Menggunakan Drive SQL pada Golang

Adapted from: [Menggunakan MYSQL pada GOLANG. Ketika belajar Golang, hampir tidak… | by Alfian Maulana Malik | Programmer Geek | Medium]

Tambahkan import berikut untuk menerapkan drive SQL pada golang:

```
import (
    "database/sql"
    "fmt"

    _ "github.com/go-sql-driver/mysql"
)
```

Lakukan perintah :

```
go mod tidy
```

agar library dapat terupdate secara otomatis

Mengestablish koneksi antar SQL dan Golang:

```
db, err := sql.Open("mysql", "root:@tcp(127.0.0.1:3306)/golang_latihan")

    if err != nil {
        panic(err.Error())
    } else {
        fmt.Println("konek")
    }

    defer db.Close()
```

Variabel “db” berguna untuk proses QUERY selanjutnya. Untuk itu, pada tahap establish ini harus tidak boleh terdapat error. Karena akan berpengaruh ke proses-proses selanjutnya.

Contoh penerapan INSERT:

```
func insert(db *sql.DB) {
    insert, err := db.Query("INSERT INTO biodata (bio_nama, bio_ttl, bio_email)
VALUES ('Pangestu', '1995-05-05', 'gusti.pangestu@binus.edu')")
    if err != nil {
        panic(err.Error())
    } else {
        insert.Close()
    }
}
```

Contoh penerapan DELETE:

```
func delete(db *sql.DB) {
    del, err := db.Query("DELETE FROM lb_user WHERE user_id=1")
    if err != nil {
        panic(err.Error())
    } else {
        del.Close()
    }
}
```

Insert data (versi 2)

Selain menerapkan db.Query, kita juga dapat menerapkan db.Prepare dan db.Exec untuk memberikan pattern pada mekanisme query yang kita lakukan:

Golang MySQL CRUD Example - Golang Docs

Select data menggunakan mekanisme REST API

Untuk melakukan select data kita perlu menampung hasil select (biasanya lebih dari 1 data) ke dalam sebuah variable. Dalam kasus ini kita dapat membuat struct seperti dibawah:

```
type Mhs struct {
    Nim   int    `json:"nim"`
    Nama  string `json:"nama"`
    Email string `json:"email"`
    Hp    string `json:"no_hp"`
}
```

Sesuaikan tipe data yang ada pada struct dengan tipe data yang ada pada Database kita. Lalu sesuaikan juga nama variable yang ada di “json:[variable]” sesuai dengan nama column pada table database kita.

Buat fungsi untuk menghandel koneksi database agar nantinya koding kita menjadi lebih rapi dengan:

```
func konek() sql.DB {
    db, err := sql.Open("mysql", "root:@tcp(127.0.0.1:3306)/la_golang")
    cekError(err)
    defer db.Close()
    return *db
}
```

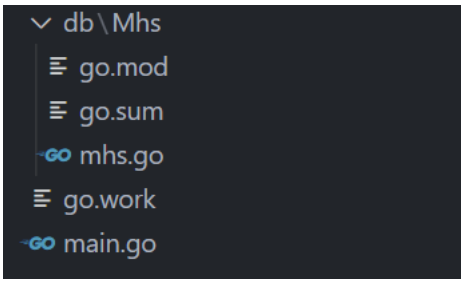
Buat fungsi untuk select data:

```
func SelectAll() []Mhs {
```

```
    db := konek()

    statement, err := db.Query("SELECT * FROM la_user")
    cekError(err)
    var mhs = Mhs{}
    var mhsArray []Mhs
    for statement.Next() {
        err = statement.Scan(&mhs.Nim, &mhs.Nama, &mhs.Email, &mhs.Hp)
        cekError(err)
        mhsArray = append(mhsArray, Mhs{Nim: mhs.Nim, Nama: mhs.Nama, Email:
mhs.Email, Hp: mhs.Hp})
    }
    for i := 0; i < len(mhsArray); i++ {
        fmt.Println(mhsArray[i])
    }
    return mhsArray
}
```

Dalam hal ini fungsi SelectAll() memiliki nilai return berupa data mahasiswa yang ada di dalam database. Kita juga dapat menghubungkannya dengan REST API, yang pertama dilakukan adalah pastikan susunan file dan folder adalah seperti ini (boleh berbeda):



Dalam hal ini, modul yang mengatur keseluruhan transaksi database pada table user/mahasiswa dijadikan satu didalam modul Mhs. Lalu buat file **main.go** untuk menghandle mekanisme multiplexer dan REST APInya.

Untuk menjalankan method select, kita dapat membuat sebuah fungsi:

```
func tampilMhs(w http.ResponseWriter, r *http.Request) {
    var mhs = mhs.SelectAll()
    str, err := json.Marshal(mhs)
    if err != nil {
        panic(err.Error())
    } else {
        w.Header().Set("Content-Type", "application/json")
        io.WriteString(w, string(str))
    }
}
```

Lalu, fungsi main() dapat kita isikan:

```
func main() {
    mux := http.NewServeMux()

    mux.HandleFunc("/tampilMhs", tampilMhs)

    http.ListenAndServe(":5050", mux)
}
```

SIMPAN data dengan mekanisme POST

Selain itu, dengna memanfaatkan mekanisme POST kita juga dapat menyimpan data pada database, yang pertama kita buat adalah fungsi untuk menghandle inputan POST dengan cara membuat fungsi simpanMhs():

```
func simpanMhs(w http.ResponseWriter, r *http.Request) {
    nama := r.FormValue("nama")
    nim := r.FormValue("nim")
    nimm, _ := strconv.ParseInt(nim, 10, 64)
    email := r.FormValue("email")
    hp := r.FormValue("no_hp")

    Mhs.Simpan(int(nimm), nama, email, hp)
}
```

Kita tangkap masing-masing variable mulai dari nama hingga no_hp. Lalu, kita parsing variable-variable tadi ke sebuah method Simpan() yang berada di file Mhs.go. Sedangkan di dalam file Mhs.go kita dapat menampungnya dengan sebuah fungsi Simpan():

```
func Simpan(nim int, nama, email, hp string) {
    db := konek()
    statement, err := db.Prepare("INSERT INTO la_user(nim, nama, email, no_hp)
VALUES (?,?,?,?)")
    cekError(err)
    respon, err := statement.Exec(nim, nama, email, hp)
    cekError(err)
    id, err := respon.LastInsertId()
    cekError(err)
    fmt.Print(id)
}
```

JWT Auth Sederhana

```
import (
    "fmt"
    "io"
    "net/http"
    "time"

    "github.com/dgrijalva/jwt-go"
    _ "github.com/dgrijalva/jwt-go"
    _ "github.com/golang-jwt/jwt"
)
```

```
var sampleSecretKey = []byte("SecretYouShouldHide")
var api_key = "1234"
```

```
func CreateJwt() (string, error) {
    token := jwt.New(jwt.SigningMethodHS256)
    claims := token.Claims.(jwt.MapClaims)
    claims["exp"] = time.Now().Add(time.Hour).Unix()
    tokenStr, err := token.SignedString(sampleSecretKey)
    cekError(err)
    return tokenStr, nil
}
```

```
func getJwt(w http.ResponseWriter, r *http.Request) {
    if r.Header["Access"] != nil {
        if r.Header["Access"][0] == api_key {
            token, err := CreateJwt()
            if err != nil {
                return
            }
            fmt.Fprintf(w, token)
        }
    }
}
```

```
func ValidateJWT(next func(w http.ResponseWriter, r *http.Request)) http.Handler
{
```

```
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {

        if r.Header["Token"] != nil {
            token, err := jwt.Parse(r.Header["Token"][0], func(t *jwt.Token)
(interface{}, error) {
                _, ok := t.Method.(*jwt.SigningMethodHMAC)
                if !ok {
                    w.WriteHeader(http.StatusUnauthorized)
                    w.Write([]byte("not authorized"))
                }
                return sampleSecretKey, nil
            })

            if err != nil {
                w.WriteHeader(http.StatusUnauthorized)
                w.Write([]byte("not authorized: " + err.Error()))
            }

            if token.Valid {
                next(w, r)
            }
        } else {
            w.WriteHeader(http.StatusUnauthorized)
            w.Write([]byte("not authorized"))
        }
    })
}
```

```
func tes(w http.ResponseWriter, r *http.Request) {
    w.Header().Set("Content-Type", "application/json")
    io.WriteString(w, string("hallo"))
}
```

```
func main() {

    http.Handle("/api", ValidateJWT(tes))
    http.HandleFunc("/jwt", getJwt)

    http.ListenAndServe(":3500", nil)

}
```