How to save enum in database as string

22

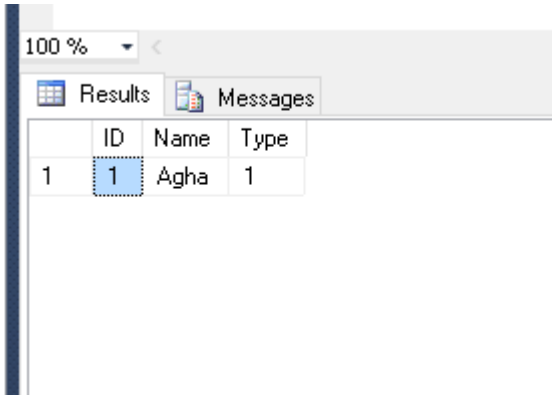
6

This is my Model Class where we have a Type which could be a Zombie or Human

```
public class User
{
    public int ID { get; set; }
    public string Name { get; set; }
    public Type Type { get; set; }
    public List<Wepon> WeposInList { get; set; }
    }

public enum Type
{   [Description("Zombie")] Zombie,
    [Description("Human")] Human
}
```

Currently it is saving data in Int



I want to save the data as Human and Zombie, not with int

c# .net entity-framework c#-4.0

1 An enumeration is a static object, there's no point in saving it to the database... Really you should define the items in the database and then generate your enum in code using a T4 template. – The Muffin Man Sep 12 '15 at 18:49

4 An enum value *is* an int. You should really save it as so. Unless you have a *very good* reason to have it as a string in the DB.. Which I suppose you don't. – Pierre-Luc Pineault Sep 12 '15 at 19:11

4 @Pierre-LucPineault : is it not dangerous to store it as an int? Wouldn't all someone have to do is re-order the enum and then immediately all the values in your database is pointing to the wrong enum without any warning – Diskdrive Jun 23 '16 at 6:13

1 @Diskdrive You can assign a specific integer to your enum so even when reordered it doesn't change (And often with powers of 2 so you can declare it as a 'Flag'). But usually you just don't go reordering enums for fun. – Pierre-Luc Pineault Jun 23 '16 at 13:26

10 @Pierre-LucPineault, what Diskdrive is referring to is a real problem. Enums can get arranged for reasons other than fun, both validly and accidentally. It's a more robust solution to store the strings; those don't change. But if they do, no data is lost, just a db update is needed. Plus, looking at the table that way is more productive, and there's no need to number the values, which is nice on non-flag enums. In fact, this is how xml serialization works. So, at least some MS developers agree with Diskdrive. – toddmo Feb 25 '17 at 0:42

5 Answers

7

I had this problem as far as I remember and honestly I don't know why didn't MS add this feature (NH can do it like since always..).

Any ways, what I usually did is use const strings classes like:

```
public static class MyEnum
{
    public const string Foo = "Foo";
    public const string Bar = "Bar";
}

public class Client
{

    public string MyVal { get; set; }

    public Client()
    {
```

Cons - as simple as can be.

Downsides - you loose type checking (though it could be enforced programmatically).

So this time I tried to think of something more ambitious. So I took the concept described by Brian (which has some downsides when e.g. a given enum is used widely across the domain). And well.. I got the following working:

A base component class to store the values:

```
[ComplexType]
public class DbEnum<TEnum>
{
    public string _ { get; set; }

    public DbEnum()
    {
        _ = default(TEnum).ToString();
    }

    protected DbEnum(TEnum value)
    {
        _ = value.ToString();
    }

    public TEnum ToEnum()
    {
        return _.ToEnum<TEnum>();
    }

    public static implicit operator DbEnum<TEnum>(TEnum value)
    {
        return new DbEnum<TEnum>(value);
    }

    public static implicit operator TEnum(DbEnum<TEnum> value)
    {
        return value.ToEnum();
    }
}
```

```
public enum PrivacyLevel
{
    Public,
    Friends,
    Private
}

public class PrivacyLevelEnum : DbEnum<PrivacyLevel>
{
    public PrivacyLevelEnum() : this(default (PrivacyLevel))
    {
    }

    public PrivacyLevelEnum(PrivacyLevel value) : base(value)
    {
    }

    public static implicit operator PrivacyLevelEnum(PrivacyLevel value)
    {
        return new PrivacyLevelEnum(value);
    }

    public static implicit operator PrivacyLevel(PrivacyLevelEnum value)
    {
        return value.ToEnum();
    }
}
```

Which gives you some boiler-plate that could be easily generated e.g. using T4 templates.

Which finally ends you up with using:

```
public class CalendarEntry : Entity
{

    public virtual PrivacyLevelEnum PrivacyLevel { get; set; } = new PrivacyLevelEnum();

}
```

But since you have implicit conversion in place, class declarations are the only ones to be aware of the helper types

Where is this method found string.ToEnum<TEnum>(); - is it an extension method – Ken Dec 5 '16 at 16:16

@Ken yes - `public static T ToEnum<T>(this string value) { return (T)Enum.Parse(typeof(T), value, true); }` – Pawel Gorczynski Jan 9 '17 at 9:42

22

You can save the enum to the db as a string, and I agree with dotctor that it is not the best idea, but if you need to, you need to make a few changes.

```
public class User
{
    public int ID { get; set; }
    public string Name { get; set; }
    public List<Wepon> WeposInList { get; set; }

    [Column("Type")]
    public string TypeString
    {
       get { return Type.ToString(); }
       private set { Type= value.ParseEnum<Type>(); }
    }

    [NotMapped]
    public Type Type { get; set; }
}
```

Add this extension class to your project.

```
public static class StringExtensions
{
    public static T ParseEnum<T>(this string value)
    {
        return (T)Enum.Parse(typeof(T), value, true);
    }
}
```

Full details are here - http://NoDogmaBlog.bryanhogan.net/2014/11/saving-enums-as-strings-with-entity-framework/

1 how would I use this to track both the label and the id in the database? I'm getting a null argument exception when I fetch a record. – TWilly Sep 14 '16 at 22:34

1 Try removing [NotMapped] and set it to some other column in the db – Bryan Sep 15 '16 at 15:27

2 the only actual answer to the OP's question. :) – toddmo Feb 25 '17 at 0:43

Wouldn't this violate SOLID? By doing so your entity would not only describe your data, it would describe business as well. What I do is to have an enum, which I would seed to the database, and then refer it from there. – Nikola Jun 27 '18 at 12:04

4

It's not a good idea to store them as string but you can create look up tables for your enums with ef enum to lookup and it is very easy to use.

answered Sep 12 '15 at 18:39

Hamid Pourjam
17.4k 8 44 63

2 Glad you like it :-) See also stackoverflow.com/q/11167665/10245 – Tim Abell Jun 1 '16 at 15:03

3 Can you please explain a bit why it is not a good idea to store enum as `nvarchar()` ? Isn't text more readable than an integer in the database? – Blaise Feb 1 '18 at 20:49

Flexibility and performance @Blaise – Hamid Pourjam Feb 3 '18 at 5:52

1

I thing, that it is much more useful to store them as `int` because you can then cast the `int` from DB very easily to the `enum` .

But if it what you desire, there are two approaches. You can save `Type.Zombie.ToString()` (or `Type.Human.ToString()` respectively) to database (which will be "Zombie"), or you can obtain the value of `DescriptionAttribute` , which you are using and save that to the DB. How to get the description is described here. - It will be also "Zombie" in this case, but it may be whatever else you write in the `Description()` .

If you use `ToString` you can then use `Enum.Parse` to get the instance of the `enum` back. If you use the description, it is not that easy.

1 Entity Framework supports enums now and there is no need to cast the int from DB to it's enum respective value . – Hamid Pourjam Sep 12 '15 at 19:26

0

```
modelBuilder.Entity<DataSet>().Property(d => d.SemanticType).HasConversion(new
EnumToStringConverter<DataSetSemanticType>());
```

answered Mar 20 at 12:14

Martin Staufcik
2,224 1 19 28

This is only available in EF core. – Seth Jun 10 at 17:27