Hari 15 - Database Part 1

Membuat REST API dengan Database MySQL

Untuk membuat REST API dengan database MySQL, kita perlu mempersiapkan project kita terlebih dahulu, pada contoh ini kita akan memberi nama projectnya api-mysql. pada contoh ini kita akan menggunakan dua package diluar packages bawaan Go. packagenya adalah https://github.com/go-sql-driver/mysql dan https://github.com/julienschmidt/httprouter. untuk installasi package cukup menjalankan perintah:

```
go get github.com/go-sql-driver/mysql
go get github.com/julienschmidt/httprouter
```

setelah melakukan installasi diatas maka akan otomatis ditambahkan file go.sum dan perubahan di file go.mod akan terlihat seperti ini:

```
module api-mysql

go 1.17

require (
  github.com/go-sql-driver/mysql v1.6.0 // indirect
  github.com/julienschmidt/httprouter v1.3.0 // indirect
)
```

Struktur Project

Silahkan buat folder dan file dengan struktur folder sederhana seperti di bawah ini :

```
.
├── main.go
├── go.mod
├── go.sum
├── config
│   └── config.go
├── movie
│   └── repository_mysql.go
├── models
│   └── movie.go
└── utils
    └── res.go
```

Penjelasan :

- **main.go**, digunakan untuk menjalankan melakukan aksi terhadap data, bisa di katakan sebuah controller.
- **utils/res.go**, digunakan untuk mencetak data dengan format JSON.
- **config/config.go**, digunakan untuk melakukan konfigurasi MySQL.
- **movie/repository_mysql.go,** digunakan untuk melakukan query ke database.
- **models/movie.go**, digunakan untuk membuat struct/ struktur.

Sekarang kita mulai isi kode di file **main.go**.

Hal yang pertama kita lakukan yaitu membuat `http server` agar dapat di jalankan di browser.

```
package main

import(
  "log"
  "net/http"
  "github.com/julienschmidt/httprouter"
)

func main() {
  router := httprouter.New()
  fmt.Println("Server Running at Port 8080")
  log.Fatal(http.ListenAndServe(":8080", router))
}
```

Lalu kita isi kode file **utils/res.go**.

```
  "encoding/json"
  "net/http"
)

func ResponseJSON(w http.ResponseWriter, p interface{}, status int) {
  ubahkeByte, err := json.Marshal(p)
  w.Header().Set("Content-Type", "application/json")

  if err != nil {
    http.Error(w, "error om", http.StatusBadRequest)
  }

  w.Header().Set("Content-Type", "application/json")
  w.WriteHeader(status)
  w.Write([]byte(ubahkeByte))
}
```

Kode di atas berguna untuk menampilkan data dengan bentuk JSON di browser. Fungsi di atas nantinya akan kita panggil dari file **main.go**.

Sekarang kita buat modelnya, silahkan buka **models/movie.go**.

```
package models

import (
  "time"
)

type (
  // Movie
  Movie struct {
    ID        int       `json:"id"`
    Title     string    `json:"title"`
    Year      int       `json:"year"`
    CreatedAt time.Time `json:"created_at"`
    UpdatedAt time.Time `json:"updated_at"`
  }
)
```

Kode di atas merupakan struct, dimana nanti digunakan untuk melakukan segala bentuk manipulasi data, seperti menampilkan data di Golang dengan database MySQL.

Jika sudah sekarang kita akan melakukan koneksi ke database MySQL.

Koneksi Golang ke Database MySQL

Sebelumnya silahkan buat database dengan nama `db_movie`. Dan nama tabel `movie`.

Berikut ini query sql nya yang bisa anda excute.

```
CREATE TABLE db_movie.movie (
    id INT auto_increment primary key,
    title VARCHAR(255) not null,
    `year` INT not null,
    created_at DATETIME not null,
    updated_at DATETIME not null
)
```

untuk membuat tabel tidak harus eksekusi query SQL seperti diatas, boleh langsung dibuat di tools seperti **phpMyadmin** , **Dbeaver dan sejenisnya.**

Sehingga skema tabel `movie` seperti gambar di bawah ini.

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| id         | int          | NO   | PRI | NULL    | auto_increment |
| title      | varchar(255) | NO   |     | NULL    |                |
| year       | int          | NO   |     | NULL    |                |
| created_at | datetime     | NO   |     | NULL    |                |
| updated_at | datetime     | NO   |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

https://github.com/go-sql-driver/mysql yang sudah di install sebelumnya.

Untuk melakukan koneksi dengan database MySQL kita menggunakan file **config/config.go**.

```
package config

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

const (
    username string = "root"
    password string = "password"
    database string = "db_movie"
)

var (
    dsn = fmt.Sprintf("%v:%v@/%v", username, password, database)
)

// HubToMySQL
func MySQL() (*sql.DB, error) {
    db, err := sql.Open("mysql", dsn)

    if err != nil {
        return nil, err
    }

    return db, nil
}
```

Silahkan ganti pengaturan koneksi ke databasenya sesuai dengan `environment` milik anda yang terdapat pada kode :

```
const (
  username string = "root"
  password string = "password"
  database string = "db_movie"
)
```

Sekarang mari kita lihat hasilnya. Apakah sudah berhasil atau belum.

Silahkan panggil fungsi `MySQL()` dari file **main.go**.

```
package main

import (
    "log"
    "net/http"
    "api-mysql/config"
    "fmt"
    "github.com/julienschmidt/httprouter"
)

func main() {
    db, e := config.MySQL()

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

    eb := db.Ping()
    if eb != nil {
        panic(eb.Error())
    }

    fmt.Println("Success")

    router := httprouter.New()
    fmt.Println("Server Running at Port 8080")
    log.Fatal(http.ListenAndServe(":8080", router))
}
```

```
C:\api-mysql>go run main.go
Success
Server Running at Port 8080
```

jika muncul hasil print "Success" berarti koneksi database sudah berhasil

---

CRUD (Create, Read, Update, Delete)

Dalam sebuah aplikasi manipulasi data yang lumrah adalah user dapat menambahkan data (create), melihat data apa saja yang sudah terinput (read), membuat perubahan pada data yang sudah ada (update) dan menghapus data yang diinginkan (delete). Kali ini kita akan membuat keempat hal tersebut. kita akan uraikan satu-persatu.

Read

hal pertama yang paling terlihat adalah menampilkan keseluruhan data yang ada pada suatu table.

pertama kita akan membuat kode di **main.go** seperti dibawah ini:

```
package main
```

```
import (
  "api-mysql/movie"
  "api-mysql/utils"
  "context"
  "fmt"
  "log"
  "net/http"

  "github.com/julienschmidt/httprouter"
)

func main() {
  router := httprouter.New()
  router.GET("/movie", GetMovie)

  fmt.Println("Server Running at Port 8080")
  log.Fatal(http.ListenAndServe(":8080", router))

}

// Read
// GetMovie
func GetMovie(w http.ResponseWriter, _ *http.Request, _ httprouter.Params)
{
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  movies, err := movie.GetAll(ctx)

  if err != nil {
    fmt.Println(err)
  }

  utils.ResponseJSON(w, movies, http.StatusOK)
}
```

Perhatikan kode di atas, di dalam method getMovie() berisi kode untuk memanggil fungsi yang ada di dalam file movie/repository_mysql.go di tandai dengan kode movie.GetAll(ctx). Nama fungsi yang di panggil adalah GetAll().

Asumsikan kita sudah punya method tersebut.

Kode di atas terdapat kode yang berisi context.WithCancel(context.Background()) Hal ini gunakan untuk membatalkan semua proses ketika ada kesalahan.

Selanjutnya ketika sudah mendapat respon dari fungsi `GetAll() maka hasil nya akan di kirimkan ke fungsi ResponseJSON agar hasilnya berupa JSON.

```
package movie

import (
  "api-mysql/config"
  "api-mysql/models"
  "context"
  "fmt"
  "log"
  "time"
)

const (
  table          = "movie"
  layoutDateTime = "2006-01-02 15:04:05"
)

// GetAll
func GetAll(ctx context.Context) ([]models.Movie, error) {
  var movies []models.Movie
  db, err := config.MySQL()

  if err != nil {
    log.Fatal("Cant connect to MySQL", err)
  }

  queryText := fmt.Sprintf("SELECT * FROM %v Order By created_at DESC",
table)
  rowQuery, err := db.QueryContext(ctx, queryText)

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

  for rowQuery.Next() {
    var movie models.Movie
    var createdAt, updatedAt string
    if err = rowQuery.Scan(&movie.ID,
      &movie.Title,
      &movie.Year,
      &createdAt,
      &updatedAt); err != nil {
      return nil, err
    }

    //  Change format string to datetime for created_at and updated_at
    movie.CreatedAt, err = time.Parse(layoutDateTime, createdAt)

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

    movie.UpdatedAt, err = time.Parse(layoutDateTime, updatedAt)
    if err != nil {
      log.Fatal(err)
    }

    movies = append(movies, movie)
  }
  return movies, nil
}
```

Perhatikan kode di atas , Baris awal dari fungsi GetAll() yaitu var movies []models.movie yang mana nantinya akan digunakan untuk memberikan nilai dari perulangan.

Setelah itu terdapat query MySQL SELECT (terkait ini nanti akan di berikan di referensi)

Ketika sudah berhasil melakukan query ke MySQL akan di lakukan perulangan dengan fungsi for.

Lalu hasil dari perulangan di masukkan ke single struct yaitu var movie models.movie. Pada baris terakhir pada perilangan menggunakan fungsi append, dimana ketika ada data setiap perulangan di tambahkan ke struct []models.movie.

Pada kode dia atas juga terdapat fungsi yang digunakan untuk mengubah format date dari MySQL ke bentuk date yang ada di golang.

Create

lalu kita lanjutkan untuk create data.

kita akan membuat kode di **main.go** seperti dibawah ini:

```
import "encoding/json"

// di dalam func main
router.POST("/movie/create", PostMovie)
```

```
// Create
// PostMovie
func PostMovie(w http.ResponseWriter, r *http.Request, _ httprouter.Params) {
  if r.Header.Get("Content-Type") != "application/json" {

   http.Error(w, "Gunakan content type application / json", http.StatusBadRequest)
    return
  }
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var mov models.Movie
  if err := json.NewDecoder(r.Body).Decode(&mov); err != nil {
    utils.ResponseJSON(w, err, http.StatusBadRequest)
    return
  }
  if err := movie.Insert(ctx, mov); err != nil {
    utils.ResponseJSON(w, err, http.StatusInternalServerError)
    return
  }
  res := map[string]string{
    "status": "Succesfully",
  }
  utils.ResponseJSON(w, res, http.StatusCreated)

}
```

Pada kode di atas langkah awal kita harus memeriksa content type pada header, apabila data tersebut berupa json maka akan di lanjutkan.

Baris berikutnya yaitu melakukan decode dari data json di tandai dengan potongan kode json.NewDecoder(r.Body).Decode(&mov). Kita menggunakan variable mov agar data yang di insert sesuai dengan model struct Movie.

Apabila sudah isi dari variable movie akan di kirim pada fungsi Insert pada file movie/repository_mysql.go. Asumsikan kita sudah memiliki fungsi tersebut.

Sekarang kita buat fungsi baru di file **movie/repository_mysql.go**. Kita akan membuat fungsi dengan nama Insert().

Berikut ini kodenya :

```
// Insert Movie
func Insert(ctx context.Context, movie models.Movie) error {
  db, err := config.MySQL()
  if err != nil {
    log.Fatal("Can't connect to MySQL", err)
  }

  queryText := fmt.Sprintf("INSERT INTO %v (title, year, created_at,
updated_at) values('%v',%v, NOW(), NOW())", table,
    movie.Title,
    movie.Year)
  _, err = db.ExecContext(ctx, queryText)

  if err != nil {
    return err
  }
  return nil
}
```

Kode di atas cukup sederhana dan mudah di mengerti. Perhatikan kode di atas, terdapat variable dengan nama `queryText` dimana variable tersebut berisi query untuk melakukan insert ke database MySQL.

Lalu agar query teks tersebut dapat di ekstekusi maka kita dapat menggunakan kode `ExecContext` dengan parameter context dan query teksnya.

Update

lalu kita lanjutkan untuk update data.

kita akan membuat kode di **main.go** seperti dibawah ini:

```
// Update
// UpdateMovie
func UpdateMovie(w http.ResponseWriter, r *http.Request, ps
httprouter.Params) {
  if r.Header.Get("Content-Type") != "application/json" {
    http.Error(w, "Gunakan content type application / json",
http.StatusBadRequest)
    return
  }

  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
```

```
  var mov models.Movie

  if err := json.NewDecoder(r.Body).Decode(&mov); err != nil {
    utils.ResponseJSON(w, err, http.StatusBadRequest)
    return
  }

  var idMovie = ps.ByName("id")

  if err := movie.Update(ctx, mov, idMovie); err != nil {
    utils.ResponseJSON(w, err, http.StatusInternalServerError)
    return
  }

  res := map[string]string{
    "status": "Succesfully",
  }

  utils.ResponseJSON(w, res, http.StatusCreated)
}
```

kode diatas hampir mirip seperti kode di create tetapi perbedaannya ada params id yang diambil dan di gunakan nantinya untuk query update

Sekarang kita buat fungsi baru di file **movie/repository_mysql.go**. Kita akan membuat fungsi dengan nama Update().

```
// Update Movie
func Update(ctx context.Context, movie models.Movie, id string) error {
  db, err := config.MySQL()
  if err != nil {
    log.Fatal("Can't connect to MySQL", err)
  }


 queryText := fmt.Sprintf("UPDATE %v set title ='%s', year = %d, updated_at = NOW() where id = %s",
    table,
    movie.Title,
    movie.Year,
    id,
  )

  _, err = db.ExecContext(ctx, queryText)
  if err != nil {
    return err
  }

  return nil
}
```

kode untuk query nya pun hampir sama, perbedaannya dari yang sebelumnya menggunakan query SQL insert sekarang menggunakan query update.

Delete

lalu terakhir kita lanjutkan untuk delete data.

kita akan membuat kode di **main.go** seperti dibawah ini:

```
// Delete
// DeleteMovie
func DeleteMovie(w http.ResponseWriter, r *http.Request, ps
httprouter.Params) {
  ctx, cancel := context.WithCancel(context.Background())
  defer cancel()
  var idMovie = ps.ByName("id")
  if err := movie.Delete(ctx, idMovie); err != nil {
    kesalahan := map[string]string{
      "error": fmt.Sprintf("%v", err),
    }
    utils.ResponseJSON(w, kesalahan, http.StatusInternalServerError)
    return
  }
  res := map[string]string{
    "status": "Succesfully",
  }
  utils.ResponseJSON(w, res, http.StatusOK)
}
```

berbeda dengan lainnya, pada DELETE kita bisa tidak menggunakan struct Movie, karena nanti pada query sql nya yang di butuhkan hanya id

kita lanjut ke kode di file **movie/repository_mysql.go**. Kita akan membuat fungsi dengan nama Delete().

```
// Delete Movie
func Delete(ctx context.Context, id string) error {
    db, err := config.MySQL()
```

```
    if err != nil {
        log.Fatal("Can't connect to MySQL", err)
    }

    queryText := fmt.Sprintf("DELETE FROM %v where id = %s", table, id)

    s, err := db.ExecContext(ctx, queryText)

    if err != nil && err != sql.ErrNoRows {
        return err
    }

    check, err := s.RowsAffected()
    fmt.Println(check)
    if check == 0 {
        return errors.New("id tidak ada")
    }

    if err != nil {
        fmt.Println(err.Error())
    }

    return nil
}
```

terlihat pada query SQL "DELETE FROM" disana hanya membutuhkan id saja, sehingga kita dapat mengurangi parameter yang sebelumnya menggunakan object dari struct diganti menjadi id saja

---

Referensi Tulisan:

- https://kodingin.com/golang-koneksi-database-mysql/
- https://kodingin.com/golang-menampilkan-data-mysql/
- https://kodingin.com/golang-insert-data-mysql/
- https://kodingin.com/golang-rest-api-update-data-mysql/
- https://kodingin.com/golang-delete-data-api-mysql/

Rating - Feedback

Berikan Rating pada posting ini:

Berikan kritik dan saran..

Submit