

ORM-

Fields/FieldsEx

1. Fields
2. FieldsEx

Fields

1. user4uid, nickname, passport, password
- 2.

```
// SELECT `uid`,`nickname` FROM `user` ORDER BY `uid` asc
g.Model("user").Fields("uid, nickname").Order("uid asc").All()
```

- 3.

```
m := g.Map{
    "uid"      : 10000,
    "nickname" : "John Guo",
    "passport"  : "john",
    "password"  : "123456",
}
g.Model(table).Fields("nickname,passport,password").Data(m).Insert()
// INSERT INTO `user`(`nickname`,`passport`,`password`) VALUES
('John Guo','john','123456')
```

FieldsEx

1. user4uid, nickname, passport, password
- 2.

```
// SELECT `uid`,`nickname` FROM `user`
g.Model("user").FieldsEx("passport, password").All()
```

- 3.

```
m := g.Map{
    "uid"      : 10000,
    "nickname" : "John Guo",
    "passport"  : "john",
    "password"  : "123456",
}
g.Model(table).FieldsEx("uid").Data(m).Insert()
// INSERT INTO `user`(`nickname`,`passport`,`password`) VALUES
('John Guo','john','123456')
```

OmitEmpty

```
map/struct nil,"",0 gdbOmitEmpty
```

```
func (m *Model) OmitEmpty() *Model
func (m *Model) OmitEmptyWhere() *Model
func (m *Model) OmitEmptyData() *Model
```

OmitEmptyWhereDataOmitEmptyWhere/OmitEmptyData

Content Menu

- [Fields/FieldsEx](#)
 - [Fields](#)
 - [FieldsEx](#)
- [OmitEmpty](#)
 - [/](#)
 -
- [OmitNil](#)
 - [do](#)
- [Filter\(\)](#)

/

/Insert, Replace, Update, Savemapstruct

```
// UPDATE `user` SET `name`='john',update_time=null WHERE `id`=1
g.Model("user").Data(g.Map{
    "name"      : "john",
    "update_time" : nil,
}).Where("id", 1).Update()
```

OmitEmpty

```
// UPDATE `user` SET `name`='john' WHERE `id`=1
g.Model("user").OmitEmpty().Data(g.Map{
    "name"      : "john",
    "update_time" : nil,
}).Where("id", 1).Update()
```

struct

```
type User struct {
    Id          int    `orm:"id"`
    Passport    string  `orm:"passport"`
    Password    string  `orm:"password"`
    NickName    string  `orm:"nickname"`
    CreateTime  string  `orm:"create_time"`
    UpdateTime  string  `orm:"update_time"`
}
user := User{
    Id          : 1,
    NickName    : "john",
    UpdateTime  : gtime.Now().String(),
}
g.Model("user").OmitEmpty().Data(user).Insert()
// INSERT INTO `user`(`id`,`nickname`,`update_time`) VALUES(1,'john','2019-10-01 12:00:00')
```



/OmitEmpty

omitemptyOmitEmpty

1. structomitemptyjsonORMstruct
2. omitemptyOmitEmptyORMstructomitemptyOmitEmptystructOmitEmptystruct

whereOmitEmpty

```
// SELECT * FROM `user` WHERE `passport`='john' LIMIT 1
r, err := g.Model("user").Where(g.Map{
    "nickname" : "",
    "passport"  : "john",
}).OmitEmpty().One()
```

```

type User struct {
    Id          int      `orm:"id"`
    Passport    string   `orm:"passport"`
    Password    string   `orm:"password"`
    NickName    string   `orm:"nickname"`
    CreateTime  string   `orm:"create_time"`
    UpdateTime  string   `orm:"update_time"`
}
user := User{
    Passport : "john",
}
r, err := g.Model("user").OmitEmpty().Where(user).One()
// SELECT * FROM `user` WHERE `passport`='john' LIMIT 1

```

OmitNil

```
map/struct nilgdbOmitNilOmitEmptyOmitNilnil""0
```

```

func (m *Model) OmitNil() *Model
func (m *Model) OmitNilWhere() *Model
func (m *Model) OmitNilData() *Model

```

OmitEmptyWhereDataOmitEmptyWhere/OmitEmptyData

do

GoFramegf gen daomake daodao/entity/dodo

do

```

// User is the golang structure of table user for DAO operations like Where
/Data.
type User struct {
    g.Meta    `orm:"table:user, do:true"`
    Id        interface{} // User ID
    Passport  interface{} // User Passport
    Password  interface{} // User Password
    Nickname  interface{} // User Nickname
    CreateAt  *gtime.Time // Created Time
    UpdateAt  *gtime.Time // Updated Time
}

```

```

dao.User.Transaction(ctx, func(ctx context.Context, tx gdb.TX) error {
    _, err = dao.User.Ctx(ctx).Data(do.User{
        Passport: in.Passport,
        Password: in.Password,
        Nickname: in.Nickname,
    }).Insert()
    return err
})

```

```
var user *entity.User
err = dao.User.Ctx(ctx).Where(do.User{
    Passport: in.Passport,
    Password: in.Password,
}).Scan(&user)
```

Filter()

```
gdb()Filter/map/struct/[]map/[]string
user4uid, nickname, passport, password
```

```
r, err := g.Model("user").Filter().Data(g.Map{
    "id"      : 1,
    "uid"     : 1,
    "passport" : "john",
    "password" : "123456",
}).Insert()
// INSERT INTO user(uid,passport,password) VALUES(1, "john", "123456")
```

~~idSQL~~



~~DataFilter/~~



GoFrame v1.15.7filterFilter