Extracted from:

SQL Antipatterns, Volume 1

Avoiding the Pitfalls of Database Programming

The Pragmatic Bookshelf

Raleigh, North Carolina

The Pragmatic Programmers

SQL Antipatterns, Volume 1

Avoiding the Pitfalls of Database Programming

Bill Karwin

edited by Jacquelyn Carter

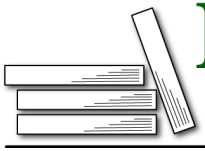
SQL Antipatterns, Volume 1

Avoiding the Pitfalls of Database Programming

Bill Karwin

The Pragmatic Bookshelf

Raleigh, North Carolina



For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit *https://pragprog.com*.

The team that produced this book includes:

CEO: Dave Rankin
COO: Janet Furlow
Managing Editor: Tammy Coron
Development Editor: Jacquelyn Carter
Copy Editor: Karen Galle
Indexing: Potomac Indexing, LLC
Layout: Gilson Graphics
Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact *support@pragprog.com*.

For international rights, please contact *rights@pragprog.com*.

ISBN-13: 978-1-68050-898-7
Encoded using the finest acid-free high-entropy binary digits.
Book version: P1.0—November 2022

To my wife Jan, my best supporter.

Science is feasible when the variables are few and can be enumerated;
when their combinations are distinct and clear.

➤ *Paul Valéry*

CHAPTER 11

31 Flavors

In a personal contact information table, the *salutation* is a good example of a column that can have only a few values. Once you support *Mr.*, *Mrs.*, *Ms.*, *Dr.*, and *Rev.*, you've accounted for virtually everyone. You could specify this list in the column definition, using a data type or a constraint, so that no one can accidentally enter an invalid string into the salutation column.

31-Flavors/intro/create-table.sql

```
CREATE TABLE PersonalContacts (
  -- other columns
  salutation VARCHAR(4)
    CHECK (salutation IN ('Mr.', 'Mrs.', 'Ms.', 'Dr.', 'Rev.')),
);
```

That should settle it—you assume there are no other salutations to support.

Unfortunately, your boss tells you that your company is opening a subsidiary in France. You need to support the salutations *M.*, *Mme.*, and *Mlle.* Your mission is to alter your contact table to permit these values. This is a delicate job and may not be possible without interrupting availability of that table.

You also thought your boss mentioned that the company is trying to open an office next month in Brazil.

Objective: Restrict a Column to Specific Values

Restricting a column's values to a fixed set of values is very useful. If we can ensure that the column never contains an invalid entry, it can simplify use of that column. For example, in the Bugs table of our example database, the status column indicates whether a given bug is *NEW*, *IN PROGRESS*, *FIXED*, and so on. The significance of each of these status values depends on how we manage bugs in our project, but the point is that the data in the column must be one of these values.

Ideally, we need the database to reject invalid data:

31-Flavors/obj/insert-invalid.sql
```
INSERT INTO Bugs (status) VALUES ('NEW'); -- OK

INSERT INTO Bugs (status) VALUES ('BANANA'); -- Error!
```

Antipattern: Specify Values in the Column Definition

Many people choose to specify the valid data values when they define the column. The column definition is part of the *metadata*—the definition of the table structure itself.

For example, you could define a *check constraint* on the column. This constraint disallows any insert or update that would make the constraint false.

31-Flavors/anti/create-table-check.sql
```
CREATE TABLE Bugs (
  -- other columns
  status   VARCHAR(20) CHECK (status IN ('NEW', 'IN PROGRESS', 'FIXED'))
);
```

MySQL supports a nonstandard data type called `ENUM` that restricts the column to a specific set of values.

31-Flavors/anti/create-table-enum.sql
```
CREATE TABLE Bugs (
  -- other columns
  status ENUM('NEW', 'IN PROGRESS', 'FIXED'),
);
```

In MySQL's implementation, you declare the values as strings, but internally the column is stored as the ordinal number of the string in the enumerated list. The storage is thus compact, but when you sort a query by this column, the default order of the result is by the ordinal value, not alphabetically by the string value. You may not expect this behavior.

Other solutions include *domains* and *user-defined types* (UDTs). You can use these to restrict a column to a specific set of values and conveniently apply the same domain or data type to several columns within your database. Unfortunately, these features are not supported in a standard way among brands of RDBMSs yet.

Finally, you could write a trigger that contains the set of permitted values and raises an error unless the `status` matches one of these values.

All of these solutions share some disadvantages. The following sections describe some of these problems.

Baskin-Robbins "31 Flavors" Ice Cream

In 1953, this famous chain of ice cream parlors offered one flavor for each day of the month. The chain used the slogan *31 Flavors* for many years.

Today, more than sixty years later, Baskin-Robbins offers twenty-one classic flavors, twelve seasonal flavors, sixteen regional flavors, as well as a variety of Bright Choices and Flavors of the Month. Even though its ice cream flavors were once an immutable set that defined its brand, Baskin-Robbins expanded its choices and made them configurable and variable.

The same thing could happen in the project for which you're designing a database—in fact, you should count on it.

What Was the Middle One?

Suppose you're developing a user interface so a user can edit bug reports. To make it guide the user to pick one of the valid status values, you choose to fill a drop-down menu control with these values. You need to query the database for an enumerated list of values that are currently allowed in the status column.

Your first instinct might be to query all the values currently in use, with a simple query like the following one:

31-Flavors/anti/distinct.sql

```
SELECT DISTINCT status FROM Bugs;
```

However, if all the bugs are new, the previous query returns only *NEW*. If you use this result to populate a user interface control for the status of bugs, you could create a chicken-and-egg situation; you can't change a bug to any status other than those currently in use.

To get the complete list of permitted status values, you need to query the definition of that column's metadata. Most SQL databases support *system views* for these kinds of queries, but using them can be complex. For example, if you used MySQL's ENUM data type, you can use the following query to query the INFORMATION_SCHEMA system views:

31-Flavors/anti/information-schema.sql

```
SELECT column_type
FROM information_schema.columns
WHERE table_schema = 'bugtracker_schema'
  AND table_name = 'bugs'
  AND column_name = 'status';
```

You can't simply get the discrete enumeration values from the INFORMATION_SCHEMA in a conventional result set. Instead, you get a string containing

the definition of the check constraint or ENUM data type. For example, the previous query in MySQL returns a column of type LONGTEXT, with the value *ENUM('NEW', 'IN PROGRESS', 'FIXED')*, including the parentheses, commas, and single quotes. You must write application code to parse this string and extract the individual quoted values before you can use them to populate a user interface control.

The queries needed to report check constraints, domains, or UDTs are progressively more complex. Most people choose the better part of valor and manually maintain a parallel list of values in application code. This is an easy way for bugs to affect your project as application data becomes out of sync with the database metadata.

Adding a New Flavor

The most common alterations are to add or remove one of the permitted values. There's no syntax to add or remove a value from an ENUM or check constraint; you can only redefine the column with a new set of values. The following is an example of adding *DUPLICATE* as one new status value in the MySQL ENUM:

31-Flavors/anti/add-enum-value.sql

```
ALTER TABLE Bugs MODIFY COLUMN status
  ENUM('NEW', 'IN PROGRESS', 'FIXED', 'DUPLICATE');
```

You need to know that the previous definition of the column allowed *NEW*, *IN PROGRESS*, and *FIXED*. This leads you back to the difficulty of querying the current set of values as described earlier.

Some database brands can't change the definition of a column unless the table is empty. You might need to dump the contents of the table, redefine the table, and then import your saved data, making the table inaccessible in the meantime. This work is common enough that it has a name: *ETL* for "extract, transform, and load." Other brands of database support restructuring a populated table with ALTER TABLE commands, but it can still be complex and expensive to perform these changes.

As a matter of policy, changing metadata—that is, changing the definition of tables and columns—should be infrequent and with attention to testing and quality assurance. If you need to change metadata to add or remove a value from an ENUM, then you either have to skip the appropriate testing or spend a lot of software engineering effort on short notice to make the change. Either way, these changes introduce risk and destabilize your project.

Old Flavors Never Die

If you make a value obsolete, you could upset historical data. For example, you change your quality control process to replace *FIXED* with two stages, *CODE COMPLETE* and *VERIFIED*:

31-Flavors/anti/remove-enum-value.sql

```
ALTER TABLE Bugs MODIFY COLUMN status
  ENUM('NEW', 'IN PROGRESS', 'CODE COMPLETE', 'VERIFIED');
```

If you remove *FIXED* from the enumeration, you need to decide what to do with bugs whose status was *FIXED*. One possible change is to update all *FIXED* bugs to *VERIFIED*. Another option is set obsolete values to null or a default value. Unfortunately, ALTER TABLE can't guess which one of these changes you want.

You may have to keep an obsolete value that old rows reference. You can't know only from the column definition which values are obsolete, so you exclude them from your user interface. Someone could still choose one of those values.

Portability Is Hard

Check constraints, domains, and UDTs are not supported uniformly among brands of SQL databases. The ENUM data type is a proprietary feature in MySQL. Each brand of database may have a different limit on the length of the list you can give in a column definition. Trigger languages vary as well. These variations make it hard to choose a solution if you need to support multiple brands of database.

How to Recognize the Antipattern

The problems with using ENUM or a check constraint arise when the set of values is not fixed. If you're considering using ENUM, first ask yourself whether the set of values are expected to change or even whether they *might* change. If so, it's probably not a good time to employ an ENUM.

- "We have to take the database offline so we can add a new choice in one of our application's menus. It should take no more than thirty minutes, if all goes well."

  This is a sign that a set of values is baked into the definition of a column. You should never need to interrupt service for a change like this.

- "The status column can have one of the following values. We shouldn't need to revise this list."

Shouldn't need to are weasel words, and this says something quite different from *can't*.

- "The list of values in the application code got out of sync with the business rules in the database—again."

  This is a risk of maintaining information in two different places.

Legitimate Uses of the Antipattern

As we discussed, `ENUM` may cause fewer problems if the set of values is unchanging. It's still difficult to query the metadata for the set of values, but you can maintain a matching list of values in application code without getting out of sync.

`ENUM` is most likely to succeed when it would make no sense to alter the set of permitted values, such as when a column represents an either/or choice with two mutually exclusive values: *LEFT/RIGHT*, *ACTIVE/IN-ACTIVE*, *ON/OFF*, *INTERNAL/EXTERNAL*, and so on.

Check constraints can be used in many ways other than simply to implement an `ENUM`-like mechanism, such as checking that a time interval's `start` is less than its `end`.