Ways to save enums in database

110

72

What is the best way to save enums into a database?

I know Java provides `name()` and `valueOf()` methods to convert enum values into a String and back. But are there any other (flexible) options to store these values?

Is there a smart way to make enums into unique numbers (`ordinal()` is not safe to use)?

Update:

Thanks for all awesome and fast answers! It was as I suspected.

However a note to 'toolkit'; That is one way. The problem is that I would have to add the same methods to each Enum type I create. Thats a lot of duplicated code and, at the moment, Java does not support any solutions for this (a Java enum cannot extend other classes).

java database enums

edited Jan 9 '18 at 9:14
pants
102 10

asked Oct 23 '08 at 14:11
user20298
725 3 11 10

2 Why is ordinal() not safe to use? – Michael Myers ♦ Oct 23 '08 at 14:11

What kind of database? MySQL has an enum type, but I don't think it's standard ANSI SQL. – Sherm Pendley Oct 23 '08 at 14:19

5 Because any enumerative additions must then be put on the end. Easy for an unsuspecting developer to mess this up and cause havoc – oxbow_lakes Oct 23 '08 at 14:23

1 I see. Guess it's a good thing I don't deal with databases much, because I probably wouldn't have thought of that until it was too late. – Michael Myers ♦ Oct 23 '08 at 16:36

151

We *never* store enumerations as numerical ordinal values anymore; it makes debugging and support way too difficult. We store the actual enumeration value converted to string:

```
public enum Suit { Spade, Heart, Diamond, Club }

Suit theSuit = Suit.Heart;

szQuery = "INSERT INTO Customers (Name, Suit) " +
          "VALUES ('Ian Boyd', %s)".format(theSuit.name());
```

and then read back with:

```
Suit theSuit = Suit.valueOf(reader["Suit"]);
```

The problem was in the past staring at Enterprise Manager and trying to decipher:

```
Name                Suit
==================  ==========
Shelby Jackson      2
Ian Boyd            1
```

verses

```
Name                Suit
==================  ==========
Shelby Jackson      Diamond
Ian Boyd            Heart
```

the latter is much easier. The former required getting at the source code and finding the numerical values that were assigned to the enumeration members.

Yes it takes more space, but the enumeration member names are short, and hard drives are cheap, and it is much more worth it to help when you're having a problem.

Additionally, if you use numerical values, you are tied to them. You cannot nicely insert or rearrange the members without having to

would have to become :

```
public enum Suit {
      Unknown = 4,
      Heart = 1,
      Club = 3,
      Diamond = 2,
      Spade = 0 }
```

in order to maintain the legacy numerical values stored in the database.

How to sort them in the database

The question comes up: lets say i wanted to order the values. Some people may want to sort them by the enum's ordinal value. Of course, ordering the cards by the numerical value of the enumeration is meaningless:

```
SELECT Suit FROM Cards
ORDER BY SuitID; --where SuitID is integer value(4,1,3,2,0)

Suit
------
Spade
Heart
Diamond
Club
Unknown
```

That's not the order we want - we want them in enumeration order:

```
SELECT Suit FROM Cards
ORDER BY CASE SuitID OF
    WHEN 4 THEN 0 --Unknown first
    WHEN 1 THEN 1 --Heart
    WHEN 3 THEN 2 --Club
    WHEN 2 THEN 3 --Diamond
    WHEN 0 THEN 4 --Spade
    ELSE 999 END
```

```
SELECT Suit FROM Cards
ORDER BY Suit; --where Suit is an enum name

Suit
-------
Club
Diamond
Heart
Spade
Unknown
```

But that's not the order we want - we want them in enumeration order:

```
SELECT Suit FROM Cards
ORDER BY CASE Suit OF
    WHEN 'Unknown' THEN 0
    WHEN 'Heart'   THEN 1
    WHEN 'Club'    THEN 2
    WHEN 'Diamond' THEN 3
    WHEN 'Space'   THEN 4
    ELSE 999 END
```

My opinion is that this kind of ranking belongs in the user interface. If you are sorting items based on their enumeration value: you're doing something wrong.

But if you wanted to really do that, i would create a `Suits` dimension table:

```
| Suit       | SuitID       | Rank           | Color  |
|------------|--------------|----------------|--------|
| Unknown    | 4            | 0              | NULL   |
| Heart      | 1            | 1              | Red    |
| Club       | 3            | 2              | Black  |
| Diamond    | 2            | 3              | Red    |
| Spade      | 0            | 4              | Black  |
```

This way, when you want to change your cards to use ***Kissing Kings*** **New Deck Order** you can change it for display purposes without throwing away all your data:

```
| Diamond    | 2           | 2           | Red     | 1          |
| Club       | 3           | 3           | Black   | -1         |
| Heart      | 1           | 4           | Red     | -1         |
```

Now we are separating an internal programming detail (enumeration name, enumeration value) with a display setting meant for users:

```
SELECT Cards.Suit
FROM Cards
   INNER JOIN Suits ON Cards.Suit = Suits.Suit
ORDER BY Suits.Rank,
   Card.Rank*Suits.CardOrder
```

edited Aug 24 '18 at 19:24

answered Oct 23 '08 at 14:21

Ian Boyd
126k 198 706 1024

21 toString is often overriden to provide display value. name() is a better choice as it's by definition the counterpart of valueOf() – ddimitrov Oct 23 '08 at 14:40

Worht noting that enum classes can be extended and you can add fields / methods. I've used this to make enums which values which correspond to bit fields (1,2,4,8...) so they can be OR'd together. – basszero Oct 23 '08 at 18:06

8 I strongly disagree with this, if enum persistence is required then should not persist names. as far as reading it back goes it is even simpler with value instead of name can just typecast it as SomeEnum enum1 = (SomeEnum)2; – mamu Sep 3 '09 at 3:49

3 mamu: What happens when the numeric equivalents change? – Ian Boyd Sep 4 '09 at 15:34

2 I would discourage anyone using this approach. Tying yourself to string representation limits code flexibility and refactoring. You should better use unique ids. Also storing strings wastes storage space. – Tautvydas Apr 18 '14 at 18:57

37

Unless you have specific performance reasons to avoid it, I would recommend using a separate table for the enumeration. Use foreign key integrity unless the extra lookup really kills you.

Suits table:

Players table

```
player_name suit_id
Ian Boyd           4
Shelby Lake        2
```

1. If you ever refactor your enumeration to be classes with behavior (such as priority), your database already models it correctly
2. Your DBA is happy because your schema is normalized (storing a single integer per player, instead of an entire string, which may or may not have typos).
3. Your database values (`suit_id`) are independent from your enumeration value, which helps you work on the data from other languages as well.

edited Jun 28 '10 at 2:09

Ian Boyd
126k 198 706 1024

answered Jan 6 '09 at 5:55

Tom
9,061 3 35 47

13 While I agree it is nice to have it normalized, and constrained in the DB, this does cause updates in two places to add a new value (code and db), which might cause more overhead. Also, spelling mistakes should be nonexistent if all updates are done programatically from the Enum name. – Jason Oct 13 '09 at 18:23

3 I agree with the comment above. An alternative enforcement mechanism at the database level would be to write a constraint trigger, which would reject inserts or updates that try to use an invalid value. – Steve Perkins Jun 21 '12 at 16:11

1 Why would I want to declare the same information in two places? Both in CODE `public enum foo {bar}` and `CREATE TABLE foo (name varchar);` that can easily get out of sync. – ebyrob Nov 11 '16 at 16:17

Re: #1, can't `enum` s already have behavior? – Charles Wood Mar 3 '17 at 20:57

If we take the accepted answer at face value, that is that the enum names are only used for manual investigations, then this answer is indeed the best option. Also, if you go on changing enumeration order or values or names, you will always have much more problems than maintaining this extra table. Especially when you only need it (and may choose to create only temporarily) for debugging and support. – afk5min Jun 1 '17 at 17:51

oxbow_lakes
109k 49 293 434

1 I'm using a hybrid approach of your solution and @Ian Boyd's solution with great success. Thanks for the tip! – technomalogical Jul 13 '09 at 14:04

5

As you say, ordinal is a bit risky. Consider for example:

```
public enum Boolean {
    TRUE, FALSE
}

public class BooleanTest {
    @Test
    public void testEnum() {
        assertEquals(0, Boolean.TRUE.ordinal());
        assertEquals(1, Boolean.FALSE.ordinal());
    }
}
```

If you stored this as ordinals, you might have rows like:

```
> SELECT STATEMENT, TRUTH FROM CALL_MY_BLUFF

"Alice is a boy"      1
"Graham is a boy"     0
```

But what happens if you updated Boolean?

```
public enum Boolean {
    TRUE, FILE_NOT_FOUND, FALSE
}
```

This means all your lies will become misinterpreted as 'file-not-found'

43.1k 14 97 129

3 +1 for the reference to thedailywtf.com/Articles/What_Is_Truth_0x3f_.aspx – Matthijs Bierman Mar 9 '12 at 9:09

3

We just store the enum name itself - it's more readable.

We did mess around with storing specific values for enums where there are a limited set of values, e.g., this enum that has a limited set of statuses that we use a char to represent (more meaningful than a numeric value):

```
public enum EmailStatus {
    EMAIL_NEW('N'), EMAIL_SENT('S'), EMAIL_FAILED('F'), EMAIL_SKIPPED('K'), UNDEFINED('-
');

    private char dbChar = '-';

    EmailStatus(char statusChar) {
        this.dbChar = statusChar;
    }

    public char statusChar() {
        return dbChar;
    }

    public static EmailStatus getFromStatusChar(char statusChar) {
        switch (statusChar) {
        case 'N':
            return EMAIL_NEW;
        case 'S':
            return EMAIL_SENT;
        case 'F':
            return EMAIL_FAILED;
        case 'K':
            return EMAIL_SKIPPED;
        default:
            return UNDEFINED;
        }
    }
```

JeeBee
16k 4 44 58

If you don't want to maintain a switch statement and can ensure that dbChar is unique you could use something like: public static EmailStatus getFromStatusChar(char statusChar) { return Arrays.stream(EmailStatus.values()) .filter(e -> e.statusChar() == statusChar) .findFirst() .orElse(UNDEFINED); } – Kuchi Nov 24 '16 at 9:51

3

For a large database, I am reluctant to lose the size and speed advantages of the numeric representation. I often end up with a database table representing the Enum.

You can enforce database consistency by declaring a foreign key -- although in some cases it might be better to not declare that as a foreign key constraint, which imposes a cost on every transaction. You can ensure consistency by periodically doing a check, at times of your choosing, with:

```
SELECT reftable.* FROM reftable
  LEFT JOIN enumtable ON reftable.enum_ref_id = enumtable.enum_id
WHERE enumtable.enum_id IS NULL;
```

The other half of this solution is to write some test code that checks that the Java enum and the database enum table have the same contents. That's left as an exercise for the reader.

edited May 24 '17 at 9:43

naXa
16.1k 11 102 152

answered Oct 23 '08 at 15:06

Roger Hayes
83 5

1 Say the average enumeration name length is 7 characters. Your `enumID` is four bytes, so you have an extra three bytes per row by using names. 3 bytes x 1 million rows is 3MB. – Ian Boyd Dec 1 '11 at 15:06

@IanBoyd: But an `enumId` surely fits in two bytes (longer enums are not possible in Java) and most of them fit in a single byte (which some DB support). The saved space is negligible, but the faster comparison and the fixed length should help. – maaartinus Jan 24 '14 at 13:23

```
        // possibly quote value?
        return name;
    }

    public static <E extends Enum<E>> E deserialize(Class<E> enumType, String dbVal) {
        // possibly handle unknown values, below throws IllegalArgEx
        return Enum.valueOf(enumType, dbVal.trim());
    }

    // Sample use:
    String dbVal = getSerializedForm(Suit.SPADE);
    // save dbVal to db in larger insert/update ...
    Suit suit = deserialize(Suit.class, dbVal);
```

answered Oct 23 '08 at 18:00

Dov Wasserman
2,418 14 13

Nice to use this with a default enum value to fall back on in deserialize. For example, catch the IllegalArgEx and return Suit.None. – Jason Oct 13 '09 at 18:21

2

All my experience tells me that safest way of persisting enums anywhere is to use additional code value or id (some kind of evolution of @jeebee answer). This could be a nice example of an idea:

```
enum Race {
    HUMAN ("human"),
    ELF ("elf"),
    DWARF ("dwarf");

    private final String code;

    private Race(String code) {
        this.code = code;
    }

    public String getCode() {
```

Now you can go with any persistence referencing your enum constants by it's code. Even if you'll decide to change some of constant names, you always can save code value (e.g. `DWARF("dwarf")` to `GNOME("dwarf")`)

Ok, dive some more deeper with this conception. Here is some utility method, that helps you find any enum value, but first lets extend our approach.

```
interface CodeValue {
    String getCode();
}
```

And let our enum implement it:

```
enum Race implement CodeValue {...}
```

This is the time for magic search method:

```
static <T extends Enum & CodeValue> T resolveByCode(Class<T> enumClass, String code) {
    T[] enumConstants = enumClass.getEnumConstants();
    for (T entry : enumConstants) {
        if (entry.getCode().equals(code)) return entry;
    }
    // In case we failed to find it, return null.
    // I'd recommend you make some log record here to get notified about wrong logic,
perhaps.
    return null;
}
```

And use it like a charm: `Race race = resolveByCode(Race.class, "elf")`

edited Jul 9 '15 at 23:28

answered Jul 9 '15 at 23:22

Metaphore

646 6 13

```
public enum FurthitMethod {

    Apple,
    Orange,
    Lemon
}
```

In the entity class, define `@Enumerated(EnumType.STRING)` :

```
@Enumerated(EnumType.STRING)
@Column(name = "Fruits")
public FurthitMethod getFuritMethod() {
    return fruitMethod;
}

public void setFruitMethod(FurthitMethod authenticationMethod) {
    this.fruitMethod= fruitMethod;
}
```

While you try to set your value to Database, String value will be persisted into Database as " `APPLE` ", " `ORANGE` " or " `LEMON` ".

edited May 27 '16 at 1:06

Unheilig
12.3k 16 57 89

answered May 27 '16 at 0:48

SaravanaC
21 3

1

Multiple values with OR relation for one, enum field. The concept for .NET with storing enum types in database like a byte or an int and using FlagsAttribute in your code.

http://blogs.msdn.com/b/efdesign/archive/2011/06/29/enumeration-support-in-entity-framework.aspx

answered Jan 31 '12 at 17:47

Kryszal
1,213 12 18