Go & MySQL

Daniel Nichter

PDX MUG

July 2015

Let’s look at...

1. Basic usage
2. Interpolation
3. Connection pools and leaks
4. Scanning results and NULL
5. Non-MySQL behaviors

Basic usage

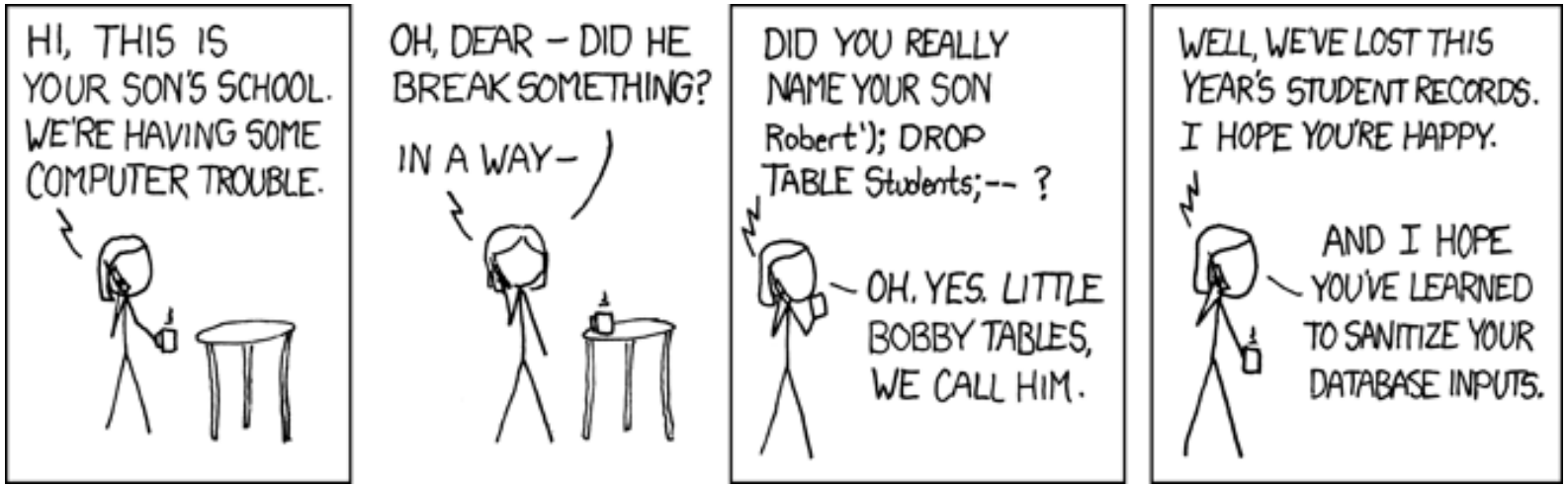
Driver:

github.com/go-sql-driver/mysql

Examples:

github.com/daniel-nichter/go-and-mysql

Interpolation



https://xkcd.com/327/

Interpolation

Without

- Prepared statements
- Binary protocol
- 3x round trip for one-off queries
- Best for repeated queries*

With

- Text protocol
- 1 query = 1 round trip
- Best for one-off queries

* *Prepared statements are per-database*. This is an undocumented MySQL limitation/gotcha.

Interpolation

- Add interpolateParams=true to DSN
- *Not in latest (v1.2)*
- Requires >= 60fe63a
- Seems immune to SQL injection
- Can Prepare() to use prepared statements

Imho, interpolateParams should be enabled by default; we do this in production at Percona.

Connection pools and leaks

“DB is a database handle representing a pool of zero or more underlying connections. It's safe for concurrent use by multiple goroutines.

The sql package creates and frees connections automatically; it also maintains a free pool of idle connections. If the database has a concept of per-connection state, such state can only be reliably observed within a transaction. Once DB. Begin is called, the returned Tx is bound to a single connection. Once Commit or Rollback is called on the transaction, that transaction's connection is returned to DB's idle connection pool. The pool size can be controlled with SetMaxIdleConns.”

http://golang.org/pkg/database/sql/#DB

Connection pools and leaks

Can you spot the leak in 07-leak-kills-mysql.go?

```
$ ./07-leak-kills-mysql
2015/07/19 16:11:02 Failed after doing important work only 16382 times :-(
```

Connection pools and leaks

The leak is not doing “defer stmt.Close()”.

```
$ ./07-leak-kills-mysql
2015/07/19 16:11:02 Failed after doing important work only 16382 times :-(
```

Connection pools and leaks

Leaks are easy to create with pools, concurrency, and goroutine.

A leak can bring down production.

Solution: read docs carefully, know what can leak, and stop it with defer.

Connection pools and leaks

Can leak:

- sql.DB
- sql.Rows
- sql.Stmt
- goroutines
- (There’s probably more)

Scanning results and NULL

Basic, built-in:

```
var col1 string
var col2 int
db.QueryRow("SELECT col1, col2 ...").Scan(&col1, &col2)
```

- 1-to-1 column-to-variable mapping (order is significant)
- Variables can be a struct fields, like &T.col1, but not structs
- If a column is nullable, its variable must be a Null* type
- Simple and direct, but verbose with many columns
- We use this in production; it helps encourage/force simplicity

Scanning results and NULL

Check out SQLX (http://jmoiron.github.io/sqlx/) if you want to scan results into structures by column-field name.

Google for others, or roll your own if you feel like learning some Go internals (reflect).

Scanning results and NULL

Null* types:

- sql.NullBool
- sql.NullFloat64
- sql.NullInt64
- sql.NullString
- mysql.NullTime

Explicitly typecast for others, e.g. uint(sql.Int64)

Scanning results and NULL

Life with Null* types:

```
type T struct {
        name string
}
var person T
var name sql.NullString
db.QueryRow(“SELECT name”).Scan(&name)
if name.Valid {
        person.Name = name.String
}
// If Go wasn’t so awesome, this might be more annoying.
```

Scanning results and NULL

Alternative life with Null* types:

```
type T struct {
        name string
}
var person T
db.QueryRow(“SELECT COALESCE(name, ‘’) ...”).Scan(&person.name)
// With Go and MySQL, I use COALESCE more than ever before.
```

Scanning results and NULL

Think and plan carefully for NULL and Go-equivalent zero values, especially if data comes from or goes to other languages.

- MySQL columns
- Data models (JSON?)
- Languages (Go, Python?, JS?)

Non-MySQL behaviors

- Defaults files (my.cnf) are not used
- localhost is not magic for socket
- Old password auth may not work
- Can be difficult to use blank password
- Uses utf8 character set by default...

Non-MySQL behaviors

Normal

```
character_set_client     | latin1
character_set_connection | latin1
character_set_database   | latin1
character_set_filesystem | binary
character_set_results    | latin1
character_set_server     | latin1
character_set_system     | utf8
```

Go MySQL Driver

```
character_set_client     | utf8
character_set_connection | utf8
character_set_database   | latin1
character_set_filesystem | binary
character_set_results    | utf8
character_set_server     | latin1
character_set_system     | utf8
```

We looked at...

1. Basic usage
2. Interpolation
3. Connection pools and leaks
4. Scanning results and NULL
5. Non-MySQL behaviors

Go & MySQL

mysql> \q
Thank you