安装和配置

GORM 是一个流行的 Go 语言 ORM（对象关系映射）库，它简化了与数据库交互的过程。GORM 支持多种数据库，包括 MySQL、PostgreSQL、SQLite 和 SQL Server。下面简述一下 GORM 的常见使用方法。

1. 安装 GORM

首先，需要安装 GORM 库，可以通过 `go get` 安装：

```
go get -u github.com/jinzhu/gorm
```

此外，还需要安装特定数据库的驱动，例如 MySQL：

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

2. 配置数据库连接

首先导入 GORM 和数据库驱动库，并配置数据库连接：

```
//"user:pass@tcp(127.0.0.1:3306)/dbname?charset=utf8mb4&parseTime=True&loc=Local"
dsn := "root:password@tcp(172.20.217.248:3306)/gorm?
charset=utf8&parseTime=True&loc=Local"
db, err := gorm.Open(mysql.Open(dsn), &gorm.Config{})
if err != nil {
    panic(err)
} else {
    log.Println("connect success", db)
}
```

假定我们有数据库名为gorm，users表结构如下：

```
CREATE TABLE users (
    id INT UNSIGNED AUTO_INCREMENT PRIMARY KEY,         -- ID字段，自增主键
    name VARCHAR(255) NOT NULL,                         -- Name字段，非空
    email VARCHAR(255) DEFAULT NULL,                    -- Email字段，可以为空
    age TINYINT UNSIGNED NOT NULL,                      -- Age字段，非负整数
    birthday DATETIME DEFAULT NULL,                     -- Birthday字段，可以为空
    member_number VARCHAR(255) DEFAULT NULL,            -- MemberNumber字段，可以为空
    activated_at DATETIME DEFAULT NULL,                 -- ActivatedAt字段，可以为空
    created_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP, -- CreatedAt字段，自动生成
当前时间
```

```
    updated_at DATETIME NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP, -- UpdatedAt字段，自动更新时间
    ignored VARCHAR(255) -- Ignored字段，未导出字段不参与表结构
);

type User struct {
    ID           uint           // Standard field for the primary key
    Name         string         // A regular string field
    Email        *string        // A pointer to a string, allowing for null values
    Age          uint8          // An unsigned 8-bit integer
    Birthday     *time.Time     // A pointer to time.Time, can be null
    MemberNumber sql.NullString // Uses sql.NullString to handle nullable strings
    ActivatedAt  sql.NullTime   // Uses sql.NullTime for nullable time fields
    CreatedAt    time.Time      // Automatically managed by GORM for creation time
    UpdatedAt    time.Time      // Automatically managed by GORM for update time
    ignored      string         // fields that aren't exported are ignored
}
```

Gorm规范

1. 约定

1. **主键**：GORM 使用一个名为 `ID` 的字段作为每个模型的默认主键。
2. **表名**：默认情况下，GORM 将结构体名称转换为 `snake_case` 并为表名加上复数形式。 例如，一个 `User` 结构体在数据库中的表名变为 `users` 。
3. **列名**：GORM 自动将结构体字段名称转换为 `snake_case` 作为数据库中的列名。
4. **时间戳字段**：GORM使用字段 `CreatedAt` 和 `UpdatedAt` 来自动跟踪记录的创建和更新时间。

遵循这些约定可以大大减少您需要编写的配置或代码量。 但是，GORM也具有灵活性，允许您根据自己的需求自定义这些设置。 您可以在GORM的约定文档中了解更多关于自定义这些约定的信息。

2. `gorm.Model`

GORM提供了一个预定义的结构体，名为 `gorm.Model` ，其中包含常用字段：

```
// gorm.Model 的定义
type Model struct {
  ID        uint           `gorm:"primaryKey"`
  CreatedAt time.Time
  UpdatedAt time.Time
  DeletedAt gorm.DeletedAt `gorm:"index"`
}
```

- **将其嵌入在您的结构体中**: 您可以直接在您的结构体中嵌入 `gorm.Model` ，以便自动包含这些字段。 这对于在不同模型之间保持一致性并利用GORM内置的约定非常有用，请参考嵌入结构。
- **包含的字段**:
  - `ID` ：每个记录的唯一标识符（主键）。
  - `CreatedAt` ：在创建记录时自动设置为当前时间。
  - `UpdatedAt` ：每当记录更新时，自动更新为当前时间。
  - `DeletedAt` ：用于软删除（将记录标记为已删除，而实际上并未从数据库中删除）。

创建记录

1. 创建单条记录

```
user := User{Name: "Jinzhu", Age: 18}
result := db.Create(&user) // 通过数据的指针来创建
log.Println("result.Error:", result.Error)
log.Println("result.RowsAffected:", result.RowsAffected)
```

`db.Create()` 返回一个 `gorm.DB` 结构体。

`gorm.DB` :

```
type DB struct {
    *Config
    Error        error
    RowsAffected int64
    Statement    *Statement
    clone        int
}

user.ID             // 返回插入数据的主键
result.Error        // 返回 error
result.RowsAffected // 返回插入记录的条数
```

2. 创建多条记录

注意⚠️⚠️⚠️⚠️：你无法向 'create' 传递结构体，你应该传入数据的指针。

```
users := []*User{
    {Name: "Jinzhu", Age: 18, Birthday: time.Now()},
    {Name: "Jackson", Age: 19, Birthday: time.Now()},
}
```

```
result := db.Create(users)

//这样写也是可以的，注意要传递指针即可。
users := []User{
    {Name: "Jinzhu", Age: 18, Birthday: time.Now()},
    {Name: "Jackson", Age: 19, Birthday: time.Now()},
}
result := db.Create(&users)
log.Println("result.Error:", result.Error)
log.Println("result.RowsAffected:", result.RowsAffected)
```

3. 选择创建记录的字段

```
user := User{
    Name:      "test",
    Email:     "hellobyteqq.com",
    Age:       18,
    Birthday:  time.Now(),
    CreatedAt: time.Now(),
    UpdatedAt: time.Now(),
}
//这样在表中插入的记录只会有 Name,Age,CreatedAt三个字段，其他字段为空。
result := db.Select("Name", "Age", "CreatedAt").Create(&user)
log.Println("result.Error:", result.Error)
log.Println("result.RowsAffected:", result.RowsAffected)

//忽略Name,Age,CreatedAt字段的值，如果没有默认值，数据库会报错
//比如:
/*
2024/12/04 15:06:58 /root/gorm/main.go:57 Error 1364 (HY000):
Field 'age' doesn't have a default value

[0.650ms] [rows:0] INSERT INTO `users`
(`name`,`email`,`birthday`,`member_number`,`activated_at`,`updated_at`) VALUES
('test','hellobyteqq.com','2024-12-04 15:06:58.504',NULL,NULL,'2024-12-04
15:06:58.504')
*/

result := db.Omit("Name", "Age", "CreatedAt").Create(&user)
log.Println("result.Error:", result.Error)
log.Println("result.RowsAffected:", result.RowsAffected)
```

4. 设置批量插入的批次大小

```
var users = []User{
    {Name: "jinzhu1", Birthday: time.Now()},
    {Name: "jinzhu2", Birthday: time.Now()},
    {Name: "jinzhu3", Birthday: time.Now()},
}
result := db.Create(&users)

for _, user := range users {
    log.Println(user.ID) // 1,2,3
}
log.Println("result.RowsAffected:", result.RowsAffected)
```

要高效地插入大量记录，请将切片传递给 `Create` 方法。 GORM 将生成一条 SQL 来插入所有数据，以返回所有主键值，并触发 `Hook` 方法。 当这些记录可以被分割成多个批次时，GORM会开启一个事务来处理它们。

```
var users = []User{{Name: "jinzhu_1"}, ...., {Name: "jinzhu_10000"}}

// batch size 100
db.CreateInBatches(users, 100)
```

`db.CreateInBatches()` 可以设置插入 `users` 表记录时的批次大小。

连接数据库的时候，可以配置参数 `&gorm.Config` 中的 `CreateInBatches` 参数。

```
&gorm.Config{
  CreateBatchSize: 1000,
}
```

5. 根据Go的Map创建

```
db.Model(&User{}).Create(map[string]interface{}{
  "Name": "jinzhu", "Age": 18,
})

// batch insert from `[]map[string]interface{}{}`
db.Model(&User{}).Create([]map[string]interface{}{
  {"Name": "jinzhu_1", "Age": 18},
  {"Name": "jinzhu_2", "Age": 20},
})
```

`Model`是什么?

```
Model specify the model you would like to run db operations
// update all users's name to `hello`
db. Model(&User{}).Update("name", "hello")
// if user's primary key is non-blank, will use it as condition, then will only
update that user's name to `hello`
db. Model(&user).Update("name", "hello")
```

6. 使用 SQL 表达式创建记录

使用`clause.Expr`可以使用SQL表达式。

`Expr`结构体如下:

```
type Expr struct {
    SQL                string
    Vars               []interface{}
    WithoutParentheses bool
}
```

```
// Create from map
db.Model(User{}).Create(map[string]interface{}{
  "Name": "jinzhu",
  "Location": clause.Expr{
      SQL: "ST_PointFromText(?)",
      Vars: []interface{}{"POINT(100 100)"}},
})

// INSERT INTO `users` (`name`,`location`) VALUES
("jinzhu",ST_PointFromText("POINT(100 100)"));
```

7. 使用Context Valuer 创建记录

```
type Location struct {
    X, Y int
}

// Scan implements the sql.Scanner interface
func (loc *Location) Scan(v interface{}) error {
  // Scan a value into struct from database driver
```

```
}

func (loc Location) GormDataType() string {
  return "geometry"
}

func (loc Location) GormValue(ctx context.Context, db *gorm.DB) clause.Expr {
  return clause.Expr{
    SQL:  "ST_PointFromText(?)",
    Vars: []interface{}{fmt.Sprintf("POINT(%d %d)", loc.X, loc.Y)},
  }
}

type User struct {
  Name     string
  Location Location
}

db.Create(&User{
  Name:     "jinzhu",
  Location: Location{X: 100, Y: 100},
})
// INSERT INTO `users` (`name`,`location`) VALUES
("jinzhu",ST_PointFromText("POINT(100 100)"))
```

8. 默认值

你可以通过结构体Tag `default` 来定义字段的默认值，示例如下：

```
type User struct {
  ID   int64
  Name string `gorm:"default:galeone"`
  Age  int64  `gorm:"default:18"`
}
```

这些默认值会被当作结构体字段的零值插入到数据库中

> **注意**，当结构体的字段默认值是零值的时候比如 `0`，`''`，`false`，这些字段值将不会被保存到数据库中，你可以使用指针类型或者Scanner/Valuer来避免这种情况。

```go
type User struct {
  gorm.Model
  Name string
  Age  *int           `gorm:"default:18"`
  Active sql.NullBool `gorm:"default:true"`
}
```

查询记录

1. 查询单条记录

```go
// 获取第一条记录（主键升序）并且存储到user变量中
var user User
db.First(&user)
// SELECT * FROM users ORDER BY id LIMIT 1;

// 获取一条记录，没有指定排序字段
db.Take(&user)
// SELECT * FROM users LIMIT 1;

// 获取最后一条记录（主键降序）
db.Last(&user)
// SELECT * FROM users ORDER BY id DESC LIMIT 1;

result := db.First(&user)
result.RowsAffected // 返回找到的记录数
result.Error        // returns error or nil

// 检查 ErrRecordNotFound 错误
errors.Is(result.Error, gorm.ErrRecordNotFound)
```