



Gorm



Gorm

+



Gorm

Introduction



Gorm adalah ORM yang cukup populer untuk bahasa Go, yang dimana Gorm telah menyediakan berbagai macam fitur seperti auto migration, eager loading, association, query method, dan lain-lain.

Dengan menggunakan Gorm, maka akan dapat mempercepat pengembangan aplikasi kita karena fitur-fitur yang telah disediakan oleh Gorm.

Untuk menginstall Gorm, maka kita perlu menjalankan perintah: `go get -u gorm.io/gorm`.

```
go get -u gorm.io/gorm
```

Gorm

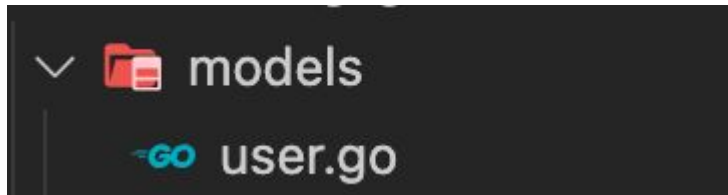
Declaring Models

Pada saat kita ingin membuat table dengan menggunakan Gorm, maka kita perlu membuat models nya terlebih dahulu.

Model tersebut dapat kita buat dengan menggunakan struct.

Misalkan contohnya kita ingin membuat table User, maka kita dapat membuat struct dengan nama User.

Disini penulis membuat folder models, lalu juga membuat file bernama user.go.



Gorm

Declaring Models

Gambar dibawah ini merupakan struct User yang dibuat pada file user.go. Untuk membuat structnya, maka seluruh property maupun nama dari structnya perlu diawali dengan huruf besar.

```
1  package models
2
3  import "time"
4
5  type User struct {
6      ID          uint    `gorm:"primaryKey"`
7      Email       string  `gorm:"not null;unique;type:varchar(191)"`
8      Products   []Product
9      CreatedAt  time.Time
10     UpdatedAt  time.Time
11 }
```



Gorm

Declaring Models(tags)

```
1 package models
2
3 import "time"
4
5 type User struct {
6     ID      uint   `gorm:"primaryKey"`
7     Email    string `gorm:"not null;unique;type:varchar(191)"`
8     Products []Product
9     CreatedAt time.Time
10    UpdatedAt time.Time
11 }
```

Gorm juga telah menyediakan tags yang dapat kita gunakan, seperti halnya property ID diberikan tag `gorm:"primaryKey"`. Ini digunakan untuk menjadikan kolom id tersebut menjadi primary key dari table User.

Kemudian kita juga dapat memberikan constraint kepada suatu kolom, seperti halnya pada property email. Terdapat tag `not null`, dan `unique`. Hal ini dilakukan agar kolom email dari table User menjadi unique column dan tidak boleh null alias `not null`.

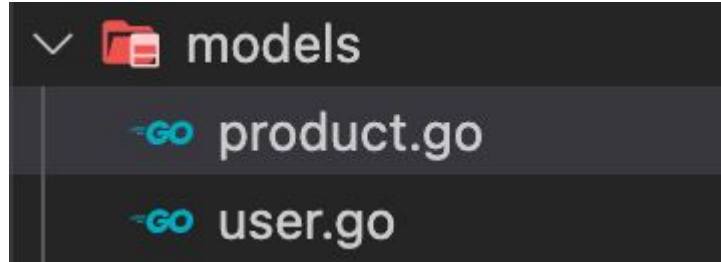
Property Email dari struct memiliki tipe data `string`, dan secara otomatis tipe datanya akan menjadi `TEXT` pada database. Namun kita dapat mengganti tipe datanya seperti contohnya `varchar`, maka kita dapat melakukannya dengan memberikan tag `type:<tipe_data>`

Gorm Association

Gorm telah menyediakan berbagai macam asosiasi, seperti one to one, one to many, dan many to many.

Misalkan contohnya selain kita ingin membuat table User, kita juga ingin membuat table Products. Table Products ini akan memiliki foreign key dari ID User, sehinggalan assosiasinya akan menjadi one to many(Satu User bisa memiliki banyak Product, dan satu Product hanya dapat memiliki satu Product).

Disini penulis telah menambahkan satu file bernama product.go pada folder models.



Gorm Association

```
1 package models
2
3 import (
4     "time"
5 )
6
7 type Product struct {
8     ID          uint   `gorm:"primaryKey"`
9     Name        string `gorm:"not null;type:varchar(191)"`
10    Brand        string `gorm:"not null;type:varchar(191)"`
11    UserID       uint
12    CreatedAt    time.Time
13    UpdatedAt    time.Time
14 }
15
```

Struct Product diatas dibuat pada file product.go. Perhatikan bahwa struct Product memiliki property UserID dengan tipe data uint. By default, property UserID tersebut akan secara otomatis menjadi foreign key karena, ketika suatu struct memiliki property yang dimana nama propertynya merupakan gabungan dari nama struct lain dengan primary key dari struct tersebut, maka Gorm secara otomatis akan menjadikan property tersebut menjadi foreign key dari struct lainnya.

Nama struct

Primary Key

```
type User struct {
    ID          string `gorm:"primaryKey"`
    ...
}
```


Gorm Association

Kemudian kita perlu menambahkan property Products pada struct User.

```
1 package models
2
3 import "time"
4
5 type User struct {
6     ID      uint    `gorm:"primaryKey"`
7     Email   string  `gorm:"not null;unique;type:varchar(191)"`
8     Products []Product
9     CreatedAt time.Time
10    UpdatedAt time.Time
11 }
```



Gorm Association

Namun ketika kita ingin agar foreign key yang dikandung oleh Product bukan dari id User, maka kita dapat membuat tag foreign key, seperti yang telah dijelaskan oleh dokumentasi dari Gorm. Silahkan kunjungi dokumentasi dari Gorm untuk penjelasan lengkapnya: https://gorm.io/docs/has_many.html.

Override Foreign Key

To define a `has many` relationship, a foreign key must exist. The default foreign key's name is the owner's type name plus the name of its primary key field

For example, to define a model that belongs to `User`, the foreign key should be `UserID`.

To use another field as foreign key, you can customize it with a `foreignKey` tag, e.g:

```
type User struct {
    gorm.Model
    CreditCards []CreditCard `gorm:"foreignKey:UserRefer"`
}

type CreditCard struct {
    gorm.Model
    Number      string
    UserRefer   uint
}
```



Connecting To Database And Table Migration

Sekarang kita membahas cara membangun koneksi pada database, dan cara memigrasi struct-struct yang telah kita buat menjadi table-table pada database.

Disini penulisan telah membuat database lokal dengan nama learning-gorm. Jenis databasenya adalah Postgresql.



Connecting To Database And Table Migration

Karena disini penulis menggunakan database Postgresql, maka dari itu penulis telah menginstall driver Postgresql.

Cara menginstallnya adalah dengan menjalankan perintah pada terminal: `go get gorm.io/driver/postgres`.

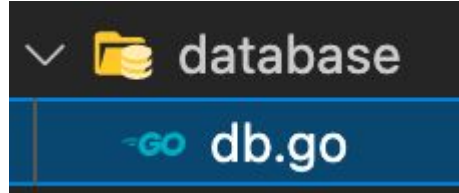
```
go get gorm.io/driver/postgres
```



Gorm

Connecting To Database And Table Migration

Disini penulis membuat folder database, dengan file db.go di dalamnya.



Connecting To Database And Table Migration

Pada baris 23, penulis menampung config untuk menghubungkan kepada database.

Kemudian pada baris 25, method Open digunakan untuk membangun koneksi kepada database. Ketika koneksi telah berhasil terbangun, maka variable db pada baris 25 akan mengandung referensi dari database dengan tipe data *gorm.DB.

Bisa diperhatikan bahwa variable db merupakan variable global yang pertama kali dibuat pada baris 18.

Method Debug pada baris 30 digunakan sebagai debugging atau logger. Kemudian di chaining dengan method AutoMigrate.

Method AutoMigrate digunakan untuk memigrasi secara otomatis dari struct-struct yang telah dibuat.

```
1 package database
2
3 import (
4     "fmt"
5     "learning-gorm/models"
6     "log"
7
8     "gorm.io/driver/postgres"
9     "gorm.io/gorm"
10 )
11
12 var (
13     host    = "localhost"
14     user    = "sofyan"
15     password = "postgres"
16     dbPort  = "5432"
17     dbname  = "learning-gorm"
18     db      *gorm.DB
19     err      error
20 )
21
22 func StartDB() {
23     config := fmt.Sprintf("host=%s user=%s password=%s dbname=%s port=%s sslmode=disable", host, user,
24         password, dbname, dbPort)
25
26     db, err = gorm.Open(postgres.Open(config), &gorm.Config{})
27     if err != nil {
28         log.Fatalf("error connecting to database :", err)
29     }
30
31     db.Debug().AutoMigrate(models.User{}, models.Product{})
32 }
```



Connecting To Database And Table Migration

Sekarang penulis akan memanggil function StartDB pada func main dari file main.go

```
1  package main
2
3  import "learning-gorm/database"
4
5  func main() {
6      database.StartDB()
7  }
```



Connecting To Database And Table Migration

Penulis akan menjalankan perintah go run main.go.

```
go run main.go
```

Maka pada terminal akan tampil logging dari method Debug.

```
go run main.go
2021/12/03 17:19:23 /Users/sofyan/go/pkg/mod/gorm.io/driver/postgres@v1.2.3/migrator.go:199 SLOW SQL >= 200
[297.686ms] [rows:1] SELECT count(*) FROM information_schema.tables WHERE table_schema = CURRENT_SCHEMA() A
2021/12/03 17:19:23 /Users/sofyan/go/pkg/mod/gorm.io/driver/postgres@v1.2.3/migrator.go:172 SLOW SQL >= 200
[290.623ms] [rows:0] CREATE TABLE "users" ("id" text,"email" varchar(191) NOT NULL UNIQUE,"created_at" time
2021/12/03 17:19:23 /Users/sofyan/go/pkg/mod/gorm.io/driver/postgres@v1.2.3/migrator.go:199
[2.173ms] [rows:1] SELECT count(*) FROM information_schema.tables WHERE table_schema = CURRENT_SCHEMA() AND
2021/12/03 17:19:23 /Users/sofyan/go/pkg/mod/gorm.io/driver/postgres@v1.2.3/migrator.go:172
[60.718ms] [rows:0] CREATE TABLE "products" ("id" text,"name" varchar(191) NOT NULL,"brand" varchar(191) NO
PRIMARY KEY ("id"),CONSTRAINT "fk_users_products" FOREIGN KEY ("user_id") REFERENCES "users"("id"))
```



Connecting To Database And Table Migration

Dan sekarang database learning-gorm sudah akan memiliki 2 table yaitu users dan products. Nama dari table nya akan otomatis menjadi plural dan nama dari tablenya akan memiliki huruf lowercase.

Struct User akan menjadi table users, dan struct Product akan menjadi products.

Seluruh nama-nama kolom pun akan menjadi lowercase.



Connecting To Database And Table Migration

Kolom-kolom dari table users.

COLUMN NAME	TYPE	DEFAULT	CONSTRAINTS
id	text	no default ▾	PRIMARY KEY +
email	character varying(191)	no default ▾	UNIQUE NOT NULL +
created_at	timestamp with time zone	no default ▾	+
updated_at	timestamp with time zone	no default ▾	+



Connecting To Database And Table Migration

Kolom-kolom dari table products.

products			
COLUMN NAME	TYPE	DEFAULT	CONSTRAINTS
id	text	no default ▾	PRIMARY KEY +
name	character varying(191)	no default ▾	NOT NULL +
brand	character varying(191)	no default ▾	NOT NULL +
user_id	character varying(64)	no default ▾	→ users.id +
created_at	timestamp with time zone	no default ▾	+
updated_at	timestamp with time zone	no default ▾	+



Create User Data

Untuk melakukan CRUD data melalui Gorm, maka kita perlu mendapatkan referensi database yang telah ditampung pada suatu variable.

Jika kita ingat kembali, referensi database tersebut telah ditampung pada variable global bernama db pada file db.go dalam folder database. Maka dari itu penulis disini menambahkan function GetDB pada file db.go yang akan mereturn variable db tersebut.

```
36 func GetDB() *gorm.DB {  
37     return db  
38 }
```



Gorm

Create User Data

File db.go

```
1 package database
2
3 import (
4     "fmt"
5     "learning-gorm/models"
6     "log"
7
8     "gorm.io/driver/postgres"
9     "gorm.io/gorm"
10 )
11
12 var (
13     host      = "localhost"
14     user      = "sofyan"
15     password  = "postgres"
16     dbPort    = "5432"
17     dbname    = "learning-gorm"
18     db        *gorm.DB
19     err       error
20 )
21
22 func StartDB() {
23     config := fmt.Sprintf("host=%s user=%s password=%s dbname=%s port=%s sslmode=disable", host, user,
24         password, dbname, dbPort)
25
26     db, err = gorm.Open(postgres.Open(config), &gorm.Config{})
27     if err != nil {
28         log.Fatal("error connecting to database :", err)
29     }
30
31     db.Debug().AutoMigrate(models.User{}, models.Product{})
32 }
33
34 func GetDB() *gorm.DB {
35     return db
36 }
```



HACKTIV8

Create User Data

Untuk melakukan CRUD data melalui Gorm, maka kita perlu mendapatkan referensi database yang telah ditampung pada suatu variable.

Jika kita ingat kembali, referensi database tersebut telah ditampung pada variable global bernama db pada file db.go dalam folder database. Maka dari itu penulis disini menambahkan function GetDB pada file db.go yang akan mereturn variable db tersebut.

```
36 func GetDB() *gorm.DB {  
37     return db  
38 }
```



Gorm

Create User Data

Kemudian pada file main.go, penulis akan membuat function bernama `createUser` yang digunakan untuk membuat data User baru.

Untuk melakukan create data, maka kita dapat menggunakan method `Create`. Kemudian method `Create` ini memerlukan data sebagai argumentnya yang dimana data tersebut perlu memiliki tipe data yang sama dengan yang ingin kita buat.

Misalkan seperti pada gambar disebelah kanan, penulis berhendak untuk membuat data User baru, alhasil data yang diberikan pada method `Create` tersebut adalah data dengan tipe data struct `User` dari folder `models` yang telah penulis buat sebelumnya dan juga sudah kita bahas.

Method untuk CRUD pada Gorm dengan property `Error` agar kita bisa langsung dapat memeriksa errornya jika memang ada.

```
9 func main() {
10     database.StartDB()
11
12     createUser("johndoe@gmail.com")
13 }
14
15 func createUser(email string) {
16     db := database.GetDB()
17
18     User := models.User{
19         Email: email,
20     }
21
22     err := db.Create(&User).Error
23
24     if err != nil {
25         fmt.Println("Error creating user data:", err)
26         return
27     }
28
29     fmt.Println("New User Data:", User)
30 }
```



Create User Data

Sekarang penulis akan menjalankan perintah `go run main.go` untuk mengeksekusi function `createUser` yang telah dibuat.

Maka data user baru sudah bertambah pada database.

```
9 func main() {
10     database.StartDB()
11
12     createUser("johndoe@gmail.com")
13 }
14
15 func createUser(email string) {
16     db := database.GetDB()
17
18     User := models.User{
19         Email: email,
20     }
21
22     err := db.Create(&User).Error
23
24     if err != nil {
25         fmt.Println("Error creating user data:", err)
26         return
27     }
28
29     fmt.Println("New User Data:", User)
30 }
```

id	email	created_at	updated_at
1	jodoe@gmail.com	2021-12-05 21:34:44.946556+07	2021-12-05 21:34:44.946556+07



Get One User Data

Untuk mendapatkan suatu data dari suatu table, maka kita dapat menggunakan method First.

Method First dapat menerima 3 parameter, parameter pertamanya adalah pointer terhadap data yang ingin dicari. Karena sekarang penulis berhendak untuk mencari data User, maka penulis memberikan tipe data struct User sebagai argumen pertama dari method First.

Kemudian parameter keduanya adalah condition dari query nya, dan yang terakhir adalah data dari conditionnya.

Method First akan mengembalikan error berupa ErrRecordNotFound jika tidak ada data yang ditemukan.

```
36 func getUserById(id uint) {
37     db := database.GetDB()
38
39     user := models.User{}
40
41     err := db.First(&user, "id = ?", id).Error
42
43     if err != nil {
44         if errors.Is(err, gorm.ErrRecordNotFound) {
45             fmt.Println("User data not found")
46             return
47         }
48         print("Error finding user:", err)
49     }
50
51     fmt.Printf("User Data: %v \n", user)
52 }
```



Get One User Data

Sekarang function `getUserById` akan dijalankan, maka hasilnya akan terlihat pada terminal.

```
12 func main() {  
13     database.StartDB()  
14  
15     // createUser("jodoe@gmail  
16     getUserById(1)  
17 }
```

```
go run main.go  
User Data: {ID:1 Email:jodoe@gmail.com Products:[] CreatedAt:2021-12-05 21:34:44.946556 +0700 WIB UpdatedAt:2021-12-05 21:34:44.946556 +0700 WIB}
```



Gorm

Update User Data

Untuk mengupdate data menggunakan Gorm, maka kita perlu menggunakan method **Model** terlebih dahulu agar hasil dari update nya dapat langsung discan sehingga kita dapat langsung mengetahui hasilnya.

Kemudian jika kita ingin membuat condition nya, maka kita dapat menggunakan method **Where** sehingga method **Model** dapat dichaining dengan method **Where**.

Dan yang terakhir kita bisa langsung menentukan data apa yang ingin di update.

Selain menggunakan method **Model**, kita juga dapat menggunakan method **Table** seperti yang telah dijelaskan pada dokumentasi dari Gorm

```
55 func updateUserById(id uint, email string) {
56     db := database.GetDB()
57
58     user := models.User{}
59
60     err := db.Model(&user).Where("id = ?", id).Updates(models.User{Email: email}).Error
61
62     if err != nil {
63         fmt.Println("Error updating user data:", err)
64         return
65     }
66     fmt.Printf("Update user's email: %v \n", user.Email)
67 }
```

```
db.Table("users").Where("id = ?", 1).Updates(map[string]interface{}{"email": "johnjohn@gmail.com"})
// UPDATE users SET name='johnjohn@gmail.com' WHERE id = 1;
```



Update User Data

Sekarang function `updateUserById` akan dijalankan, maka data user dengan id 1 akan diupdate kolom emailnya.

```
12 func main() {  
13     database.StartDB()  
14  
15     // createUser("jodoe@gmail.com")  
16     // getUserById(1)  
17     updateUserById(1, "johnjohn@gmail.com")  
18 }
```

id	email	created_at	updated_at
1	johnjohn@gmail.com	2021-12-05 21:34:44.946556+07	2021-12-05 22:04:13.838401+07



Gorm Hooks

Gorm telah menyediakan hooks yang dapat kita gunakan, salah satunya adalah method `BeforeCreate`. Method `BeforeCreate` akan tereksekusi sebelum melakukan create data. Disini penulis akan mengimplementasikan `BeforeCreate` untuk `Product`, sehingga nantinya ketika penulis ingin membuat data product maka method `BeforeCreate` tersebut akan terkesekusi.

```
11 type Product struct {
12     ID      uint   `gorm:"primaryKey"`
13     Name     string `gorm:"not null;type:varchar(191)"`
14     Brand    string `gorm:"not null;type:varchar(191)"`
15     UserID   uint
16     CreatedAt time.Time
17     UpdatedAt time.Time
18 }
19
20 func (p *Product) BeforeCreate(tx *gorm.DB) (err error) {
21
22     fmt.Println("Product Before Create()")
23
24     if len(p.Name) < 4 {
25         err = errors.New("product name is too short")
26     }
27
28     return
29 }
```



Gorm Hooks

Kemudian penulis membuat function `createProduct` untuk membuat data product baru.

Perhatikan pada gambar kedua, penulis disini akan menguji validasi yang telah penulis buat pada hook `BeforeCreate`.

```
69 func createProduct(userID uint, brand string, name string) {
70     db := database.GetDB()
71
72     Product := models.Product{
73         UserID: userID,
74         Brand:   brand,
75         Name:    name,
76     }
77
78     err := db.Create(&Product).Error
79
80     if err != nil {
81         fmt.Println("Error creating product data:", err.Error())
82         return
83     }
84
85     fmt.Println("New Product Data:", Product)
86 }
```

```
12 func main() {
13     database.StartDB()
14
15     // createUser("jodoe@gmail.com")
16     // getUserById(1)
17     // updateUserById(1, "johnjohn@gmail.com")
18     createProduct(1, "YLO", "YYY")
19 }
```



Gorm Hooks

Ketika function `createProduct` dijalankan, maka akan menghasilkan error karena panjang karakter dari property `Name` dari struct `Product` kurang dari 4.

```
└─ go run main.go
Product Before Create()
Error creating product data: product name is too short
```

Agar tidak terjadi error, maka jumlah karakter `Name` dari `Product` harus terdiri dari 4 karakter atau lebih.

```
createProduct(1, "YLO", "YYYY")
```



Gorm Hooks

Sekarang ketika function `createProduct` dijalankan, maka error tidak akan terjadi dan data product baru sudah akan terlihat pada database.

id	name	brand	user_id	created_at	updated_at
1	YYYY	YLO	1	2021-12-05 22:58:56.372542+07	2021-12-05 22:58:56.372542+07



Gorm

Eager Loading

Sekarang data user dengan id 1 sudah memiliki data product, maka kita dapat melakukan join statement. Untuk melakukan join statement, kita dapat menggunakan eager loading dari Gorm.

Caranya adalah dengan menggunakan method Preload dan kita perlu memberikan nama table untuk parameter method Preload. Walaupun nama tablenya adalah products jika kita melihat pada database, tapi tetap huruf awal dari nama table untuk parameter Preload harus menggunakan uppercase.

```
90 func getUsersWithProducts() {
91     db := database.GetDB()
92
93     users := models.User{}
94     err := db.Preload("Products").Find(&users).Error
95
96     if err != nil {
97         fmt.Println("Error getting user datas with products:", err.Error())
98         return
99     }
100
101     fmt.Println("User Datas With Products")
102     fmt.Printf("%+v", users)
103 }
```



Gorm Eager Loading

Sekarang penulis akan menjalankan function `getUsersWithProducts`, maka hasilnya pada terminal akan seperti pada gambar kedua.

```
12 func main() {  
13     database.StartDB()  
14  
15     // createUser("jodoe@gmail.com")  
16     // getUserById(1)  
17     // updateUserById(1, "johnjohn@gmail.com")  
18     // createProduct(1, "YLO", "YYYY")  
19     getUsersWithProducts()  
20 }
```

```
User Datas With Products  
{ID:1 Email:johnjohn@gmail.com Products:[{ID:1 Name:YYYY Brand:YLO UserID:1 CreatedAt:2021-12-05 22:58:56.372542 +0700 WIB UpdatedAt:2021-12-05 22:58:56.372542 +0700 WIB}] CreatedAt:2021-  
12-05 21:34:44.946556 +0700 WIB UpdatedAt:2021-12-05 22:04:13.838401 +0700 WIB}%
```



Gorm

Delete Product

Dan yang terakhir penulis akan menghapus data product. Untuk menghapus data product, maka kita bisa menggunakan method Delete.

```
105 func deleteProductById(id uint) {
106     db := database.GetDB()
107
108     product := models.Product{}
109     err := db.Where("id = ?", id).Delete(&product).Error
110
111     if err != nil {
112         fmt.Println("Error deleting product:", err.Error())
113         return
114     }
115
116     fmt.Printf("Product with id %d has been successfully deleted", id)
117 }
```



Gorm Delete Product

Sekarang penulis akan menjalankan function `deleteProductById`.

Maka data product dengan id 1 di database akan terhapus.

```
12 func main() {
13     database.StartDB()
14
15     // createUser("jodoe@gmail.com")
16     // getUserById(1)
17     // updateUserById(1, "johnjohn@gmail.com")
18     // createProduct(1, "YLO", "YYYY")
19     // getUsersWithProducts()
20     deleteProductById(1)
21 }
```

learning-gorm products Connected. PostgreSQL 13.4 LOCAL					
id	name	brand	user_id	created_at	updated_at