WEEK 1

GO WITH DATABASE SQL

IMPORTING DATABASE DRIVER

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

- gunakan go get untuk mendapatkan driver yang ingin anda pakai
- go get <nama-driver-package>

- Go menyediakan package database/sql yang bertujuan untuk berinteraksi dengan database SQL. Ada cukup banyak SQL Driver yang tersedia seperti :
  - MySQL
  - Oracle
  - Microsoft SQL Server
  - DLL

ACCESSING DATABASE

```
func main() {
    db, err := sql.Open("mysql",
            "user:password@tcp(127.0.0.1:3306)/hello")
    if err != nil {
        log.Fatal(err)
    }
    defer db.Close()
}
```

- sql.Open digunakan untuk memulai koneksi ke database
- menerima 2 parameter yaitu nama driver dan connection string
- db.Close() digunakan untuk menutup instance koneksi

- penjelasan connection string
  - user (nama user database) & password (kata sandi dari user database)
  - database hostname / ip address (127.0.0.1 atau localhost) & port (3306)
  - database name (hello)

RETRIEVING RESULT SETS

```
var (
    id int
    name string
)

rows, err := db.Query("select id, name from users where
                id = ?", 1)

if err != nil {
    log.Fatal(err)
}

defer rows.Close()

for rows.Next() {
    err := rows.Scan(&id, &name)
    if err != nil {
        log.Fatal(err)
    }
    log.Println(id, name)
}

err = rows.Err()

if err != nil {
    log.Fatal(err)
}
```

- db.Query() digunakan untuk mengirim query ke database
- melakukan iterasi ke rows menggunakan rows.Next()
- membaca kolom di setiap baris ke dalam variabel menggunakan rows.Scan()
- cek error setelah melakukan iterasi
- gunakan db.QueryRow() jika ingin menghasilkan 1 baris record saja dan chain dengan method Scan() untuk mendapatkan valuenya

PREPARING QUERIES

```
stmt, err := db.Prepare("select id, name from users
                where id = ?")

if err != nil {
    log.Fatal(err)
}

defer stmt.Close()

rows, err := stmt.Query(1)

if err != nil {
    log.Fatal(err)
}

defer rows.Close()

for rows.Next() {
    // ...
}

if err = rows.Err(); err != nil {
    log.Fatal(err)
}
```

- db.Prepare() digunakan untuk mengeksekusi query secara berulang kali
- teknik penulisannya di buat di awal agar dapat di reuse kembali
- dapat digunakan bersamaan dengan Query() atau QueryRow()
- mencegah SQL Injection

SINGLE-ROW QUERIES

```
var name string

err = db.QueryRow("select name from users where id
        = ?", 1).Scan(&name)
i
f err != nil {
    log.Fatal(err)
}
fmt.Println(name)
```

- db.QueryRow() digunakan untuk query yang menghasilkan 1 baris record saja.
- chain dengan Scan() untuk mendapatkan valuenya
- contoh code dengan menggunakan db.Prepare() dan QueryRow()

```
stmt, err := db.Prepare("select name from users where
                id = ?")
if err != nil {
    log.Fatal(err)
}

defer stmt.Close()

var name string

err = stmt.QueryRow(1).Scan(&name)

if err != nil {
    log.Fatal(err)
}

fmt.Println(name)
```

INSERT, UPDATE, DELETE WITH EXEC()

```
stmt, err := db.Prepare("INSERT INTO users(name)
                VALUES(?)")

if err != nil {
     log.Fatal(err)
}

res, err := stmt.Exec("Johny")

if err != nil {
     log.Fatal(err)
}
```

- db.Exec() digunakan untuk insert, update atau delete.
- direkomendasikan untuk esksekusi perintah tersebut menggunakan Exec()
- menggabungkan db.Prepare() dan Exec() untuk reusable statement query serta mencegah SQL Injection

```
_, err = db.Exec("insert into users VALUES
     (?)", "Dono")

if err != nil {
     log.Fatal(err)
}

fmt.Println("insert success!")
```

```
_, err := db.Exec("DELETE FROM users")  // OK
_, err := db.Query("DELETE FROM users") // BAD
```

WORKING WITH TRANSACTIONS

```
tx, err := db.Begin()

if err != nil {
    log.Fatal(err)
}

defer tx.Rollback()

stmt, err := tx.Prepare("INSERT INTO foo VALUES (?)")

if err != nil {
    log.Fatal(err)
}

defer stmt.Close() // danger!

for i := 0; i < 10; i++ {
    _, err = stmt.Exec(i)
    if err != nil {
        log.Fatal(err)
    }
}

err = tx.Commit()

if err != nil {
    log.Fatal(err)
}

// stmt.Close() runs here!
```

- `db.Begin()` digunakan untuk memulai melakukan transaksi
- `tx.Rollback()` digunakan untuk mengembalikan data dimana transaksi tersebut sebelum dijalankan
- `tx.Commit()` digunakan untuk membuat permanen data dari hasil transaksi yang telah dilakukan dan kita sudah tidak bisa mengembalikan data sesaat sebelm dilakukan transaksi