

Dokumentacja Projektu

Bazy Danych 2

Jan Pazdan

13 Czerwca 2025

1 Zdefiniowanie tematu projektu

Przetwarzanie własnych typów danych CLR UDT. Opracować API oraz jego implementację obsługującą wybrany zestaw typów własnych UDT w technologii CLR do obsługi złożonych danych pobieranych z różnych źródeł danych. Opracowane API powinno umożliwić wprowadzanie danych do zdefiniowanych struktur, wyszukiwanie danych w opracowanych strukturach oraz tworzenie odpowiednich raportów, jak również informować o błędnym ich wykorzystaniu. W ramach zadania należy zaimplementować powyżej pięciu typów własnych UDT.

2 Opis problemu

Program SQL Server umożliwia tworzenie obiektów bazy danych, które są programowane względem zestawu utworzonego w środowisku programu .NET Framework (CLR). Obiekty bazy danych, które mogą korzystać z zaawansowanego modelu programowania udostępnianego przez CLR. System typów SQL można rozszerzyć poprzez definiowanie niestandardowego typu danych do wykorzystania w programowaniu dla SQL Server. Typ zdefiniowany przez użytkownika (UDT) może być prosty lub złożony, o dowolnym stopniu skomplikowania. Może on enkapsulować (zawierać) złożone, zdefiniowane przez użytkownika zachowania. Taki typ UDT jest implementowany jako zarządzana klasa (lub struktura) w jednym z języków CLR, a następnie rejestrowany w SQL Server.

3 Opis funkcjonalności udostępnianej przez API

Projekt pozwala na pozbycie się ograniczeń dla standardowych typów danych SQL Servera poprzez reprezentację typów złożonych które pozwalają na traktowanie ich jako pojedynczych, atomowych obiektów w bazie danych. Każdy z typów posiada swoje operacje, które wykraczają poza zdolność typów podstawowych.

Aplikacja dla typu *Email* umożliwia:

- Dodanie nowego użytkownika
- Wyświetlenie wszystkich użytkowników
- Wyszukanie użytkownika po części maila.
- Usunięcie użytkownika po ID.
- Dodanie nowych użytkowników poprzez plik CSV.

Aplikacja dla typu *Vector3D* umożliwia:

- Dodanie nowego wektora.
- Wyświetlenie wszystkich wektorów.
- Usunięcie wektora po ID.
- Dodanie dwóch wektorów.
- Odejmowanie dwóch wektorów.
- Mnożenie wektora przez skalar.

- Iloczyn skalarny dwóch wektorów.
- Iloczyn wektorowy dwóch wektorów.
- Dodanie nowych wektorów poprzez plik CSV.

Aplikacja dla typu *UnitSI* umożliwia:

- Dodanie nowej wartości jednostki.
- Wyświetlenie wszystkich jednostek.
- Dodanie dwóch jednostek.
- Odejmowanie dwóch jednostek.
- Pomnożenie dwóch jednostek.
- Podzielenie dwóch jednostek.
- Mnożenie jednostki przez skalar.
- Podzielenie jednostki przez skalar.
- Wyświetlenie jednostki z przedrostkiem.
- Dodanie nowych wartości jednostek poprzez plik CSV.

Aplikacja dla typu *GeoLocation* umożliwia:

- Dodanie nowej lokalizacji.
- Wyświetlenie wszystkich lokalizacji.
- Usunięcie lokalizacji po ID.
- Wyszukanie lokalizacji najbliższej do podanej.
- Dodanie nowych lokalizacji poprzez plik CSV.

Aplikacja dla typu *ColorRGB* umożliwia:

- Dodanie nowego koloru.
- Wyświetlenie wszystkich kolorów.
- Usunięcie koloru po ID.
- Obliczenie negatywu koloru.
- Zblendowanie dwóch kolorów.
- Dodanie nowych kolorów poprzez plik CSV.

Aplikacja dla typu *MoneyType* umożliwia:

- Dodanie nowej wartości pieniężnej.
- Wyświetlenie wszystkich wartości.
- Usunięcie wartości po ID.
- Przeliczenie wartości i waluty na inną.
- Dodanie nowych wartości poprzez plik CSV.

4 Opis typów danych oraz metod udostępnionych w ramach API

4.1 Email

Typ *Email* reprezentuje znaną strukturę adresu e-mail. Służy do rozdzielania w bazie standardowego typu VARCHAR i jego pochodnych od faktycznego adresu e-mail złożonego z nazwy użytkownika (local-part) oraz domeny przedzielonych znakiem @.

Typ *Email* jest złożony z pól:

- *public string username* - Przechowuje lokalną część adresu e-mail (ciąg znaków przed znakiem @). Wszystkie przechowywane nazwy użytkowników są automatycznie konwertowane na małe litery w celu normalizacji i ułatwienia porównywania.
- *public string domain* - Przechowuje część domeny adresu e-mail (ciąg znaków po znaku @). Podobnie jak *username*, domena jest konwertowana na małe litery.
- *private bool is_Null* - Prywatna flaga wskazująca, czy instancja *Email* reprezentuje wartość NULL w bazie danych SQL Server.

Posiada właściwości:

- *public bool IsNull* - Właściwość zgodna z interfejsem *INullable*. Zwraca *true*, jeśli obiekt *Email* reprezentuje wartość NULL, w przeciwnym razie *false*.
- *public static Email Null* - Statyczna właściwość zwracająca nową instancję *Email* ustawioną na wartość NULL.

Posiada metody:

- *static Email Parse(SqlString s)* - główny sposób tworzenia instancji typu *Email* z wartości string przekazanej z SQL Servera. Jest wywoływana automatycznie, gdy wstawia się string do kolumny typu *Email* w T-SQL (np. INSERT INTO Users (ContactEmail) VALUES ('test@example.com')).
- *static bool IsValidLocalPart(string local)* - Sprawdza poprawność składniową *username* (część lokalna).
- *bool IsValidDomain(string local)* - Sprawdza poprawność składniową *domain*.
- *bool ValidateEmailUdt()* - Metoda wywołuje dwa poprzednie sposoby walidacji danych, jest wywoływana przez SQL Server.
- *string ToString()* - Konwertuje instancję typu *Email* z powrotem na string w standardowym formacie *nazwa_uzytkownika@domena.com*. Jest to metoda wywoływana, poprzez zapytanie w T-SQL (np. SELECT ContactEmail.ToString() FROM Users).
- *void Read(BinaryReader w)* - Metoda jest wywoływana przez SQL Server, gdy instancja *Email* jest odczytywana z bazy danych.
- *void Write(BinaryWriter w)* - Metoda jest wywoływana przez SQL Server, gdy instancja *Email* jest zapisywana do bazy danych.
- *bool Equals(object obj)* - Porównuje *username* i *domain* obu obiektów bez uwzględniania wielkości liter, zgodnie z normalizacją stosowaną w metodzie *Parse*.
- *int GetHashCode()* - Generuje kod skrótu na podstawie wartości *username* i *domain* (po normalizacji na małe litery), zapewniając, że równe obiekty mają ten sam kod skrótu.

4.2 Vector3D

Typ *Vector3D* reprezentuje trójwymiarowy wektor w przestrzeni euklidesowej. Służy do przechowywania i wykonywania operacji na danych wektorowych bezpośrednio w bazie danych SQL Server.

Typ *Vector3D* jest złożony z pól:

- *public float **x*** - Reprezentuje współrzędną X wektora.
- *public float **y*** - Reprezentuje współrzędną Y wektora.
- *public float **z*** - Reprezentuje współrzędną Z wektora.
- *private bool **is_Null*** - Prywatna flaga wskazująca, czy instancja *Vector3D* reprezentuje wartość NULL w bazie danych SQL Server.

Posiada właściwości:

- *public bool **IsNull*** - Właściwość zgodna z interfejsem *INullable*. Zwraca *true*, jeśli obiekt *Vector3D* reprezentuje wartość NULL, w przeciwnym razie *false*.
- *public static Vector3D **Null*** - Statyczna właściwość zwracająca nową instancję *Vector3D* ustawioną na wartość NULL.

Posiada metody:

- ***Vector3D**(float x, float y, float z)* - Konstruktor tworzy nową instancję *Vector3D* z podanych współrzędnych.
- *static Vector3D **Parse**(SqlString s)* - Główny sposób tworzenia instancji typu *Vector3D* z wartości string przekazanej z SQL Servera. Jest wywoływana automatycznie, gdy wstawia się string do kolumny typu *Vector3D* w T-SQL (np. 'INSERT INTO Objects (Position) VALUES ('[1.0,2.5,3.0]')'). Oczekiwany format to '[x,y,z]'. Metoda wykonuje walidację formatu i poprawności numerycznej komponentów, rzucając wyjątek *ArgumentException* w przypadku błędu.
- *string **ToString**()* - Konwertuje instancję typu *Vector3D* z powrotem na string w standardowym formacie '[x,y,z]'. Jest to metoda wywoływana poprzez zapytanie w T-SQL (np. 'SELECT Position.ToString() FROM Objects'). Jeśli instancja jest NULL, zwraca string NULL.
- *static Vector3D **Add**(Vector3D v1, Vector3D v2)* - Wykonuje dodawanie dwóch wektorów ('v1 + v2'), zwracając nowy wektor, którego współrzędne są sumą odpowiednich współrzędnych wektorów wejściowych. Jeśli któryś z wektorów wejściowych jest NULL, metoda zwraca NULL.
- *static Vector3D **Subtract**(Vector3D v1, Vector3D v2)* - Wykonuje odejmowanie dwóch wektorów ('v1 - v2'), zwracając nowy wektor, którego współrzędne są różnicą odpowiednich współrzędnych wektorów wejściowych. Jeśli któryś z wektorów wejściowych jest NULL, metoda zwraca NULL.
- *static Vector3D **MultiplyByScalar**(Vector3D v, float scalar)* - Mnoży wektor przez skalar, zwracając nowy wektor, którego każda współrzędna została pomnożona przez podany skalar. Jeśli wektor wejściowy jest NULL, metoda zwraca NULL.
- *static SqlDouble **DotProduct**(Vector3D v1, Vector3D v2)* - Oblicza iloczyn skalarny (dot product) dwóch wektorów. Wynikiem jest pojedyncza wartość liczbową typu 'SqlDouble' (standardowy typ SQL Servera dla liczb zmiennoprzecinkowych podwójnej precyzji). Jeśli któryś z wektorów wejściowych jest NULL, metoda zwraca NULL.
- *static Vector3D **CrossProduct**(Vector3D v1, Vector3D v2)* - Oblicza iloczyn wektorowy (cross product) dwóch wektorów. Wynikiem jest nowy wektor prostopadły do obu wektorów wejściowych. Jeśli któryś z wektorów wejściowych jest NULL, metoda zwraca NULL.
- *void **Read**(BinaryReader r)* - Metoda jest wywoływana przez SQL Server, gdy instancja *Vector3D* jest odczytywana z bazy danych. Odpowiada za deserializację danych ze strumienia binarnego i odtworzenie obiektu *Vector3D*. Odczytywane są współrzędne X, Y, Z. W przypadku pustego strumienia, obiekt jest ustawiany na NULL.
- *void **Write**(BinaryWriter w)* - Metoda jest wywoływana przez SQL Server, gdy instancja *Vector3D* jest zapisywana do bazy danych. Odpowiada za serializację wewnętrznych pól obiektu do strumienia binarnego. Zapisywane są współrzędne X, Y, Z, o ile obiekt nie jest NULL.

- *bool Equals(object obj)* - Porównuje dwie instancje *Vector3D*. Zwraca *true*, jeśli oba wektory mają identyczne współrzędne X, Y i Z. Poprawnie obsługiwane jest porównywanie obiektów NULL.
- *int GetHashCode()* - Generowany jest kod skrótu dla instancji *Vector3D* na podstawie jej współrzędnych X, Y i Z, zapewniając, że równe obiekty mają ten sam kod skrótu. Obsługiwany jest również przypadek NULL.

4.3 UnitSI

Typ *UnitSI* odpowiada wartości liczbowej wraz z odpowiadającą im jednostką miary z Międzynarodowego Układu Jednostek Miar (SI). Są to "m", "kg", "s", "A", "K", "mol", "cd".

Typ *UnitSI* jest złożony z pól:

- *public double Value* - Przechowuje wartość liczbową wielkości fizycznej.
- *public string Unit* - Przechowuje symbol jednostki miary (np. "m" dla metra, "kg" dla kilograma).
- *private bool is_Null* - Prywatna flaga wskazująca, czy instancja *UnitSI* reprezentuje wartość NULL w bazie danych SQL Server.

Posiada właściwości:

- *public bool IsNull* - Właściwość zgodna z interfejsem *INullable*. Zwraca *true*, jeśli obiekt *UnitSI* reprezentuje wartość NULL, w przeciwnym razie *false*.
- *public static UnitSI Null* - Statyczna właściwość zwracająca nową instancję *UnitSI* ustawioną na wartość NULL.

Posiada metody:

- *UnitSI(double value, string unit)* - Konstruktor tworzący instancję *UnitSI* z podanej wartości i jednostki. Wykonuje podstawową walidację wartości (czy nie jest 'NaN' lub 'Infinity') oraz jednostki.
- *static UnitSI Parse(SqlString s)* - Główny sposób tworzenia instancji *UnitSI* ze stringa w formacie "wartość [jednostka]". Metoda parsowania i walidacji formatu, wartości liczbowej oraz przynależności jednostki do bazowych jednostek SI. W przypadku błędu rzuca wyjątek *ArgumentException*.
- *string ToString()* - Konwertuje instancję *UnitSI* na string w formacie "wartość [jednostka]". Liczby są formatowane z użyciem 'CultureInfo.InvariantCulture'. Jeśli instancja jest NULL, zwraca string NULL.
- *static UnitSI Add(UnitSI u1, UnitSI u2)* - Dodaje dwie wartości *UnitSI*. Wymaga identycznych jednostek; w przeciwnym razie rzuca wyjątek *ArgumentException*.
- *static UnitSI Subtract(UnitSI u1, UnitSI u2)* - Odejmowanie dwóch wartości *UnitSI*. Wymaga identycznych jednostek; w przeciwnym razie rzuca wyjątek *ArgumentException*.
- *static UnitSI Multiply(UnitSI u1, UnitSI u2)* - Mnoży dwie wartości *UnitSI*. Jednostki są łączone w formacie "jednostka1*jednostka2" z obsługą jednostki "none".
- *static UnitSI Divide(UnitSI u1, UnitSI u2)* - Dzieli dwie wartości *UnitSI*. Obsługuje dzielenie przez zero (rzuca 'DivideByZeroException'). Jednostki są łączone w formacie "jednostka1/jednostka2" z obsługą jednostki "none".
- *static UnitSI MultiplyByScalar(UnitSI u, double scalar)* - Mnoży wartość *UnitSI* przez skalar, zwracając nową instancję z pomnożoną wartością i oryginalną jednostką.
- *static UnitSI DivideByScalar(UnitSI u, double scalar)* - Dzieli wartość *UnitSI* przez skalar. Obsługuje dzielenie przez zero (rzuca 'DivideByZeroException'). Zwraca nową instancję z podzieloną wartością i oryginalną jednostką.

- *SqlString ToPrefixedString(SqlString prefix)* - Konwertuje wartość *UnitSI* na string z wartością przeliczoną na jednostkę z podanym przedrostkiem SI. Specjalna obsługa dla "kg" i pustego przedrostka ('1 kg' → '1000 g').
- *void Read(BinaryReader r)* - Deserializuje instancję *UnitSI* ze strumienia binarnego. Odczytywana jest flaga NULL, wartość ('double') i jednostka ('string').
- *void Write(BinaryWriter w)* - Serializuje instancję *UnitSI* do strumienia binarnego. Zapisywana jest flaga NULL, wartość ('double') i jednostka ('string').
- *bool Equals(object obj)* - Porównuje dwie instancje *UnitSI*. Zwraca *true*, jeśli mają zbliżone wartości i identyczne jednostki.
- *int GetHashCode()* - Generuje kod skrótu dla instancji *UnitSI* na podstawie jej wartości i jednostki. Obsługiwany jest również przypadek NULL.

4.4 GeoLocation

Typ *GeoLocation* reprezentuje punkt geograficzny na powierzchni Ziemi za pomocą szerokości i długości geograficznej.

Typ *GeoLocation* jest złożony z pól:

- *public double Latitude* - Przechowuje szerokość geograficzną. Zakres wartości: od -90° (biegun południowy) do 90° (biegun północny).
- *public double Longitude* - Przechowuje długość geograficzną. Wartości są automatycznie normalizowane do zakresu od -180° do 180° .
- *private bool is_Null* - Prywatna flaga wskazująca, czy instancja *GeoLocation* reprezentuje wartość NULL w bazie danych SQL Server.

Posiada właściwości:

- *public bool IsNull* - Właściwość zgodna z interfejsem *INullable*. Zwraca *true*, jeśli obiekt *GeoLocation* reprezentuje wartość NULL, w przeciwnym razie *false*.
- *public static GeoLocation Null* - Statyczna właściwość zwracająca nową instancję *GeoLocation* ustawioną na wartość NULL.

Posiada metody:

- *GeoLocation(double latitude, double longitude)* - Konstruktor tworzący instancję *GeoLocation* z podanej szerokości i długości geograficznej. Wykonuje walidację zakresu szerokości oraz normalizację długości geograficznej.
- *static GeoLocation Parse(SqlString s)* - Główny sposób tworzenia instancji *GeoLocation* ze stringa. Akceptuje formaty takie jak '(lat, lon)' lub 'lat (N/S), lon (E/W)'. Metoda wykonuje parsowanie wartości oraz konwersję kierunkową, rzucając wyjątek *ArgumentException* w przypadku błędu.
- *string ToString()* - Konwertuje instancję *GeoLocation* na string w formacie '(lat, lon)', z zachowaniem odpowiedniej precyzji i znaku dla współrzędnych. Jeśli instancja jest NULL, zwraca string NULL.
- *string ToCardinalString()* - Konwertuje instancję *GeoLocation* na string w formacie kardynalnym (kierunkowym), np. '52.22 N, 21.01 E'. Jeśli instancja jest NULL, zwraca string NULL.
- *static SqlDouble DistanceKm(GeoLocation g1, GeoLocation g2)* - Oblicza odległość w kilometrach między dwoma punktami *GeoLocation* przy użyciu wzoru Haversine'a. Zwraca 'SqlDouble' lub 'SqlDouble.Null', jeśli którykolwiek z punktów jest NULL.
- *void Read(BinaryReader r)* - Deserializuje instancję *GeoLocation* ze strumienia binarnego. Odczytuje flagę NULL, a następnie szerokość i długość geograficzną.
- *void Write(BinaryWriter w)* - Serializuje instancję *GeoLocation* do strumienia binarnego. Zapisuje flagę NULL, a następnie szerokość i długość geograficzną, o ile obiekt nie jest NULL.

- *bool* ***Equals(object obj)*** - Porównuje dwie instancje *GeoLocation*. Zwraca *true*, jeśli mają zbliżone szerokości i długości geograficzne. Obsługuje również porównywanie obiektów NULL.
- *int* ***GetHashCode()*** - Generuje kod skrótu dla instancji *GeoLocation* na podstawie jej szerokości i długości geograficznej. Obsługiwany jest również przypadek NULL.
- *private bool* ***ValidateGeoLocation()*** - Prywatna metoda walidacji wywoływana przez SQL Server. Sprawdza poprawność szerokości geograficznej (zakres) oraz upewnia się, że wartości nie są 'NaN' ani 'Infinity'.

4.5 ColorRGB

Typ *ColorRGB* reprezentuje kolor w przestrzeni barw RGB (czerwień, zieleń, błękit) z opcjonalnym kanałem alfa (przezroczystość). Umożliwia przechowywanie i manipulowanie danymi kolorów bezpośrednio w bazie danych SQL Server.

Typ *ColorRGB* jest złożony z pól:

- *public byte* ***R*** - Reprezentuje składową czerwoną koloru (wartość od 0 do 255).
- *public byte* ***G*** - Reprezentuje składową zieloną koloru (wartość od 0 do 255).
- *public byte* ***B*** - Reprezentuje składową niebieską koloru (wartość od 0 do 255).
- *public byte* ***A*** - Reprezentuje składową alfa (przezroczystość) koloru (wartość od 0 do 255). Domyślnie wynosi 255 (pełna nieprzezroczystość).
- *private bool* ***is_Null*** - Prywatna flaga wskazująca, czy instancja *ColorRGB* reprezentuje wartość NULL w bazie danych SQL Server.

Posiada właściwości:

- *public bool* ***IsNull*** - Zwraca *true*, jeśli obiekt *ColorRGB* reprezentuje wartość NULL, w przeciwnym razie *false*.
- *public static ColorRGB* ***Null*** - Statyczna właściwość zwracająca nową instancję *ColorRGB* ustawioną na wartość NULL.

Posiada metody:

- ***ColorRGB(byte r, byte g, byte b, byte a = 255)*** - Konstruktor tworzący instancję *ColorRGB* z podanych wartości składowych R, G, B i opcjonalnie A.
- *static ColorRGB* ***FromHex(string hex)*** - Tworzy instancję *ColorRGB* z szesnastkowego ciągu znaków (np. '#RRGGBB' lub '#RRGGBBAA'). Rzucany jest wyjątek *ArgumentException* dla nieprawidłowego formatu.
- *string* ***ToHex()*** - Konwertuje instancję *ColorRGB* na szesnastkowy ciąg znaków (np. '#RRGGBB' lub '#RRGGBBAA', jeśli kanał alfa jest różny od 255).
- *ColorRGB* ***Negate()*** - Zwraca kolor dopełniający poprzez odwrócenie wartości składowych R, G i B (255 minus bieżąca wartość). Kanał alfa pozostaje niezmienny.
- *ColorRGB* ***Blend(ColorRGB other, double ratio)*** - Łączy bieżący kolor z innym kolorem, używając podanego współczynnika 'ratio' (od 0.0 do 1.0). Ratio 0.0 zwraca bieżący kolor, 1.0 zwraca drugi kolor.
- *string* ***ToString()*** - Konwertuje instancję *ColorRGB* na string w formacie '[R,G,B]' lub '[R,G,B,A]', jeśli kanał alfa jest różny od 255. Jeśli instancja jest NULL, zwraca NULL.
- *static ColorRGB* ***Parse(SqlString s)*** - Główny sposób tworzenia instancji *ColorRGB* ze stringa. Akceptuje formaty szesnastkowe (np. '#RRGGBB') lub tablicowe (np. '[R,G,B]' lub '[R,G,B,A]'). Rzucany jest wyjątek *ArgumentException* dla nieprawidłowego formatu.
- *private bool* ***ValidateColorRGB()*** - Prywatna metoda walidacji wywoływana przez SQL Server. Zwraca *true*, jeśli instancja nie jest NULL.

- *void* **Write**(*BinaryWriter w*) - Serializuje instancję *ColorRGB* do strumienia binarnego. Zapisuje flagę NULL, a następnie wartości R, G, B i A.
- *void* **Read**(*BinaryReader r*) - Deserializuje instancję *ColorRGB* ze strumienia binarnego. Odczytuje flagę NULL, a następnie wartości R, G, B i A.
- *bool* **Equals**(*object obj*) - Porównuje dwie instancje *ColorRGB*. Zwraca *true*, jeśli wszystkie składowe (R, G, B, A) oraz status NULL są identyczne.
- *int* **GetHashCode**() - Generuje kod skrótu dla instancji *ColorRGB* na podstawie jej wartości składowych R, G, B i A.

4.6 MoneyType

Typ *MoneyType* reprezentuje wartości pieniężne wraz z przypisaną walutą. Pozwala na przechowywanie i wykonywanie operacji na danych finansowych bezpośrednio w SQL Server, zapewniając walidację kodów walutowych ISO oraz operacje konwersji i arytmetyczne. Obsługiwane waluty to "PLN", "EUR", "USD", "GBP", "CHF", "JPY", "CNY".

Typ *MoneyType* jest złożony z pól:

- *public decimal* **Amount** - Przechowuje wartość pieniężną.
- *public string* **Currency** - Przechowuje trzyliterowy kod waluty (np. "PLN"). Kody są konwertowane na wielkie litery i walidowane względem listy obsługiwanych walut ISO.
- *private bool* **is_Null** - Prywatna flaga wskazująca, czy instancja *MoneyType* reprezentuje wartość NULL w bazie danych SQL Server.

Posiada właściwości:

- *public bool* **IsNull** - Właściwość zgodna z interfejsem *INullable*. Zwraca *true*, jeśli obiekt *MoneyType* reprezentuje wartość NULL, w przeciwnym razie *false*.
- *public static MoneyType* **Null** - Statyczna właściwość zwracająca nową instancję *MoneyType* ustawioną na wartość NULL.

Posiada metody:

- *MoneyType*(*decimal amount, string currency*) - Konstruktor tworzący instancję *MoneyType* z podanej kwoty i waluty. Waluta jest normalizowana (na wielkie litery) i walidowana.
- *static bool* **ValidateCurrency**(*string code*) - Prywatna metoda pomocnicza walidująca, czy podany kod waluty ISO znajduje się na liście obsługiwanych walut.
- *MoneyType* **ConvertTo**(*string targetCurrency*) - Konwertuje kwotę na określoną walutę docelową, wykorzystując uproszczone kursy wymiany. Zwraca nową instancję *MoneyType* z przeliczoną kwotą i walutą docelową. Rzucane są wyjątki dla nieprawidłowych lub nieobsługiwanych walut.
- *string* **ToString**() - Konwertuje instancję *MoneyType* na string w formacie 'kwota [WALUTA]' (np. '123.45 [PLN]'). Jeśli instancja jest NULL, zwraca NULL.
- *static MoneyType* **Parse**(*SqlString s*) - Główny sposób tworzenia instancji *MoneyType* ze stringa w formacie 'kwota [WALUTA]'. Parsuje wartość liczbową i walutę. Rzucane są wyjątki *ArgumentException* dla nieprawidłowego formatu lub nieznannej waluty.
- *static MoneyType* **Add**(*MoneyType m1, MoneyType m2*) - Dodaje dwie wartości *MoneyType*. Wymaga, aby waluty obu wartości były identyczne; w przeciwnym razie rzuca wyjątek *ArgumentException*.
- *static MoneyType* **Subtract**(*MoneyType m1, MoneyType m2*) - Odejmowanie dwóch wartości *MoneyType*. Wymaga identycznych walut; w przeciwnym razie rzuca wyjątek *ArgumentException*.
- *static MoneyType* **MultiplyByScalar**(*MoneyType m, decimal scalar*) - Mnoży wartość *MoneyType* przez skalar, zwracając nową instancję z pomnożoną kwotą i oryginalną walutą.

- *void Write(BinaryWriter w)* - Serializuje instancję *MoneyType* do strumienia binarnego. Zapisywana jest flaga NULL, a następnie kwota ('decimal') i waluta ('string').
- *void Read(BinaryReader r)* - Deserializuje instancję *MoneyType* ze strumienia binarnego. Odczytywana jest flaga NULL, a następnie kwota ('decimal') i waluta ('string').
- *bool Equals(object obj)* - Porównuje dwie instancje *MoneyType*. Zwraca *true*, jeśli mają identyczne kwoty i waluty (bez uwzględniania wielkości liter). Obsługuje również porównywanie obiektów NULL.
- *int GetHashCode()* - Generuje kod skrótu dla instancji *MoneyType* na podstawie jej kwoty i waluty. Obsługiwany jest również przypadek NULL.
- *private bool ValidateMoney()* - Prywatna metoda walidacji wywoływana przez SQL Server. Sprawdza poprawność kodu waluty.

5 Opis implementacji udostępnionego API przez bibliotekę

Program zorganizowany jest w Visual Studio. Podzielony został na dwa projekty. Pierwszy, *UserDefinedTypes*, opiera się na bibliotece .NET zawierający wszystkie UDT. Drugi jest aplikacją konsolową w języku C#, która obejmuje interfejs użytkownika, przekazywanie input'u do metod i komunikację aplikacji z serwerem SQL.

5.1 UserDefinedTypes

Każdy z typów wykorzystuje oznaczenia UDT dla SQL Servera tj. [Serializable], [SqlUserDefinedType] i [SqlMethod] gdzie:

- Serializable - Mówi środowisku .NET, że instancje tej struktury mogą być konwertowane na strumień bajtów (serializowane i deserializowane).
- SqlUserDefinedType - Informuje SQL Server, że ta konkretna struktura jest przeznaczona do użytku jako niestandardowy typ danych. Atrybut ten przyjmuje parametry, które konfigurują zachowanie UDT:
 - Format.UserDefined
 - IBinarySerialize
 - IsByteOrdered
 - MaxByteSize
 - ValidationMethodName
- SqlMethod - Dana metoda ma być dostępna jako funkcja lub metoda CLR UDT w T-SQL. Dzięki temu można wywoływać ją bezpośrednio w zapytaniach SQL.

Każdy z opisanych typów implementuje interfejs:

- INullable - interfejs w platformie .NET, który umożliwia reprezentowanie stanu NULL, implementacja INullable jest obowiązkowa dla niestandardowych typów, aby kolumny tego typu mogły przechowywać wartości NULL w bazie danych.
- IBinarySerialize - interfejs w platformie .NET, który umożliwi pełną kontrolę nad procesem zapisywania i odczytywania obiektu do i ze strumienia binarnego. Jest to szczególnie ważne dla UDT w SQL Serverze, ponieważ dane muszą być przechowywane w formie binarnej w kolumnach bazy danych. Interfejs ten definiują dwie metody *void Write(BinaryWriter w)* i *void Read(BinaryReader r)*.

5.2 Projekt aplikacji konsolowej

Każdy ze zdefiniowanych typów jest reprezentowany przez schemat plików, które pozwalają na jego wykorzystanie i komunikację interfejsu użytkownika z serwerem.

- [Type] - klasa nazwana na podobieństwo oryginalnie utworzonego typu. Zawiera pola odpowiadające tabeli w bazie SQL Server. Przykład dla klasy *GeoLocation*, klasa jest nazwana *Location*.

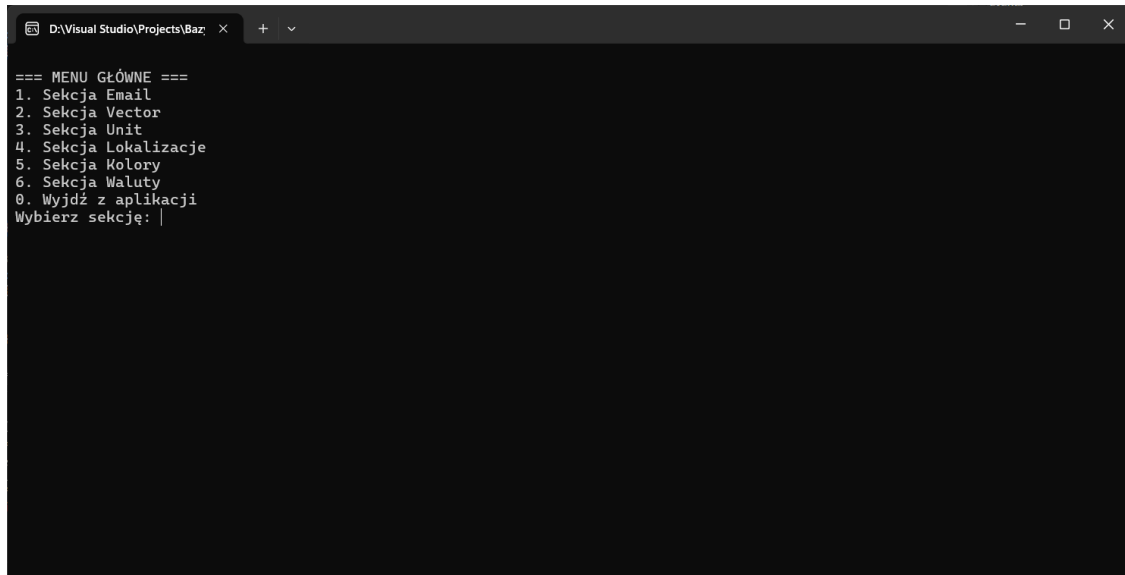
- [Type]Repository - klasa implementująca połączenie i komunikację z SQL Server, finalizuje zapytania i je przesyła.
- [Type]Service - klasa łącząca utworzony typ i jego unikatowe metody z oczekiwaną funkcjonalnością, które przekazuje do *Repository*.
- [Type]App - klasa użytkowa, która obsługuje bezpośrednie akcje użytkownika przekazywane bezpośrednio do Service.

Każda z aplikacji obsługujących typy również ma zaimplementowane zapisywanie nowych danych na bazie pliku CSV, który w pierwszym wierszu ma nazwy kolumn, które odpowiadają nazwom w tabelach SQL Server. Przy wybraniu odpowiedniej opcji należy wpisać bezwzględną ścieżkę do pliku.

Program *main* znajduje się w pliku *Program.cs*, w którym znajduje się obsługa każdej z aplikacji danego typu.

6 Przykład użycia

Użytkownik obsługuje aplikację poprzez klawiaturę. Główną akcją jest wpisywanie cyfr lub liczb, które odpowiadają danemu wydarzeniu. Po wybraniu wydarzenia następuje wyświetlenie rezultatu lub poproszenie o wprowadzenie odpowiednich danych.



```

D:\Visual Studio\Projects\Baz x + v
=== MENU GŁÓWNE ===
1. Sekcja Email
2. Sekcja Vector
3. Sekcja Unit
4. Sekcja Lokalizacje
5. Sekcja Kolory
6. Sekcja Waluty
0. Wyjdź z aplikacji
Wybierz sekcję: |

```

Rys. 6.1: Okno poprawnego uruchomienia aplikacji, pokazujące główne menu z opcjami do wyboru od 0-6

```
D:\Visual Studio\Projects\Baz x + v
=== MENU GŁÓWNE ===
1. Sekcja Email
2. Sekcja Vector
3. Sekcja Unit
4. Sekcja Lokalizacje
5. Sekcja Kolory
6. Sekcja Waluty
0. Wyjdź z aplikacji
Wybierz sekcję: 4

--- Aplikacja Zarządzania Lokalizacjami (CLR UDT GeoLocation) ---

--- Menu ---
1. Dodaj nową lokalizację
2. Wyświetl wszystkie lokalizacje
3. Usuń lokalizację po ID
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: |
```

Rys. 6.2: Okno po wybraniu opcji nr 4. *Sekcja Lokalizacje*. Pojawia się kolejne menu odpowiadające akcjom, które można wykonać na tabeli powiązanej z *GeoLocation*.

```
D:\Visual Studio\Projects\Baz x + v
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: 2

--- Wszystkie lokalizacje ---
-----
ID      Nazwa      GeoLocation      GeoLocation (N/S, E/W)
-----
1      Warszawa  (+52.2297, +21.0122)  52.2297N, 21.0122E
2      Kraków    (+50.0647, +19.945)   50.0647N, 19.945E
3      Gdańsk    (+54.352, +18.6466)   54.352N, 18.6466E
4      Wrocław  (+51.1079, +17.0385)  51.1079N, 17.0385E
5      Poznań    (+52.4064, +16.9252)  52.4064N, 16.9252E
6      Szczecin (+53.4285, +14.5528)  53.4285N, 14.5528E
7      Lublin    (+51.2465, +22.5684)  51.2465N, 22.5684E
8      Katowice (+50.2649, +19.0238)  50.2649N, 19.0238E
9      Białystok (+53.1325, +23.1688)  53.1325N, 23.1688E
10     Łódź      (+51.7592, +19.455)   51.7592N, 19.455E
-----

--- Menu ---
1. Dodaj nową lokalizację
2. Wyświetl wszystkie lokalizacje
3. Usuń lokalizację po ID
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: |
```

Rys. 6.3: Wybrana została opcja 2. *Wyświetl wszystkie lokalizacje*. Pojawiła się tabela istniejących rekordów.

```
D:\Visual Studio\Projects\Baz x + v
4 Wrocław (+51.1079, +17.0385) 51.1079N, 17.0385E
5 Poznań (+52.4064, +16.9252) 52.4064N, 16.9252E
6 Szczecin (+53.4285, +14.5528) 53.4285N, 14.5528E
7 Lublin (+51.2465, +22.5684) 51.2465N, 22.5684E
8 Katowice (+50.2649, +19.0238) 50.2649N, 19.0238E
9 Białystok (+53.1325, +23.1688) 53.1325N, 23.1688E
10 Łódź (+51.7592, +19.455) 51.7592N, 19.455E
-----
--- Menu ---
1. Dodaj nową lokalizację
2. Wyświetl wszystkie lokalizacje
3. Usuń lokalizację po ID
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: 1

Podaj nazwę lokalizacji: NOWE MIASTO
Podaj lokalizację w formacie (lat, lon) lub 'latN, lonE' (np. 52.2297N, 21.0122E): (10,100)
Dodano lokalizację: NOWE MIASTO (+10, +100)

--- Menu ---
1. Dodaj nową lokalizację
2. Wyświetl wszystkie lokalizacje
3. Usuń lokalizację po ID
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: |
```

Rys. 6.4: Została wybrana opcja 1. *Dodaj nową lokalizację*. Użytkownik został poproszony o podanie nazwy oraz współrzędnych lokalizacji. Poprawne wpisanie danych skutkuje dodaniem lokalizacji.

```
D:\Visual Studio\Projects\Baz x + v
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: 2

--- Wszystkie lokalizacje ---
-----
ID Nazwa GeoLocation GeoLocation (N/S, E/W)
-----
1 Warszawa (+52.2297, +21.0122) 52.2297N, 21.0122E
2 Kraków (+50.0647, +19.945) 50.0647N, 19.945E
3 Gdańsk (+54.352, +18.6466) 54.352N, 18.6466E
4 Wrocław (+51.1079, +17.0385) 51.1079N, 17.0385E
5 Poznań (+52.4064, +16.9252) 52.4064N, 16.9252E
6 Szczecin (+53.4285, +14.5528) 53.4285N, 14.5528E
7 Lublin (+51.2465, +22.5684) 51.2465N, 22.5684E
8 Katowice (+50.2649, +19.0238) 50.2649N, 19.0238E
9 Białystok (+53.1325, +23.1688) 53.1325N, 23.1688E
10 Łódź (+51.7592, +19.455) 51.7592N, 19.455E
11 NOWE MIASTO (+10, +100) 10N, 100E
-----

--- Menu ---
1. Dodaj nową lokalizację
2. Wyświetl wszystkie lokalizacje
3. Usuń lokalizację po ID
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: |
```

Rys. 6.5: Wybrana została opcja 2. *Wyświetl wszystkie lokalizacje*. Pojawiła się tabela istniejących rekordów wraz z nowo wpisanym rekordem.

```

D:\Visual Studio\Projects\Baz x + v
4 Wrocław (+51.1079, +17.0385) 51.1079N, 17.0385E
5 Poznań (+52.4064, +16.9252) 52.4064N, 16.9252E
6 Szczecin (+53.4285, +14.5528) 53.4285N, 14.5528E
7 Lublin (+51.2465, +22.5684) 51.2465N, 22.5684E
8 Katowice (+50.2649, +19.0238) 50.2649N, 19.0238E
9 Białystok (+53.1325, +23.1688) 53.1325N, 23.1688E
10 Łódź (+51.7592, +19.455) 51.7592N, 19.455E
11 NOWE MIASTO (+10, +100) 10N, 100E
-----
--- Menu ---
1. Dodaj nową lokalizację
2. Wyświetl wszystkie lokalizacje
3. Usuń lokalizację po ID
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: 0

Wyjście z aplikacji Lokalizacje.

=== MENU GŁÓWNE ===
1. Sekcja Email
2. Sekcja Vector
3. Sekcja Unit
4. Sekcja Lokalizacje
5. Sekcja Kolory
6. Sekcja Waluty
0. Wyjdź z aplikacji
Wybierz sekcję: |

```

Rys. 6.6: Wybrana została opcja 0. Wyjdź, co spowodowało opuszczenie sekcji lokalizacji.

```

Microsoft Visual Studio Debug x + v
11 NOWE MIASTO (+10, +100) 10N, 100E
-----
--- Menu ---
1. Dodaj nową lokalizację
2. Wyświetl wszystkie lokalizacje
3. Usuń lokalizację po ID
4. Znajdź najbliższą lokalizację do podanej
5. Dodaj rekordy przez plik CSV
0. Wyjdź
Wybierz opcję: 0

Wyjście z aplikacji Lokalizacje.

=== MENU GŁÓWNE ===
1. Sekcja Email
2. Sekcja Vector
3. Sekcja Unit
4. Sekcja Lokalizacje
5. Sekcja Kolory
6. Sekcja Waluty
0. Wyjdź z aplikacji
Wybierz sekcję: 0

Zamykanie aplikacji...

D:\Visual Studio\Projects\Bazy danych\Projekt_BD2\exe\Projekt_BD2.exe (process 30024) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|

```

Rys. 6.7: Wybrana została opcja 0. Wyjdź z aplikacji, co spowodowało opuszczenie programu i zakończenie jego działania.

7 Podsumowanie i wnioski

7.1 Podsumowanie

W ramach projektu zrealizowano implementację niestandardowych typów danych użytkownika (UDT) w środowisku SQL Server CLR, co znacząco rozszerzyło możliwości baz danych w zakresie przechowywania i manipulowania złożonymi danymi bezpośrednio na poziomie serwera. Stworzone typy UDT – Email(adres e-mail), Vector3D(wektor w trzech wymiarach), UnitSI (jednostki SI), GeoLocation (lokalizacje geograficzne), ColorRGB (kolory RGB) oraz MoneyType (wartości pieniężne z walutą) – zapewniają silne typowanie, walidację danych oraz hermetyzację logiki biznesowej, która tradycyjnie byłaby obsługiwana na poziomie aplikacji.

7.2 Wnioski

Projekt dowiódł, że implementacja niestandardowych typów danych za pomocą CLR UDT jest skutecznym i efektywnym podejściem do zarządzania złożonymi danymi w SQL Serverze. Główne korzyści płynące z tego rozwiązania to:

- Zwiększona integralność danych: Dzięki wbudowanej walidacji na poziomie typu, baza danych sama dba o to, by przechowywane dane były poprawne.
- Lepsza hermetyzacja logiki biznesowej: Operacje specyficzne dla danego typu mogą być wykonywane bezpośrednio w bazie danych za pomocą metod UDT.
- Uproszczony model danych: Zamiast przechowywać złożone dane w wielu kolumnach (np. oddzielne kolumny na wartość i jednostkę), można użyć pojedynczej kolumny typu UDT. To sprawia, że schemat bazy danych jest czystszy i łatwiejszy do zrozumienia.

8 Kod źródłowy

8.1 Struktura plików

- **Projekt_BD2**
 - **exe** - zawiera plik wykonywalny oraz dane potrzebne do połączenia z serwerem i bazą danych
 - * connection.txt
 - * Projekt_BD2.exe
 - **CSV** - zawiera przykładowe pliki CSV do użycia podczas działania programu
 - **SQL** - zawiera kluczowe komendy SQL do utworzenia assembly, typów i tabel
 - * assembly.sql
 - * table.sql
 - Program.cs
 - CSVimporter.cs
 - **Color**
 - * Color.cs
 - * ColorRepository.cs
 - * ColorService.cs
 - * ColorApp.cs
 - **Currency**
 - * Currency.cs
 - * CurrencyRepository.cs
 - * CurrencyService.cs
 - * CurrencyApp.cs
 - **Email**
 - * User.cs
 - * UserRepository.cs
 - * UserService.cs
 - * EmailApp.cs
 - **Unit**
 - * Unit.cs
 - * UnitRepository.cs
 - * UnitService.cs
 - * UnitApp.cs
 - **Vector**
 - * Vector.cs
 - * VectorRepository.cs
 - * VectorService.cs
 - * VectorApp.cs
 - **GeoLocation**
 - * Location.cs

- * LocationRepository.cs
- * LocationService.cs
- * LocationApp.cs
- **UserDefinedTypes**
 - MoneyType.cs
 - UnitSI.cs
 - Color.cs
 - Vector3D.cs
 - GeoLocation.cs
 - Email.cs
 - **bin**
 - * **Debug**
 - UserDefinedTypes.dll - plik używany w *assembly.sql*

8.2 Komendy T-SQL

Przykładowy kod T-SQL potrzebny do utworzenia assembly i typów:

8.2.1 assembly.sql

```

IF NOT EXISTS (
    SELECT name
    FROM sys.databases
    WHERE name = N'BD_Projekt'
)
BEGIN
    CREATE DATABASE test;
END
GO

drop table if exists Users;
drop table if exists Vectors;
drop table if exists Units;
drop table if exists Locations;
drop table if exists Colors;
drop table if exists Currency;

IF TYPE_ID('Email') IS NOT NULL
    DROP TYPE Email;
IF TYPE_ID('Vector3D') IS NOT NULL
    DROP TYPE Vector3D;
IF TYPE_ID('UnitSI') IS NOT NULL
    DROP TYPE UnitSI;
IF TYPE_ID('GeoLocation') IS NOT NULL
    DROP TYPE GeoLocation;
IF TYPE_ID('ColorRGB') IS NOT NULL
    DROP TYPE ColorRGB;
IF TYPE_ID('MoneyType') IS NOT NULL
    DROP TYPE MoneyType;

IF EXISTS (SELECT * FROM sys.assemblies WHERE name = 'UDTAssembly')
    DROP ASSEMBLY UDTAssembly;

```

— Należy wpisać własną ścieżkę lokalizacji plików, zazwyczaj będzie ona w strukturze
 — [POCZATEK SCIEZKI]\UserDefinedTypes\bin\Debug\UserDefinedTypes.dll

```

CREATE ASSEMBLY UDTAssembly
FROM 'D:\Visual Studio\Projects\Bazy danych\UserDefinedTypes\bin\Debug\UserDefinedTypes.dll'
WITH PERMISSION_SET = SAFE;
GO

```

```

CREATE TYPE Email
EXTERNAL NAME UDTAssembly.[Email];

```

```

CREATE TYPE Vector3D
EXTERNAL NAME UDTAssembly.[Vector3D];

```

```

CREATE TYPE UnitSI
EXTERNAL NAME UDTAssembly.[UnitSI];

```

```

CREATE TYPE GeoLocation
EXTERNAL NAME UDTAssembly.[GeoLocation];

```

```

CREATE TYPE ColorRGB
EXTERNAL NAME UDTAssembly.[ColorRGB];

```

```

CREATE TYPE MoneyType
EXTERNAL NAME UDTAssembly.[MoneyType];

```

```

GO

```

8.2.2 table.sql

```

CREATE TABLE Users (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    ContactEmail Email NOT NULL
);

```

```

CREATE TABLE Vectors(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Vec Vector3D NOT NULL
);

```

```

CREATE TABLE Units(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Ut UnitSI NOT NULL
);

```

```

CREATE TABLE Locations(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    Place VARCHAR(100) NOT NULL,
    Location GeoLocation NOT NULL
);

```

```

CREATE TABLE Colors (
    Id INT IDENTITY(1,1) PRIMARY KEY,
    ColorValue ColorRGB NOT NULL
);

```

```

CREATE TABLE Currency (

```



```
Id INT IDENTITY(1,1) PRIMARY KEY,  
MoneyValue MoneyType NOT NULL  
);
```

9 Przygotowanie do wykonania

Aby poprawnie uruchomić program należy wykonać następujące kroki:

- W SQL Server mieć dostęp do tworzenia bazy danych
- Wykonać plik *assembly.sql* z odpowiednią ścieżką do pliku *UserDefinedTypes.dll*
- Wykonać plik *tables.sql*
- Sprawdzić plik *connection.txt*, czy zawiera poprawne dane serwera i nazwę bazy danych, która w pliku *assembly.sql* oraz *connection.txt* została ustawiona na *BD_Projekt*.

10 Literatura

- https://learn.microsoft.com/en-us/sql/relational-databases/clr-integration-database-objects-working-with-user-defined-types-defining-udt-tables-and-columns?view=sql-server-ver17&utm_source=chatgpt.com
- https://learn.microsoft.com/en-us/dotnet/framework/data/adonet/sql/clr-user-defined-types?utm_source=chatgpt.com
- https://learn.microsoft.com/en-us/sql/relational-databases/clr-integration-database-objects-clr-user-defined-types?view=sql-server-ver17&utm_source=chatgpt.com
- https://en.wikipedia.org/wiki/Email_address