Persisting Data

Alex Schultz
SOFTWARE ENGINEER | AWS ML HERO
@AlexCSchultz

Overview

DB setup

Connecting to a Database

Querying Data

Executing SQL Statements

Connection Pooling

- Configuration
- Contexts

Uploading and Downloading Files

- multipart/form-data
- io.Copy

sql.Open

```
func Open(driverName, dataSourceName string) (*DB, error)
```

DB Type

- Configurable pool of zero or more connections
- Creates and frees connections automatically
- Thread-safe

database.go

```go
import “database/sql”

. . .


var DbConn *sql.DB


func SetupDatabase(){
  var err error
  DbConn, err = sql.Open(“mysql”, “root:password123@tcp(127.0.0.1:3306)/inventorydb”)
  if err != nil {
     log.Fatal(err)
  }
}
```

https://github.com/golang/go/wiki/SQLDrivers

DB.Query

```
func (db *DB) Query(query string, args …interface{}) (*Rows, error)
```

Rows Type

- Result of a query
- Use “Next” to advance
- Needs to be closed

Rows.Scan

```
func (rs *Rows) Scan(dest …interface{}) error
```

product.data.go

```
. . .
results, err := db.Query(`select productId, manufacturer, sku from products`)
if err != nil {
  log.Fatal(err)
}
defer results.Close()
products := make([]Product, 0)
for results.Next(){
  var product Product
  results.Scan(&product.ProductID, &product.Manufacturer, &product.Sku …)
  products = append(products, product)
}
```

DB.QueryRow

```
func (db *DB) QueryRow(query string, args …interface{}) *Row
```

Row.Scan

```
func (rs *Row) Scan(dest …interface{}) error
```

DB.Exec

```
func (rs *DB) Exec(query string, args …interface{}) (Result, error)
```

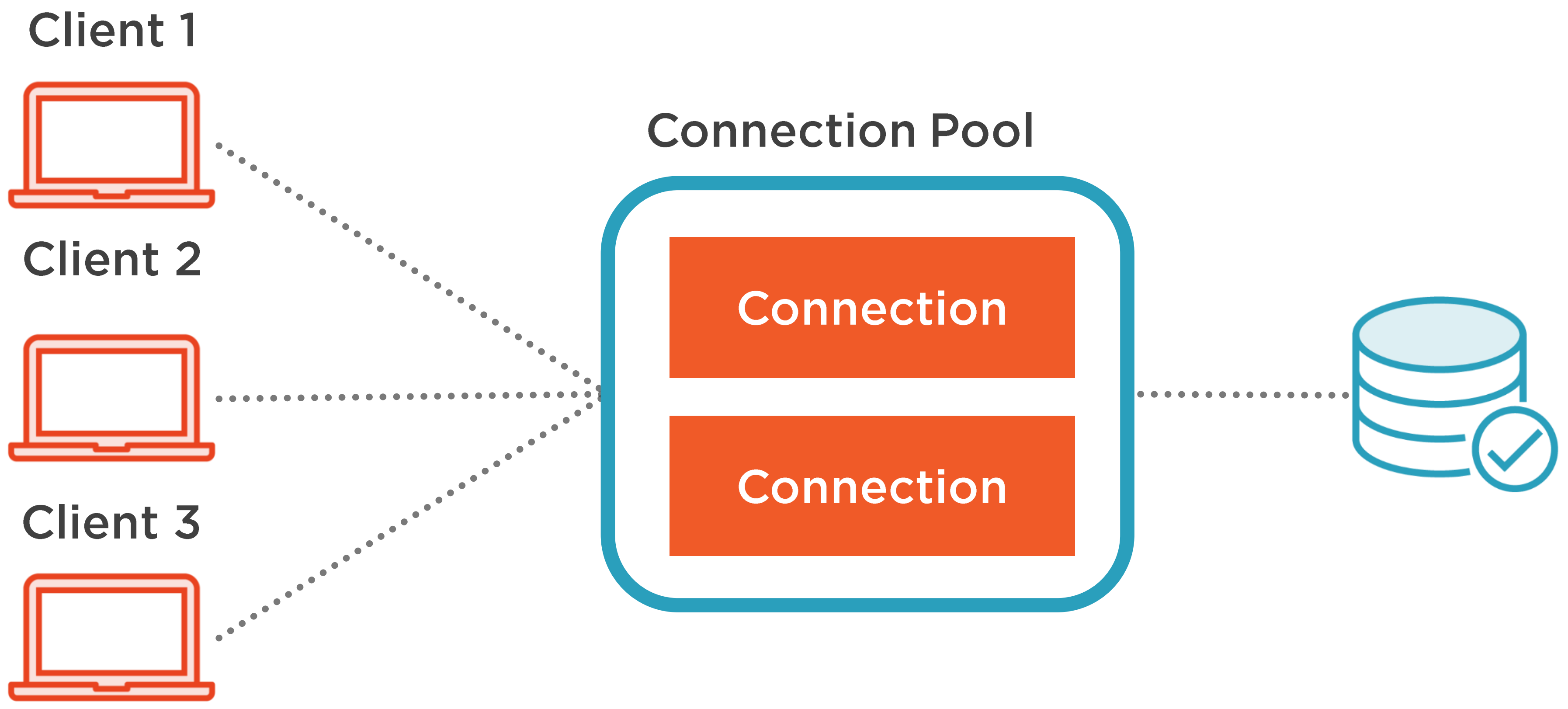
sql.Result

```
type Result interface {
  LastInsertId() (int64, error)
  RowsAffected() (int64, error)
}
```

product.data.go

```
. . .
result, err := db.Exec(`update products set sku=? where productid=?`,
   product.Sku,
   product.ProductID)
if err != nil {
  log.Fatal(err)
}
fmt.Printf("number of affected rows %d\n", result.RowsAffected())
. . .
```

Managing Connections



Connection Pooling

Connection Max Lifetime

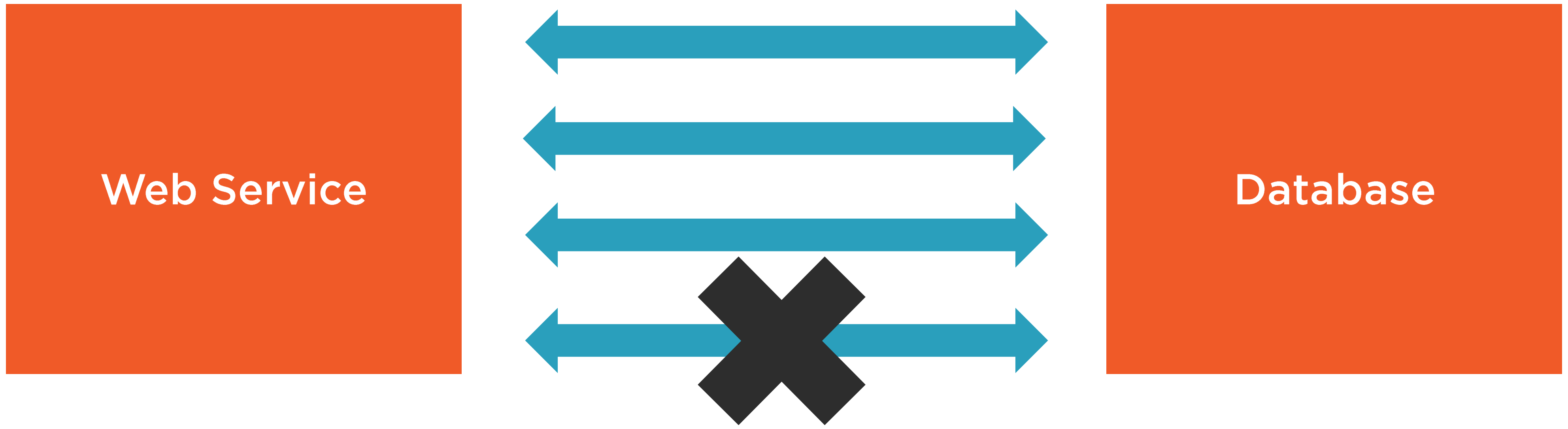
Sets the maximum amount of time a connection may be used

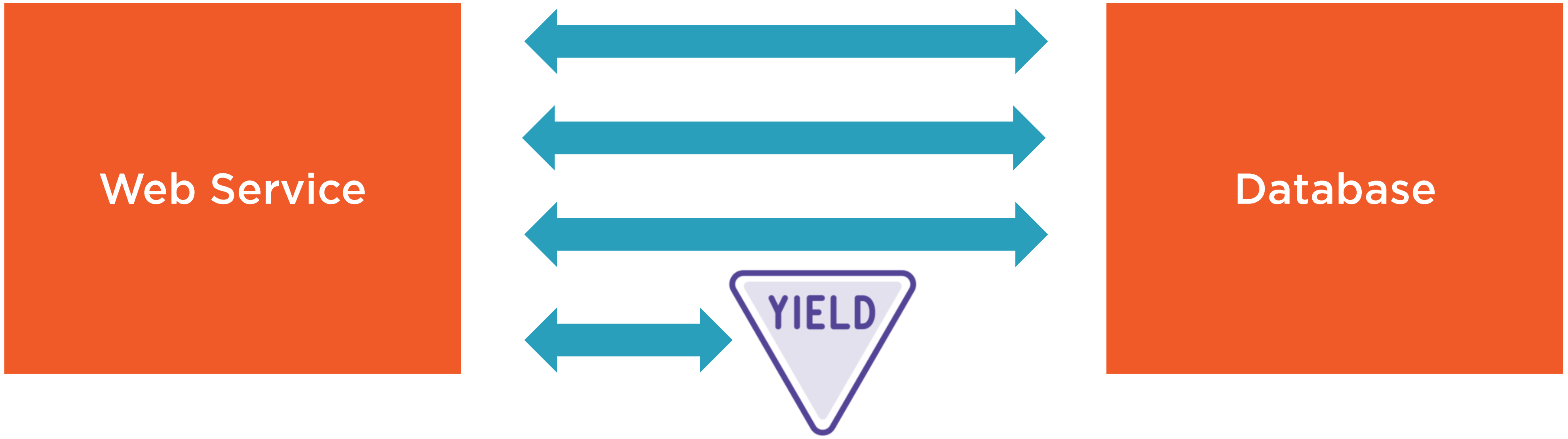
Max Idle Connections

Sets the maximum number of connections in the idle connection pool

Max Open Connections

Sets the maximum number of open connections to the database


Web Service
Database


Web Service
YIELD
Database

Context

Allows you to set a deadline, cancel a signal, or set other request-scoped values across API boundaries and between processes.

product.data.go

```
. . .
ctx, cancel := context.WithTimeout(context.Background(), 3*time.Second)
results, err := db.QueryContext(ctx, `select productId, manufacturer, sku from products`)
if err != nil {
  log.Fatal(err)
}
defer results.Close()
products := make([]Product, 0)
for results.Next(){
  results.Scan(&product.ProductID, &product.Manufacturer, &product.Sku …)
  products = append(products, product)
}
. . .
```

QueryContext

QueryRowContext

ExecContext

File Upload

base64 encode

Convert the file to a string and include in JSON payload

multipart/form-data

Uses an HTTP form to submit the raw data

Encoding.DecodeString

```
func (enc *Encoding) DecodeString(s string) ([]byte, error)
```

product.service.go

```
str := “SGVsbG8gV29ybGQ=”
output, err := base64.StdEncoding.DecodeString(str)
if err != nil {
  log.Fatal(err)
}
fmt.Printf(“%q\n”, output)
```

```
Hello World
```

Request.FormFile

```
func (r *Request) FormFile(key string) (multipart.File, *multipart.FileHeader,
error)
```

multipart.File

```
type File interface {
  io.Reader
  io.ReaderAt
  io.Seeker
  io.Closer
}
```

multipart.FileHeader

```
type FileHeader struct {
  Filename string
  Header textproto.MIMEHeader
  Size int64
}
```

receipt.service.go

```go
func uploadFileHandler(w http.ResponseWriter, r *http.Request){
  r.ParseMultiPartForm(5 << 20) // 5 Mb
  file, handler, err := r.FormFile(“uploadFileName”)
  if err != nil {
    fmt.Println(“error reading file from request”)
    return
  }
  defer file.Close()
  f, err := os.OpenFile(“./filepath/” + handler.Filename, os.O_WRONLY|os.O_CREATE, 0666)
  defer f.Close()
  io.Copy(f, file)
}
```

receipt.service.go

```go
func downloadFileHandler(w http.ResponseWriter, r *http.Request){
  filename = “gopher.png”
  file, err := os.Open(fileName)
  if err != nil {
    fmt.Println(“error reading file”)
    return
  }
  defer file.Close()
  w.Header.Set(“Content-Disposition”, “attachment; filename=“+fileName)
  io.Copy(w, file)
}
```

Summary

DB setup

Connecting to a Database

Querying Data

Executing SQL Statements

Connection Pooling

- Configuration
- Contexts

Uploading and Downloading Files

- multipart/form-data