Akademia Górniczo-Hutnicza

im. Stanisława Staszica w Krakowie

Wydział Informatyki, Elektroniki i Telekomunikacji

Instytut Informatyki

STUDIA PODYPLOMOWE
SYSTEMY BAZ DANYCH

Projekt dyplomowy

Aplikacja internetowa oferująca kursy i korepetycje z zakresu matematyki

Monika Dąbrowska

Opiekun Projektu: dr inż. Robert Marcjan

Kierownik Studiów: dr inż. Anna Zygmunt

Kraków 2023

Spis treści

1. Wstęp

Zapotrzebowanie na kursy i korepetycje z matematyki dla uczniów klas szkół podstawowych i liceów w ostatnich latach drastycznie wzrosło. Odpowiedzią na to zapotrzebowanie jest system, będący aplikacją internetową który jest przedmiotem niniejszej pracy dyplomowej.

System ten oferuje kursy i korepetycje z zakresu matematyki, umożliwa użytkownikom zdalną naukę matematyki w dogodnym dla nich miejscu i czasie. System jest przeznaczony dla uczniów szkół podstawowych, liceum, oraz dla osób przygotowujących się do egaminów maturalnych. Aplikacja oferuje poziom nauczania w zakresie podstawowym i rozszerzonym.

Głownymi funkcjonalnościami systemu są:

- Rejestracja i logowanie użytkowników - użytkownicy mają możliwość utworzenia konta, dzięki któremu mogą korzystać z funkcjonalności systemu.
- Przeglądanie kursów i korepetycji - system umożliwia użytkownikom przeglądanie oferowanych kursów i korepetycji z matematyki. Każdy kurs zawiera szczegółowe informacje takie jak: poziom, temat, czas trwania, data dodania i cenę.
- Rezerwacja i opłacenie kursów i korepetycji - użytkownicy mają możliwość rezerwacji i opłacenia kursów oraz korepetycji z poziomu systemu.
- Przeglądanie zakupionych kursów i dodanie kursów do ulubionych - użytkownicy mogą przeglądać zakupione i opłacone kursy, oraz dodać konkretne kursy do ulubionych.
- Zarządzanie rezerwacjami korepetycji - użytkownicy mogą zarządzać swoimi korepetycjami, dodać komentarze lub zrezygnować z danego terminu.
- Odtworzenie kursu - użytkownicy mogą odtworzyć zakupione i opłacone kursy.
- Przeglądanie ulubionych kursów - użytkownicy mogą przeglądać kursy dodane do ulubionych, oraz usunąć dany kurs z ulubionych.

Do zaprojektowania i implementacji bazy danych wykorzystano *SQL Server* firmy Microsoft [1], makiety interfejsu powstały przy użyciu programu *Figma* [2], natomiast elementy kodu aplikacji powstały przy użyciu języka programowania *Python* z frameworkiem *Flask* [3].

Kolejne rozdziały pracy opisują szczegółowo etapy powstawania aplikacji internetowej, począwszy od analizy wymagań (Rozdział 2), przez makiety interfejsu (Rozdział 3), projekt bazy danych (Rozdział 4), warstwy dostępu do danych (Rozdział 5), po implementację elemntów aplikacji (Rozdział 6).

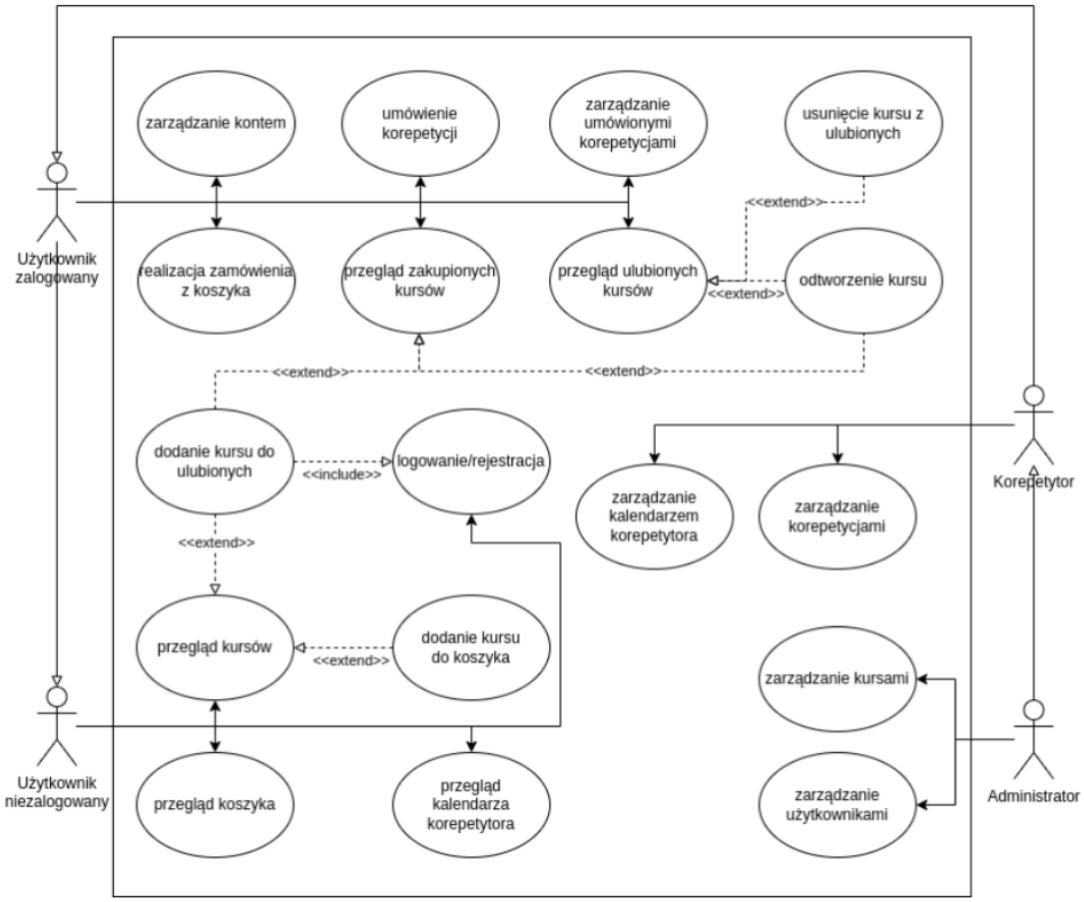
2. Analiza wymagań

Analiza wymagań jest ważnym aspektem w procesie projektu każdej aplikacji. Identyfikuje ona wymagania, jakich spełnienia oczekują użytkownicy systemu, oraz ograniczenia nakładane na jego realizację i użytkowanie. Składa się ona z kilku etapów, takich jak zbieranie wymagań, ich dokumentowanie, analiza i weryfikacja. Istnieje wiele narzędzi wykorzystywanych do modelowania wymagań. Mogą to być notatki z rozmowy z klientem, nagrania, diagramy przypadków użycia i scenariusze, prototypy GUI, czy diagramy stanów UML.

W niniejszej pracy wykorzystywana jest metoda *przypadków użycia*, czyli opis zbiorów ciągów akcji wykonywanych przez system w celu osiągniecia określonego wyniku [5].

2.1 Diagram przypadków użycia

Na Rysunku 1 znajduje się diagram przypadków użycia projektowanej aplikacji internetowej. Diagram powstał przy użyciu aplikacji *Diagrams.net* [4].



Rysunek 1: Diagram przypadków użycia projektowanej aplikacji internetowej.

2.2 Opisy i scenariusze przypadków użycia

Poniżej przedstawione zostały opisy i scenariusze przypadków użycia. Każdy element składa się z następujących części: *przypadek użycia* - czyli nazwa konkretnego przypadku, *aktorzy* - użytkownicy mogący zainicjować określoną akcję, *opis* - szczegółowy opis przypadku użycia, oraz *przypadki użycia* - przypadki rozszerzające.

Przypadek użycia: przegląd kursów.
Aktorzy: użytkownik niezalogowany, użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik przegląda dostępne kursy. Domyślnie wyświetlane są wszystkie kursy w nieokreślonym porządku. Użytkownik ma możliwość sortowania kursów: po cenie, po poziomie, po tematyce, po długości trwania, oraz po autorze kursu. Użytkownik ma możliwość wybrania danego kursu i dodania go do koszyka (*PU dodanie kursu do koszyka*).
Przypadki użycia:dodanie kursu do koszyka.

Przypadek użycia: dodanie kursu do koszyka.
Aktorzy: użytkownik niezalogowany, użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik dodaje wybrany kurs do koszyka.

Przypadek użycia: przegląd koszyka.
Aktorzy: użytkownik niezalogowany, użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik przegląda kursy dodane do koszyka. Użytkownik ma możliwość edytować koszyk: usunąć dany kurs z koszyka.

Przypadek użycia: przegląd kalendarza korepetytora.
Aktorzy: użytkownik niezalogowany, użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik przegląda kalendarz, który domyślnie wyświetla wszystkie wolne terminy i wszystkich dostępnych korepetytorów. Kalendarz wyświetlany jest tygodniami, pierwszym dniem tygodnia jest poniedziałek. Użytkownik ma możliwość wybrać konkretnego korepetytora i przejrzeć jego wolne terminy.

Przypadek użycia: realizacja zamówienia z koszyka.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik realizuje zamówienie kursów dodanych do koszyka. Realizacja zamówienia

odbywa się poprzez wybranie formy płatności. Możliwe formy płatności: przelew tradycyjny, szybki przelew PayU, lub BLIK. Po wybraniu formy płatności, użytkownik zostaje przekierowany do strony na której realizuje płatność. Po zaksięgowaniu wpłaty, wybrane kursy zostają dodane do listy zakupionych kursów użytkownika, oraz zostaje wysłany e-mail z potwierdzeniem zakupu.

Przypadek użycia: zarządzanie kontem.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik zarządza swoimi danymi. Może zmienić swoje imię i nazwisko, płeć, datę urodzenia, adres, numer telefonu. Użytkownik może zmienić również hasło. Nowe hasło musi składać się z 5 do 15 znaków, musi posiadać przynajmniej jeden znak specjalny i jedną cyfrę. Użytkownik może również usunąć swoje konto.

Przypadek użycia: umówienie korepetycji.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik na podstawie kalendarza korepetytora umawia termin korepetycji. Umawianie odbywa się poprzez wybranie konkretnego korepetytora, oraz wybranie wolnego terminu w jego kalendarzu. Po wybraniu terminu, na podany adres e-mail użytkownika i korepetytora zostaje wysłana widomość z potwierdzeniem.

Przypadek użycia: zarządzanie umówionymi korepetycjami.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik przegląda umówione korepetycje, które są wyświetlane jako lista posortowana po dacie. Użytkownik może odwołać korepetycje, lub dodać komentarz do konkretnych korepetycji. Po odwołaniu korepetycji na podany adres e-mail użytkownika, oraz korepetytora zostaje wysłana wiadomość z potwierdzeniem rezygnacji.

Przypadek użycia: zarządzanie kalendarzem korepetytora.
Aktorzy: korepetytor, administrator.
Opis: korepetytor zarządza swoim kalendarzem: wprowadza, lub usuwa dostępne terminy.

Przypadek użycia: przegląd ulubionych kursów.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik przegląda kursy dodane do ulubionych. Kursy wyświetlane są w formie listy, domyślnie w nieokreślonym porządku. Użytkownik ma możliwość odtworzenia kursu (*PU odtworzenie kursu*), a także usunięcia kursu z listy ulubionych (*PU usunięcie kursu z ulubionych*).
Przypadki użycia: odtworzenie kursu, usunięcie kursu z ulubionych.

Przypadek użycia: logowanie/rejestracja.
Aktorzy: użytkownik niezalogowany, użytkownik zalogowany, korepetytor, administrator.
Opis: niezalogowany użytkownik, nieposiadający konta rejestruje się w aplikacji. Rejestracja odbywa się poprzez wypełnienie formularza zawierającego obowiązkowe pola: imię, nazwisko, e-mail i hasło (hasło musi składać się z 5 do 15 znaków, musi posiadać przynajmniej jeden znak specjalny i jedną cyfrę). Nieobowiązkowe pola: numer telefonu, płeć, adres, data urodzenia. Po wysłaniu formularza rejestracyjnego, na podany adres e-mail zostaje wysłane powiadomienie o pomyślnej rejestracji użytkownika. Niezalogowany użytkownik posiadający konto w aplikacji, wprowadza dane uwierzytelniające: e-mail i hasło. Zalogowany użytkownik przekierowany zostaje do swojego panelu użytkownika.

Przypadek użycia: dodanie kursu do ulubionych.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik podczas przeglądania zakupionych kursów dodaje wybrany kurs do ulubionych.

Przypadek użycia: odtworzenie kursu.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik odtwarza wybrany, zakupiony kurs.

Przypadek użycia: usunięcie kursu z ulubionych.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik usuwa dany kurs z ulubionych.

Przypadek użycia: przegląd zakupionych kursów.
Aktorzy: użytkownik zalogowany, korepetytor, administrator.
Opis: użytkownik przegląda zakupione kursy. Kursy wyświetlane są w formie listy, domyślnie w nieokreślonym porządku. Użytkownik ma możliwość sortowania kursów: po poziomie, po tematyce, po długości trwania, oraz po autorze kursu. Użytkownik ma możliwość dodania kursu do ulubionych (*PU dodanie kursu do ulubionych*), a także odtworzenia kursu (*PU odtworzenie kursu*).
Przypadki użycia: dodanie kursu do ulubionych, odtworzenie kursu.

Przypadek użycia: zarządzanie korepetycjami.
Aktorzy: korepetytor, administrator.
Opis: korepetytor zarządza umówionymi korepetycjami. Może zmienić termin, odwołać korepetycje, lub dodać komentarz do konkretnych korepetycji. Przy każdorazowej zmianie przesyłana jest wiadomość na podany przez użytkownika i korepetytora adres e-mail.

Przypadek użycia: zarządzanie kursami.
Aktorzy: administrator.
Opis: administrator zarządza kursami. Może dodać i usunąć kurs, dodać, zmienić, lub usunąć wybrane szczegóły dotyczące kursu.

Przypadek użycia: zarządzanie użytkownikami.
Aktorzy: administrator.
Opis: administrator zarządza użytkownikami. Może zmienić imię, nazwisko, hasło, telefon, płeć, adres. Administrator może także dodać, lub usunąć użytkownika.

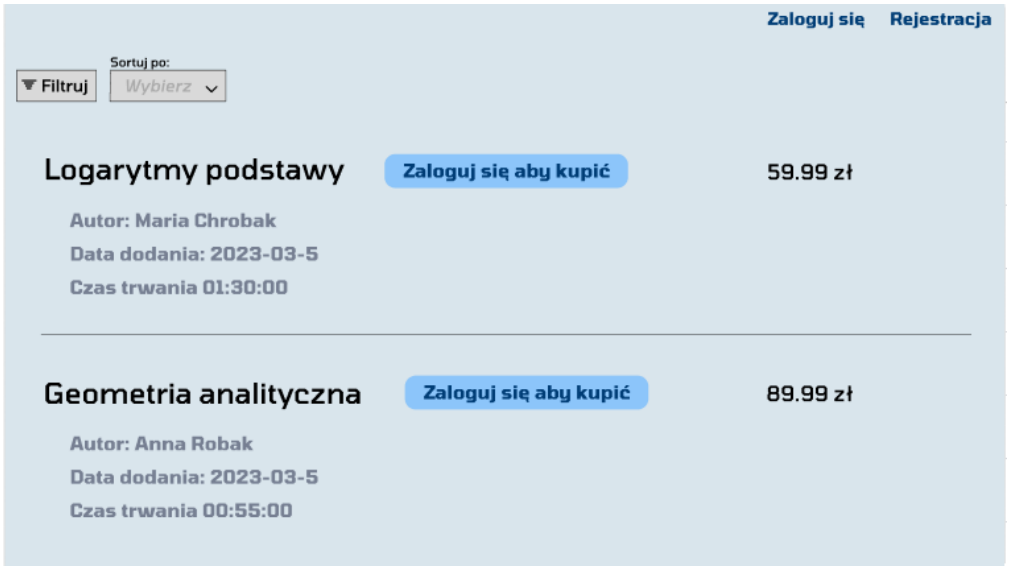
3. Makiety interfejsu aplikacji

Rozdział ten przedstawia makiety interfejsu aplikacji dla podstawowych stron aplikacji.

Na Rysunku 2 przedstawiono stronę główną, widoczną dla użytkownika niezalogowanego. Strona docelowo ma być bardzo prosta, z odnośnikami do głównych funkcji. Przycik *Kursy* kieruje do strony z listą oferowanych kursów (Rysunek 3). Przycik *Korepetycje* odpowiednio kieruje do strony z kalendarzem dostępnych korepetycji (Rysunek 4). Dodatkowo, użytkownik może się zalogować (przycisk *Zaloguj się*) lub zarejestrować (przycisk *Rejestracja*).

Rysunek 2: Makieta interfesju prezentująca stronę główną aplikacji.

Rysunek 3 prezentuje widok strony z listą i szczegółami oferowanych kursów dla użytkownika niezalogowanego. Użytkownik ma możliwość filtrowania listy wyników dzięki przyciskowi *Filtruj* (po autorze, cenie, czasie trwania, poziomie, tematyce), oraz sortowania (po dacie dodania i cenie) korzystając z przycisku *Sortuj po*. Każdy kurs posiada informacje o autorze, dacie dodania, czasie trwania i cenie. Przycisk *Zaloguj się aby kupić* przekierowuje użytkownika niezalogowanego do strony logowania.



Rysunek 3: Makieta interfesju prezentująca stronę z listą kursów.

Makieta na Rysunku 4 przedstawia widok kalendarza korepetytorów wraz z wolnymi terminami widoczny dla użytkownika niezalogowanego. Domyślnie wyświetlani są wszyscy korepetytorzy i ich wolne terminy. Kalendarz przedstawiony jest tygodniami, użytkownik może nawigować do innego tygodnia manipulując przyciskami znajdującymi się nad kalendarzem, w prawym górym rogu. Użytkownik może także wybrać konkretnego korepetytora dzięki przyciskowi *Wybierz korepetytora*. W celu rezerwacji terminu użytkownik musi się zalogować albo poprzez przycisk *Zaloguj się*, albo *Zaloguj się aby zarezerwować termin*. Terminy niedostępne, lub zarezerwowane nie są widoczne na kalendarzu.

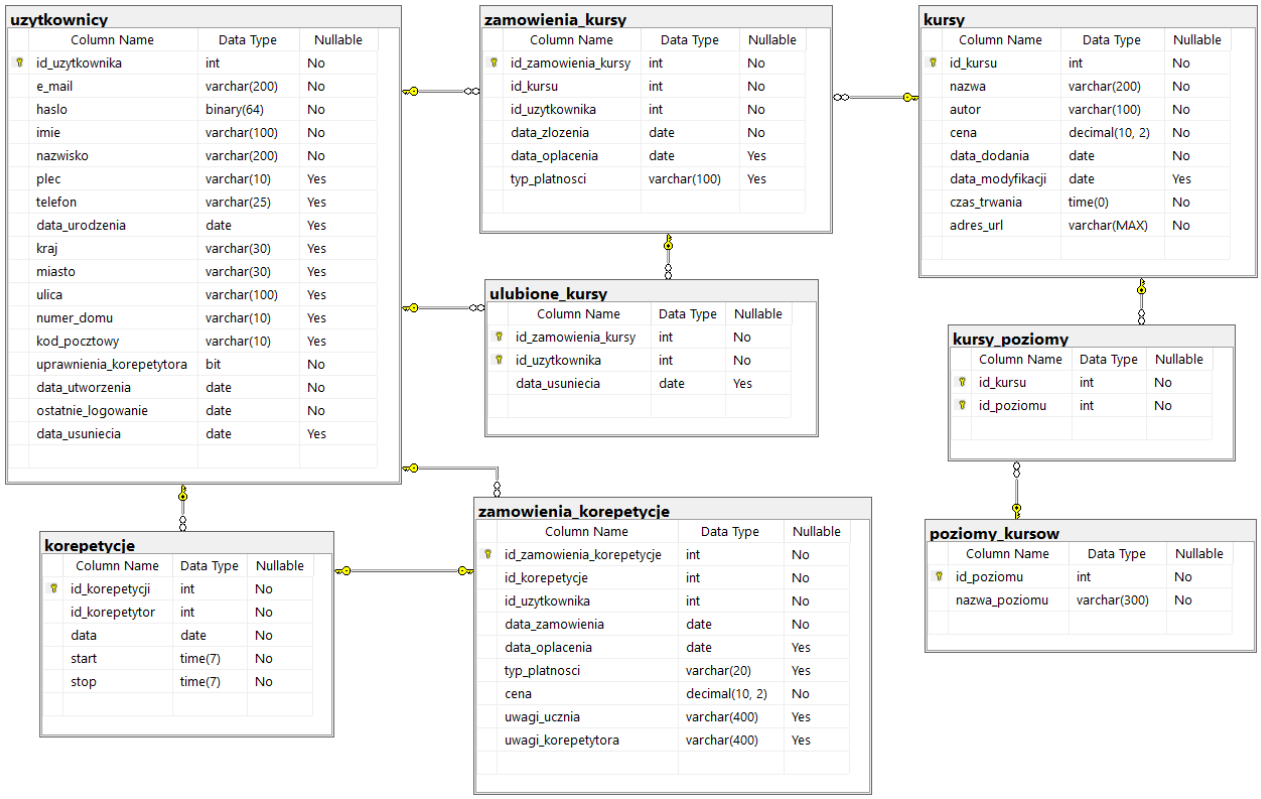
Rysunek 4: Makieta interfesju prezentująca stronę z kalendarzem korepetytorów.

4. Projekt bazy danych

Do zaprojektowania bazy danych wykorzystano *SQL Server* firmy Microsoft. Jest to system zarządzania relacyjnymi bazami danych (RDBMS), który dzięki wielu zaawansowanym funkcjonalnościom i narzędziom umożliwa tworzenie, przechowywanie, przetwarzanie i udostępnianie danych w sposób wydajny i bezpiczny [1].

4.1 Schemat

Na Rysunku 5 przedstawiono schemat bazy danych aplikacji na którym znajdują się informacje o tabelach: nazwy atrybutów, typy danych, czy atrybuty mogą mieć wartość *null* (wartość *null* oznacza brak wartości, lub nieokreśloną wartość dla danego pola). Na schemacie zaznaczono również relacje pomiędzy tabelami.



Rysunek 5: Schemat bazy danych aplikacji internetowej zaprojektowany przy użyciu systemu zarządzania bazami danych *SQL Server* firmy Microsoft [1].

4.2 Opis tabel

Poniżej znajdują się szczegółowe opisy tabel bazy danych aplikacji. Każdy opis składa się z czterech części: w pierwszej znajduje się tabela zawierająca nazwy atrybutów, typy danych, czy wartości mogą być null, oraz ewentualne komentarze. W kolejnej części opisane są warunki integralnościowe dla poszczególnych kolumn. Trzecia część przedstawia zrzut ekranu z systemu zarządzania bazą danych prezentujący przykładowe dane. Ostatnią część stanowi skrypt generujący całą tabelę.

4.2.1 uzytkownicy

uzytkownicy to tabela zawierająca szczegółowe informacje o zarejestrowanych użytkownikach aplikacji. Tabela 1, Tabela 2, Rysunek 6, oraz Skrypt 1 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_uzytkownika	int	nie	klucz główny
e_mail	varchar(200)	nie	-
haslo	binary(64)	nie	-
imie	varchar(100)	nie	-
nazwisko	varchar(200)	nie	-
plec	varchar(10)	tak	-
telefon	varchar(25)	tak	-
data_urodzenia	date	tak	-
kraj	varchar(30)	tak	-
miasto	varchar(30)	tak	-
ulica	varchar(100)	tak	-
numer_domu	varchar(10)	tak	-
kod_pocztowy	varchar(10)	tak	-
uprawnienia_korepetytora	bit	nie	-
data_utworzenia	date	nie	-
ostatnie_logowanie	date	nie	-
data_usuniecia	date	tak	-

Tabela 1: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *uzytkownicy*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
e_mail	UNIQUE	-
uprawnienia_korepetytora	DEFAULT	0
data_urodzenia	CHECK	data_urodzenia ⩽ getdate() AND data_urodzenia ⩾ dateadd(year,(-120),getdate())
data_utworzenia	CHECK	data_utworzenia ⩽ getdate()
data_usuniecia	CHECK	data_usuniecia ⩽ getdate()
kod_pocztowy	CHECK	LIKE '[0-9][0-9]-[0-9][0-9][0-9]'
ostatnie_logowanie	CHECK	ostatnie_logowanie ⩽ getdate()
telefon	CHECK	LIKE '[0-9][0-9][0-9][0-9][0-9] [0-9][0-9][0-9][0-9][0-9][0-9]'
plec	CHECK	plec IN ('Inna', 'Mezczyzna', 'Kobieta')

Tabela 2: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *uzytkownicy*.

id_uzytkownika	e_mail	haslo	imie	nazwisko	plec	telefon	data_urodzenia	kraj	miasto	ulica
1	mik@op.pl	0xDA23...	Waclaw	Ktos	Mezczyzna	048550098345	1993-01-03	Francja	Kraków	Zakopianska
2	oleks@o2.pl	0x90F2...	Aleksander	Somanski	NULL	048345621235	1988-03-05	Polska	Zagrzeb	Patryzantów
3	izu@l2.pl	0xF41F...	Izabella	Ulezanska	Inna	NULL	1997-01-08	NULL	NULL	NULL
4	vers@o2.pl	0x891B...	Kornel	Mazurek	Mezczyzna	NULL	NULL	NULL	NULL	NULL
5	lukrecjaona...	0x3E32...	Kamila	Piotrowska	Kobieta	048660631235	1982-12-12	Polska	Krakow	Okulickiego
6	stanislawse...	0x8A84...	Stanislaw	Sekielski	NULL	NULL	NULL	NULL	NULL	NULL
7	joziazuzia@...	0xCB09...	Zuzanna	Ogorska	Inna	NULL	NULL	NULL	NULL	NULL
8	pascal123...	0xBD23...	Pawel	Kalinowski	Mezczyzna	048587698345	1993-09-03	Polska	Warszawa	Partyzantów
9	toni@gmail....	0x7D11...	Toni	Loisko	NULL	NULL	1999-09-03	Polska	Warszawa	Akacjowa
10	zosia@gmai...	0x7A19...	Zofia	Macanko	NULL	NULL	NULL	NULL	NULL	NULL
11	iga@gmail.c...	0x109E...	Iga	Figa	NULL	NULL	1989-09-03	Polska	Praga	Zielona
13	ula@gmail.c...	0x109E...	Iga	Figa	NULL	NULL	1989-09-03	Polska	Praga	Zielona
14	mm@op.pl	0xCF83...	Monika	Jakas	Kobieta	NULL	NULL	NULL	NULL	Zala

numer_domu	kod_pocztowy	uprawnienia_korepetytora	data_utworzenia	ostatnie_logowanie	data_usuniecia
125B	30-978	0	2023-05-04	2023-05-04	NULL
23B/4	44-938	1	2023-05-04	2023-05-04	NULL
NULL	NULL	0	2023-05-04	2023-05-04	NULL
NULL	NULL	1	2023-04-04	2023-05-04	2023-05-04
125B/44	30-578	1	2023-05-04	2023-05-04	NULL
NULL	NULL	0	2023-05-04	2023-05-04	2023-05-04
NULL	NULL	1	2023-05-04	2023-05-04	NULL
3/65H	80-078	0	2023-05-04	2023-05-04	2023-05-04
3/6l	10-079	0	2023-05-06	2023-05-06	NULL
NULL	NULL	0	2023-05-06	2023-05-06	NULL
3B	66-777	0	2023-05-06	2023-05-06	NULL
3B	66-777	0	2023-05-06	2023-05-06	2023-05-06
NULL	NULL	0	2023-05-10	2023-05-10	NULL

Rysunek 6: Przykładowe dane pochodzące z tabeli *uzytkownicy*.

```
CREATE TABLE [dbo].[uzytkownicy](
        [id_uzytkownika] [int] IDENTITY(1,1) NOT NULL,
        [e_mail] [varchar](200) NOT NULL,
        [haslo] [binary](64) NOT NULL,
        [imie] [varchar](100) NOT NULL,
        [nazwisko] [varchar](200) NOT NULL,
        [plec] [varchar](10) NULL,
        [telefon] [varchar](25) NULL,
        [data_urodzenia] [date] NULL,
        [kraj] [varchar](30) NULL,
        [miasto] [varchar](30) NULL,
        [ulica] [varchar](100) NULL,
        [numer_domu] [varchar](10) NULL,
        [kod_pocztowy] [varchar](10) NULL,
        [uprawnienia_korepetytora] [bit] NOT NULL,
        [data_utworzenia] [date] NOT NULL,
        [ostatnie_logowanie] [date] NOT NULL,
        [data_usuniecia] [date] NULL,
CONSTRAINT [PK_uzytkownicy] PRIMARY KEY CLUSTERED
(
    [id_uzytkownika] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [e_mail] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]

ALTER TABLE [dbo].[uzytkownicy] ADD DEFAULT ((0)) FOR
[uprawnienia_korepetytora]

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([data_urodzenia]<=getdate() AND
[data_urodzenia]>=dateadd(year,(-120),getdate())))

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([data_utworzenia]<=getdate()))
```

```
ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([data_usuniecia]<=getdate()))

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([kod_pocztowy] like '[0-9][0-9]-[0-9][0-9][0-9]'))

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([ostatnie_logowanie]<=getdate()))

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([telefon] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([plec]='Inna' OR [plec]='Mezczyzna' OR [plec]='Kobieta'))
```

Skrypt 1: Skrypt generujący tabelę *uzytkownicy*.

4.2.2 kursy

kursy to tabela zawierająca szczegółowe informacje o oferowanych kursach. Tabela 3, Tabela 4, Rysunek 7, oraz Skrypt 2 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_kursu	int	nie	klucz główny
nazwa	varchar(200)	nie	-
autor	varchar(100)	nie	-
cena	decimal(10,2)	nie	-
data_dodania	date	nie	-
data_modyfikacji	date	nie	-
czas_trwania	time(0)	nie	-
adres_url	varchar(MAX)	nie	-

Tabela 3: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *kursy*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
nazwa	UNIQUE	-
data_dodania	CHECK	data_dodania ⩽ getdate()
data_modyfikacji	CHECK	data_dodania ⩽ getdate()
adres_url	CHECK	LIKE 'https://%'

Tabela 4: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *kursy*.

id_kursu	nazwa	autor	cena	data_dodania	data_modyfikacji	czas_trwania	adres_url
1	Logarytmy podstawy	Maria Chrobak	59.99	2023-03-05	2023-05-05	01:30:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ
2	Ulamki	Anna Robak	25.99	2023-02-05	NULL	00:55:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ
3	Geometria analityczna	Maria Chrobak	88.99	2023-01-05	2023-04-05	02:00:00	https://maria-https://www.youtube.com/watch?v=dQ...
4	Procenty	Marian Zora	55.55	2023-05-05	NULL	01:14:15	https://marian-https://www.youtube.com/watch?v=d...
5	Funkcja kwadratorwa	Maria Chrobak	33.99	2023-05-05	NULL	00:45:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ
6	Równania	Zofia Gut	78.90	2023-05-06	NULL	01:15:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ
7	Funkcje	Zofia Gut	50.00	2023-05-06	2023-05-06	00:35:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ
11	Zadania tekstowe	Maria Chrobak	25.00	2023-05-05	2023-05-06	01:30:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ

Rysunek 7: Przykładowe dane pochodzące z tabeli *kursy*.

```
CREATE TABLE [dbo].[kursy](
        [id_kursu] [int] IDENTITY(1,1) NOT NULL,
        [nazwa] [varchar](200) NOT NULL,
        [autor] [varchar](100) NOT NULL,
        [cena] [decimal](10, 2) NOT NULL,
        [data_dodania] [date] NOT NULL,
        [data_modyfikacji] [date] NULL,
        [czas_trwania] [time](0) NOT NULL,
        [adres_url] [varchar](max) NOT NULL,
CONSTRAINT [PK_kursu] PRIMARY KEY CLUSTERED
(
    [id_kursu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [nazwa] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
```

```
 ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]

ALTER TABLE [dbo].[kursy]  WITH CHECK ADD CHECK
(([data_dodania]<=getdate()))

ALTER TABLE [dbo].[kursy]  WITH CHECK ADD CHECK
(([data_modyfikacji]<=getdate()))

ALTER TABLE [dbo].[kursy]  WITH CHECK ADD  CONSTRAINT
[sprawdzenie_url] CHECK  (([adres_url] like 'https://%'))

ALTER TABLE [dbo].[kursy] CHECK CONSTRAINT [sprawdzenie_url]
```

Skrypt 2: Skrypt generujący tabelę *kursy*.

4.2.3 poziomy_kursow

poziomy_kursow to tabela zawierająca informacje o poziomach nauczania związanych z kursami (np. I LO podstawa, II LO rozszerzenie, matura rozszerzenie). Tabela 5, Tabela 6, Rysunek 8, oraz Skrypt 3 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_poziomu	int	nie	klucz główny
nazwa_poziomu	varchar(300)	nie	-

Tabela 5: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *poziomy_kursow*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
nazwa_poziomu	UNIQUE	-

Tabela 6: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *poziomy_kursow*.

id_poziomu	nazwa_poziomu
1	I LO podstawa
2	II LO podstawa
3	III LO podstawa
4	IV LO podstawa
6	I LO rozszerzenie
7	II LO rozszerzenie
8	III LO rozszerzenie
9	IV LO rozszerzenie
10	V SP
11	VI SP
12	VII SP
13	VIII SP
14	Egzamin 8
15	Matura podstawa

Rysunek 8: Przykładowe dane pochodzące z tabeli *poziomy_kursow*.

```
CREATE TABLE [dbo].[poziomy_kursow](
        [id_poziomu] [int] IDENTITY(1,1) NOT NULL,
        [nazwa_poziomu] [varchar](300) NOT NULL,
CONSTRAINT [PK_kursy_poziomy] PRIMARY KEY CLUSTERED
(
    [id_poziomu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [nazwa_poziomu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
       ON [PRIMARY]
) ON [PRIMARY]
```

Skrypt 3: Skrypt generujący tabelę *poziomy_kursow*.

4.2.4 kursy_poziomy

kurs_poziomy to tabela łącznikowa, łącząca tabelę *kursy* z tablą *poziomy_kursow*. Każdy kurs może należeć do wielu poziomów, oraz każdy poziom może wiązać się z wieloma kursami. Tabela 7, Tabela 8, Rysunek 9, oraz Skrypt 4 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_poziomu	int	nie	klucz główny, klucz obcy - referencja do tabeli *poziomy_kursow*
id_kursu	int	nie	klucz główny, klucz obcy - referencja do tabeli *kursy*

Tabela 7: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *kursy_poziomy*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
id_kursu	FOREIGN KEY	REFERENCES [kursy] ([id_kursu])
id_poziomu	FOREIGN KEY	REFERENCES [poziomy_kursow] ([id_poziomu])

Tabela 8: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *kursy_poziomy*.

id_kursu	id_poziomu
1	1
1	6
1	15
2	10
2	11
2	12
2	15
3	4

Rysunek 9: Przykładowe dane pochodzące z tabeli *kursy_poziomy*.

```
CREATE TABLE [dbo].[kursy_poziomy](
        [id_kursu] [int] NOT NULL,
        [id_poziomu] [int] NOT NULL,
 CONSTRAINT [PK_kursy_poziomyy] PRIMARY KEY CLUSTERED
(
    [id_kursu] ASC,
    [id_poziomu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]

ALTER TABLE [dbo].[kursy_poziomy] WITH CHECK ADD CONSTRAINT
[FK_kursy_poziomy_kursy] FOREIGN KEY([id_kursu])
REFERENCES [dbo].[kursy] ([id_kursu])

ALTER TABLE [dbo].[kursy_poziomy] CHECK CONSTRAINT
[FK_kursy_poziomy_kursy]

ALTER TABLE [dbo].[kursy_poziomy] WITH CHECK ADD CONSTRAINT
[FK_kursy_poziomy_poziomy_kursow] FOREIGN KEY([id_poziomu])
REFERENCES [dbo].[poziomy_kursow] ([id_poziomu])

ALTER TABLE [dbo].[kursy_poziomy] CHECK CONSTRAINT
[FK_kursy_poziomy_poziomy_kursow]
```

Skrypt 4: Skrypt generujący tabelę *kursy_poziomy*.

4.2.5 zamowienia_kursy

zamowienia_kursy to tabela zawierająca szczegółowe informacje o zamówionych kursach przez użytkowników aplikacji. Tabela 9, Tabela 10, Rysunek 10, oraz Skrypt 5 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_zamowienia_kursy	int	nie	klucz główny
id_kursu	int	nie	klucz główny, klucz obcy - referencja do tabeli *kursy*
id_uzytkownika	int	nie	klucz główny, klucz obcy - referencja do tabeli *uzytkownicy*
data_zlozenia	date	nie	-
data_oplacenia	date	tak	-
typ_platnosci	varchar(100)	tak	-

Tabela 9: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *zamowienia_kursy*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
id_kursu	FOREIGN KEY	REFERENCES [kursy] ([id_kursu])
id_uzytkownika	FOREIGN KEY	REFERENCES [uzytkownicy] ([id_uzytkownika])
data_zlozenia	CHECK	data_zlozenia ⩽ getdate()
data_oplacenia	CHECK	data_oplacenia ⩽ getdate()
typ_platnosci	CHECK	IN ('BLIK', 'PayU', 'przelew tradycyjny')

Tabela 10: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *zamowienia_kursy*.

id_zamowienia_kursy	id_kursu	id_uzytkownika	data_zlozenia	data_oplacenia	typ_platnosci
1	2	2	2023-05-05	NULL	NULL
2	4	1	2023-05-05	2023-05-05	Blik
3	5	6	2023-05-05	2023-05-05	PayU
4	2	7	2023-05-05	NULL	NULL
5	4	8	2023-05-05	2023-05-05	Blik
6	4	2	2023-05-05	2023-05-05	Blik
8	2	8	2023-05-06	2023-05-05	NULL
26	1	23	2023-05-14	2023-05-14	Blik
27	2	23	2023-05-14	2023-05-14	Blik
28	1	25	2023-05-14	2023-05-14	Blik
34	1	26	2023-05-14	NULL	NULL
35	2	26	2023-05-14	2023-05-14	Blik

Rysunek 10: Przykładowe dane pochodzące z tabeli *zamowienia_kursy*.

```
CREATE TABLE [dbo].[zamowienia_kursy](
        [id_zamowienia_kursy] [int] IDENTITY(1,1) NOT NULL,
        [id_kursu] [int] NOT NULL,
        [id_uzytkownika] [int] NOT NULL,
        [data_zlozenia] [date] NOT NULL,
        [data_oplacenia] [date] NULL,
        [typ_platnosci] [varchar](100) NULL,
 CONSTRAINT [PK_zamowienia_kursy] PRIMARY KEY CLUSTERED
(
    [id_zamowienia_kursy] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]

ALTER TABLE [dbo].[zamowienia_kursy]  WITH CHECK ADD CONSTRAINT
[FK_zamowienia_kursy_kursy] FOREIGN KEY([id_kursu])
REFERENCES [dbo].[kursy] ([id_kursu])

ALTER TABLE [dbo].[zamowienia_kursy] CHECK CONSTRAINT
[FK_zamowienia_kursy_kursy]

ALTER TABLE [dbo].[zamowienia_kursy] WITH CHECK ADD CONSTRAINT
[FK_zamowienia_kursy_uzytkownicy] FOREIGN KEY([id_uzytkownika])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])

ALTER TABLE [dbo].[zamowienia_kursy] CHECK CONSTRAINT
[FK_zamowienia_kursy_uzytkownicy]

ALTER TABLE [dbo].[zamowienia_kursy] WITH CHECK ADD CHECK
(([data_zlozenia]<=getdate()))

ALTER TABLE [dbo].[zamowienia_kursy] WITH CHECK ADD CHECK
(([data_oplacenia]<=getdate()))

ALTER TABLE [dbo].[zamowienia_kursy] WITH CHECK ADD CHECK
(([typ_platnosci]='BLIK' OR [typ_platnosci]='PayU'
OR [typ_platnosci]='przelew tradycyjny'))
```

Skrypt 5: Skrypt generujący tabelę *zamowienia_kursy*.

4.2.6 ulubione_kursy

ulubione_kursy to tabela łącznikowa, łącząca tabelę *zamowienia_kursy* z tabelą *uzytkownicy*. Każdy zakupiony kurs może być polubiony przez wielu użytkowników i każdy użytkownik może dodać do ulubionych wiele zakupionych kursów. Tabela 11, Tabela 12, Rysunek 11, oraz Skrypt 6 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_zamowienia_kursy	int	nie	klucz główny, klucz obcy - referencja do tabeli *zamowienia_kursy*
id_uzytkownika	int	nie	klucz główny, klucz obcy - referencja do tabeli *uzytkownicy*

Tabela 11: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *ulubione_kursy*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
id_zamowienia_kursy	FOREIGN KEY	REFERENCES .[zamowienia_kursy] ([id_zamowienia_kursy])
id_uzytkownika	FOREIGN KEY	REFERENCES [uzytkownicy] ([id_uzytkownika])

Tabela 12: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *ulubione_kursy*.

id_zamowienia_kursy	id_uzytkownika	ytkownika
1	2	
4	7	
5	8	
6	2	
8	8	
26	23	

Rysunek 11: Przykładowe dane pochodzące z tabeli *ulubione_kursy*.

```
CREATE TABLE [dbo].[ulubione_kursy](
        [id_zamowienia_kursy] [int] NOT NULL,
        [id_uzytkownika] [int] NOT NULL,
 CONSTRAINT [PK_ulubione_kursy] PRIMARY KEY CLUSTERED
(
    [id_zamowienia_kursy] ASC,
    [id_uzytkownika] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
       ON [PRIMARY]
) ON [PRIMARY]

ALTER TABLE [dbo].[ulubione_kursy] WITH CHECK ADD CONSTRAINT
[FK_ulubione_kursy_uzytkownicy] FOREIGN KEY([id_uzytkownika])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])

ALTER TABLE [dbo].[ulubione_kursy] CHECK CONSTRAINT
[FK_ulubione_kursy_uzytkownicy]

ALTER TABLE [dbo].[ulubione_kursy] WITH CHECK ADD CONSTRAINT
[FK_ulubione_kursy_zamowienia_kursy]
FOREIGN KEY ([id_zamowienia_kursy])
REFERENCES [dbo].[zamowienia_kursy] ([id_zamowienia_kursy])

ALTER TABLE [dbo].[ulubione_kursy] CHECK CONSTRAINT
[FK_ulubione_kursy_zamowienia_kursy]
```

Skrypt 6: Skrypt generujący tabelę *ulubione_kursy*.

4.2.7 korepetycje

korepetycje to tabela zawierająca szczegółowe informacje o korepetycjach. Tabela 13, Tabela 14, Rysunek 12, oraz Skrypt 7 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_korepetycji	int	nie	klucz główny
id_korepetytor	int	nie	klucz obcy - referencja do tabeli *uzytkownicy*
data	date	nie	-
start	time(7)	nie	początek czasu (godzina)
stop	time(7)	nie	koniec czasu (godzina)

Tabela 13: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *korepetycje*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
id_korepetytor	FOREIGN KEY	REFERENCES [uzytkownicy] ([id_korepetytor])

Tabela 14: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *korepetycje*.

id_korepetycji	id_korepetytor	data	start	stop
1	2	2023-06-05	09:00:00.0000000	10:00:00.0000000
2	2	2023-05-05	10:00:00.0000000	11:00:00.0000000
3	2	2023-05-05	11:30:00.0000000	12:30:00.0000000
5	5	2023-05-06	13:00:00.0000000	14:00:00.0000000
6	4	2023-05-07	09:00:00.0000000	10:00:00.0000000
7	4	2023-05-05	10:00:00.0000000	11:00:00.0000000
8	4	2023-05-05	11:00:00.0000000	12:00:00.0000000
9	5	2023-05-05	12:00:00.0000000	13:00:00.0000000

Rysunek 12: Przykładowe dane pochodzące z tabeli *korepetycje*.

```
CREATE TABLE [dbo].[korepetycje](
        [id_korepetycji] [int] IDENTITY(1,1) NOT NULL,
        [id_korepetytor] [int] NOT NULL,
        [data] [date] NOT NULL,
        [start] [time](7) NOT NULL,
        [stop] [time](7) NOT NULL,
 CONSTRAINT [PK_korepetycje] PRIMARY KEY CLUSTERED
(
    [id_korepetycji] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
       ON [PRIMARY]
) ON [PRIMARY]

ALTER TABLE [dbo].[korepetycje] WITH CHECK ADD CONSTRAINT
[FK_korepetycje_uzytkownicy] FOREIGN KEY([id_korepetytor])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])

ALTER TABLE [dbo].[korepetycje] CHECK CONSTRAINT
[FK_korepetycje_uzytkownicy]
```

Skrypt 7: Skrypt generujący tabelę *korepetycje*.

4.2.8 zamowienia_korepetycje

zamowienia_korepetycje to tabela zawierająca szczegółowe informacje o zamówionych korepetycjach. Tabela 15, Tabela 16, Rysunek 13, oraz Skrypt 8 zawierają informacje opisane w sekcji 4.2.

Nazwa atrybutu	Typ danych	Nullable	Komentarz
id_zamowienia_korepetycje	int	nie	klucz główny
id_korepetycje	int	nie	klucz obcy - referencja do tabeli *korepetycje*
id_uzytkownika	int	nie	klucz obcy - referencja do tabeli *uzytkownicy*
data_zamowienia	date	nie	-
data_oplacenia	date	tak	-
typ_platnosci	varchar(20)	tak	-
cena	decimal(10,2)	nie	-
uwagi_ucznia	varchar(400)	tak	-
uwagi_korepetytora	varchar(400)	tak	-

Tabela 15: Nazwy atrybutów, typy danych, wartości będące *nullable* i komentarze tabeli *zamowienia_korepetycje*.

Nazwa atrybutu	Rodzaj warunku	Wartość warunku
id_korepetycje	FOREIGN KEY	REFERENCES [korepetycje] ([id_korepetycje])
id_korepetycje	UNIQUE	-
id_uzytkownika	FOREIGN KEY	REFERENCES [uzytkownicy] ([id_uzytkownika])
data_zamowienia	CHECK	data_zamowienia ⩽ getdate()
data_oplacenia	CHECK	data_oplacenia ⩽ getdate()
typ_platnosci	CHECK	IN ('BLIK', 'PayU', 'przelew tradycyjny')

Tabela 16: Warunki integralnościowe nałożone na poszczególne kolumny tabeli *zamowienia_korepetycje*.

id_zamowienia_korepetycje	id_korepetycje	id_uzytkownika	data_zamowienia	data_oplacenia	typ_platnosci	cena	uwagi_ucznia	uwagi_korepetytora
1	1	1	2023-05-05	2023-05-05	Blik	60.00	braki w procentach	NULL
2	2	3	2023-05-04	NULL	PayU	60.00	NULL	NULL
3	4	3	2023-05-05	2023-05-05	przelew tradycyjny	80.00	matura rozszerzon	NULL
5	11	8	2023-05-02	2023-05-04	Blik	120.00	geometria rozszerzona	NULL
6	10	6	2023-05-01	NULL	przelew tradycyjny	90.00	NULL	procenty
8	3	3	2023-05-06	NULL	NULL	90.00	geometria	NULL
9	5	8	2023-05-06	NULL	Blik	120.00	NULL	NULL

Rysunek 13: Przykładowe dane pochodzące z tabeli *zamowienia_korepetycje*.

```
CREATE TABLE [dbo].[zamowienia_korepetycje](
        [id_zamowienia_korepetycje] [int] IDENTITY(1,1) NOT
   NULL,
        [id_korepetycje] [int] NOT NULL,
        [id_uzytkownika] [int] NOT NULL,
        [data_zamowienia] [date] NOT NULL,
        [data_oplacenia] [date] NULL,
        [typ_platnosci] [varchar](20) NULL,
        [cena] [decimal](10, 2) NOT NULL,
        [uwagi_ucznia] [varchar](400) NULL,
        [uwagi_korepetytora] [varchar](400) NULL,
 CONSTRAINT [PK_zamowienia_korepetycje] PRIMARY KEY CLUSTERED
(
    [id_zamowienia_korepetycje] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
       ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [id_korepetycje] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
       ON [PRIMARY]
) ON [PRIMARY]

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CONSTRAINT [FK_zamowienia_korepetycje_korepetycje]
FOREIGN KEY ([id_zamowienia_korepetycje])
REFERENCES [dbo].[korepetycje] ([id_korepetycji])

ALTER TABLE [dbo].[zamowienia_korepetycje] CHECK CONSTRAINT
[FK_zamowienia_korepetycje_korepetycje]

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CONSTRAINT [FK_zamowienia_korepetycje_uzytkownicy]
FOREIGN KEY([id_uzytkownika])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])

ALTER TABLE [dbo].[zamowienia_korepetycje]
CHECK CONSTRAINT [FK_zamowienia_korepetycje_uzytkownicy]
```

```
ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CHECK (([data_zamowienia]<=getdate()))

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CHECK (([data_oplacenia]<=getdate()))

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CHECK (([typ_platnosci]='BLIK' OR [typ_platnosci]='PayU'
OR [typ_platnosci]='przelew tradycyjny'))
```

Skrypt 8: Skrypt generujący tabelę *zamowienia_korepetycje*.

5. Warstwy dostępu do danych

5.1 Widoki

Widok to mechanizm pozwalający na utworzenie wirtualnej tabeli złożonej z kolumn i wierszy z innych tabel lub innych widoków, zdefiniowanych za pomocą instrukcji SELECT. Widoki są przydatne w sytuacjach, gdy często wykonywane są złożone zapytania składające się z wielu tabel. Pozwalają one na uproszczenie zapytań i łatwiejszą obsługę danych [6].

Poniżej przedstawione są widoki utworzone dla bazy danych aplikacji. Do każdego widoku dołączony jest skrypt generujący widok, oraz przykładowe działanie.

- Widok kursów i poziomów ***Vkursy_i_poziomy*** (Skrypt 9) - zawiera informacje o nazwach poziomów kursów.

```
CREATE VIEW Vkursy_i_poziomy AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy AS k
JOIN kursy_poziomy AS kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow AS p ON kp.id_poziomu = pk.id_poziomu
```

Skrypt 9: Skrypt generujący widok *Vkursy_i_poziomy*.

przykład działania widoku:

id_kursu	nazwa	nazwa_poziomu
5	Funkcja kwadratorwa	II LO podstawa
5	Funkcja kwadratorwa	Matura podstawa
7	Funkcje	I LO rozszerzenie
3	Geometria analityczna	IV LO podstawa
3	Geometria analityczna	IV LO rozszerzenie
3	Geometria analityczna	Matura podstawa
3	Geometria analityczna	Matura rozszerzenie
1	Logarytmy podstawy	I LO podstawa
1	Logarytmy podstawy	I LO rozszerzenie
1	Logarytmy podstawy	Matura podstawa

Rysunek 14: Przykładowe dane wygenerowane za pomocą widoku *Vkursy_i_poziomy*.

- Widok kursów o poziomie podstawowym ***Vkursy_podstawa*** (Skrypt 10) - przedstawia nazwy kursów i ich poziomy, które w swojej nazwie zawierają słowo *podstawa* dla liceum.

```
CREATE VIEW Vkursy_podstawa AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy AS k
JOIN kursy_poziomy AS kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow AS p ON kp.id_poziomu = p.id_poziomu
WHERE p.nazwa_poziomu LIKE '%podstawa%'
```

Skrypt 10: Skrypt generujący widok *Vkursy_podstawa*.

przykład działania widoku:

id_kursu	nazwa	nazwa_poziomu
1	Logarytmy podstawy	I LO podstawa
1	Logarytmy podstawy	Matura podstawa
2	Ulamki	Matura podstawa
3	Geometria analityczna	IV LO podstawa
3	Geometria analityczna	Matura podstawa
4	Procenty	Matura podstawa
5	Funkcja kwadratorwa	II LO podstawa

Rysunek 15: Przykładowe dane wygenerowane za pomocą widoku *Vkursy_podstawa*.

- Widok kursów o poziomie rozszerzonym ***Vkursy_rozszerzenie*** (Skrypt 11) - przedstawia nazwy kursów i ich poziomy, które w swojej nazwie zawierają słowo *rozszerzenie* dla liceum.

```
CREATE VIEW Vkursy_rozszerzenie AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy AS k
JOIN kursy_poziomy AS kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow AS p ON kp.id_poziomu = p.id_poziomu
WHERE p.nazwa_poziomu LIKE '%rozszerzenie%'
```

Skrypt 11: Skrypt generujący widok *Vkursy_rozszerzenie*.

przykład działania widoku:

id_kursu	nazwa	nazwa_poziomu
1	Logarytmy podstawy	I LO rozszerzenie
3	Geometria analityczna	IV LO rozszerzenie
3	Geometria analityczna	Matura rozszerzenie
6	Równania	I LO rozszerzenie

Rysunek 16: Przykładowe dane wygenerowane za pomocą widoku *Vkursy_rozszerzenie*.

- Widok kursów dla szkoły podstawowej ***Vkursy_szkola_podstawowa*** (Skrypt 12) - przedstawia nazwy kursów i ich poziomy, które w swojej nazwie zawierają słowo *SP* lub *Egzamin 8* dla szkoły podstawowej.

```
CREATE VIEW Vkursy_szkola_podstawowa AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy AS k
JOIN kursy_poziomy AS kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow AS p ON kp.id_poziomu = p.id_poziomu
WHERE p.nazwa_poziomu LIKE '%SP%'
      OR p.nazwa_poziomu LIKE '%Egzamin 8%'
```

Skrypt 12: Skrypt generujący widok *Vkursy_szkola_podstawowa*.

przykład działania widoku:

id_kursu	nazwa	nazwa_poziomu
2	Ulamki	V SP
2	Ulamki	VI SP
2	Ulamki	VII SP
4	Procenty	VIII SP

Rysunek 17: Przykładowe dane wygenerowane za pomocą widoku *Vkursy_szkola_podstawowa*.

- Widok aktywnych użytkowników ***Vaktywni_uzytkownicy*** (Skrypt 13) - przedstawia zarejestrowanych użytkowników aplikacji, których konto nie zostało usunięte i nie mają uprawnień korepetytora.

```
CREATE VIEW Vaktywni_uzytkownicy AS
SELECT id_uzytkownika, imie, nazwisko, telefon,
        data_urodzenia, kraj, miasto, ulica, numer_domu,
        kod_pocztowy, data_utworzenia, ostatnie_logowanie
FROM uzytkownicy
WHERE data_usuniecia IS NULL AND uprawnienia_korepetytora=0
```

Skrypt 13: Skrypt generujący widok *Vaktywni_uzytkownicy*.

przykład działania widoku:

id_uzytkownika	imie	nazwisko	telefon	data_urodzenia	kraj	miasto	ulica	numer_domu	kod_pocztowy	data_utworzenia	ostatnie_logowanie
1	Waclaw	Ktos	048550098345	1993-01-03	Francja	Kraków	Zakopianska	125B	30-978	2023-05-04	2023-05-04
3	Izabella	Ulezanska	NULL	1997-01-08	NULL	NULL	NULL	NULL	NULL	2023-05-04	2023-05-04
9	Toni	Loisko	NULL	1999-09-03	Polska	Warszawa	Akacjowa	3/6l	10-079	2023-05-06	2023-05-06
10	Zofia	Macanko	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-06	2023-05-06
11	Iga	Figa	NULL	1989-09-03	Polska	Praga	Zielona	3B	66-777	2023-05-06	2023-05-06
14	Monika	Jakas	NULL	NULL	NULL	NULL	Zala	NULL	NULL	2023-05-10	2023-05-10
17	Kodek	Rodek	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-11	2023-05-11
18	Korek	Worek	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-11	2023-05-11
19	Iza	Ryza	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-11	2023-05-11
20	Róza	Zimna	048556678345	1993-04-05	Polska	Kraków	Moja	123/2	34-700	2023-05-11	2023-05-11

Rysunek 18: Przykładowe dane wygenerowane za pomocą widoku *Vaktywni_uzytkownicy*.

- Widok nieaktywnych użytkowników ***Vnieaktywni_uzytkownicy*** (Skrypt 14) - przedstawia zarejestrowanych użytkowników aplikacji, których konto zostało usunięte i nie mają uprawnień korepetytora.

```
CREATE VIEW Vnieaktywni_uzytkownicy AS
SELECT id_uzytkownika, imie, nazwisko, telefon,
   data_urodzenia,
       kraj, miasto, ulica, numer_domu, kod_pocztowy,
       data_utworzenia, ostatnie_logowanie, data_usuniecia
FROM uzytkownicy
WHERE data_usuniecia IS NOT NULL
    AND uprawnienia_korepetytora = 0
```

Skrypt 14: Skrypt generujący widok *Vnieaktywni_uzytkownicy*.

przykład działania widoku:

id_uzytkownika	imie	nazwisko	telefon	data_urodzenia	kraj	miasto	ulica	numer_domu	kod_pocztowy	data_utworzenia	ostatnie_logowanie	data_usuniecia
6	Stanislaw	Sekielski	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-04	2023-05-04	2023-05-04
8	Pawel	Kalinowski	048587698345	1993-09-03	Polska	Warszawa	Partyzantów	3/65H	80-078	2023-05-04	2023-05-04	2023-05-04
13	Iga	Figa	NULL	1989-09-03	Polska	Praga	Zielona	3B	66-777	2023-05-06	2023-05-06	2023-05-06
25	Iza	Ryza	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-14	2023-05-14	2023-05-15
26	Iza	Ryza	NULL	NULL	NULL	NULL	NULL	55g	NULL	2023-05-14	2023-05-14	2023-05-14
27	Ola	Kapik	048776654365	1993-06-07	Polska	Kraków	NULL	34	30-798	2023-05-14	2023-05-14	2023-05-13
28	Kondrad	Skot	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-15	2023-05-15	2023-05-15
29	Konrad	Sok	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-15	2023-05-15	2023-05-15

Rysunek 19: Przykładowe dane wygenerowane za pomocą widoku *Vnieaktywni_uzytkownicy*.

- Widok korepetytorów ***Vkorepetytorzy*** (Skrypt 15) - przedstawia użytkowników, których konto nie zostało usunięte i mają uprawnienia korepetytora.

```
CREATE VIEW Vkorepetytorzy AS
SELECT id_uzytkownika, imie, nazwisko, telefon,
       data_urodzenia, kraj, miasto, ulica,
       numer_domu, kod_pocztowy, data_utworzenia,
       ostatnie_logowanie
FROM uzytkownicy
WHERE data_usuniecia IS NULL AND uprawnienia_korepetytora=1
```

Skrypt 15: Skrypt generujący widok *Vkorepetytorzy*.

przykład działania widoku:

id_uzytkownika	imie	nazwisko	telefon	data_urodzenia	kraj	miasto	ulica	numer_domu	kod_pocztowy	data_utworzenia	ostatnie_logowanie	data_usuniecia
2	Aleksander	Somanski	048345621235	1988-03-05	Polska	Zagrzeb	Patryzantów	23B/4	44-938	2023-05-04	2023-05-04	2023-05-04
5	Kamila	Piotrowska	048660631235	1982-12-12	Polska	Krakow	Okulickiego	125B/44	30-578	2023-05-04	2023-05-04	2023-05-04
7	Zuzanna	Ogorska	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-04	2023-05-04	2023-05-06
18	Korek	Worek	NULL	NULL	NULL	NULL	NULL	NULL	NULL	2023-05-11	2023-05-11	2023-05-15
20	Róza	Zimna	048556678345	1993-04-05	Polska	Kraków	Moja	123/2	34-700	2023-05-11	2023-05-11	2023-05-14

Rysunek 20: Przykładowe dane wygenerowane za pomocą widoku *Vkorepetytorzy*.

- Widok czasu trwania korepetycji ***Vczas_korepetycji*** (Skrypt 16) - przedstawia czas trwania korepetycji w danym terminie dla konkretnego korepetytora wyrażony w minutach.

```
CREATE VIEW Vczas_korepetycji AS
SELECT id_korepetycji, id_korepetytor, data, start, stop,
       DATEDIFF(MINUTE, start, stop) AS czas_trwania
FROM korepetycje
```

Skrypt 16: Skrypt generujący widok *Vczas_korepetycji*.

przykład działania widoku:

id_korepetycji	id_korepetytor	data	start	stop	czas_trwania
1	2	2023-06-05	09:00:00.0000000	10:00:00.0000000	60
2	2	2023-05-05	10:00:00.0000000	11:00:00.0000000	60
3	2	2023-05-05	11:30:00.0000000	12:30:00.0000000	60
5	5	2023-05-06	13:00:00.0000000	14:00:00.0000000	60
6	4	2023-05-07	09:00:00.0000000	10:00:00.0000000	60
7	4	2023-05-05	10:00:00.0000000	11:00:00.0000000	60
8	4	2023-05-05	11:00:00.0000000	12:00:00.0000000	60
9	5	2023-05-05	12:00:00.0000000	13:00:00.0000000	60

Rysunek 21: Przykładowe dane wygenerowane za pomocą widoku *Vczas_korepetycji*.

- Widok zamówionych, nieopłaconych kursów ***Vzamowienia_kursy_nieoplacone*** (Skrypt 17) - przedstawia informacje o zamówionych kursach przez użytkowników, które nie zostały opłacone.

```sql
CREATE VIEW Vzamowienia_kursy_nieoplacone AS
SELECT a.id_zamowienia_kursy, a.data_zlozenia, b.id_kursu,
       b.nazwa, c.id_uzytkownika, c.imie, c.nazwisko,
       c.e_mail, a.typ_platnosci, b.cena
FROM zamowienia_kursy as a
JOIN kursy b ON b.id_kursu = a.id_kursu
JOIN uzytkownicy c ON c.id_uzytkownika = a.id_uzytkownika
WHERE a.data_oplacenia IS NULL
```

Skrypt 17: Skrypt generujący widok *Vzamowienia_kursy_nieoplacone*.

przykład działania widoku:

id_zamowienia_kursy	data_zlozenia	id_kursu	nazwa	id_uzytkownika	imie	nazwisko	e_mail	typ_platnosci	data_oplacenia	cena
1	2023-05-05	2	Ulamki	2	Aleksander	Somanski	oleks@o2.pl	NULL	NULL	25.99
4	2023-05-05	2	Ulamki	7	Zuzanna	Ogorska	joziazuzia@gmail.com	NULL	NULL	25.99
34	2023-05-14	1	Logarytmy podstawy	26	Iza	Ryza	rr@op.pl	NULL	NULL	59.99
38	2023-05-14	5	Funkcja kwadratorwa	26	Iza	Ryza	rr@op.pl	Blik	NULL	33.99
41	2023-05-15	1	Logarytmy podstawy	27	Ola	Kapik	p@op.pl	Blik	NULL	59.99

Rysunek 22: Przykładowe dane wygenerowane za pomocą widoku *Vzamowienia_kursy_nieoplacone*.

- Widok zamówionych, nieopłaconych korepetycji ***Vzamowienia_korepetycje_nieoplacone*** (Skrypt 18) - przedstawia informacje o zamówionych korepetycjach przez użytkowników, które nie zostały opłacone.

```sql
CREATE VIEW Vzamowienia_korepetycje_nieoplacone AS
SELECT a.id_zamowienia_korepetycje, a.data_zamowienia,
       a.typ_platnosci, a.cena, b.id_korepetytor, b.data,
       b.start, b.stop, c.id_uzytkownika, c.imie,
       c.nazwisko, c.e_mail, c2.imie AS imie_korepetytora,
       c2.nazwisko AS nazwisko_korepetytora
FROM zamowienia_korepetycje a
JOIN korepetycje b ON a.id_korepetycje = b.id_korepetycji
JOIN uzytkownicy c ON a.id_uzytkownika = c.id_uzytkownika
JOIN uzytkownicy c2 ON b.id_korepetytor = c2.id_uzytkownika
    WHERE a.data_oplacenia is NULL
```

Skrypt 18: Skrypt generujący widok *Vzamowienia_korepetycje_nieoplacone*.

przykład działania widoku:

id_zamowienia_korepetycje	data_zamowienia	typ_platnosci	cena	id_korepetytor	data	start	stop
2	2023-05-04	PayU	60.00	2	2023-05-05	10:00:00.0000000	11:00:00.0000000
6	2023-05-01	przelew tradycyjny	90.00	2	2023-05-06	09:00:00.0000000	10:00:00.0000000
8	2023-05-06	NULL	90.00	2	2023-05-05	11:30:00.0000000	12:30:00.0000000
9	2023-05-06	Blik	120.00	5	2023-05-06	13:00:00.0000000	14:00:00.0000000

id_uzytkownika	imie	nazwisko	e_mail	imie_korepetytora	nazwisko_korepetytora
3	Izabella	Ulezanska	izu@l2.pl	Aleksander	Somanski
6	Stanislaw	Sekielski	stanislawsekielski@gmail.com	Aleksander	Somanski
3	Izabella	Ulezanska	izu@l2.pl	Aleksander	Somanski
8	Pawel	Kalinowski	pascal123@gmail.com	Kamila	Piotrowska

Rysunek 23: Przykładowe dane wygenerowane za pomocą widoku *Vzamowienia_korepetycje_nieoplacone*.

- Widok sumy cen kursów ***Vilosc_zamowien_kursow_oplaconych*** (Skrypt 19) - przedstawia informacja o sumarycznej cenie wszystkich opłaconych kursów.

```
CREATE VIEW Vilosc_zamowien_kursow_oplaconych AS
SELECT zk.id_kursu, SUM(k.cena) AS cena_suma
FROM zamowienia_kursy AS zk
JOIN kursy AS k ON zk.id_kursu = k.id_kursu
WHERE zk.data_oplacenia IS NOT NULL
GROUP BY zk.id_kursu
```

Skrypt 19: Skrypt generujący widok *Vilosc_zamowien_kursow_oplaconych*.

przykład działania widoku:

id_kursu	cena_suma
1	239.96
2	155.94
3	88.99
4	222.20
5	67.98
6	78.90

Rysunek 24: Przykładowe dane wygenerowane za pomocą widoku *Vilosc_zamowien_kursow_oplaconych*.

- Widok zamówień kursów ***Vszczegoly_zamowien_kursy*** (Skrypt 20) - przedstawia szczegółowe informacje o wszystkich zamówionych kursach.

```
CREATE VIEW Vszczegoly_zamowien_kursy AS
SELECT zk.id_zamowienia_kursy, zk.data_zlozenia,
       zk.id_kursu, zk.data_oplacenia,
       zk.typ_platnosci, k.nazwa, k.autor,
       k.adres_url, u.id_uzytkownika,
       u.imie, u.nazwisko
FROM zamowienia_kursy zk
JOIN kursy k ON zk.id_kursu = k.id_kursu
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN uzytkownicy u ON zk.id_uzytkownika = u.id_uzytkownika
WHERE zk.data_oplacenia IS NOT NULL
```

Skrypt 20: Skrypt generujący widok *Vszczegoly_zamowien_kursy*.

przykład działania widoku:

id_zamowienia_kursy	data_zlozenia	id_kursu	data_oplacenia	typ_platnosci	nazwa	autor	adres_url	id_uzytkownika	imie	nazwisko
2	2023-05-05	4	2023-05-05	Blik	Procenty	Marian Zora	https://marian-https://www.youtube.com/watch?v=d...	1	Waclaw	Ktos
3	2023-05-05	5	2023-05-05	PayU	Funkcja kwadratorwa	Maria Chrobak	https://www.youtube.com/watch?v=dQw4w9WgXcQ	6	Stanislaw	Sekielski
5	2023-05-05	4	2023-05-05	Blik	Procenty	Marian Zora	https://marian-https://www.youtube.com/watch?v=d...	8	Pawel	Kalinowski
6	2023-05-05	4	2023-05-05	Blik	Procenty	Marian Zora	https://marian-https://www.youtube.com/watch?v=d...	2	Aleksander	Somanski
8	2023-05-06	2	2023-05-05	NULL	Ulamki	Anna Robak	https://www.youtube.com/watch?v=dQw4w9WgXcQ	8	Pawel	Kalinowski
26	2023-05-14	1	2023-05-14	Blik	Logarytmy podstawy	Maria Chrobak	https://www.youtube.com/watch?v=dQw4w9WgXcQ	23	Milena	Kot
27	2023-05-14	2	2023-05-14	Blik	Ulamki	Anna Robak	https://www.youtube.com/watch?v=dQw4w9WgXcQ	23	Milena	Kot
28	2023-05-14	1	2023-05-14	Blik	Logarytmy podstawy	Maria Chrobak	https://www.youtube.com/watch?v=dQw4w9WgXcQ	25	Iza	Ryza
35	2023-05-14	2	2023-05-14	Blik	Ulamki	Anna Robak	https://www.youtube.com/watch?v=dQw4w9WgXcQ	26	Iza	Ryza

Rysunek 25: Przykładowe dane wygenerowane za pomocą widoku *Vszczegoly_zamowien_kursy*.

- Widok zamówień korepetycji ***Vszczegoly_zamowien_korepetycje*** (Skrypt 21) - przedstawia szczegółowe informacje o wszystkich zamówionych korepetycjach.

```
CREATE VIEW Vszczegoly_zamowien_korepetycje AS
SELECT zk.id_zamowienia_korepetycje, zk.data_zamowienia,
       zk.data_oplacenia, zk.typ_platnosci, zk.cena,
       zk.id_korepetycje, k.id_korepetytor, k.data,
       k.start, k.stop, c2.imie AS imie_korepetytora,
       c2.nazwisko AS nazwisko_korepetytora, u.
   id_uzytkownika,
       u.imie, u.nazwisko
FROM zamowienia_korepetycje zk
```

```
    JOIN korepetycje k ON zk.id_korepetycje = k.id_korepetycji
    JOIN uzytkownicy u ON zk.id_uzytkownika = u.id_uzytkownika
    JOIN uzytkownicy c2 ON k.id_korepetytor = c2.id_uzytkownika
        WHERE zk.data_oplacenia IS NOT NULL
```

Skrypt 21: Skrypt generujący widok *Vszczegoly_zamowien_korepetycje*.

przykład działania widoku:

id_zamowienia_korepetycje	data_zamowienia	data_oplacenia	typ_platnosci	cena	id_korepetycje	id_korepetytor	data	start
1	2023-05-05	2023-05-05	Blik	60.00	1	2	2023-06-05	09:00:00.0000000
5	2023-05-02	2023-05-04	Blik	120.00	11	2	2023-05-06	11:00:00.0000000

stop	imie_korepetytora	nazwisko_korepetytora	id_uzytkownika	imie	nazwisko
10:00:00.0000000	Aleksander	Somanski	1	Waclaw	Ktos
12:00:00.0000000	Aleksander	Somanski	8	Pawel	Kalinowski

Rysunek 26: Przykładowe dane wygenerowane za pomocą widoku *Vszczegoly_zamowien_korepetycje*.

- Widok ulubionych kursów ***Vulubione_kursy*** (Skrypt 22) - przedstawia listę kursów dodanych przed danego użytkownika do ulubionych.

```
CREATE VIEW Vulubione_kursy AS
SELECT uk.id_zamowienia_kursy, uk.id_uzytkownika,
       k.nazwa, k.autor, k.data_dodania, k.czas_trwania,
       k.adres_url
FROM ulubione_kursy AS uk
JOIN zamowienia_kursy AS zk
    ON uk.id_zamowienia_kursy = zk.id_zamowienia_kursy
JOIN kursy AS k on zk.id_kursu = k.id_kursu
```

Skrypt 22: Skrypt generujący widok *Vulubione_kursy*.

przykład działania widoku:

id_zamowienia_kursy	id_uzytkownika	nazwa	autor	data_dodania	czas_trwania	adres_url
1	2	Ulamki	Anna Robak	2023-02-05	00:55:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ
4	7	Ulamki	Anna Robak	2023-02-05	00:55:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ
5	8	Procenty	Marian Zora	2023-05-05	01:14:15	https://marian-https://www.youtube.com/watch?v=d...
6	2	Procenty	Marian Zora	2023-05-05	01:14:15	https://marian-https://www.youtube.com/watch?v=d...
8	8	Ulamki	Anna Robak	2023-02-05	00:55:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ

Rysunek 27: Przykładowe dane wygenerowane za pomocą widoku *Vulubione_kursy*.

- Widok ilości kursów zakupionych przez użytkownika ***Vilosc_zamowien_kursy_uzytkownika*** (Skrypt 23) - przedstawia ilość kursów zakupionych i opłaconych przez użytkowników.

```
CREATE VIEW Vilosc_zamowien_kursy_uzytkownika AS
SELECT zk.id_uzytkownika,
       COUNT(zk.id_zamowienia_kursy) AS ilosc_zamowien
FROM zamowienia_kursy AS zk
WHERE zk.data_oplacenia IS NOT NULL
GROUP BY zk.id_uzytkownika
```

Skrypt 23: Skrypt generujący widok *Vilosc_zamowien_kursy_uzytkownika*.

przykład działania widoku:

id_uzytkownika	ilosc_zamowien
1	1
2	1
6	1
8	2
23	2
25	1

Rysunek 28: Przykładowe dane wygenerowane za pomocą widoku *Vilosc_zamowien_kursy_uzytkownika*.

- Widok ilości korepetycji zarezerwowanych przez użytkownika ***Vilosc_zamowien_korepetycje_uzytkownika*** (Skrypt 24) - przedstawia ilość korepetycji zarezerwowanych i opłaconych przez użytkowników.

```
CREATE VIEW Vilosc_zamowien_korepetycje_uzytkownika AS
SELECT zk.id_uzytkownika,
       COUNT(zk.id_zamowienia_korepetycje) AS
       ilosc_zamowien
FROM zamowienia_korepetycje AS zk
WHERE zk.data_oplacenia IS NOT NULL
GROUP BY zk.id_uzytkownika
```

Skrypt 24: Skrypt generujący widok *Vilosc_zamowien_korepetycje_uzytkownika*.

przykład działania widoku:

id_uzytkownika	ilosc_zamowien
1	1
3	2
6	1
8	2

Rysunek 29: Przykładowe dane wygenerowane za pomocą widoku *Vilosc_zamowien_korepetycje_uzytkownika*.

- Widok szczegółów kursów ***Vkursy_poziomy_widok*** (Skrypt 25) - przedstawia listę dostępych kursów z dodatkowym polem zbierającym wszystkie poziomy danego kursu.

```
CREATE VIEW [dbo].[Vkursy_poziomy_widok] AS
SELECT k.id_kursu, k.nazwa, k.autor, k.cena,k.data_dodania,
       k.czas_trwania, k.adres_url,
       STRING_AGG(pk.nazwa_poziomu, ', ') AS poziomy_kursow
FROM kursy AS k
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow pk ON kp.id_poziomu = pk.id_poziomu
GROUP BY k.id_kursu, k.nazwa, k.autor, k.cena,
         k.data_dodania, k.czas_trwania, k.adres_url;
```

Skrypt 25: Skrypt generujący widok *Vkursy_poziomy_widok*.

przykład działania widoku:

id_kursu	nazwa	autor	cena	data_dodania	czas_trwania	adres_url	poziomy_kursow
1	Logarytmy podstawy	Maria Chrobak	59.99	2023-03-05	01:30:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ	I LO podstawa, I LO rozszerzenie, Matura podstawa
2	Ulamki	Anna Robak	25.99	2023-02-05	00:55:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ	V SP, VI SP, VII SP, Matura podstawa
3	Geometria analityczna	Maria Chrobak	88.99	2023-01-05	02:00:00	https://maria-https://www.youtube.com/watch?v=dQ...	IV LO podstawa, IV LO rozszerzenie, Matura podst...
4	Procenty	Marian Zora	55.55	2023-05-05	01:14:15	https://marian-https://www.youtube.com/watch?v=d...	VIII SP, Matura podstawa
5	Funkcja kwadratorwa	Maria Chrobak	33.99	2023-05-05	00:45:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ	II LO podstawa, Matura podstawa
6	Równania	Zofia Gut	78.90	2023-05-06	01:15:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ	I LO rozszerzenie
7	Funkcje	Zofia Gut	50.00	2023-05-06	00:35:00	https://www.youtube.com/watch?v=dQw4w9WgXcQ	I LO rozszerzenie

Rysunek 30: Przykładowe dane wygenerowane za pomocą widoku *Vkursy_poziomy_widok*.

5.2 Procedury

Procedury składowane to nazwane bloki kodu T-SQL, które są przechowywane w bazie danych jako obiekty i mogą być wywoływane przez aplikacje lub inne zapytania. Procedury składowane są przydatne w sytuacjach, gdy często wykonywane są te same zapytania lub operacje na bazie danych, a ich wyniki są potrzebne w wielu miejscach [7].

Poniżej przedstawiono procedury utworzone dla bazy danych aplikacji. Do każdej procedury dołączony jest skrypt generujący daną procedurę.

- Procedura dodania użytkownika ***DodajUzytkownika*** (Skrypt 26) - dodaje nowego użytkownika do bazy danych, szyfrując hasło algorytmem SHA2_512 [8].

```
CREATE PROCEDURE [dbo].[DodajUzytkownika]
        @email VARCHAR(200),
        @haslo VARCHAR(100),
        @imie VARCHAR(200) = NULL,
        @nazwisko VARCHAR(200) = NULL,
        @plec VARCHAR(10) = NULL,
        @telefon VARCHAR(25) = NULL,
        @data_urodzenia DATE = NULL,
        @kraj VARCHAR(30) = NULL,
        @miasto VARCHAR(30) = NULL,
        @ulica VARCHAR(100) = NULL,
        @numer_domu VARCHAR(10) = NULL,
        @kod_pocztowy VARCHAR(10) = NULL,
        @uprawnienia_korepetytora BIT = 0,
        @data_usuniecia DATE = NULL
AS
BEGIN
    -- dodaje szyfrowanie hasla
    DECLARE @zaszyfrowane_haslo VARBINARY(8000) =
            HASHBYTES('SHA2_512', @haslo);

    INSERT INTO uzytkownicy (e_mail, haslo, imie, nazwisko,
                plec, telefon, data_urodzenia, kraj, miasto
                ulica, numer_domu, kod_pocztowy,
                prawnienia_korepetytora, data_utworzenia,
                ostatnie_logowanie, data_usuniecia)
    VALUES (@email, @zaszyfrowane_haslo, @imie, @nazwisko,
            @plec, @telefon, @data_urodzenia, @kraj,
            @miasto, @ulica, @numer_domu, @kod_pocztowy,
```

```
                @uprawnienia_korepetytora, GETDATE(),
                GETDATE(), @data_usuniecia);
    END
```

Skrypt 26: Skrypt generujący procedurę składową *DodajUzytkownika*.

- Procedura edycji użytkownika ***EdytujUzytkownika*** (Skrypt 27) - edytuje dane użytkownika, szyfrując hasło algorytmem SHA2_512 [8].

```
    CREATE PROCEDURE [dbo].[EdytujUzytkownika]
            @id int,
            @haslo VARCHAR(100) = NULL,
            @imie VARCHAR(200) = NULL,
            @nazwisko VARCHAR(200) = NULL,
            @plec VARCHAR(10) = NULL,
            @telefon VARCHAR(25) = NULL,
            @data_urodzenia DATE = NULL,
            @kraj VARCHAR(30) = NULL,
            @miasto VARCHAR(30) = NULL,
            @ulica VARCHAR(100) = NULL,
            @numer_domu VARCHAR(10) = NULL,
            @kod_pocztowy VARCHAR(10) = NULL
    AS
    BEGIN
        DECLARE @zaszyfrowane_haslo VARBINARY(8000) =
                HASHBYTES('SHA2_512', @haslo)

        UPDATE uzytkownicy
        SET
            imie = ISNULL(@imie, imie),
            nazwisko = ISNULL(@nazwisko, nazwisko),
            haslo = ISNULL(@zaszyfrowane_haslo, haslo),
            plec = CASE WHEN @plec IS NULL THEN NULL ELSE
                   ISNULL(@plec, plec) END,
            telefon = CASE WHEN @telefon IS NULL THEN NULL ELSE
                      ISNULL(@telefon, telefon) END,
            data_urodzenia = CASE WHEN @data_urodzenia IS NULL
                             THEN NULL ELSE ISNULL
                             (@data_urodzenia, data_urodzenia)
       END,
            kraj = CASE WHEN @kraj IS NULL THEN NULL ELSE
                   ISNULL(@kraj, kraj) END,
```

```
            miasto = CASE WHEN @miasto IS NULL THEN NULL ELSE
                     ISNULL(@miasto, miasto) END,
            ulica = CASE WHEN @ulica IS NULL THEN NULL ELSE
                    ISNULL(@ulica, ulica) END,
            numer_domu = CASE WHEN @numer_domu IS NULL THEN
        NULL
                         ELSE ISNULL(@numer_domu, numer_domu)
        END,
            kod_pocztowy = CASE WHEN @kod_pocztowy IS NULL THEN
                           NULL ELSE ISNULL
                           (@kod_pocztowy, kod_pocztowy) END
         WHERE id_uzytkownika = @id;

    END
```

Skrypt 27: Skrypt generujący procedurę składową *EdytujUzytkownika*.

- Procedura dodania kursu do ulubionych przez użytkownika ***DodajKursDoUlubionych*** (Skrypt 28) - dodaje dany, zakupiony kurs do ulubionych użytkownika.

```
    CREATE PROCEDURE [dbo].[DodajKursDoUlubionych]
        @IdUzytkownika int,
        @IdZamowieniaKurs int
    AS
    BEGIN
        INSERT INTO ulubione_kursy (id_zamowienia_kursy,
       id_uzytkownika)
        VALUES (@IdUzytkownika, @IdZamowieniaKurs)
    END
```

Skrypt 28: Skrypt generujący procedurę składową *DodajKursDoUlubionych*.

- Procedura szukania kursu po ID poziomu ***SzukajKursowPoPoziomie*** (Skrypt 29) - wyświetla informacje o kursach o danym poziomie.

```
    CREATE PROCEDURE SzukajKursowPoPoziomie
        @IDPoziomu int
    AS
    BEGIN
        SELECT pk.nazwa_poziomu, k.nazwa AS nazwa_kursu,
```

```
        k.autor, k.cena, k.adres_url
        FROM kursy AS k
        JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
        JOIN poziomy_kursow pk ON kp.id_poziomu = pk.id_poziomu
        WHERE pk.id_poziomu = @IDPoziomu
    END
```

Skrypt 29: Skrypt generujący procedurę składową *SzukajKursowPoPoziomie*.

- Procedura sumująca ilość zamówień kursów w przedziale czasowym ***LiczbaZamowienKursuCzas*** (Skrypt 30) - wyświetla informacje o ilości zamówień opłaconych w danym przedziale czasu.

```
    CREATE PROCEDURE LiczbaZamowienKursuCzas
        @IDKursu int,
        @DataOd datetime,
        @DataDo datetime
    AS
    BEGIN
        SELECT COUNT(*) AS liczba_zamowien,
               YEAR(zamowienia_kursy.data_oplacenia) AS rok,
               MONTH(zamowienia_kursy.data_oplacenia)
               AS miesiac
        FROM zamowienia_kursy
        WHERE zamowienia_kursy.id_kursu = @IDKursu
        AND zamowienia_kursy.data_oplacenia BETWEEN
            @DataOd AND @DataDo
        GROUP BY YEAR(zamowienia_kursy.data_oplacenia),
        MONTH(zamowienia_kursy.data_oplacenia)
    END
```

Skrypt 30: Skrypt generujący procedurę składową *LiczbaZamowienKursuCzas*.

- Procedura sumująca ilość zamówień korepetycji w przedziale czasowym ***LiczbaZamowienKorepetycjiCzas*** (Skrypt 31) - wyświetla informacje o ilości zamówień opłaconych w danym przedziale czasu.

```
CREATE PROCEDURE LiczbaZamowienKorepetycjiCzas
    @IDKorepetycji int,
    @DataOd datetime,
    @DataDo datetime
AS
BEGIN
    SELECT COUNT(*) AS liczba_zamowien,
           YEAR(zamowienia_korepetycje.data_oplacenia)
           AS rok,
           MONTH(zamowienia_korepetycje.data_oplacenia)
               AS miesiac
    FROM zamowienia_korepetycje
    WHERE zamowienia_korepetycje.id_korepetycji = @IDKursu
    AND zamowienia_korepetycje.data_oplacenia BETWEEN
        @DataOd AND @DataDo
    GROUP BY YEAR(zamowienia_korepetycje.data_oplacenia),
    MONTH(zamowienia_korepetycje.data_oplacenia)
END
```

Skrypt 31: Skrypt generujący procedurę składową *LiczbaZamowienKorepetycjiCzas*.

- Procedura usuwająca nieopłacone zamówienia ***UsunNieoplaconeZamowienia*** (Skrypt 32) - usuwa zamówienia kursów i korepetycji starsze niż 30 dni, które nie zostały opłacone.

```
CREATE PROCEDURE UsunNieoplaconeZamowienia
AS
BEGIN
    DELETE FROM zamowienia_kursy
    WHERE data_oplacenia IS NULL
    AND DATEDIFF(day, data_zlozenia, GETDATE()) > 30

    DELETE FROM zamowienia_korepetycje
    WHERE data_oplacenia IS NULL
    AND DATEDIFF(day, data_zamowienia, GETDATE()) > 30
END
```

Skrypt 32: Skrypt generujący procedurę składową *UsunNieoplaconeZamowienia*.

- Procedura aktualizująca cenę kursu ***AktualizujCeneKursu*** (Skrypt 33) - zmienia cenę kursu o podany procent.

```
CREATE PROCEDURE AktualizujCeneKursu
    @IDKursu int,
    @Procent decimal(5,2)
AS
BEGIN
    DECLARE @stara_cena decimal(10,2);
    DECLARE @nowa_cena decimal(10,2);

    -- pobranie ceny kursu
    SELECT @stara_cena = cena FROM kursy
    WHERE id_kursu = @IDKursu

    -- obliczenie nowej ceny
    SELECT @nowa_cena = @stara_cena * (1 + @Procent/100);

    -- aktualizacja
    UPDATE kursy SET cena = @nowa_cena
    WHERE id_kursu = @IDKursu
END
```

Skrypt 33: Skrypt generujący procedurę składową *AktualizujCeneKursu*.

- Procedura aktualizująca cenę korepetycji ***AktualizujCeneKorepetycji*** (Skrypt 34) - zmienia cenę korepetycji o podany procent.

```
CREATE PROCEDURE AktualizujCeneKorepetycji
    @IDKorepetycji int,
    @Procent decimal(5,2)
AS
BEGIN
    DECLARE @stara_cena decimal(10,2);
    DECLARE @nowa_cena decimal(10,2);

    -- pobranie ceny korepetycji
    SELECT @stara_cena = cena FROM zamowienia_korepetycje
    WHERE id_korepetycje = @IDKorepetycji
```

```
        -- obliczenie nowej ceny
        SELECT @nowa_cena = @stara_cena * (1 + @Procent/100);

        -- aktualizacja
        UPDATE zamowienia_korepetycje SET cena = @nowa_cena
        WHERE id_korepetycje = @IDKorepetycji
    END
```

Skrypt 34: Skrypt generujący procedurę składową *AktualizujCeneKorepetycji*.

5.3 Triggery

Triggery to specjalne procedury, który są automatycznie wykonywane w odpowiedzi na określone zdarzenia w bazie danych [9].

Poniżej przedstawiony jest trigger ***usuniecie_ulubionych_kursow*** (Skrypt 35), którego zadaniem jest usunięcia kursów dodanych przez użytkownika do ulubionych w momencie w którym użytkownik usuwa swoje konto w aplikacji (usunięcie jest równoznaczne z dodaniem daty usunięcia do kolumy *data_usuniecia* tabeli *uzytkownicy* Rozdział 4.2.1).

```
CREATE TRIGGER usuniecie_ulubionych_kursow
ON uzytkownicy
AFTER UPDATE
AS
BEGIN
    IF UPDATE(data_usuniecia)
    BEGIN
    DECLARE @IDuzytkownika int
    SET @IDuzytkownika = (SELECT id_uzytkownika FROM inserted);

    DELETE FROM ulubione_kursy
    WHERE id_uzytkownika = @IDuzytkownika;
    END
END
```

Skrypt 35: Skrypt generujący trigger *usuniecie_ulubionych_kursow*.

6. Opis implementacji elementów aplikacji

6.1 Charakterystyka środowiska użytego do implementacji

Implementację aplikacji internetowej oparto na języku programowania *Python* z frameworkiem *Flask*. *Python* oferuje szerokie możliwości i jest często wykorzystywany do tworzenia aplikacji internetowych. W tym przypadku, zdecydowano się na użycie frameworka *Flask*, który jest minimalistycznym i łatwym w użyciu narzędziem do tworzenia aplikacji webowych w *Pythonie*. *Flask* umożliwia obsługę żądań HTTP, routingu, obsługi formularzy i wiele innych funkcji, które są niezbędne przy tworzeniu aplikacji internetowych [3].

Do komunikacji z bazą danych *SQL Server* wybrano bibliotekę *PyODBC*. Jest to popularne narzędzie umożliwiające połączenie się z bazami danych z poziomu aplikacji. *PyODBC* zapewnia prosty interfejs, który umożliwia wykonywanie zapytań SQL, pobieranie i zapisywanie danych w bazie [10].

Do tworzenia szablonów *HTML* z dynamicznymi danymi zastosowano silnik szablonów *Jinja*. *Jinja* jest narzędziem do generowania dynamicznych stron internetowych w *Pythonie*. Pozwala na wstawianie zmiennych, warunków, pętli i innych instrukcji programistycznych bezpośrednio w kodzie HTML [11].

6.2 Opis implementacji

Opis implementacji podzielony został na podrozdziały dotyczące konkretnych funkcjonalności aplikacji. Każdy opis posiada odpowiednie zrzuty ekranów, obrazujące działającą aplikację. Implementacja fragmentów aplikacji w języku *Python* znajduje się w Załączniku A.

6.2.1 Strona główna

Widok strony głównej aplikacji dla osoby niezalogowanej przedstawiony został na Rysunku 31. Użytkownik niezalogowany ma możliwość zalogować się (Rozdział 6.2.3), zarejestrować w aplikacji (Rozdział 6.2.4), albo przejrzeć listę dostępnych kursów (Rozdział 6.2.2).

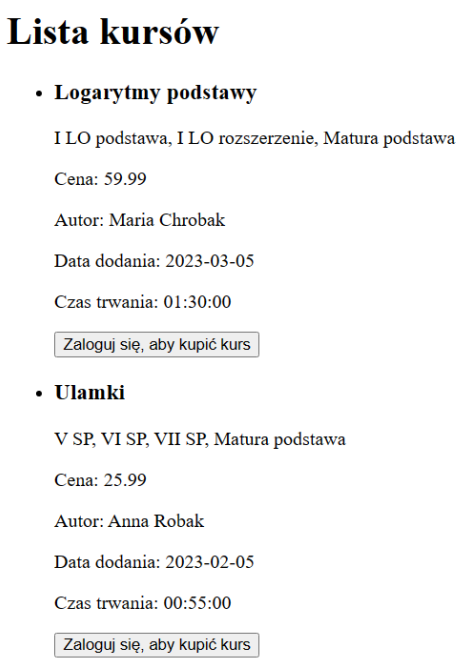
Rysunek 31: Strona główna aplikacji, widoczna dla osoby niezalogowanej.

Dla użytkownika zalogowanego funkcje na stronie głównej ulegają zmianie. Jak można zobaczyć na Rysunku 32, taki użytkownik ma możliwość przejścia do swojego konta (Rozdział 6.2.5), wylogowania się (Rozdział 6.2.3), oraz przejrzenia dostępnych kursów (Rozdział 6.2.2). Kod implementujący stronę główną aplikacji znajduje się w Załączniku A.1, Skrypt 36.

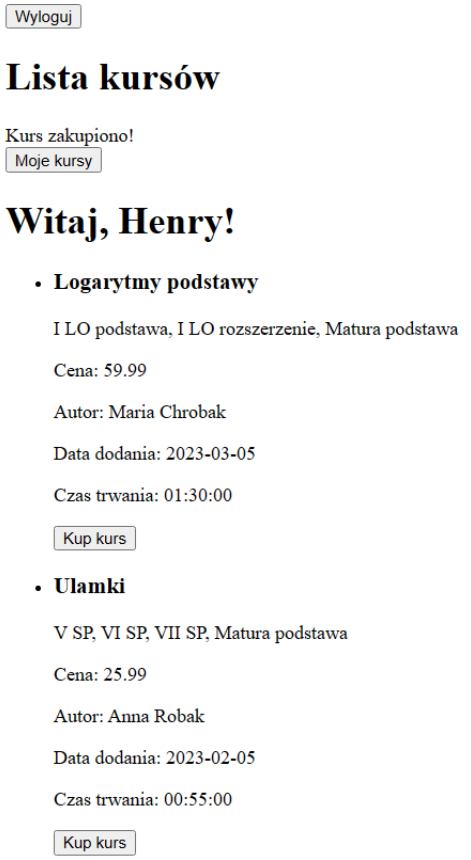
Rysunek 32: Strona główna aplikacji, widoczna dla osoby zalogowanej.

6.2.2 Lista kursów

Strona z listą oferowanych kursów dla użytkownika niezalogowanego i zalogowanego przedstawiona jest odpowiednio na Rysunku 33 i Rysunku 34. Kursy wyświetlane są przy użyciu widoku *Vkursy_poziomy_widok* (Skrypt 25). Użytkownik niezalogowany, chcąc zakupić dany kurs musi zalogować się w aplikacji (przycisk *Zaloguj się, aby kupić kurs*, Rozdział 6.2.3). Po zalogowaniu i przejściu na stronę z listą kursów (Rysunek 34) użytkownik kupuje kurs poprzez przycisk *Kup kurs*, który wprowadza nowe dane do tabeli *zamowienia_kursy* (Rozdział 4.2.5. Po zakończonej akcji zakupy, na stronie pojawia się informacja: „Kurs zakupiono!”, oraz odnośnik do strony z kursami użytkownika (Rozdział 6.2.7). Jeżeli użytkownik będzie chciał ponownie zakupić ten sam kurs, na stronie pojawi się informacja: „Kurs został już zakupiony”. Użytkownik ma także możliwość wylogowania się z aplikacji (Rozdział 6.2.3). Implementacja tej części aplikacji znajduje się w Załączniku A.2, Skrypt 37.



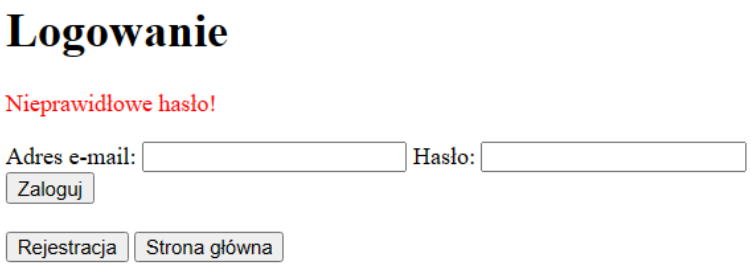
Rysunek 33: Lista dostępnych kursów, widoczna dla osoby niezalogowanej.



Rysunek 34: Lista dostępnych kursów, widoczna dla osoby zalogowanej. Widok jest z perspektywy osoby, która właśnie zakupiła kurs (powiadomienie „Kurs zakupiono!”).

6.2.3 Logowanie

Logowanie do aplikacji odbywa się poprzez wpisanie adresu e-mail i hasła podanego podczas rejestracji w odpowiednie pola na stronie. Dane porównywane są następnie z danymi bazy danych tabeli *uzytkownicy* (Rozdział 4.2.1). Jeżeli wprowadzone są one poprawnie, użytkownik zostaje przekierowany do panelu użytkownika (Rozdział 6.2.5), oraz aktualizowana jest wartość atrybutu *ostatnie_logowanie* tabeli *uzytkownicy*. Jeżeli dane wprowadzono niepoprawnie, użytkownik zostaje poinformowany o tym stosownym komunikatem (w zależności od tego czy niepoprawnie wprowadzono hasło, czy adres e-mail). Na stronie logowania znajdują się również odnośniki do strony rejestracji (Rozdział 6.2.4) i do strony głównej (Rozdział 6.2.1). Na Rysunku 35 przedstawiono widok strony po wprowadzeniu niepoprawnego hasła. W Załączniku A.3, Skrypt 38 przedstawiono kod części aplikacji związanej z logowaniem użytkownika.



Rysunek 35: Strona logowania aplikacji po podaniu nieprawidłowego hasła dla wprowadzonego adresu e-mail, który znajduje się w bazie danych.

6.2.4 Rejestracja

Rejestracja nowego użytkownika w aplikacji polega na wypełnieniu formularza przedstawionego na Rysunku 36. Obowiązkowe pola to: e-mail, hasło, imię i nazwisko. Pola, które mają posiadać określoną formę również widoczne są na przedstawionym przykładzie (płeć, telefon, data urodzenia i kod pocztowy). Wprowadzony e-mail sprawdzany jest w bazie danych (tabela *uzytkownicy*, Rozdział 4.2.1). Jeżeli użytkownik istnieje w bazie, pojawia się odpowiedni komunikat. Wprowadzone hasło jest walidowane: musi składać się z 5 do 15 znaków, z czego z przynajmniej jednej cyfry i jedenego znaku specjalnego. Jeżeli użytkownik zrezygnuje z rejestracji, może wrócić do strony głównej (przycisk *Wróć*). Nowy użytkownik wpisywany jest do bazy za pośrednictwem procedury składowania *DodajUzytkownika* (Skrypt 26). Po poprawnym zarejestrowaniu, następuje przekierowanie na strone logowania, gdzie użytkownik może zalogować się do aplikacji. Załącznik A.4, Skrypt 39 zawiera implementację tej części aplikacji.

Rysunek 36: Strona rejestracji nowego użytkownika.

6.2.5 Panel użytkownika

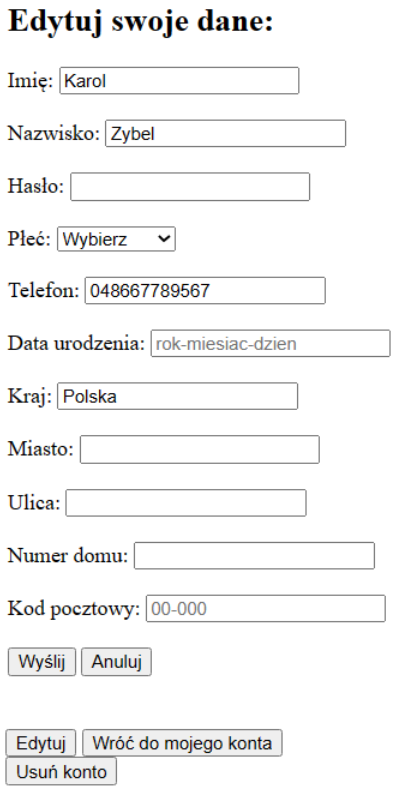
Panel użytkownika (Rysunek 37) to prosta strona zawierająca przyciski nawigujące do innych podstron aplikacji: panelu użytkownika (Rozdział 6.2.6), kursów użytkownika (Rozdział 6.2.7), ulubionych kursów użytkownika (Rozdział 6.2.8), listy oferowanych kursów (Rozdział 6.2.2), strony głównej (Rozdział 6.2.1), oraz przycisk pozwalający na wylogowanie się z aplikacji (Załącznik A.3, Skrypt 38). Skrypt implementujący panel użytkownika znajduje się w Załączniku A.5, Skrypt 40.

Rysunek 37: Panel użytkownika zalogowanego do aplikacji.

6.2.6 Dane użytkownika i ich edycja

Strona z danymi użytkownika pobiera dane z tabeli *uzytkownicy* (Rozdział 4.2.1) i wyświetla je w sposób pokazany na Rysunku 38. Użytkownik ma możliwość zmiany danych poprzez kliknięce przycisku *Edytuj* (Rysunek 39). Edycja danych odbywa się poprzez wywołanie procedury składowanej *EdytujUzytkownika* (Skrypt 27). Dodatkowo, użytkownik może także usunąć swoje konto. Usunięcie konta wprowadza aktualną datę do kolumny *data_usunięcia* tabeli *uzytkownicy*, oraz aktualizuje adres e-mail użytkownika o datę usunięcia. Dodatkowo podczas usunięcia konta wywoływany jest trigger *usuniecie_ulubionych_kursow* (Skrypt 35). Skrypt zawierający kod opisanych w tym rozdziale funkcjonalności znajduje się w Załączniku A.6, Skrypt 41.

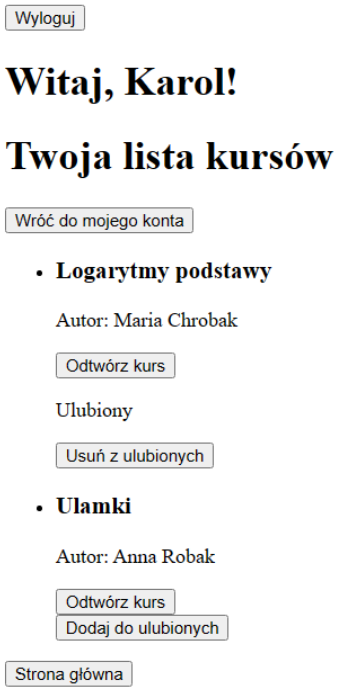
Rysunek 38: Panel użytkownika z widocznymi danymi podanymi przez użytkownika. Nieobowiązkowe pola, których użytkownik nie wypełnił są puste.



Rysunek 39: Edycja danych użytkownika.

6.2.7 Lista kursów użytkownika

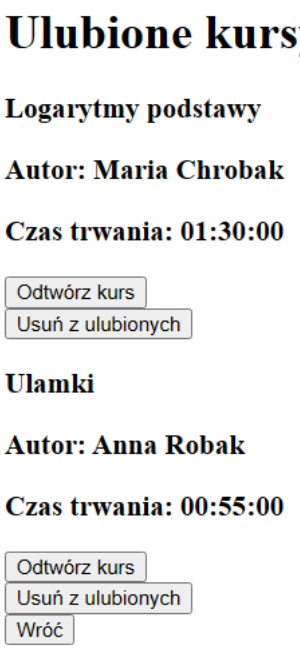
Lista zakupionych i opłaconych kursów przez użytkownika (Rysunek 40) korzysta z widoku *Vszczegoly_zamowien_kursy* (Skrypt 20). Użytkownik ma możliwość odtworzyć dany kurs (przycisk *Odtwórz kurs*), oraz dodać kurs do ulubionych (przycisk *Dodaj do ulubionych*). Dodanie kursu do ulubionych tworzy nowy rekord w tabeli *ulubione_kursy* (Rozdział 4.2.6). Po dodaniu kursu do ulubionych, przycisk *Dodaj do ulubionych* znika, pojawiaja się natomiast informacja „Ulubiony" przy kursie, oraz przycisk *Usuń z ulubionych* (który usuwa rekord z tabeli *ulubione_kursy*). Załącznik A.7, Skrypt 42 opisuje implementację opisanej funkcjonalności aplikacji.



Rysunek 40: Strona z listą zakupionych i opłaconych kursów użytkownika. Kurs *Logarytmy podstawy* został dodany do ulubionych.

6.2.8 Lista ulubionych kursów użytkownika

Ulubione kursy użytkownika (Rysunek 41) to strona bardzo podobna do listy zakupionych kursów (Rozdział 6.2.7). Wykorzystywany tutaj jest widok *Vulubione_kursy* (Skrypt 22) do pobrania informacji o kursach dodanych do ulubionych. Użytkownik może odtworzyć dany kurs (przycisk *Odtwórz kurs*), albo usunąć kurs z ulubionych (przycisk *Usuń z ulubionych*), który usuwa odpowiednie dane z tabeli *ulubione_kursy* (Rozdział 4.2.6). Ta część implementacji aplikacji znajduje się w Załączniku A.7, Skrypt 43.



Rysunek 41: Lista kursów dodanych przez użytkownika do ulubionych.

7. Podsumowanie

W ramach pracy dyplomowej przeprowadzono analizę wymagań (Rozdział 2), zaprojektowano podstawowe makiety interfejsu aplikacji (Rozdział 3) i stworzono system bazodanowy (Rozdział 4), wraz z wartwami dostępu do danych, takimi jak: widoki (Rozdział 5.1), procedury składowania (Rozdział 5.2) i triggery (Rozdział 5.3). Dodatkowo, wykonano implementację niektórych elementów aplikacji (Rozdział 6).

W celu przetestowania poprawności działania aplikacji internetowej, wykonano testy jednostkowe sprawdzające poszczególne jednostki kodu, oraz testy użytkowalności.

Stworzona aplikacji internetowa wymaga jeszcze wiele pracy przed ewentualnym wdrożeniem. Przede wszystkim nie została zaimplementowana funkcjonalność związana z rezerwowaniem i przeprowadzeniem korepetycji, czy obsługa płatności za kursy i korepetycje. Stworzone elementy platformy uznano za wystarczająca pod kątem niniejszej pracy dyplomowej.

Literatura

[1] SQL Server firmy Microsoft, https://microsoft.com, [dostęp 04.05.2023].

[2] Aplikacja internetowa Figma, https://www.figma.com/ [dostęp 15.05.2023].

[3] Framework Flask, https://flask.palletsprojects.com/en/2.3.x/ [dostęp 15.05.2023].

[4] Aplikacja Diagrams.net, https://app.diagrams.net/, [dostęp: 15.05.2023]

[5] Bernard Maj, „Studia podyplomowe - Systemy Baz Danych. Analiza i modelowanie wymagań przy użyciu wybranych metod inżynierii oprogramowania".

[6] Definicja widoków, https://www.ibm.com/docs/pl/i/7.2?topic=sql-creating-using-view, [dostęp: 08.05.2023].

[7] Definicja prodecur składowanych, https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16, [dostęp: 08.05.2023].

[8] Algorytm SHA2, https://justcryptography.com/sha-2/, [dostęp: 08.05.2023].

[9] Definicja Triggeru, https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch15.html, [dostęp: 09.05.2023].

[10] Biblioteka PyODBC, https://pypi.org/project/pyodbc/, [dostęp: 16.05.2023].

[11] Szablony Jinja, https://jinja.palletsprojects.com/en/3.1.x/, [dostęp: 16.05.2023].

A. Załącznik: implementacja elementów aplikacji

A.1 Strona główna

```
@app.route('/')
def index():
    """
    Strona główna aplikcji
    """
    if 'id_user' in session:
        msg = f"Zalogowany jako: {session['id_user']}"
        user_data = fun.user_data(email=None, id_user=session['id_user'])
        return render_template('index.html', msg=msg, user_data=user_data)
    return render_template('index.html')

# Funkcja pomocniczna
def execute_query(query, params=None):
    """
    Funkcja wykonująca zapytanie SQL i zwracająca słownik
    """
    connection_string = f'DRIVER={driver};SERVER={server}; //
                        DATABASE={database};UID={username};PWD={password}'
    conn = pyodbc.connect(connection_string)
    cursor = conn.cursor()
    if params is None:
        cursor.execute(query)
    else:
        cursor.execute(query, params)
    rows = cursor.fetchall()

    # Pobranie nazw atrybutów
    column_names = [column[0] for column in cursor.description]

    # Przygotowanie wyniku jako słownik zawierający nazwy atrybutów
    # i ich wartości
    result = []
    for row in rows:
        row_dict = {}
        for i in range(len(column_names)):
            row_dict[column_names[i]] = row[i]
        result.append(row_dict)
```

```
        # Zamknięcie połączenia z bazą danych
        cursor.close()
        conn.close()
        # Zwrócenie wyniku jako słownik
        return result

    # Funkcja pomocnicza
    def user_data(email=None, id_user=None):
        """
        Funkcja zwracająca dane użytkownika
        """
        if email is not None:
            tmp = execute_query('SELECT * FROM uzytkownicy WHERE //
                  e_mail=? AND data_usuniecia IS NULL', (email,))
        elif id_user is not None:
            tmp = execute_query('SELECT * FROM uzytkownicy WHERE //
                  id_uzytkownika=? AND data_usuniecia IS NULL', (id_user,))

        result = {}
        for i in range(len(tmp)):
            result [i]=tmp[i]

        if result:
            user_data = {
                'id_user': result[0]['id_uzytkownika'],
                'imie': result[0]['imie'],
                'nazwisko': result[0]['nazwisko'],
                'e_mail': result[0]['e_mail'],
                'haslo': result[0]['haslo'],
                'plec': result[0]['plec'],
                'telefon': result[0]['telefon'],
                'data_urodzenia': result[0]['data_urodzenia'],
                'kraj': result[0]['kraj'],
                'miasto': result[0]['miasto'],
                'ulica': result[0]['ulica'],
                'numer_domu': result[0]['numer_domu'],
                'kod_pocztowy': result[0]['kod_pocztowy'],
                'data_usuniecia': result[0]['data_usuniecia']
            }
            return user_data
        return False
```

Skrypt 36: Fragment kodu w Python implementującego stronę główną aplikacji.

A.2 Lista kursów

```
@app.route('/kursy')
def courses():
    """
    Strona z kursami dostępna dla zalogowanych
    i niezalogowanych użytkowników
    """
    # Pobranie listy kursów z bazy danych
    cursor = fun.connect_to_database().cursor()
    cursor.execute('SELECT * FROM Vkursy_poziomy_widok')
    kursy = cursor.fetchall()
    cursor.close()
    fun.connect_to_database().close()

    if 'id_user' in session:
        user_data = fun.user_data(email=None, id_user=session['id_user'])
        return render_template('kursy.html', kursy=kursy, //
                               user_data=user_data)
    return render_template('kursy.html', kursy=kursy)


@app.route('/kup-kurs/<int:id_kursu>')
def buy_course(id_kursu):
    """
    Funkcjonalność kupienie kursu.
    Jeżeli niezalogowany: kupuje kurs.
    Jeżeli niezalogowany: przekierowanie do strony logowania
    """
    if 'id_user' in session:
        # sprawdzenie czy uzytkownik juz ma kupiony dany kurs
        result0 = fun.execute_query('SELECT * FROM zamowienia_kursy WHERE //
        id_kursu=? AND id_uzytkownika=?', (id_kursu, session['id_user']))

        # jeżeli kurs widnieje w tabeli zamowienia_kursy
        if result0:
            flash('Sprawdź swoje kursy! Już masz ten kurs')
            return redirect(url_for('courses'))

        # pobieranie danych kursu
        tmp = fun.execute_query('SELECT * FROM kursy WHERE //
                                id_kursu=?', (id_kursu,))
        result = {}
```

```
            for i in range(len(tmp)):
                result[i]=tmp[i]

            if result:
                kursy = {
                    'id_kursu': result[0]['id_kursu'],
                    'nazwa': result[0]['nazwa'],
                    'autor': result[0]['autor'],
                    'cena': result[0]['cena'],
                    'data_dodania': result[0]['data_dodania'],
                    'czas_trwania': result[0]['czas_trwania'],
                    'adres_url': result[0]['adres_url'],
                }
                current_date = date.today().strftime('%Y-%m-%d')
                # Aktualizacja danych zamówienia
                # Dodanie oplacenia jako dzien dzisiejszy
                fun.execute_sql_query('INSERT INTO zamowienia_kursy (id_kursu, //
                id_uzytkownika, data_zlozenia, data_oplacenia, typ_platnosci) //
                VALUES (?, ?, ?, ?, ?)', (kursy['id_kursu'], session['id_user'], //
                current_date, current_date, 'Blik'))

                flash('Kurs zakupiono!')
                return redirect(url_for('courses'))
            else:
                return redirect(url_for('login'))
```

Skrypt 37: Fragment kodu w Python implementującego listę oferowanych kursów, oraz zakup kursu.

A.3 Logowanie

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    """
    Strona logowania
    """
    if request.method == 'POST':
        email = request.form['email']
        password = request.form['password']

        # wywołuje funkcję hashującą wprowadzone hasło,
        # które jest porównywane z hasłem z bazy danych
```

```
            hashed_password = fun.hash_password(password)

            # pobiera dane użytkownika o zadanym email lub id_user
            user_data = fun.user_data(email=email, id_user=None)

            if user_data:
                stored_password = user_data['haslo']
                if hashed_password == stored_password:
                    session['id_user'] = user_data['id_user']
                    session.permanent = True
                    app.permanent_session_lifetime = timedelta(hours=2)

                    user_data = fun.user_data(email=None,
                                id_user=session['id_user'])
                    query = f"UPDATE uzytkownicy SET ostatnie_logowanie=? //
                            WHERE id_uzytkownika=?"
                    params = (datetime.now(), session['id_user'])
                    fun.execute_sql_query(query, params)

                    # jeżeli dane są poprawne następuje wpisanie
                    # id_uzytkownika do sesji
                    # przekierowanie do strony moje-konto
                    return redirect(url_for('dashboard', user_data=user_data))
                else:
                    error_msg = "Nieprawidłowe hasło!"
                    return render_template('login.html', error=error_msg)
            else:
                error_msg = "Nieprawidłowy e-mail!"
                return render_template('login.html', error=error_msg)
        return render_template('login.html')

    @app.route('/wyloguj')
    def logout():
        """
        Funkcjonalność usuwająca id_uzytkownika z sesji
        """
        session.pop('id_user', None)
        return redirect(url_for('index'))
```

Skrypt 38: Fragment kodu w Python implementującego logowanie użytkowników do aplikacji i wylogowanie z aplikacji.

A.4 Rejestracja

```
@app.route('/rejestracja', methods=['GET', 'POST'])
def rejestracja():
    """
    Rejestracja nowego użytkownika
    """
    if request.method == 'POST':
        # Pobranie danych z formularza
        email = request.form.get('email')
        password = request.form.get('password')
        imie = request.form.get('imie')
        nazwisko = request.form.get('nazwisko')
        plec = request.form.get('plec')
        if plec == '' or plec == 'Wybierz':
            plec = None
        telefon = request.form.get("telefon")
        if telefon == '':
            telefon = None
        data_urodzenia = request.form.get('data_urodzenia')
        if data_urodzenia == '':
            data_urodzenia = None
        kraj = request.form.get('kraj')
        if kraj == '':
            kraj = None
        miasto = request.form.get('miasto')
        if miasto == '':
            miasto = None
        ulica = request.form.get('ulica')
        if ulica == '':
            ulica = None
        numer_domu = request.form.get('numer_domu')
        if numer_domu == '':
            numer_domu = None
        kod_pocztowy = request.form.get('kod_pocztowy')
        if kod_pocztowy == '':
            kod_pocztowy = None

        # Sprawdzenie, czy użytkownik o podanym emailu już istnieje
        # w bazie danych
        result = fun.execute_query('SELECT COUNT(*) FROM uzytkownicy //
                                    WHERE e_mail=?', (email,))
```

```
            if result[0][''] > 0:
                error_msg = 'Użytkownik o podanym e-mailu już istnieje'
                return render_template('rejestracja.html', error=error_msg)

            # Sprawdzenie, czy hasło spełnia wymagania
            if len(password) < 5 or len(password) > 15 //
                or not any(c.isdigit() for c in password) //
                or not any(c.isalnum() for c in password):

                error_msg = 'Hasło powinno zawierać od 5 do 15 znaków, jeden //
                            znak specjalny i jedną cyfrę'
                return render_template('rejestracja.html', error=error_msg)

            # Wywołanie procedurę składowaną DodajUzytkownika
            query = "EXEC DodajUzytkownika ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?"
            params = (email, password, imie, nazwisko, plec, telefon, //
                data_urodzenia, kraj, miasto, ulica, numer_domu, kod_pocztowy)

            fun.execute_sql_query(query, params)

            msg = 'Rejestracja przebiegła pomyślnie. Możesz teraz się zalogować'
            return render_template('login.html', rejestracja=msg)

        return render_template('rejestracja.html')
```

Skrypt 39: Fragment kodu w Python implementującego rejestrację nowego użytkownika.

A.5 Panel użytkownika

```
@app.route('/moje-konto')
def dashboard():
    """
    Strona dostępna po zalogowaniu
    """
    # pobieranie danych użytkownika
    user_data = fun.user_data(email=None, id_user=session['id_user'])
    if user_data:
        return render_template('moje-konto.html', user_data=user_data)
```

Skrypt 40: Fragment kodu w Python implementującego panel użytkownika.

A.6 Dane użytkownika i ich edycja

```
@app.route('/rejestracja', methods=['GET', 'POST'])
@app.route('/moje-dane')
def user_data():
    """
    Strona zawierająca szczegółowe informacje o danym użytkowniku
    """
    # pobieranie danych użytkownika
    user_data = fun.user_data(email=None, id_user=session['id_user'])
    if user_data:
        return render_template('moje-dane.html', user_data=user_data)

@app.route('/usun-konto', methods=['POST'])
def delet_account():
    user_id = session.get('id_user')

    if user_id:
        user_data = fun.user_data(email=None, id_user=session['id_user'])
        new_email=f"[deleted]_{time.time()}_{user_data['e_mail']}"
        query = f"UPDATE uzytkownicy SET data_usuniecia=?, e_mail=? //
                WHERE id_uzytkownika=?"
        params = (datetime.now(), new_email, session['id_user'])
        fun.execute_sql_query(query, params)
    return redirect('/wyloguj')

@app.route('/edytuj-dane', methods=['GET', 'POST'])
def edit_data():
    """
    Strona pozwalająca na edycję danych
    """
    if request.method == 'POST':
        # pobieranie danych użytkownika
        user_data = fun.user_data(email=None, id_user=session['id_user'])

        # pobieranie danych z formularza
        password = request.form.get('password')
        imie = request.form.get('imie')
        if imie == '':
            imie = None
        nazwisko = request.form.get('nazwisko')
        if nazwisko == '':
```

```
        nazwisko = None
    plec = request.form.get('plec')
    if plec == '' or plec == 'Wybierz':
        plec = None
    telefon = request.form.get("telefon")
    if telefon == '':
        telefon = None
    data_urodzenia = request.form.get('data_urodzenia')
    if data_urodzenia == '':
        data_urodzenia = None
    kraj = request.form.get('kraj')
    if kraj == '':
        kraj = None
    miasto = request.form.get('miasto')
    if miasto == '':
        miasto = None
    ulica = request.form.get('ulica')
    if ulica == '':
        ulica = None
    numer_domu = request.form.get('numer_domu')
    if numer_domu == '':
        numer_domu = None
    kod_pocztowy = request.form.get('kod_pocztowy')
    if kod_pocztowy == '':
        kod_pocztowy = None

    # sprawdzenie, czy hasło spełnia wymagania
    if password != '':
        if len(password) < 5 or len(password) > 15
            or not any(c.isdigit() for c in password)
            or not any(c.isalnum() for c in password):
            msg = 'Hasło powinno zawierać od 5 do 15 znaków, jeden //
                   znak specjalny i jedną cyfrę'
            return render_template('edytuj-dane.html', msg=msg)

    # wywołanie procedury składowanej EdytujUzytkownika
    query = "EXEC EdytujUzytkownika ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?"
    params = (session['id_user'], password, imie, nazwisko, plec, //
              telefon, data_urodzenia, kraj, miasto, ulica, //
              numer_domu, kod_pocztowy)
    fun.execute_sql_query(query, params)
```

```
            # pobranie danych użytkownika
            user_data = fun.user_data(email=None, id_user=session['id_user'])
            if user_data:
                msg = 'Zmiana danych przebiegła pomyślnie'
                return render_template('moje-dane.html', msg=msg, //
                                       user_data=user_data)

        user_data = fun.user_data(email=None, id_user=session['id_user'])
        return render_template('edytuj-dane.html', user_data=user_data)
```

Skrypt 41: Fragment kodu w Python implementującego panel z danymi użytkownika, usunięcie konta, oraz edycję danych użytkownika.

A.7 Lista kursów, oraz lista ulubionych kursów użytkownika

```
@app.route('/moje-kursy')
def my_courses():
    """
    Strona wyświetlająca zakupione kursy
    """
    # Pobranie zakupionych kursów z bazy danych
    cursor = fun.connect_to_database().cursor()
    cursor.execute(f"SELECT * FROM Vszczegoly_zamowien_kursy WHERE \\
                   id_uzytkownika={session['id_user']}")
    moje_kursy = cursor.fetchall()
    cursor.close()
    fun.connect_to_database().close()

    if 'id_user' in session:
        user_data = fun.user_data(email=None, id_user=session['id_user'])
        # Pobieranie ulubionych kursów użytkownika
        cursor = fun.connect_to_database().cursor()
        cursor.execute(f"SELECT * FROM ulubione_kursy WHERE \\
                       id_uzytkownika={session['id_user']}")
        ulubione_kursy = cursor.fetchall()
        cursor.close()
        fun.connect_to_database().close()

        ulubione_kursy_ids = [tup[0] for tup in ulubione_kursy]
        return render_template('moje-kursy.html', moje_kursy=moje_kursy, //
            user_data=user_data, ulubione_kursy_ids=ulubione_kursy_ids)
```

```
    return render_template('kursy.html', moje_kursy=moje_kursy)

@app.route('/dodaj-do-ulubionych', methods=['POST'])
def add_to_favorites():
    """
    Funkcjonalność dodania danego kursu do ulubionych
    """
    if 'id_user' in session:
        id_zamowienia_kursy = request.form.get('id_zamowienia_kursy')
        id_user = session['id_user']
        # Sprawdzenie, czy kurs już istnieje w tabeli ulubione_kursy
        cursor = fun.connect_to_database().cursor()

        cursor.execute(f"SELECT * FROM ulubione_kursy WHERE //
                       id_uzytkownika={id_user} AND //
                       id_zamowienia_kursy={id_zamowienia_kursy}")
        existing_favorite = cursor.fetchone()
        cursor.close()
        fun.connect_to_database().close()

        if existing_favorite:
            flash("Kurs jest już w ulubionych.")
        else:
            # Dodanie kursu do ulubionych
            fun.execute_query("INSERT INTO ulubione_kursy //
                              (id_zamowienia_kursy, id_uzytkownika) //
                              VALUES (?, ?)", (id_zamowienia_kursy, id_user))
            return redirect(url_for('my_courses'))

    return redirect('/moje-kursy')

@app.route('/odtworz/<int:id_zamowienia_kursy>')
def play(id_zamowienia_kursy):
    """
    Odtworzenie kursu z adresu URL
    """
    # Pobierz URL z bazy danych na podstawie kurs_id
    cursor = fun.connect_to_database().cursor()
    cursor.execute('SELECT adres_url FROM Vszczegoly_zamowien_kursy //
                   WHERE id_zamowienia_kursy = ?', (id_zamowienia_kursy,))
    adres_url = cursor.fetchone()[0]
    return render_template('odtworz-kurs.html', adres_url=adres_url)
```

```
@app.route('/ulubione-kursy')
def favorite_courses():
    """
    Strona zawierająca listę ulubionych kursów
    """
    if 'id_user' in session:
        id_user = session['id_user']

        # Pobranie ulubionych kursów użytkownika z bazy danych
        cursor = fun.connect_to_database().cursor()
        cursor.execute(f"SELECT * FROM Vulubione_kursy WHERE //
                       id_uzytkownika={id_user}")
        ulubione_kursy = cursor.fetchall()
        cursor.close()
        fun.connect_to_database().close()

        print(ulubione_kursy)

        return render_template('ulubione-kursy.html', //
                               ulubione_kursy=ulubione_kursy)
    # Przekierowanie do strony logowania,
    # jeśli użytkownik nie jest zalogowany
    return redirect('/login')


@app.route('/usun-ulubiony-kurs', methods=['POST'])
def remove_favorite_course():
    """
    Funkcjonalność usuwająca ulubiony kurs z bazy danych
    """
    if 'id_user' in session:
        id_zamowienia_kursy = request.form.get('id_zamowienia_kursy')
        redirect_page = request.form.get('redirect_page')
        id_user = session['id_user']

        # Usuwanie rekordu w tabeli ulubione_kursy
        fun.execute_sql_query("DELETE FROM ulubione_kursy WHERE //
                              id_uzytkownika=? AND id_zamowienia_kursy=?",
                              (id_user, id_zamowienia_kursy))

        flash("Kurs został usunięty z ulubionych.")
```

```
        if redirect_page == 'ulubione-kursy':
            return redirect('/ulubione-kursy')
        elif redirect_page == 'moje-kursy':
            return redirect('/moje-kursy')
```

Skrypt 42: Fragment kodu w Python implementującego listę zakupionych i opłaconych kursów przez użytkownika, odtworzenie kursu, dodanie kursu do ulubionych i usunięcie kursu z ulubionych.

```
@app.route('/ulubione-kursy')
def favorite_courses():
    """
    Strona zawierająca listę ulubionych kursów
    """
    if 'id_user' in session:
        id_user = session['id_user']

        # Pobranie ulubionych kursów użytkownika z bazy danych
        cursor = fun.connect_to_database().cursor()
        cursor.execute(f"SELECT * FROM Vulubione_kursy WHERE //
                       id_uzytkownika={id_user}")
        ulubione_kursy = cursor.fetchall()
        cursor.close()
        fun.connect_to_database().close()

        print(ulubione_kursy)

        return render_template('ulubione-kursy.html', //
                               ulubione_kursy=ulubione_kursy)
    # Przekierowanie do strony logowania,
    # jeśli użytkownik nie jest zalogowany
    return redirect('/login')
```

Skrypt 43: Fragment kodu w Python implementującego stronę z listą ulubionych kursów użytkownika.

B. Załącznik: skrypt tworzący bazę danych

```
USE [u_mdabrows]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[kursy](
  [id_kursu] [int] IDENTITY(1,1) NOT NULL,
  [nazwa] [varchar](200) NOT NULL,
  [autor] [varchar](100) NOT NULL,
  [cena] [decimal](10, 2) NOT NULL,
  [data_dodania] [date] NOT NULL,
  [data_modyfikacji] [date] NULL,
  [czas_trwania] [time](0) NOT NULL,
  [adres_url] [varchar](max) NOT NULL,
 CONSTRAINT [PK_kursu] PRIMARY KEY CLUSTERED
(
  [id_kursu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, PTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [nazwa] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, PTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE TABLE [dbo].[poziomy_kursow](
    [id_poziomu] [int] IDENTITY(1,1) NOT NULL,
    [nazwa_poziomu] [varchar](300) NOT NULL,
 CONSTRAINT [PK_kursy_poziomy] PRIMARY KEY CLUSTERED
(
    [id_poziomu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [nazwa_poziomu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]
GO


SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[kursy_poziomy](
    [id_kursu] [int] NOT NULL,
    [id_poziomu] [int] NOT NULL,
 CONSTRAINT [PK_kursy_poziomyy] PRIMARY KEY CLUSTERED
(
    [id_kursu] ASC,
    [id_poziomu] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vkursy_i_poziomy] AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy k
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow p ON kp.id_poziomu = p.id_poziomu;
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vkursy_podstawa] AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy k
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow p ON kp.id_poziomu = p.id_poziomu
WHERE p.nazwa_poziomu LIKE '%podstawa%';
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vkursy_rozszerzenie] AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy k
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow p ON kp.id_poziomu = p.id_poziomu
WHERE p.nazwa_poziomu LIKE '%rozszerzenie%';
GO
```

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vkursy_szkola_podstawowa] AS
SELECT k.id_kursu, k.nazwa, p.nazwa_poziomu
FROM kursy k
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow p ON kp.id_poziomu = p.id_poziomu
WHERE p.nazwa_poziomu LIKE '%SP%'
      OR p.nazwa_poziomu LIKE '%Egzamin 8%';
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[uzytkownicy](
  [id_uzytkownika] [int] IDENTITY(1,1) NOT NULL,
  [e_mail] [varchar](200) NOT NULL,
  [haslo] [binary](64) NOT NULL,
  [imie] [varchar](100) NOT NULL,
  [nazwisko] [varchar](200) NOT NULL,
  [plec] [varchar](10) NULL,
  [telefon] [varchar](25) NULL,
  [data_urodzenia] [date] NULL,
  [kraj] [varchar](30) NULL,
  [miasto] [varchar](30) NULL,
  [ulica] [varchar](100) NULL,
  [numer_domu] [varchar](10) NULL,
  [kod_pocztowy] [varchar](10) NULL,
  [uprawnienia_korepetytora] [bit] NOT NULL,
  [data_utworzenia] [date] NOT NULL,
  [ostatnie_logowanie] [date] NOT NULL,
  [data_usuniecia] [date] NULL,
 CONSTRAINT [PK_uzytkownicy] PRIMARY KEY CLUSTERED
(
    [id_uzytkownika] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
```

```
        IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS = ON,
        ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [e_mail] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE = OFF,
        IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
        ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vaktywni_uzytkownicy] AS
SELECT id_uzytkownika, imie, nazwisko, telefon, data_urodzenia,
        kraj, miasto, ulica, numer_domu, kod_pocztowy,
        data_utworzenia, ostatnie_logowanie
FROM uzytkownicy
WHERE data_usuniecia IS NULL AND uprawnienia_korepetytora = 0;
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vkorepetytorzy] AS
SELECT id_uzytkownika, imie, nazwisko, telefon, data_urodzenia,
        kraj, miasto, ulica, numer_domu, kod_pocztowy,
        data_utworzenia, ostatnie_logowanie
FROM uzytkownicy
WHERE uprawnienia_korepetytora = 1 AND data_usuniecia IS NULL;
GO
```

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[korepetycje](
  [id_korepetycji] [int] IDENTITY(1,1) NOT NULL,
  [id_korepetytor] [int] NOT NULL,
  [data] [date] NOT NULL,
  [start] [time](7) NOT NULL,
  [stop] [time](7) NOT NULL,
 CONSTRAINT [PK_korepetycje] PRIMARY KEY CLUSTERED
(
    [id_korepetycji] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vczas_korepetycji] AS
SELECT id_korepetycji, id_korepetytor, data, start, stop,
       DATEDIFF(MINUTE, start, stop) AS czas_trwania
FROM korepetycje;
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE VIEW [dbo].[Vnieaktywni_uzytkownicy] AS
SELECT id_uzytkownika, imie, nazwisko, telefon, data_urodzenia,
       kraj, miasto, ulica, numer_domu, kod_pocztowy,
       data_utworzenia, ostatnie_logowanie, data_usuniecia
FROM uzytkownicy
WHERE data_usuniecia IS NOT NULL
    AND uprawnienia_korepetytora = 0;
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[zamowienia_kursy](
    [id_zamowienia_kursy] [int] IDENTITY(1,1) NOT NULL,
    [id_kursu] [int] NOT NULL,
    [id_uzytkownika] [int] NOT NULL,
    [data_zlozenia] [date] NOT NULL,
    [data_oplacenia] [date] NULL,
    [typ_platnosci] [varchar](100) NULL,
 CONSTRAINT [PK_zamowienia_kursy] PRIMARY KEY CLUSTERED
(
    [id_zamowienia_kursy] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vzamowienia_kursy_nieoplacone] AS
SELECT a.id_zamowienia_kursy, a.data_zlozenia, b.id_kursu,
       b.nazwa, c.id_uzytkownika, c.imie, c.nazwisko, c.e_mail,
       a.typ_platnosci, a.data_oplacenia, b.cena
FROM zamowienia_kursy a
```

```
JOIN kursy b ON b.id_kursu = a.id_kursu
JOIN uzytkownicy c ON c.id_uzytkownika = a.id_uzytkownika
WHERE a.data_oplacenia IS NULL;
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[zamowienia_korepetycje](
  [id_zamowienia_korepetycje] [int] IDENTITY(1,1) NOT NULL,
  [id_korepetycje] [int] NOT NULL,
  [id_uzytkownika] [int] NOT NULL,
  [data_zamowienia] [date] NOT NULL,
  [data_oplacenia] [date] NULL,
  [typ_platnosci] [varchar](20) NULL,
  [cena] [decimal](10, 2) NOT NULL,
  [uwagi_ucznia] [varchar](400) NULL,
  [uwagi_korepetytora] [varchar](400) NULL,
 CONSTRAINT [PK_zamowienia_korepetycje] PRIMARY KEY CLUSTERED
(
    [id_zamowienia_korepetycje] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
ON [PRIMARY],

UNIQUE NONCLUSTERED
(
    [id_korepetycje] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
 ON [PRIMARY]
) ON [PRIMARY]
GO

SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vzamowienia_korepetycje_nieoplacone] AS
SELECT a.id_zamowienia_korepetycje, a.data_zamowienia,
       a.typ_platnosci, a.cena, b.id_korepetytor, b.data,
       b.start, b.stop, c.id_uzytkownika, c.imie, c.nazwisko,
       c.e_mail, c2.imie AS imie_korepetytora,
       c2.nazwisko AS nazwisko_korepetytora
FROM zamowienia_korepetycje a
JOIN korepetycje b ON a.id_korepetycje = b.id_korepetycji
JOIN uzytkownicy c ON a.id_uzytkownika = c.id_uzytkownika
JOIN uzytkownicy c2 ON b.id_korepetytor = c2.id_uzytkownika
WHERE a.data_oplacenia IS NULL
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vilosc_zamowien_kursow_oplaconych] AS
SELECT zk.id_kursu, SUM(k.cena) cena_suma
FROM zamowienia_kursy zk
JOIN kursy k ON zk.id_kursu = k.id_kursu
GROUP BY zk.id_kursu
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vilosc_zamowien_korepetecyje_uzytkownika] AS
SELECT zk.id_uzytkownika,
       COUNT(id_zamowienia_korepetycje) AS ilosc_zamowien
FROM zamowienia_korepetycje zk
WHERE zk.data_oplacenia IS NOT NULL
GROUP BY zk.id_uzytkownika
GO
```

```
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vilosc_zamowien_kursy_uzytkownika] AS
SELECT zk.id_uzytkownika,
       COUNT(id_zamowienia_kursy) AS ilosc_zamowien
FROM zamowienia_kursy zk
WHERE zk.data_oplacenia IS NOT NULL
GROUP BY zk.id_uzytkownika
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vszczegoly_zamowien_kursy] AS
SELECT zk.id_zamowienia_kursy, zk.data_zlozenia,
       zk.id_kursu, zk.data_oplacenia, zk.typ_platnosci,
       k.nazwa, k.autor, k.adres_url, u.id_uzytkownika,
       u.imie, u.nazwisko
FROM zamowienia_kursy as zk
JOIN kursy k ON zk.id_kursu = k.id_kursu
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN uzytkownicy u ON zk.id_uzytkownika = u.id_uzytkownika
WHERE zk.data_oplacenia IS NOT NULL
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vszczegoly_zamowien_korepetycje] AS
SELECT zk.id_zamowienia_korepetycje, zk.data_zamowienia,
       zk.data_oplacenia, zk.typ_platnosci, zk.cena,
       zk.id_korepetycje, k.id_korepetytor, k.data,
```

```
       k.start, k.stop, c2.imie as imie_korepetytora,
       c2.nazwisko AS nazwisko_korepetytora,
       u.id_uzytkownika, u.imie, u.nazwisko
FROM zamowienia_korepetycje as zk
JOIN korepetycje k ON zk.id_korepetycje = k.id_korepetycji
JOIN uzytkownicy u ON zk.id_uzytkownika = u.id_uzytkownika
JOIN uzytkownicy c2 ON k.id_korepetytor = c2.id_uzytkownika
WHERE zk.data_oplacenia IS NOT NULL
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vkursy_poziomy_widok] AS
SELECT k.id_kursu, k.nazwa, k.autor, k.cena,
       k.data_dodania, k.czas_trwania, k.adres_url,
       STRING_AGG(pk.nazwa_poziomu, ', ') AS poziomy_kursow
FROM kursy k
JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
JOIN poziomy_kursow pk ON kp.id_poziomu = pk.id_poziomu
GROUP BY k.id_kursu, k.nazwa, k.autor, k.cena, k.data_dodania,
         k.czas_trwania, k.adres_url;
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE TABLE [dbo].[ulubione_kursy](
    [id_zamowienia_kursy] [int] NOT NULL,
    [id_uzytkownika] [int] NOT NULL,
 CONSTRAINT [PK_ulubione_kursy] PRIMARY KEY CLUSTERED
(
    [id_zamowienia_kursy] ASC,
    [id_uzytkownika] ASC
)WITH (PAD_INDEX=OFF, STATISTICS_NORECOMPUTE=OFF,
       IGNORE_DUP_KEY=OFF, ALLOW_ROW_LOCKS=ON,
       ALLOW_PAGE_LOCKS=ON, OPTIMIZE_FOR_SEQUENTIAL_KEY=OFF)
```

```
 ON [PRIMARY]
) ON [PRIMARY]
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE VIEW [dbo].[Vulubione_kursy] AS
SELECT uk.id_zamowienia_kursy, uk.id_uzytkownika,
       k.nazwa, k.autor, k.data_dodania,
       k.czas_trwania, k.adres_url
FROM ulubione_kursy uk
JOIN zamowienia_kursy zk
    ON uk.id_zamowienia_kursy = zk.id_zamowienia_kursy
JOIN kursy k on zk.id_kursu = k.id_kursu
GO

ALTER TABLE [dbo].[uzytkownicy] ADD DEFAULT ((0)) FOR
[uprawnienia_korepetytora]
GO

ALTER TABLE [dbo].[korepetycje] WITH CHECK ADD CONSTRAINT
[FK_korepetycje_uzytkownicy] FOREIGN KEY([id_korepetytor])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])
GO

ALTER TABLE [dbo].[korepetycje] CHECK CONSTRAINT
[FK_korepetycje_uzytkownicy]
GO

ALTER TABLE [dbo].[kursy_poziomy] WITH CHECK ADD CONSTRAINT
[FK_kursy_poziomy_kursy] FOREIGN KEY([id_kursu])
REFERENCES [dbo].[kursy] ([id_kursu])
GO

ALTER TABLE [dbo].[kursy_poziomy] CHECK CONSTRAINT
[FK_kursy_poziomy_kursy]
GO

ALTER TABLE [dbo].[kursy_poziomy] WITH CHECK ADD CONSTRAINT
```

```
[FK_kursy_poziomy_poziomy_kursow] FOREIGN KEY([id_poziomu])
REFERENCES [dbo].[poziomy_kursow] ([id_poziomu])
GO

ALTER TABLE [dbo].[kursy_poziomy] CHECK CONSTRAINT
[FK_kursy_poziomy_poziomy_kursow]
GO

ALTER TABLE [dbo].[ulubione_kursy] WITH CHECK ADD  CONSTRAINT
[FK_ulubione_kursy_uzytkownicy] FOREIGN KEY([id_uzytkownika])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])
GO

ALTER TABLE [dbo].[ulubione_kursy] CHECK CONSTRAINT
[FK_ulubione_kursy_uzytkownicy]
GO

ALTER TABLE [dbo].[ulubione_kursy] WITH CHECK ADD  CONSTRAINT
[FK_ulubione_kursy_zamowienia_kursy]
FOREIGN KEY ([id_zamowienia_kursy])
REFERENCES [dbo].[zamowienia_kursy] ([id_zamowienia_kursy])
GO

ALTER TABLE [dbo].[ulubione_kursy] CHECK CONSTRAINT
[FK_ulubione_kursy_zamowienia_kursy]
GO

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK ADD
CONSTRAINT [FK_zamowienia_korepetycje_korepetycje]
FOREIGN KEY([id_zamowienia_korepetycje])
REFERENCES [dbo].[korepetycje] ([id_korepetycji])
GO

ALTER TABLE [dbo].[zamowienia_korepetycje] CHECK CONSTRAINT
[FK_zamowienia_korepetycje_korepetycje]
GO

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK ADD
CONSTRAINT [FK_zamowienia_korepetycje_uzytkownicy]
FOREIGN KEY([id_uzytkownika])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])
GO
```

```
ALTER TABLE [dbo].[zamowienia_korepetycje] CHECK CONSTRAINT
[FK_zamowienia_korepetycje_uzytkownicy]
GO

ALTER TABLE [dbo].[zamowienia_kursy] WITH CHECK ADD
CONSTRAINT [FK_zamowienia_kursy_kursy]
FOREIGN KEY([id_kursu])
REFERENCES [dbo].[kursy] ([id_kursu])
GO

ALTER TABLE [dbo].[zamowienia_kursy] CHECK CONSTRAINT
[FK_zamowienia_kursy_kursy]
GO

ALTER TABLE [dbo].[zamowienia_kursy] WITH CHECK ADD CONSTRAINT
[FK_zamowienia_kursy_uzytkownicy]
FOREIGN KEY([id_uzytkownika])
REFERENCES [dbo].[uzytkownicy] ([id_uzytkownika])
GO

ALTER TABLE [dbo].[zamowienia_kursy] CHECK CONSTRAINT
[FK_zamowienia_kursy_uzytkownicy]
GO

ALTER TABLE [dbo].[kursy] WITH CHECK ADD CHECK
(([data_dodania]<=getdate()))
GO

ALTER TABLE [dbo].[kursy] WITH CHECK ADD CHECK
(([data_modyfikacji]<=getdate()))
GO

ALTER TABLE [dbo].[kursy] WITH CHECK ADD CHECK
(([data_modyfikacji]<=getdate()))
GO

ALTER TABLE [dbo].[kursy] WITH CHECK ADD CHECK
(([data_modyfikacji]<=getdate()))
GO

ALTER TABLE [dbo].[kursy] WITH CHECK ADD CHECK
(([data_modyfikacji]<=getdate()))
GO
```

```
ALTER TABLE [dbo].[kursy] WITH CHECK ADD  CONSTRAINT
[sprawdzenie_url] CHECK  (([adres_url] like 'https://%'))
GO

ALTER TABLE [dbo].[kursy] CHECK CONSTRAINT [sprawdzenie_url]
GO

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([data_urodzenia]<=getdate() AND
[data_urodzenia]>=dateadd(year,(-120),getdate())))
GO

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([data_utworzenia]<=getdate()))
GO

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([data_usuniecia]<=getdate()))
GO

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([kod_pocztowy] like '[0-9][0-9]-[0-9][0-9][0-9]'))
GO

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([ostatnie_logowanie]<=getdate()))
GO

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([telefon] like
'[0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]')
   )
GO

ALTER TABLE [dbo].[uzytkownicy] WITH CHECK ADD CHECK
(([plec]='Inna' OR [plec]='Mezczyzna' OR [plec]='Kobieta'))
GO

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CHECK  (([data_zamowienia]<=getdate()))
GO
```

```
ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CHECK  (([data_oplacenia]<=getdate()))
GO

ALTER TABLE [dbo].[zamowienia_korepetycje] WITH CHECK
ADD CHECK (([typ_platnosci]='BLIK' OR [typ_platnosci]='PayU'
OR [typ_platnosci]='przelew tradycyjny'))
GO

ALTER TABLE [dbo].[zamowienia_kursy] WITH CHECK ADD CHECK
(([data_zlozenia]<=getdate()))
GO

ALTER TABLE [dbo].[zamowienia_kursy]  WITH CHECK ADD CHECK
(([data_oplacenia]<=getdate()))
GO

ALTER TABLE [dbo].[zamowienia_kursy]  WITH CHECK ADD CHECK
(([data_zlozenia]<=getdate()))
GO

ALTER TABLE [dbo].[zamowienia_kursy]  WITH CHECK ADD CHECK
(([data_oplacenia]<=getdate()))
GO

ALTER TABLE [dbo].[zamowienia_kursy]  WITH CHECK ADD CHECK
(([typ_platnosci]='BLIK' OR [typ_platnosci]='PayU'
OR [typ_platnosci]='przelew tradycyjny'))
GO

ALTER TABLE [dbo].[zamowienia_kursy]  WITH CHECK ADD CHECK
(([typ_platnosci]='BLIK' OR [typ_platnosci]='PayU'
OR [typ_platnosci]='przelew tradycyjny'))
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
```

```
CREATE PROCEDURE [dbo].[AktualizujCeneKorepetycji]
    @IDKorepetycji int,
    @Procent decimal(5,2)
AS
BEGIN
    DECLARE @stara_cena decimal(10,2);
    DECLARE @nowa_cena decimal(10,2);

    -- pobranie ceny kursu
    SELECT @stara_cena = cena FROM zamowienia_korepetycje
    WHERE id_korepetycje = @IDKorepetycji

    -- obliczenie nowej ceny
    SELECT @nowa_cena = @stara_cena * (1 + @Procent/100);

    -- aktualizacja
    UPDATE zamowienia_korepetycje SET cena = @nowa_cena
    WHERE id_korepetycje = @IDKorepetycji
END
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[AktualizujCeneKursu]
    @IDKursu int,
    @Procent decimal(5,2)
AS
BEGIN
    DECLARE @stara_cena decimal(10,2);
    DECLARE @nowa_cena decimal(10,2);

    -- pobranie ceny kursu
    SELECT @stara_cena = cena FROM kursy
    WHERE id_kursu = @IDKursu

    -- obliczenie nowej ceny
    SELECT @nowa_cena = @stara_cena * (1 + @Procent/100);
```

```
    -- aktualizacja
    UPDATE kursy SET cena = @nowa_cena
    WHERE id_kursu = @IDKursu
END
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[DodajKursDoUlubionych]
    @IdUzytkownika int,
    @IdZamowieniaKurs int
AS
BEGIN
    INSERT INTO
        ulubione_kursy (id_zamowienia_kursy, id_uzytkownika)
    VALUES (@IdUzytkownika, @IdZamowieniaKurs)
END
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO


CREATE PROCEDURE [dbo].[DodajUzytkownika]
    @email VARCHAR(200),
    @haslo VARCHAR(100),
    @imie VARCHAR(200) = NULL,
    @nazwisko VARCHAR(200) = NULL,
    @plec VARCHAR(10) = NULL,
    @telefon VARCHAR(25) = NULL,
    @data_urodzenia DATE = NULL,
    @kraj VARCHAR(30) = NULL,
    @miasto VARCHAR(30) = NULL,
    @ulica VARCHAR(100) = NULL,
    @numer_domu VARCHAR(10) = NULL,
```

```
    @kod_pocztowy VARCHAR(10) = NULL,
    @uprawnienia_korepetytora BIT = 0,
    @data_usuniecia DATE = NULL
AS
BEGIN
    -- dodaje szyfrowanie hasla
    DECLARE @zaszyfrowane_haslo VARBINARY(8000) =
            HASHBYTES('SHA2_512', @haslo);

    INSERT INTO uzytkownicy (e_mail, haslo, imie, nazwisko,
                plec, telefon, data_urodzenia, kraj,
                miasto, ulica, numer_domu,
                kod_pocztowy, uprawnienia_korepetytora,
                data_utworzenia, ostatnie_logowanie,
                data_usuniecia)
    VALUES (@email, @zaszyfrowane_haslo, @imie, @nazwisko,
            @plec, @telefon, @data_urodzenia, @kraj,
            @miasto, @ulica, @numer_domu, @kod_pocztowy,
            @uprawnienia_korepetytora, GETDATE(),
            GETDATE(), @data_usuniecia);
END
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[EdytujUzytkownika]
    @id int,
    @haslo VARCHAR(100) = NULL,
    @imie VARCHAR(200) = NULL,
    @nazwisko VARCHAR(200) = NULL,
    @plec VARCHAR(10) = NULL,
    @telefon VARCHAR(25) = NULL,
    @data_urodzenia DATE = NULL,
    @kraj VARCHAR(30) = NULL,
    @miasto VARCHAR(30) = NULL,
    @ulica VARCHAR(100) = NULL,
    @numer_domu VARCHAR(10) = NULL,
    @kod_pocztowy VARCHAR(10) = NULL
AS
```

```
BEGIN
    DECLARE @zaszyfrowane_haslo VARBINARY(8000) =
            HASHBYTES('SHA2_512', @haslo);

    UPDATE uzytkownicy
    SET
     imie = ISNULL(@imie, imie),
     nazwisko = ISNULL(@nazwisko, nazwisko),
     haslo = ISNULL(@zaszyfrowane_haslo, haslo),
     plec = CASE WHEN @plec IS NULL THEN NULL ELSE
      ISNULL(@plec, plec) END,
     telefon = CASE WHEN @telefon IS NULL THEN NULL ELSE
      ISNULL(@telefon, telefon) END,
     data_urodzenia = CASE WHEN @data_urodzenia IS NULL THEN
      NULL ELSE ISNULL(@data_urodzenia, data_urodzenia) END,
     kraj = CASE WHEN @kraj IS NULL THEN NULL ELSE
       ISNULL(@kraj, kraj) END,
     miasto = CASE WHEN @miasto IS NULL THEN NULL ELSE
      ISNULL(@miasto, miasto) END,
     ulica = CASE WHEN @ulica IS NULL THEN NULL ELSE
      ISNULL(@ulica, ulica) END,
     numer_domu = CASE WHEN @numer_domu IS NULL THEN NULL
      ELSE ISNULL(@numer_domu, numer_domu) END,
     kod_pocztowy = CASE WHEN @kod_pocztowy IS NULL THEN
      NULL ELSE ISNULL(@kod_pocztowy, kod_pocztowy) END
    WHERE id_uzytkownika = @id;
END
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[LiczbaZamowienKorepetycjeCzas]
    @IDKorepetycje int,
    @DataOd datetime,
    @DataDo datetime
AS
BEGIN
    SELECT COUNT(*) AS liczba_zamowien, YEAR(
   zamowienia_korepetycje.data_oplacenia) AS rok,
```

```
        MONTH(zamowienia_korepetycje.data_oplacenia) AS miesiac
        FROM zamowienia_korepetycje
        WHERE
            zamowienia_korepetycje.id_korepetycje = @IDKorepetycje
        AND zamowienia_korepetycje.data_oplacenia
            BETWEEN @DataOd AND @DataDo
        GROUP BY YEAR(zamowienia_korepetycje.data_oplacenia),
                 MONTH(zamowienia_korepetycje.data_oplacenia)
    END
    GO

    SET ANSI_NULLS ON
    GO

    SET QUOTED_IDENTIFIER ON
    GO

    CREATE PROCEDURE [dbo].[LiczbaZamowienKursuCzas]
        @IDKursu int,
        @DataOd datetime,
        @DataDo datetime
    AS
    BEGIN
        SELECT COUNT(*) AS liczba_zamowien,
            YEAR(zamowienia_kursy.data_oplacenia) AS rok,
            MONTH(zamowienia_kursy.data_oplacenia) AS miesiac
        FROM zamowienia_kursy
        WHERE zamowienia_kursy.id_kursu = @IDKursu
        AND zamowienia_kursy.data_oplacenia
            BETWEEN @DataOd AND @DataDo
        GROUP BY YEAR(zamowienia_kursy.data_oplacenia),
        MONTH(zamowienia_kursy.data_oplacenia)
    END
    GO

    SET ANSI_NULLS ON
    GO

    SET QUOTED_IDENTIFIER ON
    GO
```

```
CREATE PROCEDURE [dbo].[SzukajKursowPoPoziomie]
    @IDPoziomu int
AS
BEGIN
    SELECT pk.nazwa_poziomu, k.nazwa AS nazwa_kursu, k.autor,
           k.cena, k.adres_url
    FROM kursy k
    JOIN kursy_poziomy kp ON k.id_kursu = kp.id_kursu
    JOIN poziomy_kursow pk ON kp.id_poziomu = pk.id_poziomu
    WHERE pk.id_poziomu = @IDPoziomu
END
GO

SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO

CREATE PROCEDURE [dbo].[UsunNieoplaconeZamowienia]
AS
BEGIN
    DELETE FROM zamowienia_kursy
    WHERE data_oplacenia IS NULL
    AND DATEDIFF(day, data_zlozenia, GETDATE()) > 30

    DELETE FROM zamowienia_korepetycje
    WHERE data_oplacenia IS NULL
    AND DATEDIFF(day, data_zamowienia, GETDATE()) > 30
END
GO
```

Skrypt 44: Skrypt generujący bazę danych aplikacji.

Literatura

[1] SQL Server firmy Microsoft, https://microsoft.com, [dostęp 04.05.2023].

[2] Aplikacja internetowa Figma, https://www.figma.com/ [dostęp 15.05.2023].

[3] Framework Flask, https://flask.palletsprojects.com/en/2.3.x/ [dostęp 15.05.2023].

[4] Aplikacja Diagrams.net, https://app.diagrams.net/, [dostęp: 15.05.2023]

[5] Bernard Maj, „Studia podyplomowe - Systemy Baz Danych. Analiza i modelowanie wymagań przy użyciu wybranych metod inżynierii oprogramowania".

[6] Definicja widoków, https://www.ibm.com/docs/pl/i/7.2?topic=sql-creating-using-view, [dostęp: 08.05.2023].

[7] Definicja prodecur składowanych, https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16, [dostęp: 08.05.2023].

[8] Algorytm SHA2, https://justcryptography.com/sha-2/, [dostęp: 08.05.2023].

[9] Definicja Triggeru, https://docs.oracle.com/cd/A57673_01/DOC/server/doc/SCN73/ch15.html, [dostęp: 09.05.2023].

[10] Biblioteka PyODBC, https://pypi.org/project/pyodbc/, [dostęp: 16.05.2023].

[11] Szablony Jinja, https://jinja.palletsprojects.com/en/3.1.x/, [dostęp: 16.05.2023].