# Task

Given a database table where none of the column data types are correct (except one) read in data to the correct data type, for example, date/time are stored as a string, numeric types stored as strings. Write code which strongly types these columns and when performing updates or additions send back data in the database table column type, string. Hey wait a minute, one column, SomePrice is the correct type but we want to perform a conversion to the following model

What is a struct? [See here](https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct)

public readonly struct Dollars

{

    public Dollars(decimal amount) => Amount = amount;

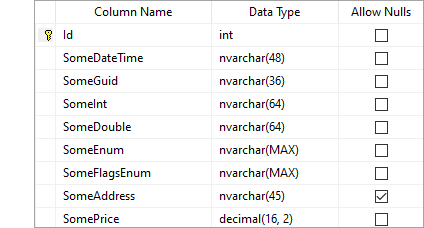
    public decimal Amount { get; }

    public override string ToString() => Amount.ToString("C");

}

**Database name**: SomeDatabase

**Table name**: SomeEntities, schema below



## Models

SomeEntity is the main model

|  |  |
| --- | --- |
| public readonly struct Dollars  {      public Dollars(decimal amount) => Amount = amount;      public decimal Amount { get; }      public override string ToString() => Amount.ToString("C");  } |  |
| public enum SomeEnum  {      First,      Second,      Third  } | [Flags]  public enum SomeFlagsEnum  {      First = 1,      Second = 2,      Third = 4  } |
| public class SomeEntity  {      public int Id { get; set; }      public DateTime SomeDateTime { get; set; }      public Guid SomeGuid { get; set; }      public int SomeInt { get; set; }      public double SomeDouble { get; set; }      public SomeEnum SomeEnum { get; set; }      public SomeFlagsEnum SomeFlagsEnum { get; set; }      public IPAddress SomeAddress { get; set; }      public Dollars SomePrice { get; set; }      public override string ToString() => Id.ToString();  } |  |

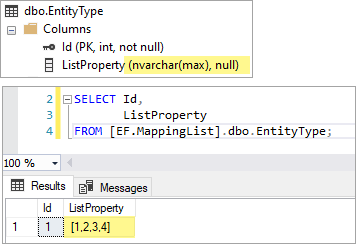
When working with EF Core, we use a generic type HasConversion method with pre-defined conversions for most transformations configured in a DbContext for properties which require transformations to and from the database table. This means in business code, if a column is a string in the database table and in code a DateTime, code as a DateTime.

|  |
| --- |
| public new virtual PropertyBuilder<TProperty> HasConversion<TProvider>()      => (PropertyBuilder<TProperty>)base.HasConversion<TProvider>(); |

A developer can create their own converter. In this case specify the UTC (Coordinated Universal Time or UTC is the primary time standard by which the world regulates clocks and time.) for a DateTime.

|  |
| --- |
| var dateTimeConverter = new ValueConverter<DateTime, DateTime>(      datetime => datetime,      dateTime =>          DateTime.SpecifyKind(dateTime, DateTimeKind.Utc)); |

A complex example which stores a l**ist of int** in an **nvarchar** column data type.



Note I broke the rule of 1 character variables, here it’s not going to matter if you understand the code.

|  |
| --- |
| modelBuilder      .Entity<EntityType>()      .Property(e => e.ListProperty)      .HasConversion(          v => JsonSerializer.Serialize(v, null),          v => JsonSerializer.Deserialize<List<int>>(v, null),          new ValueComparer<List<int>>(              (list1, list2) => list1.SequenceEqual(list2),              c => c.Aggregate(0, (a, v) =>                  HashCode.Combine(a, v.GetHashCode())),              c => c.ToList())); |

EF Core code

|  |
| --- |
| var entity = new EntityType { ListProperty = new List<int> { 1, 2, 3 } }; |

Typically a team will have at last one developer who fully understands write this and if it was copied off the web, worked without any understanding than that is a problem waiting to happen.

## Using a data provider

When working with a data provider all conversions are performed in SELECT, UPDATE and INSERT statements.

Using the same model above let’s update a record.

T-SQL

|  |
| --- |
| UPDATE dbo.SomeEntities    SET        SomeDateTime = @SomeDateTime,        SomeGuid = @SomeGuid,        SomeInt = @SomeInt,        SomeEnum = @SomeEnum,        SomePrice = @SomePrice  WHERE Id = @Identifier; |

Partial code

|  |
| --- |
| cmd.Parameters.Add("@SomeDateTime", SqlDbType.NVarChar).Value = entity.SomeDateTime      .**ToString**(CultureInfo.InvariantCulture);    cmd.Parameters.Add("@SomeGuid", SqlDbType.NVarChar).Value = entity.SomeGuid.**ToString**();  cmd.Parameters.Add("@SomeInt", SqlDbType.NVarChar).Value = entity.SomeInt;  cmd.Parameters.Add("@SomeEnum", SqlDbType.NVarChar).Value = entity.SomeEnum;  cmd.Parameters.Add("@SomePrice", SqlDbType.Decimal).Value = entity.SomePrice.Amount;  cmd.Parameters.Add("@Identifier", SqlDbType.Int).Value = entity.Id; |

Produces the following (a language extension is used to expose values of parameters)

|  |
| --- |
| UPDATE dbo.SomeEntities      SET          SomeDateTime = '06/01/2022 15:03:51',          SomeGuid = 'c180074e-617e-4453-803a-869c615e1d3b',          SomeInt = '1111',          SomeEnum = 'Third',          SomePrice = 0  WHERE Id = 1; |

Developers have been writing code this (not always right e.g. string concat parameters etc) for decades.

Issues

* Have to perform conversions each time data is needed or changed.
* If a column name changes in the database table, the developer must find all occurrences and change/test while EF Core will do this for us.

## Back to EF Core

One issue with conversions currently is the inability to handle null in some cases. The EF Core engineers may have resolved this in EF Core 6.

### How do value conversions work

* It converts it into a .NET type that does have known sql mapping
* Taking a .NET type that has no corresponding sql types
* It then goes ahead with the actual conversion to a sql type.

### Number 1 used conversion is enumerations

Here we have a model for wines and an enumeration for wine categories with reverse navigations as the data is in two tables.

|  |
| --- |
| modelBuilder      .Entity<WineVariant>()      .Property(e => e.WineVariantId)      .HasConversion<int>();    modelBuilder      .Entity<WineVariant>().HasData(          Enum.GetValues(typeof(WineVariantId))              .Cast<WineVariantId>()              .Select(e => new WineVariant()              {                  WineVariantId = e,                  Name = e.ToString()              })      ); |

The code is repeatable, book and book category

|  |
| --- |
| modelBuilder      .Entity<Book>()      .Property(e => e.BookCategory)      .HasConversion<int>();    modelBuilder      .Entity<BookVariant>().HasData(          Enum.GetValues(typeof(BookCategory))              .Cast<BookCategory>()              .Select(e => new BookVariant()              {                  BookCategoryId = e,                  Name = e.ToString()              })      ); |

And as we go through various code samples pretty much any common and uncommon conversions can be done.

Last example, setup a property for Color.

|  |
| --- |
| public Color Color { get; set; } |

Setup the conversion

|  |
| --- |
| modelBuilder.Entity<Person>()      .Property(person => person.Color)      .HasConversion(          color => color.Name,          value => Color.FromName(value)); |

Read data back

|  |
| --- |
| private static void ReadPeople()  {      using var context = new PeopleContext();      var people = context.People.ToList();      foreach (var person in people)      {          WriteLine($"{person.Id,-3}{person.Color.Name}");      }  } |