

GORM

Gorm 连接数据库实例

什么是 ORM

Golang 数据库连接以及 CURD 测试

Gorm 模型定义

`gorm.Model`

模型定义的默认值

Gorm 增删改查

增加：

查询

更新

保存所有字段

更新单个列

更新多个列

更新选定字段

更新 Hook

批量更新

更新的记录数

删除

删除一条记录

根据主键删除

批量删除

软删除

Gorm 连接数据库实例

什么是 ORM

- O: object 对象

- R : relationship 关系
- M: map 映射

即对象和关系型数据库之间的映射

优点：提升开发效率

缺点：1. 牺牲性能

2.不够灵活

3.弱化 sql 能力

Golang 数据库连接以及 CURD 测试

```
1 package test
2
3 import (
4     "gorm.io/driver/mysql"
5     "gorm.io/gorm"
6     "log"
7     "os"
8     "testing"
9 )
10
11 type UserInfo struct { // 数据库的表名为 user_infos
12     ID      int
13     Name    string
14     Hobby   string
15 }
16
17 const (
18     dbDriver = "mysql"
19     dbSource = "root:123456@tcp(127.0.0.1:3306)/blog?charset=utf8mb4&parse
Time=True&loc=Local"
20 )
21
22 var DB *gorm.DB
23
24 // TestMain:Golang 约定 TestMain 函数是所有单元测试的入口
25 func TestMain(m *testing.M) {
26     db, err := gorm.Open(mysql.Open(dbSource), &gorm.Config{})
27     if err != nil {
28         log.Fatal("can not connect db:", err)
29     }
30     DB = db
31     os.Exit(m.Run())
32 }
33
34 // Go中的每个单元测试函数都必须以 Test 开头, 并且以 testing.T 作为输入参数。
35
36 // TestAddUser 新增用户
37 func TestAddUser(t *testing.T) {
38     u1 := UserInfo{2, "max", "run"}
39     DB.Create(u1)
40 }
41
42 // TestUpdateUser 更新用户
43 func TestUpdateUser(t *testing.T) {
44     u1 := UserInfo{2, "max2", "run"}
```

```

45     DB.Update(u1)
46 }
47
48 // TestDeleteUser 删除用户
49 func TestDeleteUser(t *testing.T) {
50     DB.Delete(UserInfo{2, "max2", "run"})
51 }
52
53 // TestGetUser 获取用户
54 func TestGetUser(t *testing.T) {
55     DB.Find(UserInfo{2, "max", "run"})
56 }
57

```

Note:

TestMain:Golang 约定 TestMain 函数是所有单元测试的入口

Go中的每个单元测试函数都必须以 Test 开头，并且以 testing.T 作为输入参数

Gorm 模型定义

在使用 ORM 工具时，通常我们需要在代码中定义 Model 与数据库中的数据表进行映射，在 GORM 中模型通常是正常定义的结构体、基本的 Go 类型或它们的指针（如果结构体比较大时，传递指针会比较节约性能）。

gorm.Model

为了方便模型定义，GORM 内置了一个 `gorm.Model` 结构体。`gorm.Model` 是一个包含了 `ID`、`CreatedAt`、`UpdatedAt`、`DeletedAt` 四个字段的结构体

```
1 // gorm.Model
2 type Model struct {
3     ID      uint `gorm:"primarykey"`
4     CreatedAt time.Time
5     UpdatedAt time.Time
6     DeletedAt DeletedAt `gorm:"index"`
7 }
8 // 嵌入其他结构体
9 // type User struct {
10 //     gorm.Model
11 // }
```

这个模型中已经定义了主键，你可以选择使用或不使用，使用时可以直接嵌入其他结构体中。如果不使用这个模型，需要自己通过添加结构体 tag: `gorm:"primarykey"` 来指定主键。

模型定义的默认值

主键默认值： GORM 会默认使用结构体中的 `ID` 作为主键，除非你特意声明了其他字段

表名默认值：

1. 表名默认为结构体名称的复数。
2. 也可以通过 `tableName()` 自定义表名

```
1 type UserInfo struct { // 数据库的表名为 user_infos
2     ID int
3     Name string
4     Hobby string
5 }
6 // 将表名 userinfos 改为 users
7 (UserInfo)TableName()string {
8     return "users"
9 }
```

3. 还可以通过 `db.Table()` 指定结构体的表名：

```
1 type User struct{
2     ID int
3     Name string
4     Hobby string
5 }
6 // 使用 User 结构体创建一个 userTable 表
7 db.Table("userTable").CreateTable(&User{})
```

4. 可以通过 `DefaultTableNameHandler` 修改表名创建时的默认规则。

```
1 gorm.DefaultTableNameHandler = func (db *gorm.DB, defaultTableName string) string {
2     return "SMS_" + defaultTableName; // 注意，这个方法只能作用与你使用 GORM 默认的表名时才会生效
3 }
```

列名默认值：

1. 如果你的字段名是由两个单词组成的，GORM 会自动使用下划线分隔。

比如： `CreatedAt ==> created_at`

2. 也可以使用 tag 来指定列名

```
1 Age `gorm:"column:user_age" default:18`
```

Gorm 增删改查

在执行操作前加入 `debug()` 函数可以打印 sql 语句。

增加：

创建记录：

```

1 u1 := table.UserInfo{2, "max", "run"}
2 DB.Create(u1)

```

数据库字段插入时可以指定一个默认值比如：

```

1 type User struct {
2     Name string `gorm:"default:'max'"`
3 }

```

如果没有指定 `Name` 的值，也就是该字段的值为空。GORM 会忽略该字段，但是如果设定了默认值，就会将空值改为默认值。

如果一定要将某个字段的值置为空，有两种方法：

可以使用指针类型的数据来实现：

```

1 type User struct {
2     Name *string `gorm:"default:'max'"`
3 }
4 // 在创建时：
5 u := User{Name:new(string),age:19}

```

还可以使用 `Scanner/Valuer`：

```

1 type User struct {
2     Name sql.NullString `gorm:"default:'max'"` //这个结构体类型实现了 Scanner/
    Valuer 接口
3 }
4 // 创建
5 u := User{Name:sql.NullString{String:"",Valid:true},age:18}
6

```

```

// ...
type NullString struct {
    // ...
}

```

类型 NullString 实现了 2 个接口

- Scanner (in database/sql/sql.go) ... not NULL
- Valuer (in database/sql/driver/types.go)

查询

`db.First()` 根据主键排序，查询第一条数据，这里必须传入指针类型的结构体

`db.Last()` 与上面查询相反

`db.Find()` 需要传入结构体切片的指针，返回的是所有记录

```
1 func TestGetUser(t *testing.T) {
2
3     var u table.UserInfo
4     // 根据主键查询第一个记录
5     DB.First(&u) // First 和 Last 会根据主键排序，分别查询第一条和最后一条数据。只
    有在目标 struct 是指针或者通过 db.Model() 指定 model 时，该方法才有效。
6     // 这里只能传指针，我的理解是查询到的结果需要传入原结构体而不是副本。
7     fmt.Println(u)
8     // DB.Find() 也是需要传入结构体切片的指针，返回的是所有记录
9     //DB.Find(table.UserInfo{2, "max", "run"})
10
11
12     // works because model is specified using `db.Model()`
13     result := map[string]interface{}{}
14     db.Model(&User{}).First(&result)
15
16     // doesn't work
17     result := map[string]interface{}{}
18     db.Table("users").First(&result)
19 }
```

关于 first 为什么必须传入指针，官方文档的解释：

The `First` and `Last` methods will find the first and last record (respectively) as ordered by primary key. **They only work when a pointer to the destination struct is passed to the methods as argument or when the model is specified using `db.Model()`.** Additionally, if no primary key is defined for relevant model, then the model will be ordered by the first field.

根据 struct 和 map 查询：


```
1 // Struct
2 db.Where(&User{Name: "jinzhu", Age: 20}).First(&user)
3 // SELECT * FROM users WHERE name = "jinzhu" AND age = 20 ORDER BY id LIMIT 1;
4
5 // Map
6 db.Where(map[string]interface{}{"name": "jinzhu", "age": 20}).Find(&users)
7 // SELECT * FROM users WHERE name = "jinzhu" AND age = 20;
8
9 // Slice of primary keys
10 db.Where([]int64{20, 21, 22}).Find(&users)
11 // SELECT * FROM users WHERE id IN (20, 21, 22);
```

注意：当通过结构体查询时，GORM 只会通过非零字段进行查询，如果你的某个字段值为零，那么将不会作为构建查询的条件。

```
1 // 未找到就插入
2 db.FirstOrInit(&user, User{Name: "non_existing"})
3 // user -> User{Name: "non_existing"}
4
```

Limit 指定从数据库中检索出的最大记录数

```
1 db.Limit(3).Find(&users)
2 // SELECT * FROM users LIMIT 3;
```

Offset 指定开始返回记录前要跳过的记录数

```
1 db.Offset(3).Find(&users)
2 // SELECT * FROM users OFFSET 3;
```

Count 该 model 能获取的记录总数

```
1 db.Where("name = ?", "max").Or("name = ?", "maxs").Find(&users).Count()&count
```

注意: `Count` 必须是链式查询的最后一个操作, 因为他会覆盖前面的 `SELECT`, 但是如果里面使用了 `count` 时不会覆盖。

更新

保存所有字段

`db.Save()` 默认更新表里的所有字段, 即使字段是零值

```
1 // 对主键为 3 的字段进行 update
2 user.ID = 3
3 user.Name = "xiaoming"
4 user.Hobby = "game"
5 db.Save(&user)
6 // 未指定主键, 使用 create
7 user.Name = "xiaoming"
8 user.Hobby = "game"
9 db.Save(&user)
```

`Save()` 是一个组合型函数, 如果保存的值里不包含主键, 它就会执行 `Create()` 函数, 否则就会在所有字段上执行 `Update()` 函数。

NOTE: 不要在使用 `Save()` 时使用 `Model()`, 这是未定义的行为。

更新单个列

`db.Update()` 更新单个字段。当使用 `Update()` 更新单个列时, 必须指定条件, 否则会出现 `ErrMissingWhereClause` 错误, 当使用了 `Model` 方法, 且该对象主键有值, 该值会被用于构建条件。

```

1 // 使用 update 更新指定字段
2 db.Model(&table.UserInfo{}).Where("name = ?", "max").Update("name", "hello")
3
4 // 当指定 记录的主键时，主键会自动用来成为更新的条件
5 user.ID = 1
6 db.Model(&user).Update("name", "max")
7 // 表中 name=max 的字段都会被更新为 name=hello

```

更新多个列

`db.Updates()` 更新多个字段

```

1 // 使用 map or struct 更新多个字段，只能更新非零值字段
2 //db.Model(&user).Where("id = ?", 1).Updates(map[string]interface{}{"name": "max", "hobby": "ride"})
3 db.Model(&user).Where("id = ?", 1).Updates(table.UserInfo{
4     ID:      1,
5     Name:    "max2",
6     Hobby:   "code",
7 })

```

注意：

1. 使用 map or struct 更新多个字段，
2. struct 只能更新非零值字段，如果要确保更新某个字段，可以使用 map 或者 `SELECT` 指定

更新选定字段

指定字段更新： `Select()`

忽略字段： `Omit()`

```

1 // 指定字段更新:
2 user.ID = 1
3 //这里只选择了 name 字段, 所以哪怕其他字段不同也只更新 name
4 db.Model(&user).Select("name").Updates(map[string]interface{}{"name": "max
  1", "hobby": "cook"})
5 // 这里忽略了 name, 所以会更新其他字段
6 db.Model(&user).Omit("name").Updates(map[string]interface{}{"name": "test"
  , "hobby": "cook"})
7
8 // 更新零值字段: (注意提前指定 user 的主键)
9 db.Model(&user).Select("name").Updates(table.UserInfo{
10     ID:    1,
11     Name:  "",
12     Hobby: "code",
13 })

```

更新 Hook

对于更新操作, GORM 支持 BeforeSave、BeforeUpdate、AfterSave、AfterUpdate 钩子, 这些方法将在更新记录时被调用。

```

1 func (u *UserInfo) BeforeUpdate(tx *gorm.DB) (err error) {
2     if u.ID == 1 {
3         return errors.New("ID = 1 的字段不能被修改")
4     }
5     return
6 }

```

批量更新

如果不指定结构体的主键的值, 那么 GORM 会进行批量更新:

```
1 // 更新所有 name = xiaoming 的字段
2 db.Model(&table.UserInfo{}).Where("name = ?", "xiaoming").Updates(table.UserInfo{
3     Name: "xiaohua",
4     Hobby: "IT",
5 })
```

更新的记录数

```
1 // 通过 `RowsAffected` 得到更新的记录数
2 result := db.Model(User{}).Where("ID = ?", "1").Updates(User{Name: "hello",
3     hobby: "code"})
4 // UPDATE users SET name='hello', age=18 WHERE role = 'admin;
5 result.RowsAffected // 更新的记录数
6 result.Error         // 更新的错误
```

删除

删除一条记录

删除之前必须指定字段，否则会触发批量删除

```
1 // 删除一条记录
2 user.ID = 4
3 db.Delete(&user)
4 // 条件删除
5 db.Where("name = ?", "xiaohua").Delete(&user) // 删除了 id = 4, name = xiaohua 的字段
```

根据主键删除

GORM 允许通过内联条件指定主键来检索对象，但只支持整型数值，因为 string 可能导致 SQL 注入。

```
1 db.Delete(&User{}, 10)
2 // DELETE FROM users WHERE id = 10;
3
4 db.Delete(&User{}, "10")
5 // DELETE FROM users WHERE id = 10;
6
7 db.Delete(&users, []int{1,2,3})
8 // DELETE FROM users WHERE id IN (1,2,3);
9
```

批量删除

如果删除时不指定主键，GORM 会删除所有匹配的字段

```
1 db.Where("email LIKE ?", "%jinzhu%").Delete(&Email{})
2 // DELETE from emails where email LIKE "%jinzhu%";
3
4 db.Delete(&Email{}, "email LIKE ?", "%jinzhu%")
5 // DELETE from emails where email LIKE "%jinzhu%";
```

软删除

如果您的模型包含了一个 `gorm.deletedat` 字段（`gorm.Model` 已经包含了该字段），它就会获得软删除的特性。

拥有软删除能力的模型调用 `Delete` 时，记录不会被从数据库中真正删除。但 GORM 会将 `DeletedAt` 置为当前时间，并且你不能再通过正常的查询方法找到该记录。

可以使用 `unscoped()` 找到该记录并删除

```
1 // 找到被软删除的记录
2 db.Unscoped().Where("age = 20").Find(&users)
3 // SELECT * FROM users WHERE age = 20;
4 //永久删除该记录:
5 db.Unscoped().Delete(&order)
6 // DELETE FROM orders WHERE id=10;
```