

Go Programming Microservice Into The Specialization

Sesi 3 : SQL & API Implementation





Database

Database

Komponen utama penyusun suatu app yang terdiri dari sekumpulan data atau informasi yang tersimpan secara sistematis.

Ada banyak sekali jenis database, namun yang paling sering digunakan adalah Relational Database dan Non-Relational Database.

Relational Database

Pada relational database, data disimpan dalam sebuah skema yang ditampilkan seperti tabel (terdiri dari baris dan kolom).

Setiap data pada relational database diidentifikasi menggunakan key atau primary key.

Untuk memanipulasi data pada relational database digunakan sebuah bahasa yang disebut dengan Structured Query Language (SQL), oleh karena itu Relational Database juga disebut dengan SQL database.

Contoh Relational Database:

- [MySQL](#)
- [MariaDB](#)
- [PostgreSQL](#)
- [SQL Server](#)
- [Oracle](#)

Introduction of PostgreSQL

Sebuah *relational database manajemen system* (RDBMS) yang dikembangkan oleh tim relawan yang ada di seluruh dunia yang bersifat *open source*, artinya siapa saja bisa mengembangkannya. PostgreSQL tidak dikelola oleh perusahaan atau badan swasta lainnya, sehingga *source code* (kode program) yang tersedia bisa di dapatkan secara gratis.

Perbedaan yang paling mendasar antara postgres (sebutan untuk postgresSQL) dengan sistem relasional standar adalah, kemampuan postgres yang memungkinkan *user* untuk mendefinisikan SQL-nya sendiri, terutama untuk pembuatan *function*.

Fitur-fitur dari PostgreSQL

Point-in-time recovery

mengizinkan server terus-menerus diback-up sehingga seandainya sebuah disk drive gagal bekerja, database dapat dikembalikan di titik dimana kegagalan itu terjadi.

Savepoints

berguna bagi database developer yang membutuhkan penanganan error dalam transaksi yang kompleks, yaitu suatu fitur yang mengizinkan suatu bagian tertentu dari transaksi database untuk dibatalkan tanpa mempengaruhi sisa transaksi yang lain.

Introduction of PostgreSQL

Fitur-fitur dari PostgreSQL

Tablespaces

Berguna untuk memilih disk mana yang harus digunakan untuk menyimpan database, schema, table atau index. Sehingga kinerja PostgreSQL dalam menangani database raksasa berukuran ratusan gigabyte sampai puluhan terabyte dapat tetap terjaga.

Inheritance = Pewarisan

Mewariskan objek yang dimiliki ke pada objek yang diturunkan ,dan bersifat menyeluruh.

Kelas yang mewariskan biasa di sebut super class / class induk

Kelas yang diwariskan biasa di sebut sub class / kelas anak

Rule

Tindakan custom yang bisa kita definisikan dieksekusi saat sebuah tabel di-INSERT, UPDATE, atau DELETE.Selain itu sistem rule ini memungkinkan kita mengendalikan bagaimana data kita diubah atau diambil.

OO

Fitur OO seperti pewarisan tabel dan tipe data, atau tipe data array yang kadang praktis untuk menyimpan banyak item data di dalam satu record.Dengan adanya kemampuan OO ini maka di PostgreSQL, kita dapat mendefinisikan sebuah tabel yang mewarisi definisi tabel lain.

Installation

Selanjutnya kita butuh untuk mempersiapkan *database* postgresQL yang akan kita gunakan.

Yang dibutuhkan:

- Chrome atau firefox
- Postgre sql
 - Masuk ke website <https://www.postgresql.org>
 - Klik *download*
 - Pilih OS
 - Pilih Postgres.app
 - Klik *download*, dan *install* versi yang paling stabil
Postgres.app with PostgreSQL 14.6, or 15.1
 - Set *automation* di *preferences*
"Security" → "Privacy" → "Automation" → "Postgres"
 - Klik 2x salah satu *user*

Install masing-masing kebutuhan

Installation

Buka postgresQL 14/15

Kita bisa menjalankan *query* lewat *console* atau terminal, seperti masuk ke *database* tertentu, melihat semua daftar *database* yang ada, membuat *database* baru, dll.

Selain menggunakan *console* kita juga bisa mengelola postgresQL menggunakan *Grafical User Interface* (GUI). Salah satu GUI yang terkenal untuk postgresQL adalah pgAdmin. pgAdmin adalah *platform opensource* yang digunakan untuk manajemen, pengembangan, serta manipulasi *database* posgreSQL.

*instruksi:

- Kunjungi *website*-nya <https://www.pgadmin.org/>
- Klik download
- Pilih OS dan pilih **pgadmin terbaru**
- Setelah diinstal, *drag* ke desktop
- Klik *icon* 2x
- *Set password* (catat supaya tidak lupa)
- Buka *servers*
Biasanya setiap kita buka server kita akan ditanyakan password yang sudah kita buat sebelumnya.
- Buat database baru melalui pgAdmin



Go Sql

Go Sql

Pada materi kali ini, kita akan belajar bagaimana cara menggunakan package *database/sql* dengan mengintegrasikannya dengan sebuah sql driver. Karena sekarang kita akan menggunakan database *Postgresql*, maka dari itu kita perlu terlebih dahulu menginstall driver dari *Postgresql* dengan menjalankan perintah seperti pada gambar dibawah ini.

```
go get -u github.com/lib/pq
```

Creating Database

Setelah selesai menginstall driver dari *Postgresql*, maka kita sekarang perlu membuat database terlebih dahulu yang hanya akan mengandung satu table saja. Maka dari itu kita bisa menjalankan perintah dibawah ini pada GUI PGAdmin atau GUI lainnya maupun langsung melewati *psql* di terminal.

```
1 CREATE db-go-sql;  
2  
3 CREATE TABLE employees (  
4     id SERIAL PRIMARY KEY,  
5     full_name varchar(50) NOT NULL,  
6     email TEXT UNIQUE NOT NULL,  
7     age INT NOT NULL,  
8     division varchar(20) NOT NULL  
9 );
```

Connecting To Database

Sekarang kita akan mencoba untuk membangun koneksi ke database *Postgresql*. Maka dari itu ikutilah syntax-syntax seperti pada gambar disebelah kanan.

Bisa kita lihat pada line 7, kita memberikan tanda underscore _ saat kita mengimport diver dari *Postgresql* yang telah kita install sebelumnya. Hal ini dilakukan karena sebetulnya kita tidak akan menggunakan syntax apapun dari driver *Postgresql*. Yang kita perlu lakukan adalah hanya meregristrasikan nya dengan cara mengimportnya saja agar package *databases/sql* mengetahui tentang driver jenis apa yang kita pakai.

Line 10 - 16, kita membuat variable global berupa konstanta yang kita gunakan untuk menyimpan seluruh informasi dari *Postgresql* pada sistem kita. Perlu diingat bahwa perihal nilai *user* dan *password* dapat berbeda-beda antara penulis dan teman-teman sekalian. Tergantung dari konfigurasi *Postgresql* yang kita berikan pada saat pertama kali kita menginstallnya.

```
1 package main
2
3 import (
4     "database/sql"
5     "fmt"
6
7     _ "github.com/lib/pq"
8 )
9
10 const (
11     host      = "localhost"
12     port      = 5432
13     user       = "postgres"
14     password   = "postgres"
15     dbname    = "db-go-sql"
16 )
17
18 var (
19     db *sql.DB
20     err error
21 )
22
23 func main() {
24     psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
25         "password=%s dbname=%s sslmode=disable",
26         host, port, user, password, dbname)
27
28     db, err = sql.Open("postgres", psqInfo)
29     if err != nil {
30         panic(err)
31     }
32     defer db.Close()
33
34     err = db.Ping()
35     if err != nil {
36         panic(err)
37     }
38
39     fmt.Println("Successfully connected to database")
40 }
```

Connecting To Database

Pada line 18 - 21, kita membuat 2 variable yaitu *db* dan *err*. Variable *db* memiliki tipe data pointer dari struct *sql.DB* yang nantinya akan kita reassign dengan object dari *sql.DB* pada saat kita membangun koneksi pada function *main*. Variable ini kita buat menjadi variable global agar variable ini dapat kita gunakan secara global melalui berbagai *function*.

Line 24, kita menggabungkan seluruh info dari *Postgresql* yang telah kita buat pada line diatasnya menjadi sebuah string panjang yang kita simpan pada variable *psqlInfo*.

Line 28, kita menggunakan function *Open* yang berasal dari package *database/sql*. Function *Open* menerima 2 parameter yaitu nama dari driver yang kita pakai, dan sebuah string berupa informasi tentang bagaimana cara *database/sql* membangun koneksi nya kepada database kita sesuai dengan driver yang kita gunakan.

```
1 package main
2
3 import (
4     "database/sql"
5     "fmt"
6     _ "github.com/lib/pq"
7 )
8
9
10 const (
11     host     = "localhost"
12     port     = 5432
13     user     = "postgres"
14     password = "postgres"
15     dbname   = "db-go-sql"
16 )
17
18 var (
19     db *sql.DB
20     err error
21 )
22
23 func main() {
24     psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
25         "password=%s dbname=%s sslmode=disable",
26         host, port, user, password, dbname)
27
28     db, err = sql.Open("postgres", psqlInfo)
29     if err != nil {
30         panic(err)
31     }
32     defer db.Close()
33
34     err = db.Ping()
35     if err != nil {
36         panic(err)
37     }
38
39     fmt.Println("Successfully connected to database")
40 }
```

Connecting To Database

Pada line 34, kita memanggil method *Ping*. Kita dapat menggunakan method *Ping* karena setelah penggunaan function *Open* berhasil tereksekusi, maka function *Open* akan mengembalikan nilai berupa pointer dari struct *sql.DB* yang dimana struct tersebut memiliki method *Ping*.

Pemanggilan method *Ping* merupakan hal yang sangat penting karena function *Open* tidak akan membangun koneksi ke database, melainkan hanya berfungsi untuk memvalidasi argumen-argumen yang diberikan. Dengan memanggil method *Ping*, maka kita dapat membangun koneksi ke database sekaligus memeriksa jika string panjang berupa info yang kita berikan pada function *Open* merupakan info yang 100% valid.

```
1 package main
2
3 import (
4     "database/sql"
5     "fmt"
6     _ "github.com/lib/pq"
7 )
8
9
10 const (
11     host     = "localhost"
12     port     = 5432
13     user     = "postgres"
14     password = "postgres"
15     dbname   = "db-go-sql"
16 )
17
18 var (
19     db *sql.DB
20     err error
21 )
22
23 func main() {
24     psqlInfo := fmt.Sprintf("host=%s port=%d user=%s "+
25         "password=%s dbname=%s sslmode=disable",
26         host, port, user, password, dbname)
27
28     db, err = sql.Open("postgres", psqInfo)
29     if err != nil {
30         panic(err)
31     }
32     defer db.Close()
33
34     err = db.Ping()
35     if err != nil {
36         panic(err)
37     }
38
39     fmt.Println("Successfully connected to database")
40 }
```

Data Struct

Sekarang kita akan mencoba untuk membuat data employee baru melalui package *database/sql*. Pertama-tama, kita akan membuat terlebih dahulu sebuah struct Employee seperti pada gambar di bawah.

```
type Employee struct {  
    ID          int  
    Full_name   string  
    Email       string  
    Age         int  
    Division    string  
}
```

Create Data

Kemudian buatlah sebuah function bernama *CreateEmployee* seperti pada gambar di sebelah kanan. Pada line 54 - 56, kita membuat sebuah sql statement yang merupakan sebuah statement untuk melakukan create data pada database. Lalu tanda **\$1, \$2 dst..** merupakan representasi dari nilai-nilai yang akan kita masukkan nantinya. Lalu terdapat penulisan **Returning *** yang memiliki arti bahwa kita ingin mendapat nilai-nilai dari seluruh field yang berasal dari data yang baru dibuat.

```
50 func CreateEmployee() {
51     var employee = Employee{}
52
53     sqlStatement := `
54     INSERT INTO employees (full_name, email, age, division)
55     VALUES ($1, $2, $3, $4)
56     Returning *
57     `
58
59     err = db.QueryRow(sqlStatement, "Airell Jordan", "airell@mail.com", 23, "IT").
60         Scan(&employee.ID, &employee.Full_name, &employee.Email, &employee.Age, &employee.Division)
61
62     if err != nil {
63         panic(err)
64     }
65
66     fmt.Printf("New Employee Data : %+v\n", employee)
67
68 }
```

Create Data

Line 59 - 60, kita menggunakan method *QueryRow*. Method ini digunakan untuk mengeksekusi sebuah query sql tergantung dari statement sql yang kita berikan. Karena statement sql yang kita berikan bertujuan untuk create data, maka method *QueryRow* ini akan berfungsi untuk membuat data baru dengan nilai yang kita berikan pada parameter kedua dari method *QueryRow*. Parameter kedua dari *QueryRow* merupakan sebuah parameter *variadic* dan nilai-nilai yang kita berikan adalah nilai-nilai yang akan me-replace statement yang kita buat pada line 55 **VALUES(\$1, \$2, \$3, \$4)**.

```
50 func CreateEmployee() {
51     var employee = Employee{}
52
53     sqlStatement := `
54     INSERT INTO employees (full_name, email, age, division)
55     VALUES ($1, $2, $3, $4)
56     Returning *
57     `
58
59     err = db.QueryRow(sqlStatement, "Airell Jordan", "airell@mail.com", 23, "IT").
60         Scan(&employee.ID, &employee.Full_name, &employee.Email, &employee.Age, &employee.Division)
61
62     if err != nil {
63         panic(err)
64     }
65
66     fmt.Printf("New Employee Data : %+v\n", employee)
67
68 }
```


Create Data

Kemudian method `QueryRow` kita chaning dengan method `Scan` agar kita dapat mendapatkan nilai-nilai balikan dari statement yang telah kita buat. Karena statement kita menggunakan **Returning ***, maka kita akan mendapatkan seluruh seluruh nilai dari field-field yang berasal dari data yang baru dibuat. Dan nilai-nilai balikan tersebut akan kita simpan kedalam field-field dari variable `employee` pada line 51.

Sekarang kita akan memanggil function `CreateEmployee` didalam function `main` seperti pada gambar kedua.

```
47     fmt.Println("Successfully connected to database")
48
49     CreateEmployee()
50 }
```

Create Data

Mari kita jalankan aplikasi kita dengan perintah **go run main.go** dan kita akan melihat hasilnya seperti pada gambar di bawah ini.

```
Successfully connected to database  
New Employee Data : {ID:1 Full_name:Airell Jordan Email:airell@mail.com Age:23 Division:IT}
```

Dan kita juga dapat melihat data yang baru kita buat pada database kita seperti pada gambar dibawah ini.

id	full_name	email	age	division
1	Airell Jordan	airell@mail.com	23	IT

Get Data

Sekarang kita akan mencoba untuk mendapatkan data-data employee. Maka dari itu buatlah sebuah function dengan nama *GetEmployees* seperti pada gambar di sebelah kanan.

Pada line 77, kita menggunakan method *Query* yang dimana method *Query* biasa digunakan untuk mendapatkan banyak data dari suatu table pada database dikarenakan method ini dapat mengembalikan satu atau lebih rows dari suatu table pada database. Dan kita juga harus menutup rows tersebut dengan method *Close* yang kita panggil dengan keyword *defer*.

Line 84, kita melakukan perulangan/looping sebanyak data yang kita dapatkan dengan acuan *rows.Next*. Method *rows.Next* akan menghasilkan nilai true selama data nya masih ada, namun jika sudah tidak ada maka dia akan me-return false dan proses looping akan terhenti

```
72 func GetEmployees() {  
73     var results = []Employee{}  
74  
75     sqlStatement := `SELECT * from employees`  
76  
77     rows, err := db.Query(sqlStatement)  
78  
79     if err != nil {  
80         panic(err)  
81     }  
82     defer rows.Close()  
83  
84     for rows.Next() {  
85         var employee = Employee{}  
86  
87         err = rows.Scan(&employee.ID, &employee.Full_name, &employee.Email, &employee.Age, &employee.Division)  
88  
89         if err != nil {  
90             panic(err)  
91         }  
92  
93         results = append(results, employee)  
94     }  
95  
96  
97     fmt.Println("Employee datas:", results)  
98 }
```

Get Data

Pada line 87, kita melakukan proses scanning seperti yang telah kita lakukan sebelumnya.

Pada line 93, setelah proses scanning selesai untuk satu row, maka kita akan memasukkan data dari row tersebut kedalam variable *results* yang telah kita siapkan pada line 73.

```
72 func GetEmployees() {
73     var results = []Employee{}
74
75     sqlStatement := `SELECT * from employees`
76
77     rows, err := db.Query(sqlStatement)
78
79     if err != nil {
80         panic(err)
81     }
82     defer rows.Close()
83
84     for rows.Next() {
85         var employee = Employee{}
86
87         err = rows.Scan(&employee.ID, &employee.Full_name, &employee.Email, &employee.Age, &employee.Division)
88
89         if err != nil {
90             panic(err)
91         }
92
93         results = append(results, employee)
94     }
95
96     fmt.Println("Employee datas:", results)
97 }
98 }
```

Get Data

Sekarang mari kita panggil function *GetEmployees* pada function *main* dan jangan lupa untuk meng-comment pemanggilan function *CreateEmployee* seperti pada gambar pertama di sebelah kanan.

Jika kita jalankan kembali aplikasi kita, maka kita akan melihat hasilnya seperti pada gambar kedua.

```
// CreateEmployee()  
GetEmployees()  
}
```

```
Successfully connected to database  
Employee datas: [{1 Airell Jordan airell@mail.com 23 IT}]
```

Update Data

```
101 func UpdateEmployee() {
102     sqlStatement := `
103     UPDATE employees
104     SET full_name = $2, email = $3, division = $4, age = $5
105     WHERE id = $1;`
106     res, err := db.Exec(sqlStatement, 1, "Airell Jordan Hidayat", "airellhidayat@gmail.com", "CurDevs", 24)
107     if err != nil {
108         panic(err)
109     }
110     count, err := res.RowsAffected()
111     if err != nil {
112         panic(err)
113     }
114     fmt.Println("Updated data amount:", count)
115 }
```

Sekarang kita akan mencoba untuk mengupdate data yang sudah kita buat. Mari kita buat suatu function dengan nama *UpdateEmployee* seperti pada gambar diatas. Bisa kita lihat pada line 106, sekarang kita menggunakan method *Exec* untuk mengupdate data. Sebetulnya kita juga bisa mengupdate sebuah data dengan menggunakan method *QueryRow*, namun dianjurkan untuk menggunakan method *Exec* untuk melakukan proses insert, update, dan delete. Dengan menggunakan method *Exec*, kita tidak dapat mendapatkan data yang baru saja diupdate maupun data yang baru kita buat, namun kita dapat menggunakan method *RowsAffected* untuk mengetahui berapa jumlah row atau data yang baru diupdate. Method *RowsAffected* didapatkan dari interface *sql.Result* yang merupakan nilai kembalian dari method *Exec*.

Update Data

Sekarang mari kita panggil function *UpdateEmployee* dan jangan lupa untuk meng-comment pemanggilan function *CreateEmployee* dan *GetEmployees* seperti pada gambar pertama di sebelah kanan.

Jika kita jalankan pada aplikasi kita, maka hasilnya akan seperti pada gambar kedua.

```
// CreateEmployee()  
// GetEmployees()  
UpdateEmployee()  
}
```

```
Successfully connected to database  
Updated data amount: 1
```

Delete Data

Untuk yang terakhir, kita akan menghapus data employee. Maka dari itu, buat lah satu function bernama *DeleteEmployee* seperti pada gambar di sebelah kanan. Setelah itu mari kita panggil function *DeleteEmployee* pada function *main* dan jangan lupa untuk meng-comment 3 function yang sudah kita buat sebelumnya seperti pada gambar kedua.

Jika kita jalankan aplikasi kita, maka hasilnya akan seperti pada gambar ketiga.

```
117 func DeleteEmployee() {
118     sqlStatement := `
119     DELETE from employees
120     WHERE id = $1;
121     `
122     res, err := db.Exec(sqlStatement, 1)
123     if err != nil {
124         panic(err)
125     }
126     count, err := res.RowsAffected()
127     if err != nil {
128         panic(err)
129     }
130     fmt.Println("Deleted data amount:", count)
131 }
```

```
// CreateEmployee()
// GetEmployees()
// UpdateEmployee()
DeleteEmployee()
}
```

```
└─ go run main.go
Successfully connected to database
Deleted data amount: 1
```