETIME,
    EndDate       DATETIME
)
AS
BEGIN
    INSERT INTO @Schedule
    SELECT
        'Course' AS ActivityType,
        c.CourseName + ' - ' + cm.ModuleName AS ActivityName,
        cm.Date AS StartDate,
        DATEADD(MINUTE, DATEDIFF(MINUTE, 0, cm.DurationTime), cm.Date) AS
    EndDate
    FROM CourseParticipants cp
    JOIN Courses c ON c.CourseID = cp.CourseID
    JOIN CourseModules cm ON cm.CourseID = c.CourseID
    WHERE cp.StudentID = @StudentID;

    INSERT INTO @Schedule
    SELECT
        'Webinar' AS ActivityType,
        w.WebinarName AS ActivityName,
        w.WebinarDate AS StartDate,
        DATEADD(MINUTE, DATEDIFF(MINUTE, 0, w.DurationTime), w.WebinarDate) AS
    EndDate
    FROM WebinarDetails wd
    JOIN Webinars w ON w.WebinarID = wd.WebinarID
    WHERE wd.StudentID = @StudentID;

    INSERT INTO @Schedule
    SELECT
        'StudiesClass' AS ActivityType,
        sc.ClassName AS ActivityName,
        sc.[Date] AS StartDate,
        DATEADD(MINUTE, DATEDIFF(MINUTE, 0, sc.DurationTime), sc.[Date]) AS
    EndDate
    FROM StudiesClassAttendance sca
    JOIN StudiesClass sc ON sc.StudyClassID = sca.StudyClassID
    WHERE sca.StudentID = @StudentID;

    RETURN;
END;
GO
```

9.13 Generowanie raportu ocen dla danego studenta

Opis: Funkcja `dbo.ufnGetStudentGrades` zwraca tabelaryczny zbiór raportu ocen studenta dla poszczególnych przedmiotów. Dane pobierane są z tabel `SubjectGrades` oraz `Subject`.

```
CREATE OR ALTER FUNCTION dbo.ufnGetStudentGrades
(
    @StudentID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        sg.SubjectID,
        s.SubjectName,
        sg.SubjectGrade
    FROM SubjectGrades sg
    JOIN Subject s ON s.SubjectID = sg.SubjectID
    WHERE sg.StudentID = @StudentID
);
GO
```

9.14 Generowanie listy obecności dla danego kursu

Opis: Funkcja `dbo.ufnGetCourseAttendanceList` tworzy tabelaryczną listę obecności studentów w kursie. Łączy tabele `CoursesAttendance`, `CourseModules` oraz `Students`, aby wyświetlić m.in. nazwy modułów i dane studentów.

```
CREATE OR ALTER FUNCTION dbo.ufnGetCourseAttendanceList
(
    @CourseID INT
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        ca.ModuleID,
        cm.ModuleName,
        ca.StudentID,
        s.FirstName,
        s.LastName,
        ca.Attendance AS WasPresent
    FROM CoursesAttendance ca
    JOIN CourseModules cm ON cm.ModuleID = ca.ModuleID
    JOIN Students s ON s.StudentID = ca.StudentID
    WHERE cm.CourseID = @CourseID
);
GO
```

9.15 Generowanie listy obecności dla danego studenta

Opis: Funkcja `dbo.ufnGetStudentAllAttendances` generuje tabelaryczny raport obecności studenta, łącząc dane dotyczące zajęć w kursach, studiach oraz webinarach. Dla każdej aktywności podawany jest typ, nazwa, data wydarzenia oraz status obecności.

```
CREATE OR ALTER FUNCTION dbo.ufnGetStudentAllAttendances
(
    @StudentID INT
)
RETURNS @Attendances TABLE
(
    ActivityType  VARCHAR(20),
    ActivityName  VARCHAR(50),
    DateOfEvent   DATETIME,
    WasPresent    BIT
)
AS
BEGIN
    INSERT INTO @Attendances
    SELECT
        'Course' AS ActivityType,
        c.CourseName + ' - ' + cm.ModuleName,
        cm.Date,
        ca.Attendance
    FROM CoursesAttendance ca
    JOIN CourseModules cm ON cm.ModuleID = ca.ModuleID
    JOIN Courses c ON c.CourseID = cm.CourseID
    WHERE ca.StudentID = @StudentID;

    INSERT INTO @Attendances
    SELECT
        'StudiesClass',
        sc.ClassName,
        sc.[Date],
        sca.Attendance
    FROM StudiesClassAttendance sca
    JOIN StudiesClass sc ON sc.StudyClassID = sca.StudyClassID
    WHERE sca.StudentID = @StudentID;

    INSERT INTO @Attendances
    SELECT
        'Webinar',
        w.WebinarName,
        w.WebinarDate,
        wd.Complete
    FROM WebinarDetails wd
    JOIN Webinars w ON w.WebinarID = wd.WebinarID
    WHERE wd.StudentID = @StudentID;

    RETURN;
END;
GO
```

10 Procedury (stored procedures)

Poniżej przedstawiono zestawienie procedur składowanych wykorzystywanych w systemie wraz z ich kodem oraz omówieniem funkcjonalności.

10.1 Dodawanie nowego kursu (spAddCourse)

Cel: Dodaje nowy kurs oraz odpowiadającą mu aktywność do bazy danych.

Parametry wejściowe:

- `@CourseName` – nazwa kursu,
- `@CourseDescription` – opcjonalny opis kursu,
- `@CoursePrice` – cena kursu,
- `@CourseCoordinatorID` – identyfikator koordynatora kursu,
- `@ActivityTitle` – tytuł aktywności przypisanej do kursu,
- `@ActivityPrice` – cena aktywności,
- `@ActivityActive` – status aktywności (domyślnie 1 – aktywna).

Działanie: Procedura generuje nowe identyfikatory dla aktywności i kursu przy użyciu wyrażenia `ISNULL(MAX(...),0)+1`, wstawia rekordy do tabel `Activities` i `Courses`, a następnie zwraca nowe identyfikatory.

```
CREATE OR ALTER PROCEDURE dbo.spAddCourse
  @CourseName           VARCHAR(50),
  @CourseDescription    TEXT       = NULL,
  @CoursePrice          MONEY,
  @CourseCoordinatorID  INT,
  @ActivityTitle        VARCHAR(50),
  @ActivityPrice        MONEY,
  @ActivityActive       BIT = 1
AS
BEGIN
    SET NOCOUNT ON;

    -- Wygenerowanie nowego ActivityID
    DECLARE @NewActivityID INT = (
        SELECT ISNULL(MAX(ActivityID), 0) + 1
        FROM Activities
    );

    INSERT INTO Activities (ActivityID, Price, Title, Active)
    VALUES (@NewActivityID, @ActivityPrice, @ActivityTitle, @ActivityActive);

    -- Wygenerowanie nowego CourseID
    DECLARE @NewCourseID INT = (
        SELECT ISNULL(MAX(CourseID), 0) + 1
        FROM Courses
    );

    INSERT INTO Courses (CourseID, ActivityID, CourseName, CourseDescription,
    CoursePrice, CourseCoordinatorID)
    VALUES (
        @NewCourseID,
        @NewActivityID,
        @CourseName,
```

```
        @CourseDescription,
        @CoursePrice,
        @CourseCoordinatorID
    );

    SELECT @NewCourseID AS CreatedCourseID, @NewActivityID AS CreatedActivityID;
END;
GO
```

10.2 Usuwanie kursu (spRemoveCourse)

Cel: Usuwa kurs oraz powiązaną z nim aktywność.

Parametry wejściowe:

- `@CourseID` – identyfikator kursu.

Działanie: Procedura wyszukuje `ActivityID` powiązane z kursem. W przypadku nieznalezienia kursu generowany jest błąd. Następnie usuwa rekordy z tabel `Courses` oraz `Activities`.

```
CREATE OR ALTER PROCEDURE dbo.spRemoveCourse
  @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;

    -- źZnajd ąpowizany ActivityID
    DECLARE @ActivityID INT;

    SELECT @ActivityID = ActivityID
    FROM Courses
    WHERE CourseID = @CourseID;

    IF @ActivityID IS NULL
    BEGIN
        RAISERROR('Course not found.', 16, 1);
        RETURN;
    END;

    -- ęUsunicie z Courses
    DELETE FROM Courses
    WHERE CourseID = @CourseID;

    -- ęUsunicie z Activities (opcjonalne)
    DELETE FROM Activities
    WHERE ActivityID = @ActivityID;

    PRINT 'Course and related Activity removed successfully.';
END;
GO
```

10.3 Rejestracja studenta na kurs (`spRegisterStudentInCourse`)

Cel: Dodaje studenta do listy uczestników wybranego kursu.

Parametry wejściowe:

- `@CourseID` – identyfikator kursu,
- `@StudentID` – identyfikator studenta.

Działanie: Sprawdza, czy dany student nie jest już zapisany na kurs. W przypadku braku wpisu, wstawia nowy rekord do tabeli `CourseParticipants`, a następnie wyświetla komunikat powodzenia.

```
CREATE OR ALTER PROCEDURE dbo.spRegisterStudentInCourse
  @CourseID  INT,
  @StudentID INT
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS (
        SELECT 1 FROM CourseParticipants
        WHERE CourseID = @CourseID AND StudentID = @StudentID
    )
    BEGIN
        RAISERROR('Student is already registered in this course.', 16, 1);
        RETURN;
    END;

    INSERT INTO CourseParticipants (CourseID, StudentID)
    VALUES (@CourseID, @StudentID);

    PRINT 'Student registered successfully.';
END;
GO
```

10.4 Wypisanie studenta z kursu (spUnregisterStudentFromCourse)

Cel: Usuwa studenta z listy uczestników kursu.

Parametry wejściowe:

- `@CourseID` – identyfikator kursu,
- `@StudentID` – identyfikator studenta.

Działanie: Procedura usuwa rekord z tabeli `CourseParticipants` odpowiadający danemu studentowi i kursowi. Jeśli rekord nie został usunięty (co oznacza, że wpis nie istniał), generowany jest błąd.

```
CREATE OR ALTER PROCEDURE dbo.spUnregisterStudentFromCourse
  @CourseID  INT,
  @StudentID INT
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM CourseParticipants
    WHERE CourseID = @CourseID
      AND StudentID = @StudentID;

    IF @@ROWCOUNT = 0
    BEGIN
        RAISERROR('Student not found in that course.', 16, 1);
    END
    ELSE
    BEGIN
        PRINT 'Student unregistered successfully.';
    END
END;
GO
```

10.5 Aktualizacja ceny aktywności (`spUpdateActivityPrice`)

Cel: Umożliwia zmianę ceny aktywności.

Parametry wejściowe:

- `@ActivityID` – identyfikator aktywności,
- `@NewPrice` – nowa cena.

Działanie: Procedura aktualizuje pole `Price` w tabeli `Activities` dla podanego `ActivityID`. W przypadku braku rekordu wyświetlany jest błąd.

```
CREATE OR ALTER PROCEDURE dbo.spUpdateActivityPrice
  @ActivityID INT,
  @NewPrice   MONEY
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Activities
    SET Price = @NewPrice
    WHERE ActivityID = @ActivityID;

    IF @@ROWCOUNT = 0
        RAISERROR('Activity not found.', 16, 1);
    ELSE
        PRINT 'Activity price updated successfully.';
END;
GO
```

10.6 Zarządzanie harmonogramem nauczyciela (`spAddTeacherSchedule`)

Cel: Dodaje wpis do harmonogramu nauczyciela, sprawdzając, czy nie zachodzi konflikt terminowy.

Parametry wejściowe:

- `@TeacherID` – identyfikator nauczyciela,
- `@ClassID` – identyfikator klasy,
- `@CourseModuleID` – opcjonalny identyfikator modułu kursu,
- `@StudiesSubjectID` – opcjonalny identyfikator przedmiotu studiów,
- `@DayOfWeek` – dzień tygodnia,
- `@StartTime` – godzina rozpoczęcia,
- `@EndTime` – godzina zakończenia,
- `@TranslatorID` – opcjonalny identyfikator tłumacza.

Działanie: Procedura najpierw sprawdza, czy dla tego samego nauczyciela i dnia tygodnia następuje kolizja godzinowa. Jeżeli nie, generuje nowy identyfikator dla wpisu w harmonogramie, wstawia rekord do tabeli `Schedule` i zwraca `NewScheduleID`.

```
CREATE OR ALTER PROCEDURE dbo.spAddTeacherSchedule
  @TeacherID         INT,
  @ClassID           INT,
  @CourseModuleID    INT = NULL,
  @StudiesSubjectID INT = NULL,
  @DayOfWeek         VARCHAR(10),
  @StartTime         TIME,
  @EndTime           TIME,
  @TranslatorID      INT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    -- Sprawdzenie kolizji godzinowej
    IF EXISTS (
        SELECT 1
        FROM Schedule
        WHERE TeacherID = @TeacherID
          AND DayOfWeek = @DayOfWeek
          AND (@StartTime < EndTime AND @EndTime > StartTime)
    )
    BEGIN
        RAISERROR('Collision in the teacher schedule!', 16, 1);
        RETURN;
    END;

    DECLARE @NewScheduleID INT = (
        SELECT ISNULL(MAX(ScheduleID), 0) + 1
        FROM Schedule
    );

    INSERT INTO Schedule (
        ScheduleID, ClassID, CourseModuleID, StudiesSubjectID,
        DayOfWeek, StartTime, EndTime,
        TeacherID, TranslatorID
    )
    VALUES (
```

```
        @NewScheduleID, @ClassID, @CourseModuleID, @StudiesSubjectID,
        @DayOfWeek, @StartTime, @EndTime,
        @TeacherID, @TranslatorID
    );

    SELECT @NewScheduleID AS NewScheduleID;
    PRINT 'Schedule added successfully.';
END;
GO
```

10.7 Wyszukiwanie dostępnych aktywności (`spFindAvailableActivities`)

Cel: Wyszukuje aktywności (webinary, kursy i zajęcia studiów) w określonym przedziale czasowym.

Parametry wejściowe:

- `@StartDate` – data początkowa,
- `@EndDate` – data końcowa.

Działanie: Za pomocą operatora `UNION` łączy wyniki zapytań wyszukujących webinary, kursy (na podstawie dat modułów) oraz zajęcia studiów, filtrując tylko aktywne rekordy.

```
CREATE OR ALTER PROCEDURE dbo.spFindAvailableActivities
  @StartDate DATETIME,
  @EndDate   DATETIME
AS
BEGIN
    SET NOCOUNT ON;

    -- Webinary w tym przedziale
    SELECT
        'Webinar' AS ActivityType,
        w.WebinarID AS ActivityID,
        w.WebinarName AS ActivityName,
        w.WebinarDate AS [Date],
        a.Price AS ActivityPrice
    FROM Webinars w
    JOIN Activities a ON a.ActivityID = w.ActivityID
    WHERE w.WebinarDate >= @StartDate
      AND w.WebinarDate < @EndDate
      AND a.Active = 1

    UNION

    -- Kursy (daty łmoduów)
    SELECT
        'Course' AS ActivityType,
        c.CourseID AS ActivityID,
        c.CourseName AS ActivityName,
        cm.Date AS [Date],
        a.Price AS ActivityPrice
    FROM Courses c
    JOIN CourseModules cm ON cm.CourseID = c.CourseID
    JOIN Activities a ON a.ActivityID = c.ActivityID
    WHERE cm.Date >= @StartDate
      AND cm.Date < @EndDate
      AND a.Active = 1

    UNION

    -- Studia (ęzajcia StudiesClass)
    SELECT
        'StudiesClass' AS ActivityType,
        sc.StudyClassID AS ActivityID,
        sc.ClassName AS ActivityName,
        sc.[Date] AS [Date],
        a.Price AS ActivityPrice
    FROM StudiesClass sc
    JOIN Activities a ON a.ActivityID = sc.ActivityID
    WHERE sc.[Date] >= @StartDate
      AND sc.[Date] < @EndDate
```

```
50        AND a.Active = 1;
51 END;
52 GO
```

10.8 Rejestracja nowego nauczyciela (`spAddTeacher`)

Cel: Dodaje nowego nauczyciela do bazy oraz przypisuje mu języki nauczania (przekazane w formie CSV).

Parametry wejściowe:

- `@FirstName` – imię,
- `@LastName` – nazwisko,
- `@HireDate` – data zatrudnienia (opcjonalnie),
- `@Phone` – numer telefonu (opcjonalnie),
- `@Email` – adres e-mail,
- `@LanguagesCSV` – lista identyfikatorów języków w formacie CSV.

Działanie: Procedura generuje nowe ID nauczyciela, wstawia rekord do tabeli `Teachers`, a następnie, jeśli przekazano listę języków, wykorzystuje funkcję `STRING_SPLIT` do masowego przypisania języków w tabeli `TeacherLanguages`.

```
CREATE OR ALTER PROCEDURE dbo.spAddTeacher
  @FirstName VARCHAR(30),
  @LastName  VARCHAR(30),
  @HireDate  DATE        = NULL,
  @Phone     VARCHAR(15) = NULL,
  @Email     VARCHAR(60),
  @LanguagesCSV VARCHAR(MAX) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewTeacherID INT = (
        SELECT ISNULL(MAX(TeacherID), 0) + 1
        FROM Teachers
    );

    INSERT INTO Teachers (TeacherID, FirstName, LastName, HireDate, Phone, Email
    )
    VALUES (@NewTeacherID, @FirstName, @LastName, @HireDate, @Phone, @Email);

    PRINT 'Teacher created with ID = ' + CAST(@NewTeacherID AS VARCHAR(10));

    IF @LanguagesCSV IS NOT NULL AND LEN(@LanguagesCSV) > 0
    BEGIN
        ;WITH CTE_Lang AS (
            SELECT value AS LangID
            FROM STRING_SPLIT(@LanguagesCSV, ',')
        )
        INSERT INTO TeacherLanguages (TeacherID, LanguageID)
        SELECT @NewTeacherID, CAST(LangID AS INT)
        FROM CTE_Lang;
    END;

    PRINT 'Teacher languages assigned.';
END;
GO
```

10.9 Rejestracja nowego tłumacza (`spAddTranslator`)

Cel: Dodaje nowego tłumacza do bazy oraz przypisuje mu obsługiwane języki (w formacie CSV).

Parametry wejściowe:

- `@FirstName` – imię,
- `@LastName` – nazwisko,
- `@HireDate` – data zatrudnienia (opcjonalnie),
- `@Phone` – numer telefonu (opcjonalnie),
- `@Email` – adres e-mail,
- `@LanguagesCSV` – lista ID języków w formacie CSV.

Działanie: Procedura generuje nowe ID tłumacza, wstawia rekord do tabeli `Translators`, a następnie przypisuje tłumaczowi języki (przy użyciu `STRING_SPLIT` i wstawiając rekordy do `TranslatorsLanguages`).

```
CREATE OR ALTER PROCEDURE dbo.spAddTranslator
  @FirstName    VARCHAR(30),
  @LastName     VARCHAR(30),
  @HireDate     DATE         = NULL,
  @Phone        VARCHAR(15) = NULL,
  @Email        VARCHAR(60),
  @LanguagesCSV VARCHAR(MAX) = NULL
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewTranslatorID INT = (
        SELECT ISNULL(MAX(TranslatorID), 0) + 1
        FROM Translators
    );

    INSERT INTO Translators (TranslatorID, FirstName, LastName, HireDate, Phone,
    Email)
    VALUES (@NewTranslatorID, @FirstName, @LastName, @HireDate, @Phone, @Email);

    PRINT 'Translator created with ID = ' + CAST(@NewTranslatorID AS VARCHAR(10)
    );

    IF @LanguagesCSV IS NOT NULL AND LEN(@LanguagesCSV) > 0
    BEGIN
        ;WITH CTE_Lang AS (
            SELECT value AS LangID
            FROM STRING_SPLIT(@LanguagesCSV, ',')
        )
        INSERT INTO TranslatorsLanguages (TranslatorID, LanguageID)
        SELECT @NewTranslatorID, CAST(LangID AS INT)
        FROM CTE_Lang;
    END;

    PRINT 'Translator languages assigned.';
END;
GO
```

10.10 Tworzenie nowego webinaru (`spAddWebinar`)

Cel: Dodaje nowy webinar do systemu oraz tworzy powiązaną z nim aktywność.

Parametry wejściowe:

- `@WebinarName` – nazwa webinaru,
- `@WebinarDescription` – opis webinaru,
- `@WebinarPrice` – cena webinaru,
- `@TeacherID` – identyfikator nauczyciela,
- `@LanguageID` – identyfikator języka,
- `@VideoLink` – link do materiału wideo,
- `@WebinarDate` – data webinaru,
- `@DurationTime` – czas trwania,
- `@ActivityTitle` – tytuł powiązanej aktywności,
- `@ActivityPrice` – cena aktywności,
- `@ActivityActive` – status aktywności.

Działanie: Procedura generuje nowe ID aktywności oraz webinaru, wstawia rekordy do tabel `Activities` i `Webinars`, a następnie zwraca nowe identyfikatory.

```
CREATE OR ALTER PROCEDURE dbo.spAddWebinar
  @WebinarName        VARCHAR(50),
  @WebinarDescription TEXT,
  @WebinarPrice       MONEY,
  @TeacherID          INT,
  @LanguageID         INT,
  @VideoLink          VARCHAR(50),
  @WebinarDate        DATETIME,
  @DurationTime       TIME(0),
  @ActivityTitle      VARCHAR(50),
  @ActivityPrice      MONEY,
  @ActivityActive     BIT = 1
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewActivityID INT = (
        SELECT ISNULL(MAX(ActivityID), 0) + 1
        FROM Activities
    );

    INSERT INTO Activities (ActivityID, Price, Title, Active)
    VALUES (@NewActivityID, @ActivityPrice, @ActivityTitle, @ActivityActive);

    DECLARE @NewWebinarID INT = (
        SELECT ISNULL(MAX(WebinarID), 0) + 1
        FROM Webinars
    );

    INSERT INTO Webinars (
        WebinarID, ActivityID, TeacherID, WebinarName,
        WebinarPrice, VideoLink, WebinarDate, DurationTime,
```

```
        WebinarDescription, LanguageID
    )
    VALUES (
        @NewWebinarID, @NewActivityID, @TeacherID, @WebinarName,
        @WebinarPrice, @VideoLink, @WebinarDate, @DurationTime,
        @WebinarDescription, @LanguageID
    );

    SELECT @NewWebinarID AS NewWebinarID, @NewActivityID AS NewActivityID;
END;
GO
```

10.11 Usuwanie webinaru (`spRemoveWebinar`)

Cel: Usuwa webinar i powiązaną z nim aktywność.

Parametry wejściowe:

- `@WebinarID` – identyfikator webinaru.

Działanie: Procedura wyszukuje `ActivityID` powiązane z danym webinarem, a następnie usuwa rekordy z tabel `Webinars` i `Activities`. W przypadku braku webinaru generuje błąd.

```
CREATE OR ALTER PROCEDURE dbo.spRemoveWebinar
  @WebinarID INT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @ActivityID INT;
    SELECT @ActivityID = ActivityID
    FROM Webinars
    WHERE WebinarID = @WebinarID;

    IF @ActivityID IS NULL
    BEGIN
        RAISERROR('Webinar not found.', 16, 1);
        RETURN;
    END;

    DELETE FROM Webinars
    WHERE WebinarID = @WebinarID;

    DELETE FROM Activities
    WHERE ActivityID = @ActivityID;

    PRINT 'Webinar and related Activity removed.';
END;
GO
```

10.12 Dodawanie przedmiotu do planu studiów (`spAddSubjectToStudies`)

Cel: Dodaje nowy przedmiot do planu studiów.

Parametry wejściowe:

- `@StudiesID` – identyfikator studiów,
- `@CoordinatorID` – identyfikator koordynatora,
- `@SubjectName` – nazwa przedmiotu,
- `@SubjectDescription` – opcjonalny opis przedmiotu.

Działanie: Procedura generuje nowe `SubjectID`, wstawia rekord do tabeli `Subject`, a następnie zwraca nowe ID przedmiotu.

```
CREATE OR ALTER PROCEDURE dbo.spAddSubjectToStudies
  @StudiesID          INT,
  @CoordinatorID      INT,
  @SubjectName        VARCHAR(50),
  @SubjectDescription TEXT = NULL
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewSubjectID INT = (
        SELECT ISNULL(MAX(SubjectID), 0) + 1
        FROM Subject
    );

    INSERT INTO Subject (
        SubjectID, StudiesID, CoordinatorID,
        SubjectName, SubjectDescription
    )
    VALUES (
        @NewSubjectID,
        @StudiesID,
        @CoordinatorID,
        @SubjectName,
        @SubjectDescription
    );

    SELECT @NewSubjectID AS NewSubjectID;
END;
GO
```

10.13 Automatyczne rozdzielanie studentów do grup (`spAutoAssignStudentsT`

Cel: Automatycznie przypisuje studentów, którzy nie mają przydzielonej grupy, do jednej z dwóch grup (przykładowo przy użyciu prostej logiki modulo).

Działanie: Procedura aktualizuje kolumnę `GroupID` w tabeli `Students` (przy założeniu, że taka kolumna istnieje) używając operatora warunkowego.

```
CREATE OR ALTER PROCEDURE dbo.spAutoAssignStudentsToGroups
AS
BEGIN
    SET NOCOUNT ON;

    -- Przypisanie grupy: parzyste -> grupa 1, nieparzyste -> grupa 2
    UPDATE Students
    SET GroupID = CASE WHEN (StudentID % 2) = 0 THEN 1 ELSE 2 END
    WHERE GroupID IS NULL;

    PRINT 'Auto assignment done.';
END;
GO
```

10.14 Aktualizacja danych nauczyciela (`spUpdateTeacherData`)

Cel: Aktualizuje dane osobowe nauczyciela.

Parametry wejściowe:

- `@TeacherID` – identyfikator nauczyciela,
- `@FirstName`, `@LastName`, `@Phone`, `@Email`, `@HireDate` – nowe dane (opcjonalnie).

Działanie: Procedura aktualizuje rekord w tabeli `Teachers`, wykorzystując funkcję `COALESCE`, aby zachować istniejące dane, gdy nowe nie zostaną przekazane.

```
CREATE OR ALTER PROCEDURE dbo.spUpdateTeacherData
  @TeacherID INT,
  @FirstName VARCHAR(30)  = NULL,
  @LastName  VARCHAR(30)  = NULL,
  @Phone     VARCHAR(15)  = NULL,
  @Email     VARCHAR(60)  = NULL,
  @HireDate  DATE         = NULL
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Teachers
    SET
        FirstName = COALESCE(@FirstName, FirstName),
        LastName  = COALESCE(@LastName, LastName),
        Phone     = COALESCE(@Phone, Phone),
        Email     = COALESCE(@Email, Email),
        HireDate  = COALESCE(@HireDate, HireDate)
    WHERE TeacherID = @TeacherID;

    IF @@ROWCOUNT = 0
        RAISERROR('Teacher not found.', 16, 1);
    ELSE
        PRINT 'Teacher data updated successfully.';
END;
GO
```

10.15 Aktualizacja danych studenta (spUpdateStudentData)

Cel: Aktualizuje dane osobowe studenta.

Parametry wejściowe:

- `@StudentID` – identyfikator studenta,
- `@FirstName`, `@LastName`, `@Address`, `@CityID`, `@PostalCode`, `@Phone`, `@Email` – nowe dane (opcjonalnie).

Działanie: Podobnie jak dla nauczyciela, procedura aktualizuje rekord w tabeli `Students`, używając `COALESCE`.

```
CREATE OR ALTER PROCEDURE dbo.spUpdateStudentData
  @StudentID INT,
  @FirstName VARCHAR(30)  = NULL,
  @LastName  VARCHAR(30)  = NULL,
  @Address   VARCHAR(30)  = NULL,
  @CityID    INT          = NULL,
  @PostalCode VARCHAR(10) = NULL,
  @Phone     VARCHAR(15)  = NULL,
  @Email     VARCHAR(60)  = NULL
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Students
    SET
        FirstName  = COALESCE(@FirstName, FirstName),
        LastName   = COALESCE(@LastName, LastName),
        [Address]  = COALESCE(@Address, [Address]),
        CityID     = COALESCE(@CityID, CityID),
        PostalCode = COALESCE(@PostalCode, PostalCode),
        Phone      = COALESCE(@Phone, Phone),
        Email      = COALESCE(@Email, Email)
    WHERE StudentID = @StudentID;

    IF @@ROWCOUNT = 0
        RAISERROR('Student not found.', 16, 1);
    ELSE
        PRINT 'Student data updated successfully.';
END;
GO
```

10.16 Aktualizacja danych osobowych (RODO) (`spUpdateRODO`)

Cel: Aktualizuje lub wstawia dane dotyczące zgód RODO dla studenta.

Parametry wejściowe:

- `@StudentID` – identyfikator studenta,
- `@Withdraw` – flaga informująca o wycofaniu zgody (1 – wycofana, 0 – zgoda aktywna).

Działanie: Procedura sprawdza, czy dla danego studenta istnieje wpis w tabeli `RODO_Table`. Jeśli tak, aktualizuje dane; w przeciwnym razie wstawia nowy rekord. Data zapisywana jest jako bieżąca.

```
CREATE OR ALTER PROCEDURE dbo.spUpdateRODO
  @StudentID INT,
  @Withdraw  BIT
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @Today DATE = CONVERT(DATE, GETDATE());

    IF EXISTS (SELECT 1 FROM RODO_Table WHERE StudentID = @StudentID)
    BEGIN
        UPDATE RODO_Table
        SET [Date] = @Today,
            Withdraw = @Withdraw
        WHERE StudentID = @StudentID;
        PRINT 'RODO data updated.';
    END
    ELSE
    BEGIN
        INSERT INTO RODO_Table (StudentID, [Date], Withdraw)
        VALUES (@StudentID, @Today, @Withdraw);
        PRINT 'RODO data inserted.';
    END
END;
GO
```

10.17 Dodanie stażu (Internship) (`spAddInternship`)

Cel: Dodaje rekord stażu powiązanego z danymi studiów.

Parametry wejściowe:

- `@StudiesID` – identyfikator studiów,
- `@StartDate` – data rozpoczęcia stażu.

Działanie: Procedura generuje nowe `InternshipID`, wstawia rekord do tabeli `Internship`, a następnie zwraca nowe ID.

```
CREATE OR ALTER PROCEDURE dbo.spAddInternship
  @StudiesID INT,
  @StartDate DATETIME
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewInternshipID INT = (
        SELECT ISNULL(MAX(InternshipID), 0) + 1
        FROM Internship
    );

    INSERT INTO Internship (InternshipID, StudiesID, StartDate)
    VALUES (@NewInternshipID, @StudiesID, @StartDate);

    SELECT @NewInternshipID AS InternshipID;
END;
GO
```

10.18 Dodanie studenta do bazy (`spAddStudent`)

Cel: Wstawia nowego studenta do tabeli `Students`.

Parametry wejściowe:

- `@FirstName` – imię,
- `@LastName` – nazwisko,
- `@Address` – adres,
- `@CityID` – identyfikator miasta,
- `@PostalCode` – kod pocztowy,
- `@Phone` – numer telefonu (opcjonalnie),
- `@Email` – adres e-mail.

Działanie: Generuje nowe `StudentID`, wstawia rekord do tabeli i zwraca ID nowego studenta.

```
CREATE OR ALTER PROCEDURE dbo.spAddStudent
  @FirstName  VARCHAR(30),
  @LastName   VARCHAR(30),
  @Address    VARCHAR(30),
  @CityID     INT,
  @PostalCode VARCHAR(10),
  @Phone      VARCHAR(15) = NULL,
  @Email      VARCHAR(60)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewStudentID INT = (
        SELECT ISNULL(MAX(StudentID), 0) + 1
        FROM Students
    );

    INSERT INTO Students (
        StudentID, FirstName, LastName, [Address],
        CityID, PostalCode, Phone, Email
    )
    VALUES (
        @NewStudentID, @FirstName, @LastName, @Address,
        @CityID, @PostalCode, @Phone, @Email
    );

    SELECT @NewStudentID AS StudentID;
END;
GO
```

10.19 Usunięcie studenta z bazy (spRemoveStudent)

Cel: Usuwa studenta z bazy danych.

Parametry wejściowe:

- `@StudentID` – identyfikator studenta.

Działanie: Procedura usuwa rekord z tabeli `Students`. Jeśli nie znajdzie studenta, zgłasza błąd.

```
CREATE OR ALTER PROCEDURE dbo.spRemoveStudent
  @StudentID INT
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM Students
    WHERE StudentID = @StudentID;

    IF @@ROWCOUNT = 0
        RAISERROR('Student not found.', 16, 1);
    ELSE
        PRINT 'Student removed successfully.';
END;
GO
```

10.20 Dodanie pracownika (Employee) do bazy (`spAddEmployee`)

Cel: Dodaje nowego pracownika do tabeli `Employees`.

Parametry wejściowe:

- `@FirstName` – imię,
- `@LastName` – nazwisko,
- `@HireDate` – data zatrudnienia (opcjonalnie),
- `@EmployeeTypeID` – identyfikator typu pracownika,
- `@Phone` – numer telefonu (opcjonalnie),
- `@Email` – adres e-mail.

Działanie: Procedura generuje nowe `EmployeeID`, wstawia rekord do tabeli `Employees` oraz zwraca ID nowo dodanego pracownika.

```
CREATE OR ALTER PROCEDURE dbo.spAddEmployee
  @FirstName       VARCHAR(30),
  @LastName        VARCHAR(30),
  @HireDate        DATE         = NULL,
  @EmployeeTypeID  INT,
  @Phone           VARCHAR(15)  = NULL,
  @Email           VARCHAR(60)
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewEmployeeID INT = (
        SELECT ISNULL(MAX(EmployeeID), 0) + 1
        FROM Employees
    );

    INSERT INTO Employees (
        EmployeeID, FirstName, LastName, HireDate,
        EmployeeTypeID, Phone, Email
    )
    VALUES (
        @NewEmployeeID, @FirstName, @LastName, @HireDate,
        @EmployeeTypeID, @Phone, @Email
    );

    SELECT @NewEmployeeID AS EmployeeID;
END;
GO
```

10.21 Usunięcie pracownika z bazy (spRemoveEmployee)

Cel: Usuwa pracownika z tabeli Employees.

Parametry wejściowe:

- @EmployeeID – identyfikator pracownika.

Działanie: Procedura usuwa rekord z tabeli Employees, a w przypadku braku rekordu generuje błąd.

```
CREATE OR ALTER PROCEDURE dbo.spRemoveEmployee
  @EmployeeID INT
AS
BEGIN
    SET NOCOUNT ON;

    DELETE FROM Employees
    WHERE EmployeeID = @EmployeeID;

    IF @@ROWCOUNT = 0
        RAISERROR('Employee not found.', 16, 1);
    ELSE
        PRINT 'Employee removed successfully.';
END;
GO
```

10.22 Tworzenie modułu kursu (`spAddCourseModule`)

Cel: Tworzy nowy moduł kursu. W zależności od typu modułu (stacjonarny, online synchroniczny, online asynchroniczny) dodaje dodatkowy rekord do odpowiedniej tabeli.

Parametry wejściowe:

- `@CourseID` – identyfikator kursu,
- `@ModuleName` – nazwa modułu,
- `@ModuleDate` – data modułu,
- `@DurationTime` – czas trwania,
- `@TeacherID` – identyfikator nauczyciela,
- `@LanguageID` – identyfikator języka,
- `@TranslatorID` – opcjonalny identyfikator tłumacza,
- `@ModuleType` – typ modułu,
- `@ClassID` – dla modułów stacjonarnych,
- `@LinkOrVideo` – link lub materiał video dla modułów online,
- `@Limit` – limit miejsc (dla modułów stacjonarnych).

Działanie: Procedura wstawia rekord do tabeli `CourseModules`, a następnie w zależności od typu modułu, wstawia rekord do jednej z dodatkowych tabel: `StationaryModule`, `OnlineSyncModule` lub `OnlineAsyncModule`.

```
CREATE OR ALTER PROCEDURE dbo.spAddCourseModule
  @CourseID     INT,
  @ModuleName   VARCHAR(50),
  @ModuleDate   DATETIME,
  @DurationTime TIME(0),
  @TeacherID    INT,
  @LanguageID   INT,
  @TranslatorID INT = NULL,
  @ModuleType   VARCHAR(20),  -- 'stationary'/'online_sync'/'online_async'
  @ClassID      INT = NULL,   -- dla stacjonarnego
  @LinkOrVideo  VARCHAR(60) = NULL, -- link lub video
  @Limit        INT = 0       -- limit miejsc
AS
BEGIN
    SET NOCOUNT ON;

    DECLARE @NewModuleID INT = (
        SELECT ISNULL(MAX(ModuleID), 0) + 1
        FROM CourseModules
    );

    INSERT INTO CourseModules (
        ModuleID, CourseID, ModuleName, [Date], DurationTime,
        TeacherID, TranslatorID, LanguageID
    )
    VALUES (
        @NewModuleID, @CourseID, @ModuleName, @ModuleDate, @DurationTime,
        @TeacherID, @TranslatorID, @LanguageID
    );

```

```
    IF @ModuleType = 'stationary'
    BEGIN
        INSERT INTO StationaryModule (StationaryModuleID, ClassID, [Limit])
        VALUES (@NewModuleID, @ClassID, @Limit);
    END
    ELSE IF @ModuleType = 'online_sync'
    BEGIN
        INSERT INTO OnlineSyncModule (OnlineSyncModuleID, Link)
        VALUES (@NewModuleID, @LinkOrVideo);
    END
    ELSE IF @ModuleType = 'online_async'
    BEGIN
        INSERT INTO OnlineAsyncModule (OnlineAsyncModuleID, Video)
        VALUES (@NewModuleID, @LinkOrVideo);
    END
    ELSE
    BEGIN
        RAISERROR('Unknown module type.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END;

    PRINT 'Course module created with ID = ' + CAST(@NewModuleID AS VARCHAR(10))
    ;
END;
GO
```

10.23 Pobieranie kursu euro (`spAddEuroRate`)

Cel: Upsert (aktualizacja lub wstawienie) rekordu kursu euro dla określonej daty.

Parametry wejściowe:

- `@Rate` – kurs euro,
- `@RateDate` – opcjonalna data (jeśli nie podana, używany jest bieżący dzień).

Działanie: Procedura wykorzystuje instrukcję `MERGE` do wstawienia nowego rekordu lub aktualizacji istniejącego w tabeli `EuroExchangeRate`. Po wykonaniu wyświetla komunikat potwierdzający operację.

```
CREATE OR ALTER PROCEDURE dbo.spAddEuroRate
  @Rate DECIMAL(10,2),
  @RateDate DATETIME = NULL
AS
BEGIN
    SET NOCOUNT ON;

    IF @RateDate IS NULL
        SET @RateDate = CONVERT(DATETIME, CONVERT(DATE, GETDATE()));

    MERGE EuroExchangeRate AS tgt
    USING (SELECT @RateDate AS [Date], @Rate AS Rate) AS src
      ON (tgt.[Date] = src.[Date])
    WHEN MATCHED THEN
        UPDATE SET Rate = src.Rate
    WHEN NOT MATCHED THEN
        INSERT ([Date], Rate)
        VALUES (src.[Date], src.Rate)
    OUTPUT $action AS MergeAction;

    PRINT 'Euro rate upsert completed.';
END;
GO
```

10.24 Lista „dłużników” (spGetDebtors)

Cel: Zwraca listę studentów, którzy mają zamówienia nieopłacone (status różny od „Paid”).

Działanie: Łączy tabele: `Orders`, `OrderDetails`, `OrderPaymentStatus`, `Activities` oraz `Students`, filtrując wyniki według statusu płatności.

```
CREATE OR ALTER PROCEDURE dbo.spGetDebtors
AS
BEGIN
    SET NOCOUNT ON;

    SELECT DISTINCT
        S.StudentID,
        S.FirstName,
        S.LastName,
        O.OrderID,
        OPS.OrderPaymentStatus,
        OD.ActivityID,
        A.Title AS ActivityTitle
    FROM dbo.Orders O
    INNER JOIN dbo.OrderDetails OD ON O.OrderID = OD.OrderID
    INNER JOIN dbo.OrderPaymentStatus OPS ON O.PaymentURL = OPS.PaymentURL
    INNER JOIN dbo.Activities A ON A.ActivityID = OD.ActivityID
    INNER JOIN dbo.Students S ON O.StudentID = S.StudentID
    WHERE OPS.OrderPaymentStatus <> 'Paid'
    ORDER BY S.StudentID;
END;
GO
```

10.25 Raport frekwencji (spGenerateCourseAttendanceReport)

Cel: Generuje raport frekwencji dla kursu, studiów lub webinaru.

Parametry wejściowe:

- `@CourseID` – identyfikator kursu.

Działanie: Procedura łączy tabele `CoursesAttendance`, `CourseModules` oraz `Students`, sortując wyniki według daty modułu i identyfikatora studenta.

```
CREATE OR ALTER PROCEDURE dbo.spGenerateCourseAttendanceReport
  @CourseID INT
AS
BEGIN
    SET NOCOUNT ON;

    SELECT
        ca.ModuleID,
        cm.ModuleName,
        ca.StudentID,
        s.FirstName,
        s.LastName,
        ca.Attendance AS WasPresent
    FROM CoursesAttendance ca
    JOIN CourseModules cm ON cm.ModuleID = ca.ModuleID
    JOIN Students s ON s.StudentID = ca.StudentID
    WHERE cm.CourseID = @CourseID
    ORDER BY cm.Date, ca.StudentID;
END;
GO
```

10.26 Procedura oznaczania obecności studenta (spMarkCourseModuleAttenda

Cel: Rejestruje obecność studenta na module kursu.

Parametry wejściowe:

- `@ModuleID` – identyfikator modułu,
- `@StudentID` – identyfikator studenta,
- `@WasPresent` – status obecności (1 – obecny, 0 – nieobecny).

Działanie: Procedura próbuje zaktualizować rekord w tabeli `CoursesAttendance`. Jeśli rekord nie istnieje, wstawia nowy.

```
CREATE OR ALTER PROCEDURE dbo.spMarkCourseModuleAttendance
  @ModuleID  INT,
  @StudentID INT,
  @WasPresent BIT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE CoursesAttendance
    SET Attendance = @WasPresent
    WHERE ModuleID = @ModuleID
      AND StudentID = @StudentID;

    IF @@ROWCOUNT = 0
    BEGIN
        INSERT INTO CoursesAttendance (ModuleID, StudentID, Attendance)
        VALUES (@ModuleID, @StudentID, @WasPresent);
    END;
END;
GO
```

10.27 Procedura oznaczania obecności studenta (spMarkStudiesClassAttenda

Cel: Rejestruje obecność studenta na zajęciach studiów.

Parametry wejściowe:

- `@StudyClassID` – identyfikator zajęć,
- `@StudentID` – identyfikator studenta,
- `@WasPresent` – status obecności (1 – obecny, 0 – nieobecny).

Działanie: Procedura aktualizuje rekord w tabeli `StudiesClassAttendance`. Jeśli rekord nie istnieje, wstawia nowy.

```
CREATE OR ALTER PROCEDURE dbo.spMarkStudiesClassAttendance
  @StudyClassID INT,
  @StudentID    INT,
  @WasPresent   BIT
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE StudiesClassAttendance
    SET Attendance = @WasPresent
    WHERE StudyClassID = @StudyClassID
      AND StudentID = @StudentID;

    IF @@ROWCOUNT = 0
    BEGIN
        INSERT INTO StudiesClassAttendance (StudyClassID, StudentID, Attendance)
        VALUES (@StudyClassID, @StudentID, @WasPresent);
    END;
END;
GO
```

10.28 Podsumowanie zastosowania procedur w projekcie

Zastosowano zestaw procedur składowanych, które odpowiadają za kluczowe operacje na bazie danych. Główne funkcjonalności procedur obejmują:

- Dodawanie, aktualizację i usuwanie rekordów dotyczących kursów, webinarów, studentów, nauczycieli oraz pracowników.
- Zarządzanie rejestracją na kursy oraz automatyczne przypisywanie studentów do grup.
- Obsługę harmonogramów zajęć i modułów kursów z kontrolą konfliktów czasowych.
- Przetwarzanie danych dotyczących płatności, frekwencji, staży oraz zgód RODO.

Dzięki zastosowaniu procedur składowanych, logika biznesowa została scentralizowana po stronie bazy danych, co zapewnia:

- Wysoką spójność operacji na danych,
- Zwiększenie bezpieczeństwa i integralności danych,
- Ułatwienie modyfikacji oraz rozwoju systemu poprzez centralizację logiki operacyjnej.

10.29 Uprawnienia dla użytkowników

W ramach systemu przyznano uprawnienia do wykonywania funkcji skalarnych oraz odczytu wyników z funkcji tabelarycznych. Poniżej przedstawiono przykłady poleceń GRANT, które umożliwiają dostęp do odpowiednich obiektów bazy danych dla wybranych ról użytkowników.

Uprawnienia dla funkcji skalarnych (EXECUTE)

```
GRANT EXECUTE ON OBJECT::dbo.ufnGetOrderTotal
    TO Role_Student, Role_Admin, Role_Employee;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnGetStationaryModuleFreeSlots
    TO Role_Student, Role_Admin, Role_Employee;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnCountActiveCoursesInPeriod
    TO Role_Admin, Role_Employee;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnGetSubjectAverageGrade
    TO Role_Student, Role_Teacher, Role_Admin;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnGetFreeSeatsInBuilding
    TO Role_Admin, Role_Employee;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnConvertActivityPriceToEUR
    TO Role_Admin, Role_Employee;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnGetCourseTotalHours
    TO Role_Admin, Role_Employee, Role_Teacher;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnGetWebinarTotalHours
    TO Role_Admin, Role_Employee;
GO

GRANT EXECUTE ON OBJECT::dbo.ufnGetCourseTotalParticipants
    TO Role_Admin, Role_Employee, Role_Teacher;
GO
```

Uprawnienia dla funkcji tabelarycznych (SELECT)

```
GRANT SELECT ON OBJECT::dbo.ufnGetTeachersByLanguage
    TO Role_Admin, Role_Employee, Role_Student;
GO

GRANT SELECT ON OBJECT::dbo.ufnListActivitiesByLanguage
    TO Role_Student, Role_Admin, Role_Employee;
GO

GRANT SELECT ON OBJECT::dbo.ufnGetStudentSchedule
    TO Role_Student, Role_Teacher, Role_Admin;
GO

GRANT SELECT ON OBJECT::dbo.ufnGetStudentGrades
    TO Role_Student, Role_Teacher, Role_Admin;
GO

GRANT SELECT ON OBJECT::dbo.ufnGetCourseAttendanceList
    TO Role_Teacher, Role_Admin;
GO

GRANT SELECT ON OBJECT::dbo.ufnGetStudentAllAttendances
    TO Role_Student, Role_Teacher, Role_Admin;
GO
```

Dzięki tym uprawnieniom role użytkowników (takie jak Role_Student, Role_Teacher, Role_Admin oraz Role_Employee) mają zapewniony dostęp do funkcji, które są im niezbędne do realizacji operacji na danych. Uprawnienia do funkcji skalarnych przyznawane są poprzez polecenie `EXECUTE`, natomiast do funkcji tabelarycznych – poprzez polecenie `SELECT`.

11 Opis generowania danych przy pomocy skryptu w języku Python i biblioteki Faker

W celu zapewnienia wystarczającej ilości przykładowych rekordów w bazie danych, przygotowano dedykowany skrypt w języku Python, korzystający z biblioteki `Faker` oraz modułów `random` i `datetime`. Skrypt ten umożliwia **automatyczne wytworzenie oraz wyświetlenie** instrukcji `INSERT` dla wszystkich tabel w bazie.

11.1 Główne etapy działania skryptu

1. **Konfiguracja liczby rekordów** – Na początku pliku zdefiniowane są zmienne (np. `NUM_STUDENTS`, `NUM_ORDERS`, `NUM_COURSES`), pozwalające określić liczbę wierszy, jaka ma zostać wygenerowana w każdej tabeli. Dzięki temu możemy w łatwy sposób *skalować* ilość danych w bazie w zależności od potrzeb testowych.

2. **Wykorzystanie biblioteki `Faker`** – Skrypt używa biblioteki `Faker`, która umożliwia generowanie danych wyglądających realistycznie (m.in. imion, nazwisk, adresów e-mail, nazw firm, numerów telefonów). Dzięki temu testowana baza danych może bardziej przypominać system z danymi rzeczywistymi, co ułatwia wykrywanie ewentualnych problemów z wydajnością czy walidacją rekordów.

3. **Tworzenie poszczególnych zbiorów danych** – Skrypt generuje kolekcje słowników (np. `employees_data`, `students_data`) dla każdej tabeli, na podstawie ustalonych reguł:

   - *Klucze główne (PK)* – najczęściej rosnące identyfikatory `ID`.
   - *Klucze obce (FK)* – losowy wybór z już utworzonych zbiorów danych (np. `CityID` jest wybierany spośród istniejących miast).
   - *Losowe wartości* – np. daty zatrudnienia, stawki walut, ceny w `MONEY`, pola tekstowe z `Faker`.

4. **Formatowanie dla SQL** – Przy pomocy funkcji pomocniczych, takich jak:

   - `quote_str(s)` – zamienia pojedyncze apostrofy na podwójne, aby uniknąć błędów składni w instrukcjach `INSERT`,
   - `format_date(d)` – zamienia obiekt `date` na ciąg znaków `'YYYY-MM-DD'`,
   - `format_datetime(dt)` – formatuje daty i czasy do `'YYYY-MM-DD HH:MM:SS'`,
   - `format_money(val)` – wyświetla kwotę z dwoma miejscami po przecinku,

   skrypt generuje poprawne polecenia `INSERT INTO ... VALUES (...)` we właściwej kolejności (najpierw rekordy w tabelach niezależnych, następnie rekordy zawierające klucze obce).

5. **Wyjście skryptu** – Po zakończeniu pracy, skrypt drukuje wszystkie instrukcje `INSERT` na standardowe wyjście (`stdout`) w kolejności, która odpowiada zależnościom między tabelami. Można je przekierować do pliku (np. `> data_test_insert.sql`) i następnie uruchomić w środowisku bazy danych.

Powyższe podejście pozwala wypełnić bazę przykładowymi danymi bez konieczności ręcznego tworzenia dziesiątek instrukcji `INSERT`. Dzięki temu znacznie przyspiesza się proces rozwoju i testowania aplikacji, a także umożliwia weryfikację poprawności implementacji *relacji* i *zależności* (klucze obce, ograniczenia, itp.) w bazie danych.

12 Schemat bazy danych

W bazie danych wyróżniono **7 obszarów** tematycznych:

1. **Timetable** – zawiera m.in. *Schedule* (harmonogram zajęć) i *Buildings* (sale).
2. **University People** – tabele dotyczące nauczycieli (*Teachers*), tłumaczy (*Translators*), pracowników (*Employees*), typów stanowisk (*EmployeeTypes*) oraz języków (*Languages, TeacherLanguages, TranslatorsLanguages*).
3. **Studies** – opis kierunków (*Studies*), przedmiotów (*Subject*), ocen (*SubjectGrades*) oraz poszczególnych zajęć (*StudiesClass*) z rejestrem obecności (*StudiesClassAttendance*).
4. **Students** – dane osobowe studentów, w tym adresy i informacje o mieście/kraju (*Cities, Countries*), a także zgody RODO (*RODO_Table*).
5. **Courses** – obejmują główne tabele (*Courses, CourseModules*) z uczestnikami (*CourseParticipants*) i obecnościami (*CoursesAttendance*); *Activities* stanowi wspólną bazę aktywności (kursy, webinary, itp.).
6. **Webinars** – przechowują informacje o webinariach (*Webinars*) wraz z postępem studenta (*WebinarDetails*).
7. **Orders** – realizuje proces składania i przetwarzania zamówień (*Orders, OrderDetails, ShoppingCart, OrderPaymentStatus*), z uwzględnieniem *EuroExchangeRate* w przypadku wielowalutowości.

Na następnej stronie przedstawiono wizualny diagram z kluczami obcymi i nazwami tabel. Poszczególne grupy (np. Timetable, Courses, Webinars) oznaczono kolorami, aby ułatwić identyfikację relacji *jeden-do-wielu* oraz kluczy głównych i obcych.

Schedule

ScheduleID	int	PK
ClassID	int	FK
CourseModuleID	int	N FK
StudiesSubjectID	int	N FK
DayOfWeek	varchar(10)	
StartTime	time	
EndTime	time	
TeacherID	int	FK
TranslatorID	int	N FK

Buildings

ClassID	int	PK
BuildingName	varchar(30)	
RoomNumber	varchar(30)	

Timetable

Translators

TranslatorID	int	PK
FirstName	varchar(30)	
LastName	varchar(30)	
HireDate	date	N
Phone	varchar(15)	N
Email	varchar(60)	

Teachers

TeacherID	int	PK
FirstName	varchar(30)	
LastName	varchar(30)	
HireDate	date	N
Phone	varchar(15)	N
Email	varchar(60)	

Employees

EmployeeID	int	PK
FirstName	varchar(30)	
LastName	varchar(30)	
HireDate	date	N
EmployeeTypeID	int	FK
Phone	varchar(15)	N
Email	varchar(60)	

TranslatorsLanguages

TranslatorID	int	PK FK
LanguageID	int	PK FK

TeacherLanguages

TeacherID	int	PK FK
LanguageID	int	PK FK

EmployeeTypes

EmployeeTypeID	int	PK
EmployeeTypeName	varchar(30)	

Languages

LanguageID	int	PK
LanguageName	varchar(40)	

University People

CourseParticipant

CourseID	int	PK FK
StudentID	int	PK FK

CoursesAttendance

ModuleID	int	PK FK
StudentID	int	PK FK
Attendance	bit	

StationaryModule

StationaryModuleID	int	PK FK
ClassID	int	FK
Limit	int	

Courses

CourseID	int	PK
ActivityID	int	FK
CourseName	varchar(50)	
CourseDescription	text	N
CoursePrice	money	
CourseCoordinatorI	int	FK

OnlineSyncModule

OnlineSyncMo	int	PK FK
Link	varchar(60)	

OnlineAsyncModule

OnlineAsyncM	int	PK FK
Video	varchar(60)	

CourseModules

ModuleID	int	PK
CourseID	int	FK
ModuleName	varchar(50)	
Date	datetime	
DurationTime	time(0)	
TeacherID	int	FK
TranslatorID	int	N FK
LanguageID	int	FK

Courses

Students

StudentID	int	PK
FirstName	varchar(30)	
LastName	varchar(30)	
Address	varchar(30)	
CityID	int	FK
PostalCode	varchar(10)	
Phone	varchar(15)	N
Email	varchar(60)	

Cities

CityID	int	PK
CityName	varchar(40)	
CountryID	int	FK

Countries

CountryID	int	PK
CountryName	varchar(40)	

RODO_Table

StudentID	int	PK FK
Date	date	
Withdraw	bit	

Students

SubjectGrades

SubjectID	int	PK FK
StudentID	int	PK FK
SubjectGrade	int	

StudiesClassAttendance

StudyClassID	int	PK FK
StudentID	int	PK FK
Attendance	bit	

Subject

SubjectID	int	PK
StudiesID	int	FK
CoordinatorID	int	FK
SubjectName	varchar(50)	
SubjectDescription	text	N

StudiesClass

StudyClassID	int	PK
SubjectID	int	FK
ActivityID	int	FK
TeacherID	int	FK
ClassName	varchar(50)	
ClassPrice	money	
Date	datetime	
DurationTime	time(0)	N
LanguageID	int	N FK
TranslatorID	int	N FK
LimitClass	int	

Studies

StudiesID	int	PK
ActivityID	int	FK
StudiesName	varchar(50)	
StudiesDescription	text	N
StudiesEntryFeePrice	money	
Syllabus	text	
StudiesEmployee	int	FK
Limit	int	

StationaryClass

StationaryClassID	int	PK FK
ClassID	int	FK
Limit	int	

OnlineSyncClass

OnlineSyncClassID	int	PK FK
Link	varchar(50)	

Internship

InternshipID	int	PK
StudiesID	int	FK
StartDate	datetime	

OnlineAsyncClass

OnlineAsyncClas	int	PK FK
Video	varchar(50)	

InternshipAttendance

InternshipID	int	PK FK
StudentID	int	PK FK
Attendance	bit	

Studies

Webinars

WebinarID	int	PK
ActivityID	int	FK
TeacherID	int	FK
WebinarName	varchar(50)	
WebinarPrice	money	
VideoLink	varchar(50)	
WebinarDate	datetime	
DurationTime	time(0)	
WebinarDescripti	text	
LanguageID	int	FK

WebinarDetails

StudentID	int	PK FK
WebinarID	int	PK FK
Complete	bit	
AvailableD	date	

Webinars

Orders

OrderID	int	PK
StudentID	int	FK
OrderDate	datetime	FK
PaymentURL	int	FK
EmployeeHandlin	int	FK

OrderPaymentStatus

PaymentURL	int	PK
OrderPaymentStat	varchar(20)	
PaidDate	datetime	N

Activities

ActivityID	int	PK
Price	money	
Title	varchar(50)	
Active	bit	

OrderDetails

OrderID	int	PK FK
ActivityID	int	PK FK

EuroExchangeRate

Date	datetime	PK
Rate	decimal(10,2)	

ShoppingCart

StudentID	int	PK FK
ActivityID	int	PK FK

Orders

13 Podsumowanie

Przedstawiona baza danych została zaprojektowana w celu kompleksowej obsługi procesów edukacyjnych. Główne cechy struktury bazy to:

- **Modułowa organizacja danych:** Zbiór tabel został podzielony na tematyczne obszary, m.in. zarządzanie kursami (tabele `Courses`, `CourseModules`, `CourseParticipants`), obsługa studenckich danych osobowych oraz zamówień (`Students`, `Orders`, `ShoppingCart`), jak również zarządzanie personelem dydaktycznym (`Teachers`, `Translators`, `Employees`).
- **Integralność i spójność danych:** Rozbudowana struktura kluczy głównych i obcych pozwala na utrzymanie spójności pomiędzy tabelami, co jest dodatkowo wspierane przez mechanizmy takie jak triggery, procedury przechowywane oraz funkcje. Dzięki temu operacje w bazie zachowują wysoki poziom integralności, eliminując nieprawidłowe lub sprzeczne wpisy.
- **Elastyczność i rozszerzalność:** System ról (np. `Role_Admin`, `Role_Teacher`, `Role_Student`, `Role_Translator`, `Role_Employee`) umożliwia precyzyjne zarządzanie uprawnieniami, co pozwala na łatwą adaptację bazy do zmieniających się potrzeb użytkowników. Modułowe podejście umożliwia też dalszą rozbudowę bazy o nowe tabele czy mechanizmy operacyjne, jeśli zajdzie taka potrzeba.
- **Wsparcie dla wieloaspektowych procesów edukacyjnych:** Baza uwzględnia różne formy prowadzenia zajęć – od kursów i webinarów po zajęcia stacjonarne i studia – dzięki czemu umożliwia kompleksowe zarządzanie informacjami dotyczącymi terminów, obecności uczestników, stawek, a także relacji między zajęciami a osobami odpowiedzialnymi za ich realizację.

Podział Pracy podczas Projektu

- **Maciej Kmąk** (T = 40)
  - Redakcja dokumentacji (T = 5)
  - Implementacja tabel bazy danych (T = 10)
  - Generowanie danych do bazy (T = 5)
  - Drobne końcowe poprawki implementacyjne (T = 5)
  - Role i przykładowi użytkownicy (T = 5)
  - Projekt i schemat bazy danych (Współudział) (T = 10)
- **Jakub Stachecki** (T = 30)
  - Projekt i schemat bazy danych (Współudział) (T = 10)
  - Triggery (T = 10)
  - Widoki (T = 10)
- **Kacper Wdowiak** (T = 30)
  - Projekt i schemat bazy danych (Współudział) (T = 10)
  - Procedury (T = 10)
  - Funkcje (T = 10)