



AGH

**AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE**

Podstawy Baz Danych

Projekt 2024/25

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

Spis treści

1	Wprowadzenie	5
2	Role i uprawnienia	6
2.1	Możliwe role użytkowników	6
2.2	Uprawnienia przypisane do ról	6
2.2.1	Role_Admin (Administrator)	7
2.2.2	Role_Teacher (Nauczyciel)	7
2.2.3	Role_Student (Student)	7
2.2.4	Role_Translator (Tłumacz)	8
2.2.5	Role_Employee (Pracownik administracyjny)	8
2.3	Podsumowanie zakresu ról	8
3	Struktura wszystkich tabel	9
3.1	Activities	9
3.2	Buildings	9
3.3	Cities	9
3.4	Countries	10
3.5	CourseModules	10
3.6	CourseParticipants	10
3.7	Courses	11
3.8	CoursesAttendance	11
3.9	EmployeeTypes	11
3.10	Employees	11
3.11	EuroExchangeRate	12
3.12	Internship	12
3.13	InternshipAttendance	12
3.14	Languages	12
3.15	OnlineAsyncClass	12
3.16	OnlineAsyncModule	13
3.17	OnlineSyncClass	13
3.18	OnlineSyncModule	13
3.19	OrderDetails	13
3.20	OrderPaymentStatus	14
3.21	Orders	14
3.22	RODO_Table	14
3.23	Schedule	14
3.24	ShoppingCart	15
3.25	StationaryClass	15
3.26	StationaryModule	15
3.27	Students	15
3.28	Studies	16
3.29	StudiesClass	17
3.30	StudiesClassAttendance	17
3.31	Subject	17
3.32	SubjectGrades	18
3.33	TeacherLanguages	18
3.34	Teachers	18
3.35	Translators	18

3.36	TranslatorsLanguages	18
3.37	WebinarDetails	19
3.38	Webinars	19
4	Relacje (klucze obce)	20
5	Warunki Integralnościowe	23
5.1	Klucze główne i obce	23
5.2	Wartości NULL i NOT NULL – logika warunków w tabelach	23
5.3	Triggery wspomagające integralność danych	26
5.4	Procedury i funkcje wspomagające integralność	26
5.5	Inicjalizacja tabel i zależności	26
5.6	Podsumowanie	26
6	Triggery	27
6.1	TR_OrderPaymentStatus_AfterInsert	27
6.2	TR_OrderPaymentStatus_AfterUpdate_PaymentSuccess	28
6.3	TR_Courses_AfterDelete	29
6.4	TR_Webinars_AfterInsert_UniqueTeacherWebinar	30
6.5	TR_Teachers_InsteadOfDelete_BlockIfActive	31
6.6	TR_Students_AfterInsert_AddCity	32
7	Widoki (views)	33
7.1	Lista widoków	33
7.2	Widok v_StudentCourses	33
7.3	Widok v_CourseModulesDetailed	34
7.4	Widok v_OrdersFull	34
7.5	Widok v_ScheduleDetailed	35
7.6	Widok v_StudentGrades	36
7.7	Uprawnienia do poszczególnych widoków	36
7.8	Podsumowanie	37
8	Indeksy w bazie danych	38
8.1	Indeksy dla studentów	38
8.2	Indeksy dla kursów	38
8.3	Indeksy dla zamówień	38
8.4	Indeksy dla nauczycieli i tłumaczy	38
8.5	Indeksy dla webinarów i harmonogramu	39
8.6	Podsumowanie	39
9	Funkcje (functions)	40
9.1	Obliczanie całkowitej kwoty zamówienia	40
9.2	Sprawdzanie dostępności miejsca w grupie kursowej	41
9.3	Zliczanie aktywnych kursów w danym okresie	42
9.4	Pobieranie średniej ocen z przedmiotu	43
9.5	Obliczanie wolnych miejsc w budynku	44
9.6	Konwersja walut w cenach aktywności (EUR/PLN)	45
9.7	Pobieranie listy nauczycieli dla danego języka	46
9.8	Obliczanie liczby zajęć w kursie	46
9.9	Wyświetlanie listy aktywności dostępnych dla danego języka	47
9.10	Obliczanie sumarycznego czasu trwania webinaru	48

9.11	Obliczanie liczby uczestników w kursie	48
9.12	Harmonogram zajęć dla studenta	49
9.13	Generowanie raportu ocen dla danego studenta	50
9.14	Generowanie listy obecności dla danego kursu	50
9.15	Generowanie listy obecności dla danego studenta	51
10	Procedury (stored procedures)	52
10.1	Dodawanie nowego kursu (spAddCourse)	52
10.2	Usuwanie kursu (spRemoveCourse)	53
10.3	Rejestracja studenta na kurs (spRegisterStudentInCourse)	54
10.4	Wypisanie studenta z kursu (spUnregisterStudentFromCourse)	55
10.5	Aktualizacja ceny aktywności (spUpdateActivityPrice)	56
10.6	Zarządzanie harmonogramem nauczyciela (spAddTeacherSchedule)	57
10.7	Wyszukiwanie dostępnych aktywności (spFindAvailableActivities)	59
10.8	Rejestracja nowego nauczyciela (spAddTeacher)	61
10.9	Rejestracja nowego tłumacza (spAddTranslator)	62
10.10	Tworzenie nowego webinaru (spAddWebinar)	63
10.11	Usuwanie webinaru (spRemoveWebinar)	65
10.12	Dodawanie przedmiotu do planu studiów (spAddSubjectToStudies)	66
10.13	Automatyczne rozdzielanie studentów do grup (spAutoAssignStudentsToGroups)	67
10.14	Aktualizacja danych nauczyciela (spUpdateTeacherData)	68
10.15	Aktualizacja danych studenta (spUpdateStudentData)	69
10.16	Aktualizacja danych osobowych (RODO) (spUpdateRODO)	70
10.17	Dodanie stażu (Internship) (spAddInternship)	71
10.18	Dodanie studenta do bazy (spAddStudent)	72
10.19	Usunięcie studenta z bazy (spRemoveStudent)	73
10.20	Dodanie pracownika (Employee) do bazy (spAddEmployee)	74
10.21	Usunięcie pracownika z bazy (spRemoveEmployee)	75
10.22	Tworzenie modułu kursu (spAddCourseModule)	76
10.23	Pobieranie kursu euro (spAddEuroRate)	78
10.24	Lista „dłużników” (spGetDebtors)	79
10.25	Raport frekwencji (spGenerateCourseAttendanceReport)	80
10.26	Procedura oznaczania obecności studenta (spMarkCourseModuleAttendance)	81
10.27	Procedura oznaczania obecności studenta (spMarkStudiesClassAttendance)	82
10.28	Podsumowanie zastosowania procedur w projekcie	82
10.29	Uprawnienia dla użytkowników	83
11	Opis generowania danych przy pomocy skryptu w języku Python i biblioteki Faker	85
11.1	Główne etapy działania skryptu	85
12	Schemat bazy danych	86
13	Podsumowanie	88

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 administracyjny,
- **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)

```
1 GRANT SELECT, INSERT, UPDATE, DELETE, ALTER, REFERENCES
2 ON SCHEMA::dbo
3 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)

```
1 GRANT SELECT, UPDATE ON dbo.CoursesAttendance TO Role_Teacher;
2 GRANT SELECT, UPDATE ON dbo.StudiesClassAttendance TO Role_Teacher;
3 GRANT SELECT ON dbo.Students TO Role_Teacher;
4 GRANT SELECT ON dbo.Courses TO Role_Teacher;
5 GRANT SELECT ON dbo.Studies TO Role_Teacher;
6 GRANT SELECT ON dbo.Translators TO Role_Teacher;
7 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)

```
1 GRANT SELECT ON dbo.Activities TO Role_Student;
2 GRANT SELECT ON dbo.Courses TO Role_Student;
3 GRANT SELECT ON dbo.Webinars TO Role_Student;
4 GRANT SELECT ON dbo.Studies TO Role_Student;
5 GRANT SELECT, INSERT, DELETE ON dbo.ShoppingCart TO Role_Student;
6 GRANT SELECT, INSERT ON dbo.Orders TO Role_Student;
7 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)

```
1 GRANT SELECT ON dbo.Webinars TO Role_Translator;  
2 GRANT SELECT ON dbo.CourseModules TO Role_Translator;  
3 GRANT SELECT ON dbo.Translators TO Role_Translator;  
4 GRANT SELECT ON dbo.TranslatorsLanguages TO Role_Translator;  
5 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)

```
1 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Orders TO Role_Employee;  
2 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.OrderDetails TO Role_Employee;  
3 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Studies TO Role_Employee;  
4 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Students TO Role_Employee;  
5 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Courses TO Role_Employee;  
6 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Teachers TO Role_Employee;  
7 GRANT SELECT, INSERT, UPDATE, DELETE ON dbo.Translators TO Role_Employee;  
8 GRANT SELECT, UPDATE ON dbo.OrderPaymentStatus TO Role_Employee;  
9 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:

```
1 CREATE TABLE Activities (  
2     ActivityID      INT          NOT NULL ,  
3     Price           MONEY       NOT NULL ,  
4     Title           VARCHAR(50) NOT NULL ,  
5     Active          BIT         NOT NULL ,  
6     CONSTRAINT PK_Activities PRIMARY KEY (ActivityID)  
7 );
```

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:

```
1 CREATE TABLE Courses (  
2     CourseID          INT          NOT NULL,  
3     ActivityID        INT          NOT NULL,  -- FK do Activities  
4     CourseName        VARCHAR(50)  NOT NULL,  
5     CourseDescription TEXT          NULL,  
6     CoursePrice       MONEY        NOT NULL,  
7     CourseCoordinatorID INT        NOT NULL,  -- FK do Teachers  
8     CONSTRAINT PK_Courses PRIMARY KEY (CourseID)  
9 );
```

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 po-prawnie 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:

```
1 CREATE TABLE CourseModules (  
2     ModuleID          INT          NOT NULL,  
3     CourseID          INT          NOT NULL,  -- FK do Courses  
4     ModuleName        VARCHAR(50)  NOT NULL,  
5     Date              DATETIME     NOT NULL,  
6     DurationTime      TIME(0)      NOT NULL,  
7     TeacherID         INT          NOT NULL,  -- FK do Teachers  
8     TranslatorID      INT          NULL,      -- FK do Translators  
9     LanguageID        INT          NOT NULL,  -- FK do Languages  
10    CONSTRAINT PK_CourseModules PRIMARY KEY (ModuleID)  
11 );
```

Opis: - Wszystkie kolumny, które definiują podstawowe informacje o module (np. identyfikator mo-dułu, 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:

```
1 CREATE TABLE Employees (  
2     EmployeeID        INT          NOT NULL,  
3     FirstName         VARCHAR(30)  NOT NULL,  
4     LastName          VARCHAR(30)  NOT NULL,  
5     HireDate          DATE          NULL,  
6     EmployeeTypeID    INT          NOT NULL,  -- FK do EmployeeTypes  
7     Phone             VARCHAR(15)  NULL,  
8     Email             VARCHAR(60)  NOT NULL,  
9     CONSTRAINT PK_Employees PRIMARY KEY (EmployeeID)  
10 );
```

Opis: - Pola EmployeeID, FirstName, LastName, EmployeeTypeID oraz Email są krytyczne dla iden-tyfikacji 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:

```

1 CREATE TABLE OrderPaymentStatus (
2     PaymentURL          INT          NOT NULL,
3     OrderPaymentStatus  VARCHAR(20)  NOT NULL,
4     PaidDate            DATETIME     NULL,
5     CONSTRAINT PK_OrderPaymentStatus PRIMARY KEY (PaymentURL)
6 );

```

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:

```

1 CREATE TABLE Schedule (
2     ScheduleID          INT          NOT NULL,
3     ClassID             INT          NOT NULL, -- FK do Buildings
4     CourseModuleID      INT          NULL,   -- FK do CourseModules
5     StudiesSubjectID    INT          NULL,   -- FK do Subject
6     DayOfWeek           VARCHAR(10) NOT NULL,
7     StartTime           TIME         NOT NULL,
8     EndTime             TIME         NOT NULL,
9     TeacherID           INT          NOT NULL, -- FK do Teachers
10    TranslatorID         INT          NULL,   -- FK do Translators
11    CONSTRAINT PK_Schedule PRIMARY KEY (ScheduleID)
12 );

```

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

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

```
1 CREATE OR ALTER TRIGGER TR_OrderPaymentStatus_AfterInsert
2 ON dbo.OrderPaymentStatus
3 AFTER INSERT
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7
8     UPDATE ops
9     SET
10         OrderPaymentStatus =
11             CASE
12                 WHEN i.OrderPaymentStatus IS NULL THEN 'Pending'
13                 ELSE i.OrderPaymentStatus
14             END,
15         PaidDate =
16             CASE
17                 WHEN i.OrderPaymentStatus = 'Paid' THEN GETDATE()
18                 ELSE ops.PaidDate
19             END
20 FROM dbo.OrderPaymentStatus ops
21 JOIN inserted i ON ops.PaymentURL = i.PaymentURL;
22 END;
23 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

```
1 CREATE OR ALTER TRIGGER TR_OrderPaymentStatus_AfterUpdate_PaymentSuccess
2 ON dbo.OrderPaymentStatus
3 AFTER UPDATE
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7     UPDATE ops
8     SET PaidDate = GETDATE()
9     FROM dbo.OrderPaymentStatus ops
10    JOIN inserted i ON ops.PaymentURL = i.PaymentURL
11    WHERE i.OrderPaymentStatus = 'Paid';
12
13    ;WITH Changed AS
14    (
15        SELECT i.PaymentURL
16        FROM inserted i
17        JOIN deleted d ON i.PaymentURL = d.PaymentURL
18        WHERE i.OrderPaymentStatus = 'Paid'
19              AND d.OrderPaymentStatus <> 'Paid'
20    )
21    SELECT PaymentURL
22    INTO #Changed
23    FROM Changed;
24
25    INSERT INTO WebinarDetails (StudentID, WebinarID, Complete, AvailableDue)
26    SELECT
27        o.StudentID,
28        wb.WebinarID,
29        0 AS Complete,
30        DATEADD(DAY, 30, GETDATE()) AS AvailableDue
31    FROM #Changed ch
32    JOIN dbo.Orders o ON o.PaymentURL = ch.PaymentURL
33    JOIN dbo.OrderDetails od ON od.OrderID = o.OrderID
34    JOIN dbo.Webinars wb ON wb.ActivityID = od.ActivityID;
35
36    INSERT INTO CourseParticipants (CourseID, StudentID)
37    SELECT
38        c.CourseID,
39        o.StudentID
40    FROM #Changed ch
41    JOIN dbo.Orders o ON o.PaymentURL = ch.PaymentURL
42    JOIN dbo.OrderDetails od ON od.OrderID = o.OrderID
43    JOIN dbo.Courses c ON c.ActivityID = od.ActivityID;
44
45    INSERT INTO StudiesClassAttendance (StudyClassID, StudentID, Attendance)
46    SELECT
47        sc.StudyClassID,
48        o.StudentID,
49        0
50    FROM #Changed ch
51    JOIN dbo.Orders o ON o.PaymentURL = ch.PaymentURL
52    JOIN dbo.OrderDetails od ON od.OrderID = o.OrderID
53    JOIN dbo.Studies st ON st.ActivityID = od.ActivityID
54    JOIN dbo.StudiesClass sc ON sc.ActivityID = st.ActivityID;
55
56    DROP TABLE #Changed;
57 END;
58 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

```
1 CREATE OR ALTER TRIGGER TR_Courses_AfterDelete
2 ON dbo.Courses
3 AFTER DELETE
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7
8     DELETE cp
9     FROM dbo.CourseParticipants cp
10    JOIN deleted d ON cp.CourseID = d.CourseID;
11
12    DELETE ca
13    FROM dbo.CoursesAttendance ca
14    JOIN dbo.CourseModules cm ON ca.ModuleID = cm.ModuleID
15    JOIN deleted d ON cm.CourseID = d.CourseID;
16
17    DELETE cm
18    FROM dbo.CourseModules cm
19    JOIN deleted d ON cm.CourseID = d.CourseID;
20
21    PRINT 'All related participants, attendance records, and modules removed.';
22 END;
23 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

```
1 CREATE OR ALTER TRIGGER TR_Webinars_AfterInsert_UniqueTeacherWebinar
2 ON dbo.Webinars
3 AFTER INSERT
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7
8     IF EXISTS (
9         SELECT 1
10        FROM dbo.Webinars w
11        JOIN inserted i ON
12            w.TeacherID = i.TeacherID
13            AND w.WebinarName = i.WebinarName
14            AND w.WebinarID <> i.WebinarID
15    )
16    BEGIN
17        RAISERROR('Cannot add duplicate webinar for the same teacher.', 16, 1);
18        ROLLBACK TRANSACTION;
19        RETURN;
20    END;
21 END;
22 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

```
1 CREATE OR ALTER TRIGGER TR_Teachers_InsteadOfDelete_BlockIfActive
2 ON dbo.Teachers
3 INSTEAD OF DELETE
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7     IF EXISTS (
8         SELECT 1
9         FROM deleted d
10        JOIN CourseModules cm ON cm.TeacherID = d.TeacherID
11        JOIN Courses c ON c.CourseID = cm.CourseID
12        JOIN Activities a ON a.ActivityID = c.ActivityID
13        WHERE a.Active = 1
14    )
15    OR EXISTS (
16        SELECT 1
17        FROM deleted d
18        JOIN Webinars w ON w.TeacherID = d.TeacherID
19        JOIN Activities a ON a.ActivityID = w.ActivityID
20        WHERE a.Active = 1
21    )
22    BEGIN
23        RAISERROR('Cannot delete teacher: assigned to active classes.', 16, 1);
24        ROLLBACK TRANSACTION;
25        RETURN;
26    END;
27
28    DELETE t
29    FROM dbo.Teachers t
30    JOIN deleted d ON t.TeacherID = d.TeacherID;
31 END;
32 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

```
1 CREATE OR ALTER TRIGGER TR_Students_AfterInsert_AddCity
2 ON dbo.Students
3 AFTER INSERT
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7
8     INSERT INTO dbo.Cities (CityID, CityName, CountryID)
9     SELECT DISTINCT
10         i.CityID,
11         'Unknown',
12         1
13 FROM inserted i
14 LEFT JOIN dbo.Cities c ON c.CityID = i.CityID
15 WHERE c.CityID IS NULL;
16 END;
17 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 modulem 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

```
1 CREATE OR ALTER VIEW dbo.v_StudentCourses AS
2 SELECT
3     s.StudentID,
4     s.FirstName AS StudentFirstName,
5     s.LastName  AS StudentLastName,
6     s.Email     AS StudentEmail,
7     c.CourseID,
8     c.CourseName,
9     c.CoursePrice
10 FROM dbo.Students s
11 JOIN dbo.CourseParticipants cp ON s.StudentID = cp.StudentID
12 JOIN dbo.Courses c ON cp.CourseID = c.CourseID;
13 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

```

1 CREATE OR ALTER VIEW dbo.v_CourseModulesDetailed AS
2 SELECT
3     cm.ModuleID,
4     cm.ModuleName,
5     cm.[Date] AS ModuleDate,
6     cm.DurationTime,
7     t.TeacherID,
8     t.FirstName AS TeacherFirstName,
9     t.LastName AS TeacherLastName,
10    tr.TranslatorID,
11    tr.FirstName AS TranslatorFirstName,
12    tr.LastName AS TranslatorLastName,
13    l.LanguageName AS ModuleLanguage,
14    c.CourseID,
15    c.CourseName
16 FROM dbo.CourseModules cm
17 JOIN dbo.Teachers t ON cm.TeacherID = t.TeacherID
18 LEFT JOIN dbo.Translators tr ON cm.TranslatorID = tr.TranslatorID
19 JOIN dbo.Languages l ON cm.LanguageID = l.LanguageID
20 JOIN dbo.Courses c ON cm.CourseID = c.CourseID;
21 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

```

1 CREATE OR ALTER VIEW dbo.v_OrdersFull AS
2 SELECT
3     o.OrderID,
4     o.OrderDate,
5     ops.OrderPaymentStatus,
6     ops.PaidDate,
7     s.StudentID,
8     s.FirstName AS StudentFirstName,
9     s.LastName AS StudentLastName,
10    e.EmployeeID,
11    e.FirstName AS EmployeeFirstName,
12    e.LastName AS EmployeeLastName,
13    SUM(a.Price) AS TotalOrderPrice
14 FROM dbo.Orders o
15 JOIN dbo.OrderPaymentStatus ops ON o.PaymentURL = ops.PaymentURL
16 JOIN dbo.Students s ON o.StudentID = s.StudentID
17 JOIN dbo.Employees e ON o.EmployeeHandling = e.EmployeeID
18 JOIN dbo.OrderDetails od ON o.OrderID = od.OrderID
19 JOIN dbo.Activities a ON od.ActivityID = a.ActivityID
20 GROUP BY
21     o.OrderID, o.OrderDate, ops.OrderPaymentStatus, ops.PaidDate,
22     s.StudentID, s.FirstName, s.LastName,
23     e.EmployeeID, e.FirstName, e.LastName;
24 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`

```
1 CREATE OR ALTER VIEW dbo.v_ScheduleDetailed AS
2 SELECT
3     sch.ScheduleID,
4     sch.DayOfWeek,
5     sch.StartTime,
6     sch.EndTime,
7     t.TeacherID,
8     t.FirstName AS TeacherFirstName,
9     t.LastName AS TeacherLastName,
10    tr.TranslatorID,
11    tr.FirstName AS TranslatorFirstName,
12    tr.LastName AS TranslatorLastName,
13    b.BuildingName,
14    b.RoomNumber,
15    cm.ModuleID,
16    cm.ModuleName,
17    sb.SubjectID,
18    sb.SubjectName
19 FROM dbo.Schedule sch
20 JOIN dbo.Buildings b ON sch.ClassID = b.ClassID
21 JOIN dbo.Teachers t ON sch.TeacherID = t.TeacherID
22 LEFT JOIN dbo.Translators tr ON sch.TranslatorID = tr.TranslatorID
23 LEFT JOIN dbo.CourseModules cm ON sch.CourseModuleID = cm.ModuleID
24 LEFT JOIN dbo.Subject sb ON sch.StudiesSubjectID = sb.SubjectID;
25 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

```

1 CREATE OR ALTER VIEW dbo.v_StudentGrades AS
2 SELECT
3     sg.StudentID,
4     st.FirstName AS StudentFirstName,
5     st.LastName AS StudentLastName,
6     sb.SubjectID,
7     sb.SubjectName,
8     sb.CoordinatorID,
9     tch.FirstName AS CoordinatorFirstName,
10    tch.LastName AS CoordinatorLastName,
11    s.StudiesID,
12    s.StudiesName,
13    sg.SubjectGrade
14 FROM dbo.SubjectGrades sg
15 JOIN dbo.Students st ON sg.StudentID = st.StudentID
16 JOIN dbo.Subject sb ON sg.SubjectID = sb.SubjectID
17 JOIN dbo.Teachers tch ON sb.CoordinatorID = tch.TeacherID
18 JOIN dbo.Studies s ON sb.StudiesID = s.StudiesID;
19 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.

```

1 GRANT SELECT ON OBJECT::dbo.v_StudentCourses
2 TO Role_Admin, Role_Employee;
3 GO
4
5 GRANT SELECT ON OBJECT::dbo.v_CourseModulesDetailed
6 TO Role_Admin, Role_Employee, Role_Student, Role_Teacher, Role_Translator;
7 GO
8
9 GRANT SELECT ON OBJECT::dbo.v_OrdersFull
10 TO Role_Admin, Role_Employee, Role_Student;
11 GO
12
13 GRANT SELECT ON OBJECT::dbo.v_ScheduleDetailed
14 TO Role_Admin, Role_Employee, Role_Student, Role_Teacher, Role_Translator;
15 GO
16
17 GRANT SELECT ON OBJECT::dbo.v_StudentGrades
18 TO Role_Admin, Role_Employee, Role_Teacher;
19 GO

```

Upewnienia 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.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetOrderTotal
2 (
3     @OrderID INT
4 )
5 RETURNS MONEY
6 AS
7 BEGIN
8     DECLARE @Total MONEY;
9
10    SELECT @Total = SUM(A.Price)
11    FROM OrderDetails OD
12    JOIN Activities A ON A.ActivityID = OD.ActivityID
13    WHERE OD.OrderID = @OrderID;
14
15    IF @Total IS NULL
16        SET @Total = 0;
17
18    RETURN @Total;
19 END;
20 GO
```

Przykładowe wywołanie:

```
1 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`.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetStationaryModuleFreeSlots
2 (
3     @ModuleID INT
4 )
5 RETURNS INT
6 AS
7 BEGIN
8     DECLARE @Limit INT, @Count INT, @FreeSlots INT;
9
10    SELECT @Limit = SM.[Limit]
11    FROM StationaryModule SM
12    WHERE SM.StationaryModuleID = @ModuleID;
13
14    IF @Limit IS NULL
15    BEGIN
16        RETURN -1;
17    END;
18
19    SELECT @Count = COUNT(*)
20    FROM CoursesAttendance CA
21    WHERE CA.ModuleID = @ModuleID
22        AND CA.Attendance = 1;
23
24    SET @FreeSlots = @Limit - ISNULL(@Count,0);
25
26    RETURN @FreeSlots;
27 END;
28 GO
```

Przykładowe wywołanie:

```
1 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 mieszczące się w przedziale czasowym.

```
1 CREATE OR ALTER FUNCTION dbo.ufnCountActiveCoursesInPeriod
2 (
3     @StartDate DATETIME,
4     @EndDate   DATETIME
5 )
6 RETURNS INT
7 AS
8 BEGIN
9     DECLARE @Count INT;
10
11     WITH ActiveCourses AS
12     (
13         SELECT DISTINCT c.CourseID
14         FROM Courses c
15         JOIN Activities a ON a.ActivityID = c.ActivityID
16         JOIN CourseModules cm ON cm.CourseID = c.CourseID
17         WHERE a.Active = 1
18             AND cm.Date >= @StartDate
19             AND cm.Date < @EndDate
20     )
21     SELECT @Count = COUNT(*)
22     FROM ActiveCourses;
23
24     RETURN @Count;
25 END;
26 GO
```

Przykładowe wywołanie:

```
1 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`. Ocenę są rzutowane na typ `DECIMAL(5,2)` i uśredniane. W przypadku braku ocen wynik jest ustawiany na 0.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetSubjectAverageGrade
2 (
3     @SubjectID INT
4 )
5 RETURNS DECIMAL(5,2)
6 AS
7 BEGIN
8     DECLARE @Average DECIMAL(5,2);
9
10    SELECT @Average = AVG(CAST(SubjectGrade AS DECIMAL(5,2)))
11    FROM SubjectGrades
12    WHERE SubjectID = @SubjectID;
13
14    IF @Average IS NULL
15        SET @Average = 0;
16
17    RETURN @Average;
18 END;
19 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.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetFreeSeatsInBuilding
2 (
3     @ClassID INT
4 )
5 RETURNS INT
6 AS
7 BEGIN
8     DECLARE @FreeSeats INT;
9
10    ;WITH BuildingRooms AS
11    (
12        SELECT SC.StationaryClassID, SC.[Limit]
13        FROM StationaryClass SC
14        WHERE SC.ClassID = @ClassID
15    ),
16    Occupancy AS
17    (
18        SELECT
19            br.StationaryClassID,
20            COUNT(*) AS Occupied
21        FROM BuildingRooms br
22        JOIN StudiesClass sc ON sc.StudyClassID = br.StationaryClassID
23        JOIN StudiesClassAttendance sca ON sca.StudyClassID = sc.StudyClassID
24        GROUP BY br.StationaryClassID
25    )
26    SELECT @FreeSeats = SUM(br.[Limit] - ISNULL(o.Occupied, 0))
27    FROM BuildingRooms br
28    LEFT JOIN Occupancy o ON o.StationaryClassID = br.StationaryClassID;
29
30    RETURN ISNULL(@FreeSeats, 0);
31 END;
32 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.

```
1 CREATE OR ALTER FUNCTION dbo.ufnConvertActivityPriceToEUR
2 (
3     @ActivityID INT,
4     @RateDate   DATETIME
5 )
6 RETURNS DECIMAL(10,2)
7 AS
8 BEGIN
9     DECLARE @PLN MONEY, @Rate DECIMAL(10,2), @EUR DECIMAL(10,2);
10
11     SELECT @PLN = Price
12     FROM Activities
13     WHERE ActivityID = @ActivityID;
14
15     IF @PLN IS NULL
16         RETURN 0;
17
18     SELECT TOP(1) @Rate = Rate
19     FROM EuroExchangeRate
20     WHERE [Date] <= @RateDate
21     ORDER BY [Date] DESC;
22
23     IF @Rate IS NULL
24         RETURN 0;
25
26     SET @EUR = CAST(@PLN AS DECIMAL(10,2)) / @Rate;
27
28     RETURN @EUR;
29 END;
30 GO
```

Przykładowe wywołanie:

```
1 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`.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetTeachersByLanguage
2 (
3     @LanguageID INT
4 )
5 RETURNS TABLE
6 AS
7 RETURN
8 (
9     SELECT t.TeacherID,
10            t.FirstName,
11            t.LastName,
12            t.Email
13     FROM Teachers t
14     JOIN TeacherLanguages tl ON tl.TeacherID = t.TeacherID
15     WHERE tl.LanguageID = @LanguageID
16 );
17 GO
```

Przykładowe wywołanie:

```
1 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).

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetCourseTotalHours
2 (
3     @CourseID INT
4 )
5 RETURNS DECIMAL(5,2)
6 AS
7 BEGIN
8     DECLARE @TotalMinutes INT;
9
10    SELECT @TotalMinutes = SUM(DATEDIFF(MINUTE, 0, DurationTime))
11    FROM CourseModules
12    WHERE CourseID = @CourseID;
13
14    IF @TotalMinutes IS NULL
15        SET @TotalMinutes = 0;
16
17    RETURN CAST(@TotalMinutes AS DECIMAL(5,2)) / 60;
18 END;
19 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.

```
1 CREATE OR ALTER FUNCTION dbo.ufnListActivitiesByLanguage
2 (
3     @LanguageID INT
4 )
5 RETURNS @Result TABLE
6 (
7     ActivityType VARCHAR(20),
8     ActivityName VARCHAR(50),
9     ActivityDate DATETIME,
10    LanguageID INT,
11    Price MONEY
12 )
13 AS
14 BEGIN
15     INSERT INTO @Result
16     SELECT
17         'Webinar' AS ActivityType,
18         w.WebinarName AS ActivityName,
19         w.WebinarDate AS ActivityDate,
20         w.LanguageID,
21         a.Price
22     FROM Webinars w
23     JOIN Activities a ON a.ActivityID = w.ActivityID
24     WHERE w.LanguageID = @LanguageID
25           AND a.Active = 1;
26
27     INSERT INTO @Result
28     SELECT
29         'CourseModule' AS ActivityType,
30         c.CourseName + ' - ' + cm.ModuleName AS ActivityName,
31         cm.Date AS ActivityDate,
32         cm.LanguageID,
33         a.Price
34     FROM CourseModules cm
35     JOIN Courses c ON c.CourseID = cm.CourseID
36     JOIN Activities a ON a.ActivityID = c.ActivityID
37     WHERE cm.LanguageID = @LanguageID
38           AND a.Active = 1;
39
40     INSERT INTO @Result
41     SELECT
42         'StudiesClass' AS ActivityType,
43         sc.ClassName AS ActivityName,
44         sc.[Date] AS ActivityDate,
45         sc.LanguageID,
46         a.Price
47     FROM StudiesClass sc
48     JOIN Activities a ON a.ActivityID = sc.ActivityID
49     WHERE sc.LanguageID = @LanguageID
50           AND a.Active = 1;
51
52     RETURN;
53 END;
54 GO
```

Przykładowe wywołanie:

```
1 SELECT *
2 FROM dbo.ufnListActivitiesByLanguage(3)
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.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetWebinarTotalHours
2 (
3     @WebinarID INT
4 )
5 RETURNS DECIMAL(5,2)
6 AS
7 BEGIN
8     DECLARE @Minutes INT;
9
10    SELECT @Minutes = DATEDIFF(MINUTE, 0, DurationTime)
11    FROM Webinars
12    WHERE WebinarID = @WebinarID;
13
14    IF @Minutes IS NULL
15        SET @Minutes = 0;
16
17    RETURN CAST(@Minutes AS DECIMAL(5,2)) / 60;
18 END;
19 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`.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetCourseTotalParticipants
2 (
3     @CourseID INT
4 )
5 RETURNS INT
6 AS
7 BEGIN
8     DECLARE @TotalParticipants INT;
9
10    SELECT @TotalParticipants = COUNT(*)
11    FROM CoursesAttendance
12    WHERE ModuleID = @CourseID
13        AND Attendance = 1;
14
15    RETURN @TotalParticipants;
16 END;
17 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`.

```

1 CREATE OR ALTER FUNCTION dbo.ufnGetStudentSchedule
2 (
3     @StudentID INT
4 )
5 RETURNS @Schedule TABLE
6 (
7     ActivityType VARCHAR(20),
8     ActivityName VARCHAR(50),
9     StartDate    DATETIME,
10    EndDate       DATETIME
11 )
12 AS
13 BEGIN
14     INSERT INTO @Schedule
15     SELECT
16         'Course' AS ActivityType,
17         c.CourseName + ' - ' + cm.ModuleName AS ActivityName,
18         cm.Date AS StartDate,
19         DATEADD(MINUTE, DATEDIFF(MINUTE, 0, cm.DurationTime), cm.Date) AS
20         EndDate
21     FROM CourseParticipants cp
22     JOIN Courses c ON c.CourseID = cp.CourseID
23     JOIN CourseModules cm ON cm.CourseID = c.CourseID
24     WHERE cp.StudentID = @StudentID;
25
26     INSERT INTO @Schedule
27     SELECT
28         'Webinar' AS ActivityType,
29         w.WebinarName AS ActivityName,
30         w.WebinarDate AS StartDate,
31         DATEADD(MINUTE, DATEDIFF(MINUTE, 0, w.DurationTime), w.WebinarDate) AS
32         EndDate
33     FROM WebinarDetails wd
34     JOIN Webinars w ON w.WebinarID = wd.WebinarID
35     WHERE wd.StudentID = @StudentID;
36
37     INSERT INTO @Schedule
38     SELECT
39         'StudiesClass' AS ActivityType,
40         sc.ClassName AS ActivityName,
41         sc.[Date] AS StartDate,
42         DATEADD(MINUTE, DATEDIFF(MINUTE, 0, sc.DurationTime), sc.[Date]) AS
43         EndDate
44     FROM StudiesClassAttendance sca
45     JOIN StudiesClass sc ON sc.StudyClassID = sca.StudyClassID
46     WHERE sca.StudentID = @StudentID;
47
48     RETURN;
49 END;
50 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`.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetStudentGrades
2 (
3     @StudentID INT
4 )
5 RETURNS TABLE
6 AS
7 RETURN
8 (
9     SELECT
10         sg.SubjectID,
11         s.SubjectName,
12         sg.SubjectGrade
13     FROM SubjectGrades sg
14     JOIN Subject s ON s.SubjectID = sg.SubjectID
15     WHERE sg.StudentID = @StudentID
16 );
17 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.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetCourseAttendanceList
2 (
3     @CourseID INT
4 )
5 RETURNS TABLE
6 AS
7 RETURN
8 (
9     SELECT
10         ca.ModuleID,
11         cm.ModuleName,
12         ca.StudentID,
13         s.FirstName,
14         s.LastName,
15         ca.Attendance AS WasPresent
16     FROM CoursesAttendance ca
17     JOIN CourseModules cm ON cm.ModuleID = ca.ModuleID
18     JOIN Students s ON s.StudentID = ca.StudentID
19     WHERE cm.CourseID = @CourseID
20 );
21 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.

```
1 CREATE OR ALTER FUNCTION dbo.ufnGetStudentAllAttendances
2 (
3     @StudentID INT
4 )
5 RETURNS @Attendances TABLE
6 (
7     ActivityType VARCHAR(20),
8     ActivityName VARCHAR(50),
9     DateOfEvent DATETIME,
10    WasPresent BIT
11 )
12 AS
13 BEGIN
14     INSERT INTO @Attendances
15     SELECT
16         'Course' AS ActivityType,
17         c.CourseName + ' - ' + cm.ModuleName,
18         cm.Date,
19         ca.Attendance
20     FROM CoursesAttendance ca
21     JOIN CourseModules cm ON cm.ModuleID = ca.ModuleID
22     JOIN Courses c ON c.CourseID = cm.CourseID
23     WHERE ca.StudentID = @StudentID;
24
25     INSERT INTO @Attendances
26     SELECT
27         'StudiesClass',
28         sc.ClassName,
29         sc.[Date],
30         sca.Attendance
31     FROM StudiesClassAttendance sca
32     JOIN StudiesClass sc ON sc.StudyClassID = sca.StudyClassID
33     WHERE sca.StudentID = @StudentID;
34
35     INSERT INTO @Attendances
36     SELECT
37         'Webinar',
38         w.WebinarName,
39         w.WebinarDate,
40         wd.Complete
41     FROM WebinarDetails wd
42     JOIN Webinars w ON w.WebinarID = wd.WebinarID
43     WHERE wd.StudentID = @StudentID;
44
45     RETURN;
46 END;
47 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddCourse
2   @CourseName      VARCHAR(50),
3   @CourseDescription TEXT          = NULL,
4   @CoursePrice     MONEY,
5   @CourseCoordinatorID INT,
6   @ActivityTitle   VARCHAR(50),
7   @ActivityPrice   MONEY,
8   @ActivityActive  BIT = 1
9 AS
10 BEGIN
11     SET NOCOUNT ON;
12
13     -- Wygenerowanie nowego ActivityID
14     DECLARE @NewActivityID INT = (
15         SELECT ISNULL(MAX(ActivityID), 0) + 1
16         FROM Activities
17     );
18
19     INSERT INTO Activities (ActivityID, Price, Title, Active)
20     VALUES (@NewActivityID, @ActivityPrice, @ActivityTitle, @ActivityActive);
21
22     -- Wygenerowanie nowego CourseID
23     DECLARE @NewCourseID INT = (
24         SELECT ISNULL(MAX(CourseID), 0) + 1
25         FROM Courses
26     );
27
28     INSERT INTO Courses (CourseID, ActivityID, CourseName, CourseDescription,
29     CoursePrice, CourseCoordinatorID)
30     VALUES (
31         @NewCourseID,
32         @NewActivityID,
```

```
33         @CourseDescription ,
34         @CoursePrice ,
35         @CourseCoordinatorID
36     );
37
38     SELECT @NewCourseID AS CreatedCourseID , @NewActivityID AS CreatedActivityID;
39 END;
40 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spRemoveCourse
2     @CourseID INT
3 AS
4 BEGIN
5     SET NOCOUNT ON;
6
7     -- Źnajd powiazany ActivityID
8     DECLARE @ActivityID INT;
9
10    SELECT @ActivityID = ActivityID
11    FROM Courses
12    WHERE CourseID = @CourseID;
13
14    IF @ActivityID IS NULL
15    BEGIN
16        RAISERROR('Course not found.', 16, 1);
17        RETURN;
18    END;
19
20    -- Usuniecie z Courses
21    DELETE FROM Courses
22    WHERE CourseID = @CourseID;
23
24    -- Usuniecie z Activities (opcjonalne)
25    DELETE FROM Activities
26    WHERE ActivityID = @ActivityID;
27
28    PRINT 'Course and related Activity removed successfully.';
29 END;
30 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spRegisterStudentInCourse
2   @CourseID INT,
3   @StudentID INT
4 AS
5 BEGIN
6   SET NOCOUNT ON;
7
8   IF EXISTS (
9     SELECT 1 FROM CourseParticipants
10    WHERE CourseID = @CourseID AND StudentID = @StudentID
11  )
12  BEGIN
13    RAISERROR('Student is already registered in this course.', 16, 1);
14    RETURN;
15  END;
16
17  INSERT INTO CourseParticipants (CourseID, StudentID)
18  VALUES (@CourseID, @StudentID);
19
20  PRINT 'Student registered successfully.';
21 END;
22 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spUnregisterStudentFromCourse
2   @CourseID INT,
3   @StudentID INT
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7
8     DELETE FROM CourseParticipants
9     WHERE CourseID = @CourseID
10    AND StudentID = @StudentID;
11
12    IF @@ROWCOUNT = 0
13    BEGIN
14        RAISERROR('Student not found in that course.', 16, 1);
15    END
16    ELSE
17    BEGIN
18        PRINT 'Student unregistered successfully.';
19    END
20 END;
21 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spUpdateActivityPrice
2   @ActivityID INT,
3   @NewPrice   MONEY
4 AS
5 BEGIN
6   SET NOCOUNT ON;
7
8   UPDATE Activities
9   SET Price = @NewPrice
10  WHERE ActivityID = @ActivityID;
11
12  IF @@ROWCOUNT = 0
13    RAISERROR('Activity not found.', 16, 1);
14  ELSE
15    PRINT 'Activity price updated successfully.';
16 END;
17 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddTeacherSchedule
2     @TeacherID      INT,
3     @ClassID        INT,
4     @CourseModuleID INT = NULL,
5     @StudiesSubjectID INT = NULL,
6     @DayOfWeek       VARCHAR(10),
7     @StartTime       TIME,
8     @EndTime         TIME,
9     @TranslatorID    INT = NULL
10 AS
11 BEGIN
12     SET NOCOUNT ON;
13
14     -- Sprawdzenie kolizji godzinowej
15     IF EXISTS (
16         SELECT 1
17         FROM Schedule
18         WHERE TeacherID = @TeacherID
19             AND DayOfWeek = @DayOfWeek
20             AND (@StartTime < EndTime AND @EndTime > StartTime)
21     )
22     BEGIN
23         RAISERROR('Collision in the teacher schedule!', 16, 1);
24         RETURN;
25     END;
26
27     DECLARE @NewScheduleID INT = (
28         SELECT ISNULL(MAX(ScheduleID), 0) + 1
29         FROM Schedule
30     );
31
32     INSERT INTO Schedule (
33         ScheduleID, ClassID, CourseModuleID, StudiesSubjectID,
34         DayOfWeek, StartTime, EndTime,
35         TeacherID, TranslatorID
36     )
37     VALUES (
```

```
38         @NewScheduleID, @ClassID, @CourseModuleID, @StudiesSubjectID,  
39         @DayOfWeek, @StartTime, @EndTime,  
40         @TeacherID, @TranslatorID  
41     );  
42  
43     SELECT @NewScheduleID AS NewScheduleID;  
44     PRINT 'Schedule added successfully.';  
45 END;  
46 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 wyszukiwujących webinary, kursy (na podstawie dat modułów) oraz zajęcia studiów, filtrując tylko aktywne rekordy.

```
1 CREATE OR ALTER PROCEDURE dbo.spFindAvailableActivities
2     @StartDate DATETIME,
3     @EndDate   DATETIME
4 AS
5 BEGIN
6     SET NOCOUNT ON;
7
8     -- Webinarzy w tym przedziale
9     SELECT
10         'Webinar' AS ActivityType,
11         w.WebinarID AS ActivityID,
12         w.WebinarName AS ActivityName,
13         w.WebinarDate AS [Date],
14         a.Price AS ActivityPrice
15 FROM Webinars w
16 JOIN Activities a ON a.ActivityID = w.ActivityID
17 WHERE w.WebinarDate >= @StartDate
18        AND w.WebinarDate < @EndDate
19        AND a.Active = 1
20
21 UNION
22
23 -- Kursy (daty i modułów)
24 SELECT
25     'Course' AS ActivityType,
26     c.CourseID AS ActivityID,
27     c.CourseName AS ActivityName,
28     cm.Date AS [Date],
29     a.Price AS ActivityPrice
30 FROM Courses c
31 JOIN CourseModules cm ON cm.CourseID = c.CourseID
32 JOIN Activities a ON a.ActivityID = c.ActivityID
33 WHERE cm.Date >= @StartDate
34        AND cm.Date < @EndDate
35        AND a.Active = 1
36
37 UNION
38
39 -- Studia (zajęcia StudiesClass)
40 SELECT
41     'StudiesClass' AS ActivityType,
42     sc.StudyClassID AS ActivityID,
43     sc.ClassName AS ActivityName,
44     sc.[Date] AS [Date],
45     a.Price AS ActivityPrice
46 FROM StudiesClass sc
47 JOIN Activities a ON a.ActivityID = sc.ActivityID
48 WHERE sc.[Date] >= @StartDate
49        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**.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddTeacher
2   @FirstName VARCHAR(30),
3   @LastName  VARCHAR(30),
4   @HireDate  DATE          = NULL,
5   @Phone     VARCHAR(15) = NULL,
6   @Email     VARCHAR(60),
7   @LanguagesCSV VARCHAR(MAX) = NULL
8 AS
9 BEGIN
10    SET NOCOUNT ON;
11
12    DECLARE @NewTeacherID INT = (
13        SELECT ISNULL(MAX(TeacherID), 0) + 1
14        FROM Teachers
15    );
16
17    INSERT INTO Teachers (TeacherID, FirstName, LastName, HireDate, Phone, Email)
18    VALUES (@NewTeacherID, @FirstName, @LastName, @HireDate, @Phone, @Email);
19
20    PRINT 'Teacher created with ID = ' + CAST(@NewTeacherID AS VARCHAR(10));
21
22    IF @LanguagesCSV IS NOT NULL AND LEN(@LanguagesCSV) > 0
23    BEGIN
24        ;WITH CTE_Lang AS (
25            SELECT value AS LangID
26            FROM STRING_SPLIT(@LanguagesCSV, ',')
27        )
28        INSERT INTO TeacherLanguages (TeacherID, LanguageID)
29        SELECT @NewTeacherID, CAST(LangID AS INT)
30        FROM CTE_Lang;
31    END;
32
33    PRINT 'Teacher languages assigned.';
34 END;
35 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).

```
1 CREATE OR ALTER PROCEDURE dbo.spAddTranslator
2   @FirstName    VARCHAR(30),
3   @LastName     VARCHAR(30),
4   @HireDate     DATE          = NULL,
5   @Phone        VARCHAR(15) = NULL,
6   @Email        VARCHAR(60),
7   @LanguagesCSV VARCHAR(MAX) = NULL
8 AS
9 BEGIN
10    SET NOCOUNT ON;
11
12    DECLARE @NewTranslatorID INT = (
13        SELECT ISNULL(MAX(TranslatorID), 0) + 1
14        FROM Translators
15    );
16
17    INSERT INTO Translators (TranslatorID, FirstName, LastName, HireDate, Phone,
18        Email)
19    VALUES (@NewTranslatorID, @FirstName, @LastName, @HireDate, @Phone, @Email);
20
21    PRINT 'Translator created with ID = ' + CAST(@NewTranslatorID AS VARCHAR(10))
22    );
23
24    IF @LanguagesCSV IS NOT NULL AND LEN(@LanguagesCSV) > 0
25    BEGIN
26        ;WITH CTE_Lang AS (
27            SELECT value AS LangID
28            FROM STRING_SPLIT(@LanguagesCSV, ',')
29        )
30        INSERT INTO TranslatorsLanguages (TranslatorID, LanguageID)
31        SELECT @NewTranslatorID, CAST(LangID AS INT)
32        FROM CTE_Lang;
33    END;
34
35    PRINT 'Translator languages assigned.';
36 END;
37 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddWebinar
2   @WebinarName      VARCHAR(50),
3   @WebinarDescription TEXT,
4   @WebinarPrice     MONEY,
5   @TeacherID        INT,
6   @LanguageID       INT,
7   @VideoLink        VARCHAR(50),
8   @WebinarDate      DATETIME,
9   @DurationTime     TIME(0),
10  @ActivityTitle     VARCHAR(50),
11  @ActivityPrice     MONEY,
12  @ActivityActive    BIT = 1
13 AS
14 BEGIN
15   SET NOCOUNT ON;
16
17   DECLARE @NewActivityID INT = (
18     SELECT ISNULL(MAX(ActivityID), 0) + 1
19     FROM Activities
20   );
21
22   INSERT INTO Activities (ActivityID, Price, Title, Active)
23   VALUES (@NewActivityID, @ActivityPrice, @ActivityTitle, @ActivityActive);
24
25   DECLARE @NewWebinarID INT = (
26     SELECT ISNULL(MAX(WebinarID), 0) + 1
27     FROM Webinars
28   );
29
30   INSERT INTO Webinars (
31     WebinarID, ActivityID, TeacherID, WebinarName,
32     WebinarPrice, VideoLink, WebinarDate, DurationTime,
```

```
33     WebinarDescription, LanguageID
34 )
35 VALUES (
36     @NewWebinarID, @NewActivityID, @TeacherID, @WebinarName,
37     @WebinarPrice, @VideoLink, @WebinarDate, @DurationTime,
38     @WebinarDescription, @LanguageID
39 );
40
41 SELECT @NewWebinarID AS NewWebinarID, @NewActivityID AS NewActivityID;
42 END;
43 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 webinarzem, a następnie usuwa rekordy z tabel Webinars i Activities. W przypadku braku webinaru generuje błąd.

```
1 CREATE OR ALTER PROCEDURE dbo.spRemoveWebinar
2   @WebinarID INT
3 AS
4 BEGIN
5     SET NOCOUNT ON;
6
7     DECLARE @ActivityID INT;
8     SELECT @ActivityID = ActivityID
9     FROM Webinars
10    WHERE WebinarID = @WebinarID;
11
12    IF @ActivityID IS NULL
13    BEGIN
14        RAISERROR('Webinar not found.', 16, 1);
15        RETURN;
16    END;
17
18    DELETE FROM Webinars
19    WHERE WebinarID = @WebinarID;
20
21    DELETE FROM Activities
22    WHERE ActivityID = @ActivityID;
23
24    PRINT 'Webinar and related Activity removed.';
25 END;
26 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddSubjectToStudies
2   @StudiesID          INT,
3   @CoordinatorID      INT,
4   @SubjectName        VARCHAR(50),
5   @SubjectDescription TEXT = NULL
6 AS
7 BEGIN
8     SET NOCOUNT ON;
9
10    DECLARE @NewSubjectID INT = (
11        SELECT ISNULL(MAX(SubjectID), 0) + 1
12        FROM Subject
13    );
14
15    INSERT INTO Subject (
16        SubjectID, StudiesID, CoordinatorID,
17        SubjectName, SubjectDescription
18    )
19    VALUES (
20        @NewSubjectID,
21        @StudiesID,
22        @CoordinatorID,
23        @SubjectName,
24        @SubjectDescription
25    );
26
27    SELECT @NewSubjectID AS NewSubjectID;
28 END;
29 GO
```

10.13 Automatyczne rozdzielanie studentów do grup (spAutoAssignStudentsToGroups)

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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAutoAssignStudentsToGroups
2 AS
3 BEGIN
4     SET NOCOUNT ON;
5
6     -- Przypisanie grupy: parzyste -> grupa 1, nieparzyste -> grupa 2
7     UPDATE Students
8     SET GroupID = CASE WHEN (StudentID % 2) = 0 THEN 1 ELSE 2 END
9     WHERE GroupID IS NULL;
10
11     PRINT 'Auto assignment done.';
12 END;
13 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spUpdateTeacherData
2   @TeacherID INT,
3   @FirstName VARCHAR(30) = NULL,
4   @LastName  VARCHAR(30) = NULL,
5   @Phone     VARCHAR(15) = NULL,
6   @Email     VARCHAR(60) = NULL,
7   @HireDate  DATE        = NULL
8 AS
9 BEGIN
10    SET NOCOUNT ON;
11
12    UPDATE Teachers
13    SET
14        FirstName = COALESCE(@FirstName, FirstName),
15        LastName  = COALESCE(@LastName, LastName),
16        Phone     = COALESCE(@Phone, Phone),
17        Email     = COALESCE(@Email, Email),
18        HireDate  = COALESCE(@HireDate, HireDate)
19    WHERE TeacherID = @TeacherID;
20
21    IF @@ROWCOUNT = 0
22        RAISERROR('Teacher not found.', 16, 1);
23    ELSE
24        PRINT 'Teacher data updated successfully.';
25 END;
26 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spUpdateStudentData
2   @StudentID INT,
3   @FirstName VARCHAR(30) = NULL,
4   @LastName  VARCHAR(30) = NULL,
5   @Address   VARCHAR(30) = NULL,
6   @CityID    INT          = NULL,
7   @PostalCode VARCHAR(10) = NULL,
8   @Phone     VARCHAR(15) = NULL,
9   @Email     VARCHAR(60) = NULL
10 AS
11 BEGIN
12     SET NOCOUNT ON;
13
14     UPDATE Students
15     SET
16         FirstName = COALESCE(@FirstName, FirstName),
17         LastName  = COALESCE(@LastName, LastName),
18         [Address] = COALESCE(@Address, [Address]),
19         CityID    = COALESCE(@CityID, CityID),
20         PostalCode = COALESCE(@PostalCode, PostalCode),
21         Phone     = COALESCE(@Phone, Phone),
22         Email     = COALESCE(@Email, Email)
23     WHERE StudentID = @StudentID;
24
25     IF @@ROWCOUNT = 0
26         RAISERROR('Student not found.', 16, 1);
27     ELSE
28         PRINT 'Student data updated successfully.';
29 END;
30 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spUpdateRODO
2   @StudentID INT,
3   @Withdraw BIT
4 AS
5 BEGIN
6   SET NOCOUNT ON;
7
8   DECLARE @Today DATE = CONVERT(DATE, GETDATE());
9
10  IF EXISTS (SELECT 1 FROM RODO_Table WHERE StudentID = @StudentID)
11  BEGIN
12    UPDATE RODO_Table
13    SET [Date] = @Today,
14        Withdraw = @Withdraw
15    WHERE StudentID = @StudentID;
16    PRINT 'RODO data updated.';
17  END
18  ELSE
19  BEGIN
20    INSERT INTO RODO_Table (StudentID, [Date], Withdraw)
21    VALUES (@StudentID, @Today, @Withdraw);
22    PRINT 'RODO data inserted.';
23  END
24 END;
25 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddInternship
2   @StudiesID INT,
3   @StartDate DATETIME
4 AS
5 BEGIN
6   SET NOCOUNT ON;
7
8   DECLARE @NewInternshipID INT = (
9     SELECT ISNULL(MAX(InternshipID), 0) + 1
10    FROM Internship
11   );
12
13   INSERT INTO Internship (InternshipID, StudiesID, StartDate)
14   VALUES (@NewInternshipID, @StudiesID, @StartDate);
15
16   SELECT @NewInternshipID AS InternshipID;
17 END;
18 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddStudent
2   @FirstName VARCHAR(30),
3   @LastName  VARCHAR(30),
4   @Address   VARCHAR(30),
5   @CityID    INT,
6   @PostalCode VARCHAR(10),
7   @Phone     VARCHAR(15) = NULL,
8   @Email     VARCHAR(60)
9 AS
10 BEGIN
11     SET NOCOUNT ON;
12
13     DECLARE @NewStudentID INT = (
14         SELECT ISNULL(MAX(StudentID), 0) + 1
15         FROM Students
16     );
17
18     INSERT INTO Students (
19         StudentID, FirstName, LastName, [Address],
20         CityID, PostalCode, Phone, Email
21     )
22     VALUES (
23         @NewStudentID, @FirstName, @LastName, @Address,
24         @CityID, @PostalCode, @Phone, @Email
25     );
26
27     SELECT @NewStudentID AS StudentID;
28 END;
29 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spRemoveStudent
2   @StudentID INT
3 AS
4 BEGIN
5     SET NOCOUNT ON;
6
7     DELETE FROM Students
8     WHERE StudentID = @StudentID;
9
10    IF @@ROWCOUNT = 0
11        RAISERROR('Student not found.', 16, 1);
12    ELSE
13        PRINT 'Student removed successfully.';
14 END;
15 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddEmployee
2   @FirstName      VARCHAR(30),
3   @LastName       VARCHAR(30),
4   @HireDate       DATE          = NULL,
5   @EmployeeTypeID INT,
6   @Phone          VARCHAR(15)   = NULL,
7   @Email          VARCHAR(60)
8 AS
9 BEGIN
10    SET NOCOUNT ON;
11
12    DECLARE @NewEmployeeID INT = (
13        SELECT ISNULL(MAX(EmployeeID), 0) + 1
14        FROM Employees
15    );
16
17    INSERT INTO Employees (
18        EmployeeID, FirstName, LastName, HireDate,
19        EmployeeTypeID, Phone, Email
20    )
21    VALUES (
22        @NewEmployeeID, @FirstName, @LastName, @HireDate,
23        @EmployeeTypeID, @Phone, @Email
24    );
25
26    SELECT @NewEmployeeID AS EmployeeID;
27 END;
28 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spRemoveEmployee
2   @EmployeeID INT
3 AS
4 BEGIN
5     SET NOCOUNT ON;
6
7     DELETE FROM Employees
8     WHERE EmployeeID = @EmployeeID;
9
10    IF @@ROWCOUNT = 0
11        RAISERROR('Employee not found.', 16, 1);
12    ELSE
13        PRINT 'Employee removed successfully.';
14 END;
15 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddCourseModule
2   @CourseID      INT,
3   @ModuleName    VARCHAR(50),
4   @ModuleDate    DATETIME,
5   @DurationTime  TIME(0),
6   @TeacherID     INT,
7   @LanguageID    INT,
8   @TranslatorID  INT = NULL,
9   @ModuleType    VARCHAR(20), -- 'stationary'/'online_sync'/'online_async'
10  @ClassID        INT = NULL, -- dla stacjonarnego
11  @LinkOrVideo    VARCHAR(60) = NULL, -- link lub video
12  @Limit          INT = 0      -- limit miejsc
13 AS
14 BEGIN
15     SET NOCOUNT ON;
16
17     DECLARE @NewModuleID INT = (
18         SELECT ISNULL(MAX(ModuleID), 0) + 1
19         FROM CourseModules
20     );
21
22     INSERT INTO CourseModules (
23         ModuleID, CourseID, ModuleName, [Date], DurationTime,
24         TeacherID, TranslatorID, LanguageID
25     )
26     VALUES (
27         @NewModuleID, @CourseID, @ModuleName, @ModuleDate, @DurationTime,
28         @TeacherID, @TranslatorID, @LanguageID
29     );
30
```

```
31 IF @ModuleType = 'stationary'
32 BEGIN
33     INSERT INTO StationaryModule (StationaryModuleID, ClassID, [Limit])
34     VALUES (@NewModuleID, @ClassID, @Limit);
35 END
36 ELSE IF @ModuleType = 'online_sync'
37 BEGIN
38     INSERT INTO OnlineSyncModule (OnlineSyncModuleID, Link)
39     VALUES (@NewModuleID, @LinkOrVideo);
40 END
41 ELSE IF @ModuleType = 'online_async'
42 BEGIN
43     INSERT INTO OnlineAsyncModule (OnlineAsyncModuleID, Video)
44     VALUES (@NewModuleID, @LinkOrVideo);
45 END
46 ELSE
47 BEGIN
48     RAISERROR('Unknown module type.', 16, 1);
49     ROLLBACK TRANSACTION;
50     RETURN;
51 END;
52
53 PRINT 'Course module created with ID = ' + CAST(@NewModuleID AS VARCHAR(10))
54 ;
55 END;
56 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spAddEuroRate
2   @Rate DECIMAL(10,2),
3   @RateDate DATETIME = NULL
4 AS
5 BEGIN
6   SET NOCOUNT ON;
7
8   IF @RateDate IS NULL
9     SET @RateDate = CONVERT(DATETIME, CONVERT(DATE, GETDATE()));
10
11  MERGE EuroExchangeRate AS tgt
12  USING (SELECT @RateDate AS [Date], @Rate AS Rate) AS src
13  ON (tgt.[Date] = src.[Date])
14  WHEN MATCHED THEN
15    UPDATE SET Rate = src.Rate
16  WHEN NOT MATCHED THEN
17    INSERT ([Date], Rate)
18    VALUES (src.[Date], src.Rate)
19  OUTPUT $action AS MergeAction;
20
21  PRINT 'Euro rate upsert completed.';
22 END;
23 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spGetDebtors
2 AS
3 BEGIN
4     SET NOCOUNT ON;
5
6     SELECT DISTINCT
7         S.StudentID,
8         S.FirstName,
9         S.LastName,
10        O.OrderID,
11        OPS.OrderPaymentStatus,
12        OD.ActivityID,
13        A.Title AS ActivityTitle
14 FROM    dbo.Orders O
15 INNER JOIN    dbo.OrderDetails OD ON O.OrderID = OD.OrderID
16 INNER JOIN    dbo.OrderPaymentStatus OPS ON O.PaymentURL = OPS.PaymentURL
17 INNER JOIN    dbo.Activities A ON A.ActivityID = OD.ActivityID
18 INNER JOIN    dbo.Students S ON O.StudentID = S.StudentID
19 WHERE OPS.OrderPaymentStatus <> 'Paid'
20 ORDER BY S.StudentID;
21 END;
22 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.

```
1 CREATE OR ALTER PROCEDURE dbo.spGenerateCourseAttendanceReport
2   @CourseID INT
3 AS
4 BEGIN
5     SET NOCOUNT ON;
6
7     SELECT
8         ca.ModuleID,
9         cm.ModuleName,
10        ca.StudentID,
11        s.FirstName,
12        s.LastName,
13        ca.Attendance AS WasPresent
14 FROM CoursesAttendance ca
15 JOIN CourseModules cm ON cm.ModuleID = ca.ModuleID
16 JOIN Students s ON s.StudentID = ca.StudentID
17 WHERE cm.CourseID = @CourseID
18 ORDER BY cm.Date, ca.StudentID;
19 END;
20 GO
```


10.26 Procedura oznaczania obecności studenta (spMarkCourseModuleAttendance)

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.

```
1 CREATE OR ALTER PROCEDURE dbo.spMarkCourseModuleAttendance
2   @ModuleID INT,
3   @StudentID INT,
4   @WasPresent BIT
5 AS
6 BEGIN
7     SET NOCOUNT ON;
8
9     UPDATE CoursesAttendance
10    SET Attendance = @WasPresent
11    WHERE ModuleID = @ModuleID
12          AND StudentID = @StudentID;
13
14    IF @@ROWCOUNT = 0
15    BEGIN
16        INSERT INTO CoursesAttendance (ModuleID, StudentID, Attendance)
17        VALUES (@ModuleID, @StudentID, @WasPresent);
18    END;
19 END;
20 GO
```

10.27 Procedura oznaczania obecności studenta (spMarkStudiesClassAttendance)

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.

```
1 CREATE OR ALTER PROCEDURE dbo.spMarkStudiesClassAttendance
2   @StudyClassID INT,
3   @StudentID    INT,
4   @WasPresent   BIT
5 AS
6 BEGIN
7     SET NOCOUNT ON;
8
9     UPDATE StudiesClassAttendance
10    SET Attendance = @WasPresent
11    WHERE StudyClassID = @StudyClassID
12          AND StudentID = @StudentID;
13
14    IF @@ROWCOUNT = 0
15    BEGIN
16        INSERT INTO StudiesClassAttendance (StudyClassID, StudentID, Attendance)
17        VALUES (@StudyClassID, @StudentID, @WasPresent);
18    END;
19 END;
20 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)

```
1 GRANT EXECUTE ON OBJECT::dbo.ufnGetOrderTotal
2     TO Role_Student, Role_Admin, Role_Employee;
3 GO
4
5 GRANT EXECUTE ON OBJECT::dbo.ufnGetStationaryModuleFreeSlots
6     TO Role_Student, Role_Admin, Role_Employee;
7 GO
8
9 GRANT EXECUTE ON OBJECT::dbo.ufnCountActiveCoursesInPeriod
10    TO Role_Admin, Role_Employee;
11 GO
12
13 GRANT EXECUTE ON OBJECT::dbo.ufnGetSubjectAverageGrade
14    TO Role_Student, Role_Teacher, Role_Admin;
15 GO
16
17 GRANT EXECUTE ON OBJECT::dbo.ufnGetFreeSeatsInBuilding
18    TO Role_Admin, Role_Employee;
19 GO
20
21 GRANT EXECUTE ON OBJECT::dbo.ufnConvertActivityPriceToEUR
22    TO Role_Admin, Role_Employee;
23 GO
24
25 GRANT EXECUTE ON OBJECT::dbo.ufnGetCourseTotalHours
26    TO Role_Admin, Role_Employee, Role_Teacher;
27 GO
28
29 GRANT EXECUTE ON OBJECT::dbo.ufnGetWebinarTotalHours
30    TO Role_Admin, Role_Employee;
31 GO
32
33 GRANT EXECUTE ON OBJECT::dbo.ufnGetCourseTotalParticipants
34    TO Role_Admin, Role_Employee, Role_Teacher;
35 GO
```

Uprawnienia dla funkcji tabelarycznych (SELECT)

```
1 GRANT SELECT ON OBJECT::dbo.ufnGetTeachersByLanguage
2   TO Role_Admin, Role_Employee, Role_Student;
3 GO
4
5 GRANT SELECT ON OBJECT::dbo.ufnListActivitiesByLanguage
6   TO Role_Student, Role_Admin, Role_Employee;
7 GO
8
9 GRANT SELECT ON OBJECT::dbo.ufnGetStudentSchedule
10  TO Role_Student, Role_Teacher, Role_Admin;
11 GO
12
13 GRANT SELECT ON OBJECT::dbo.ufnGetStudentGrades
14  TO Role_Student, Role_Teacher, Role_Admin;
15 GO
16
17 GRANT SELECT ON OBJECT::dbo.ufnGetCourseAttendanceList
18  TO Role_Teacher, Role_Admin;
19 GO
20
21 GRANT SELECT ON OBJECT::dbo.ufnGetStudentAllAttendances
22  TO Role_Student, Role_Teacher, Role_Admin;
23 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.

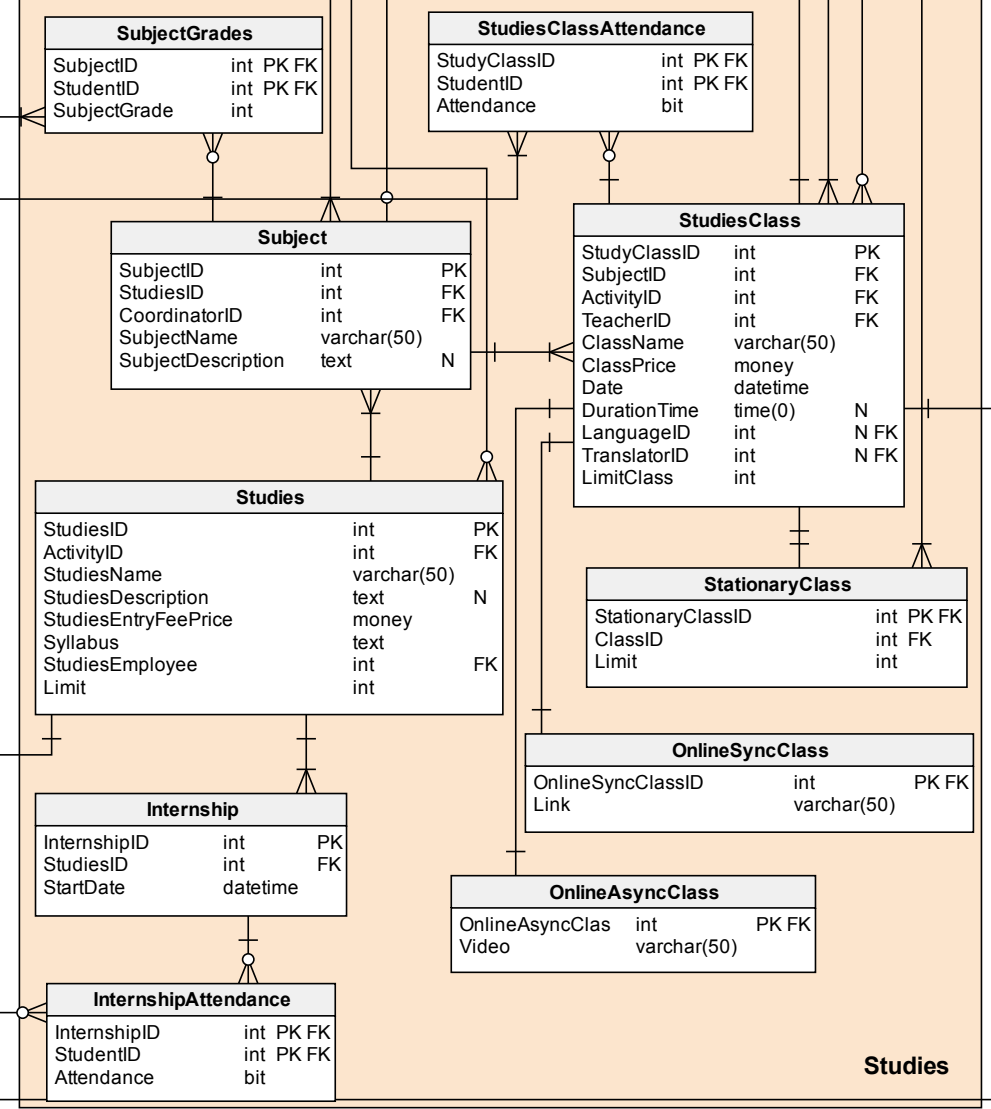
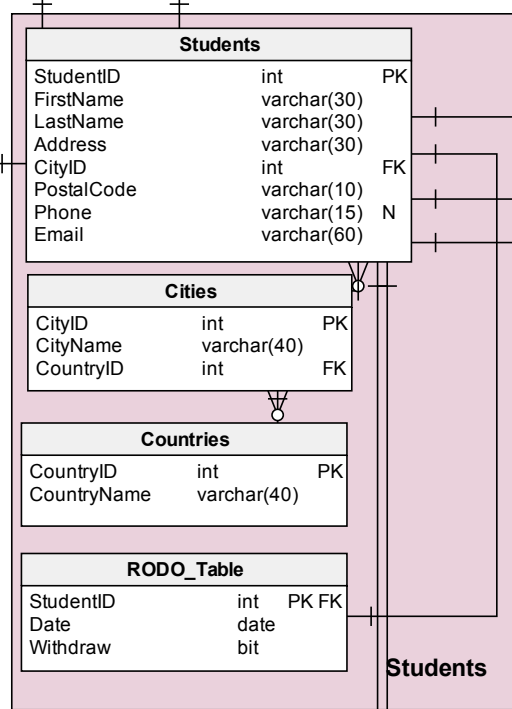
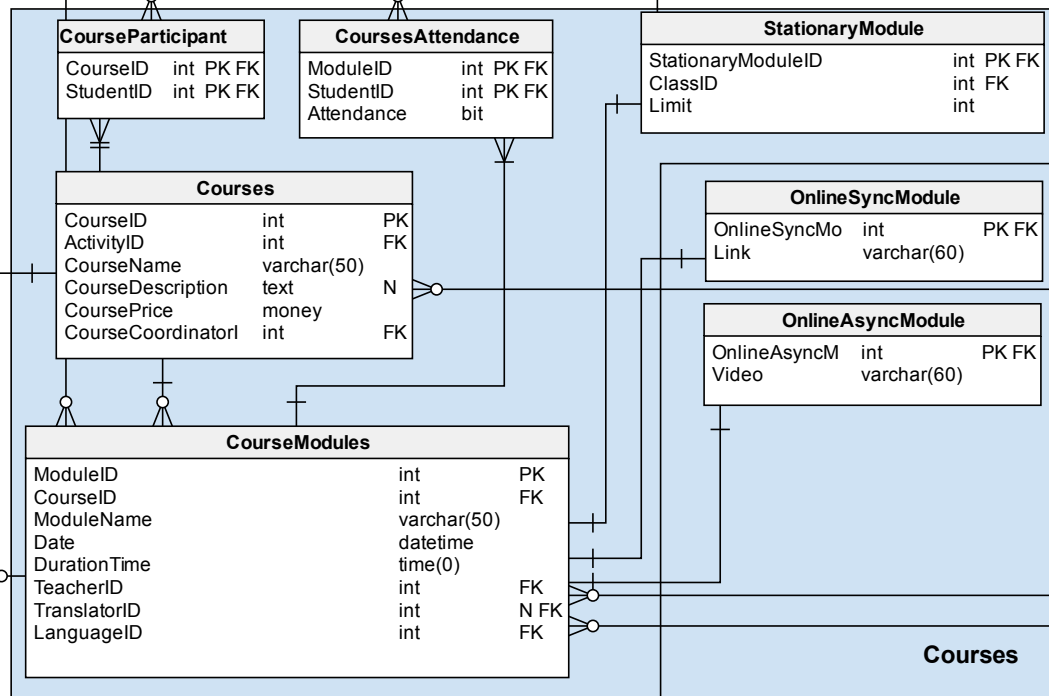
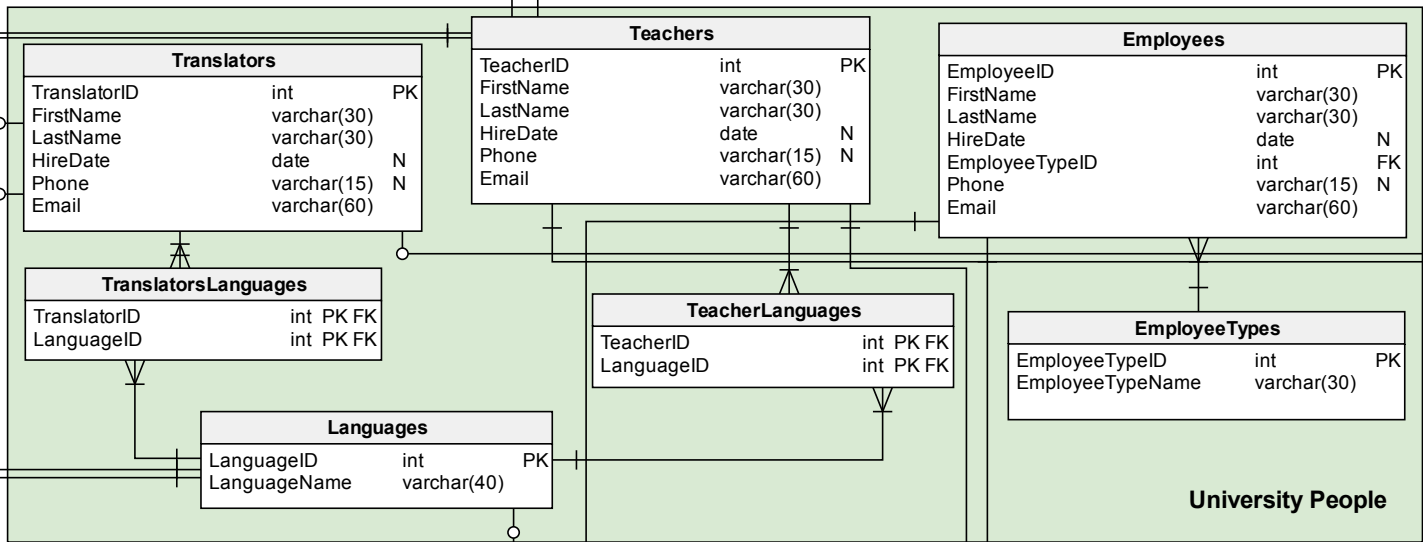
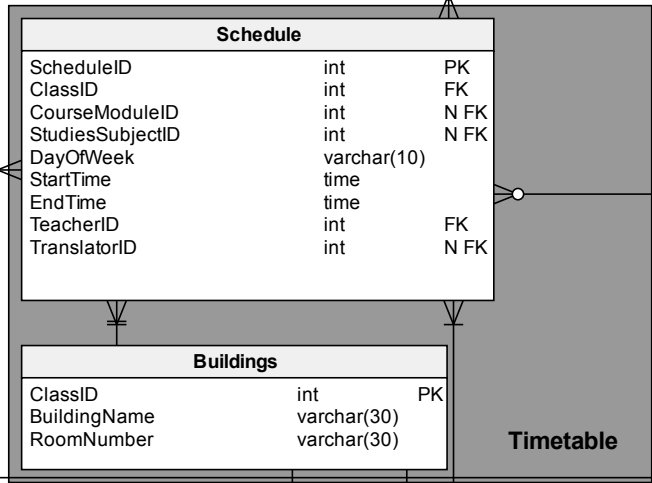
Powyższe podejście pozwala wypełnić bazę przykładowymi danymi bez konieczności ręcznego tworzenia dziesiątek instrukcji **INSERT**. Dzięki temu znacznie przyspiesza się proces rozwoju i testowania aplikacji, a także umożliwia weryfikację poprawności implementacji *relacji* i *zależności* (klucze obce,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



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 triggerzy, 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)
 - Triggerzy (T = 10)
 - Widoki (T = 10)
- **Kacper Wdowiak** (T = 30)
 - Projekt i schemat bazy danych (Współudział) (T = 10)
 - Procedury (T = 10)
 - Funkcje (T = 10)