Character encoding issues in MySQL

January 19, 2023

Modern databases are ubiquitous, they're used everywhere by applications and scripts, many of which you may have to interact with if you're working in the IT world. Creating a new database and table, then storing data that you can then retrieve later on, seems like a fairly straightforward process. But databases have a lot of settings, and there's a reason corporations hire well trained DBAs to maintain these databases. There are pitfalls you may need to watch for, one of the more common ones being character encoding issues.

If you use MySQL or MariaDB as a database, and if you ever ran into the following error, then this may be the post you've been waiting for:

```
Incorrect string value: '\xF0\x9F\x98\xF0' for column 'ti
```

What this basically says is that the data you're trying to store in the database doesn't fit with the encoding available. The data may be some type of binary, or maybe just text in another language that requires more letters than available in the ASCII set. The result may also not be that straightforward.

Back in the good old days, computers assumed everything would fit in the ASCII character set. This means the letters A to Z, numbers and some extra punctuation. As long as you're storing English text, it would work fine, but it's not hard to see how this breaks down when you switch to French or Chinese. If you're trying to save emoji, then forget about it. So the UTF-8 encoding was invented to provide more memory space for the additional characters. However, for a number of reasons, you don't want to use UTF-8 for your MySQL or MariaDB database. Instead, these are the commands you probably want:

```
ALTER DATABASE databasename CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
ALTER TABLE tablename CONVERT TO CHARACTER SET utf8mb4 COLLATE
utf8mb4_unicode_ci;
```

This command will switch the default character set and collation for a database or a table. The character set is basically UTF-8 with extra fixes, and it should allow every type of currently existing text to fit in your database. The collation defines how this data is stored and how comparisons work. In this case, the last *_ci* means that comparisons will be case-insensitive. That's important to know, because if you use this setting, then these two strings will be considered to be equal:

```
"Hello World" == "hello world"
```

If instead you want to store case-sensitive data, then use the *utf8mb4_bin* collation instead. This will consider text to be like binary data, and any difference at all will cause the comparison to return false.

There are a lot more to be said about character sets and collations, and if you look at the list of available values supported by the MySQL engine, you may be surprised to see hundreds of possibilities. At the end of the day, the settings you use depend on your needs, but these values give a good everyday default.