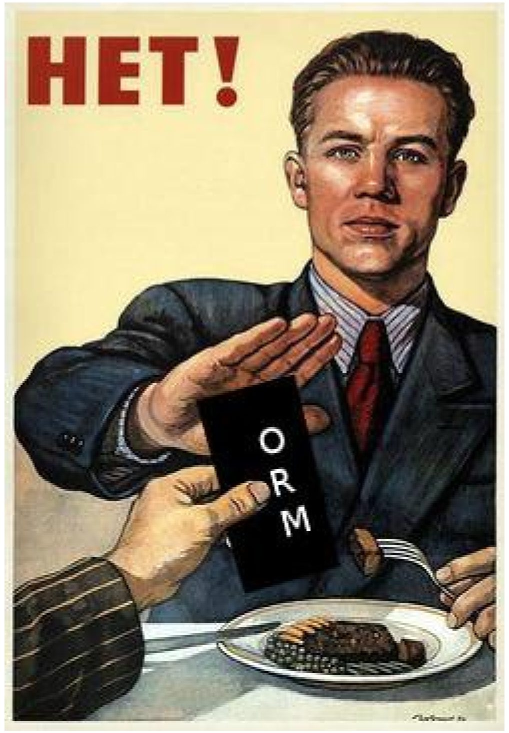
Using prepared SQL statements in Go

without any pain

Max Chechel
github.com/hexdigest

How do we access our databases?



Plain old SQL queries

What’s a prepared SQL statement?

A prepared statement is a feature used to execute the same or similar database statements repeatedly with high efficiency

1. Prepare: The statement template is created by the application and sent to the DBMS. Certain values are left unspecified
   INSERT INTO PRODUCT (name, price) VALUES (?, ?)

2. The DBMS compiles (parses, optimizes, builds query plan) the statement template, and stores the result without executing it.

3. Execute: At a later time, the application binds values for the parameters and the DBMS executes the statement

How regular SQL statements are executed in Go

```
func main() {
	db, err := sql.Open("mysql", "root:root@tcp(localhost:3306)/mysql")
	if err != nil {
		panic(err)
	}
	const query = `SELECT CONCAT("Hello ", ?, "!")`
	var s string
	if err := db.QueryRow(query, "World").Scan(&s); err != nil {
		panic(err)
	}
	fmt.Println(s)
}
```

How regular SQL statements are executed in Go

```
$ go run regular.go

Hello World!
```

How regular SQL statements are executed in Go

Connect	root@localhost on mysql using TCP/IP
Query	SELECT @@max_allowed_packet
Prepare	SELECT CONCAT("Hello ", ?, "!")
Execute	SELECT CONCAT("Hello ", 'World', "!")
Close stmt	

Let’s prepare a SQL statement

```
func main() {
	db, err := sql.Open("mysql", "root:root@tcp(localhost:3306)/mysql")
	if err != nil {
		panic(err)
	}
	stmt, err := db.Prepare(`SELECT CONCAT("Hello ", ?, "!")`)
	if err != nil {
		panic(err)
	}
	var s string
	if err := stmt.QueryRow("World").Scan(&s); err != nil {
		panic(err)
	}
	fmt.Println(s)
}
```

Let’s prepare a SQL statement

Connect	root@localhost on mysql using TCP/IP
Query	SELECT @@max_allowed_packet
Prepare	SELECT CONCAT("Hello ", ?, "!")
Execute	SELECT CONCAT("Hello ", 'World', "!")
~~**Close stmt**~~	

Two main approaches of using prepared statements

1. Initialize prepared statements on program start and use them after
2. “Initialize” prepared statement in place

Initialization on program start

```
var stmtHello *sql.Stmt

func init() {
	db, err := sql.Open("mysql", "root:root@tcp(localhost:3306)/mysql")
	if err != nil {
		panic(err)
	}
	stmtHello, err = db.Prepare(`SELECT CONCAT("Hello ", ?, "!")`)
	if err != nil {
		panic(err)
	}
}
```

Initialization on program start

```go
func main() {
	var s string
	if err := stmtHello.QueryRow("World").Scan(&s); err != nil {
		panic(err)
	}
	fmt.Println(s)
}
```

Does anybody remember what the actual query looks like?

In place initialization

```
type statementsRegistry struct {
	db         *sql.DB
	statements map[string]*sql.Stmt
}
func (sr *statementsRegistry) statement(s string) (*sql.Stmt, error) {
	if stmt, ok := sr.statements[s]; ok {
		return stmt, nil
	}
	stmt, err := sr.db.Prepare(s)
	if err != nil {
		return nil, err
	}
	sr.statements[s] = stmt
	return stmt, nil
}
```

In place initialization

```
func main() {
	db, err := sql.Open("mysql", "root:root@tcp(localhost:3306)/mysql")
	if err != nil { panic(err) }
	sr := newStatementsRegistry(db)
	stmt, err := sr.statement(`SELECT CONCAT("Hello ", ?, "!")`)
	if err != nil {
		panic(err)
	}
	var s string
	if err := stmt.QueryRow("World").Scan(&s); err != nil {
		panic(err)
	}
	fmt.Println(s)
}
```

Disadvantages of using prepared statements

1. More code → more bugs
2. Dynamic queries → memory leaks
3. Inconvenient to use

Can we fix this?

Yes!

github.com/hexdigest/prep

Using prep

```
$ prep -f github.com/meetup/demo

$ cat github.com/meetup/demo/prepared_statements.go
package main

var prepStatements = []string{
	"SELECT CONCAT(\"Hello \", ?, \"!\")",
}
```

Using prep

```
func main() {
	var db prep.Connector
	var err error
	db, err = sql.Open("mysql", "root:root@tcp(localhost:3306)/mysql")
	if err != nil {
		panic(err)
	}
	db, err = prep.NewConnection(db, prepStatements)
	if err != nil {
		panic(err)
	}
	const query = `SELECT CONCAT("Hello ", ?, "!")`
	var s string
	if err := db.QueryRow(query, "World").Scan(&s); err != nil {
		panic(err)
	}
	fmt.Println(s)
}
```

Use prepared SQL statements!

Questions???

Max Chechel
github.com/hexdigest