Gorm

1. 使用 database/sql 和 使用 gorm 的区别
2. database/sql ：

Go没有内置驱动支持任何数据库，用户可以基于驱动接口开发相应数据库的驱动。使用该方法直接将在代码里硬编码sql语句。

// 创建表

func CreateTable(DB \*sql.DB) {

    sql := `CREATE TABLE IF NOT EXISTS users(

     id INT(4) PRIMARY KEY AUTO\_INCREMENT NOT NULL,

     username VARCHAR(64),

    password VARCHAR(64),

    status INT(4),

    createtime INT(10)

    ); `

    if \_, err := DB.Exec(sql); err != nil {

        fmt.Println("create table failed:", err)

        return

    }

    fmt.Println("create table successd")

}

1. gorm 对象关系映射（以下第2点）

type User struct {

    Id   int    `gorm:"size:11;primary\_key;AUTO\_INCREMENT;not null" json:"id"`

    Age  int    `gorm:"size:11;DEFAULT NULL" json:"age"`

    Name string `gorm:"size:255;DEFAULT NULL" json:"name"`

    //gorm后添加约束，json后为对应mysql里的字段

}

func main() {

    DB, err := gorm.Open("mysql", "root:123456@tcp(127.0.0.1:3306)/test?charset=utf8")

    if err != nil {

        fmt.Println("failed to connect database:", err)

        return

    } else {

        fmt.Println("connect database success")

        // 创建表方法封装

        DB.AutoMigrate(&User{}) //通过 User 对象结构创建表

        fmt.Println("create table success")

    }

    defer DB.Close()

}

1. orm

orm使对象和数据库进行关系映射。实际上就是对数据库的操作进行封装，开发人员操作代码对象即是操作数据库。

1. gorm

支持多表映射、事务、读写分离

1. import

    "github.com/jinzhu/gorm"

    \_ "github.com/jinzhu/gorm/dialects/mysql"

或者

"gorm.io/gorm"  
 "gorm.io/driver/sqlite"

1. 对数据库的基本操作在官方文档概述中，包括连接数据库、迁移、建表、查询、更新、删除。
2. 创建连接的方法二：（推荐使用，可以配置连接时的参数）

db, err := gorm.Open(mysql.New(mysql.Config{

        DSN:               "root:fantastic0918@tcp(127.0.0.1:3306)/gormtest?charset=utf8mb4&parseTime=true&loc=Local",

        DefaultStringSize: 171, //默认长度

    }), &gorm.Config{

        SkipDefaultTransaction: false, //禁止在事务里执行写入操作

        NamingStrategy: schema.NamingStrategy{

            TablePrefix:   "table\_", //加表名前缀，如"t\_User"

            SingularTable: true,     //表名后不会加S

        },

        DisableForeignKeyConstraintWhenMigrating: true, //不会自动建立物理外键（此时你应该使用逻辑外键，代码里键外键关系）

    })

1. 对数据库的操作方法二：

M := db.Migrator() //返回的M是一个接口Migator,通过调用这个接口中的方法对数据库和表进行操作

M的函数：

HasTable( )/CreateTable( )/RenameTable( )/DropTable(“t\_user” ）

*https://gorm.io/zh\_CN/docs/migration.html*

1. 连接池

sqlDB, err := db.DB()// 获取通用数据库对象 sql.DB，然后使用其提供的功能

defer sqlDB.Close()

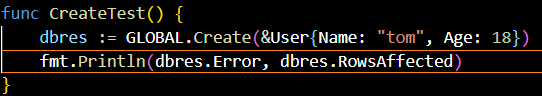
1. 常用标签示例：



1. 创建并输出命令结果

db, \_ = gorm.Open（.......）

var GLOBAL \*gorm.DB = db



批量创建

func CreateTest() {

    dbres := GLOBAL.Create(&[]User{

        {Name: "tom", Age: 18},

        {Name: "jerry", Age: 18},

        {Name: "johny", Age: 19},

        {Name: "json", Age: 20},

    })

    fmt.Println(dbres.Error, dbres.RowsAffected)

}

1. 查询记录；first查符合条件的第一条、Take随机、Find全部（需声明一个结构体切片来存放结果）

（1）

func TestFind() {

    var result = make(map[string]interface{}) //将查找结果存入result字典中

    db.Model(&User{}).First(&result, 3)       //查找的时候一定要先传入模型，查找id=3的记录（没有3则是查找第一条记录）

    fmt.Println(result)

}

func TestFind2() {

    var u1 User

    db.Model(&User{}).Take(&u1) //随机查找一条记录存入结构体u1中

    db.Model(&User{}).Last(&u1) //查找最后一条记录结果存入结构体u1中

}

1. 条件查询

func TestFind() {

    var u1 []User

    db.Where("name = ? and age = ?", "jerry", 18).Or("name = ?", "json").Find(&u1)

    fmt.Println(u1)

}

1. 内联条件查询

db.First(&user, "id = ?", "string\_primary\_key")

1. 查询时智能选择字段

type UserInfo struct {

    Name string

    Age  uint8

}

func TestFind() {

    var u1 []UserInfo

    db.Model(&User{}).Where(" age = ?", 18).Or("name = ?", "json").Find(&u1)

    fmt.Println(u1)

}

//只查询到UserInfo中声明的Name和Age

1. 更新

update：只更新你选择的字段

updates：更新所有字段，结构体0值不参与更新

save：无论如何都更新所有内容，包括0值

db.Model(&User{}).Where("Name = ?", "jerry").Update("age", 19)

1. 删除
   1. 软删除（记录仍在，记录delete\_at字段）：

    db.Where("name = ?", "tom").Delete(&User{})

* 1. 硬删除（直接删除记录）

    db.Unscoped().Where("name = ?", "tom").Delete(&User{})

1. 原生SQL

type Result struct {

    ID   int

    Name string

    Age  int

  }

var result Result

db.Raw("SELECT id, name, age FROM users WHERE name = ?", 3).Scan(&result)

1. 一对一关系
   1. belongs to

type Company struct {}

type Worker struct {

...

CompanyID uint

    Company Company

}

worker结构体内嵌有company字段：worker belongs to company

此时执行db.AutoMigrate(&Worker{}) 会创建表worker和表company

也会一起创建company表中的数据

db.Mode(&w).Association(“Company”).Append(&c)//w主动建立联系

db.Mode(&w).Association(“Company”).Replace(&c,&c2)//换联系

db.Mode(&w).Association(“Company”).Delete(&c)//删除联系

db.Mode(&w).Association(“Company”).Clear()//清理联系（null）

db.Model(&c).Association(“Worker”)......//和以上四个类似

* 1. has one ( 以下是Company has Worker ）

type Company struct {

    gorm.Model

    Name   string

    Worker Worker

}

type Worker struct {

    gorm.Model

    Name      string

    CompanyID uint //联系

创建表Company时不会一起创建表Worker；但是建好表Worker后，添加company的记录时，会连着将和它有联系的worker一起添加进worker表中

* 1. 查询

要想查询出本身嵌有的其它结构体字段时需要使用预加载

没有使用预加载：db.First(&Company{},2) //查询到了Company表id为2的记录，但是记录中没有内嵌结构体字段的信息

使用预加载：db.Preload(“Worker”).First(&Company{},2) //可以查到此company含有的Worker的信息

belongs to的预加载：db.Preload(“Company”).First(&Worker{},1)

1. 一对多

将has one中的Company内嵌的结构体字段Worker变成切片[ ]Worker即可

    db.Preload("Worker").First(&Company{})

此条会查询出第一个“Company”和其拥有的全部“worker”（必须预加载）

db.Preload("Worker", "name=?", "worker2").First(&Company{}) //带条件的预加载

1. 链式预加载

适用情况：Company has Worker；Worker has Info

db.Preload(“Worker.Info”).Preload(“Worker”).First(&Company{})

此时可以在两个Preload()里使用预加载

相当于：

db.Preload(“Worker.Info”).First(&Company{})//此时仅在Info可以预加载

（preload仅在所在的那层可以使用预加载）

1. joins （只适用于一对一对一情况）

    db.Preload("Worker", func(db \*gorm.DB) \*gorm.DB {

        return db.Joins("Info").Where("Age>20")

    }).First(&Company{})

查询第一条company，包含其年龄大于20的员工和员工信息

1. 多对多关系

type Company struct {

    gorm.Model

    Name   string

    Worker []Worker `gorm:"many2many:worker\_company"`

}

type Worker struct {

    gorm.Model

    Name    string

    Company []Company `gorm:"many2many:worker\_company"`

    Info    Info

}

type Info struct {

    gorm.Model

    Salary   int

    WorkerID uint

}

有时间需要补习一下

1. 引用和关联标签
2. 多态及其关系标签：

polymorphic 指定多态类型（结构体上面的那个标签名）

polymorphicvalue 指定多态值（数据库用来记录是哪个结构体的那个字段的值）

1. 引用标签

foreignKey

references

joinForeignKey

joinReferences

1. 事务

*<https://gorm.io/zh_CN/docs/transactions.html>*

默认事务 SkipDefaultTransaction: true //true是确保自己错误的sql不会被执行

1. 自定义数据类型

自定义的数据类型必须实现 [Scanner](https://pkg.go.dev/database/sql" \l "Scanner" \t "https://gorm.io/zh_CN/docs/_blank) 和 [Valuer](https://pkg.go.dev/database/sql/driver" \l "Valuer" \t "https://gorm.io/zh_CN/docs/_blank) 接口，以便让 GORM 知道如何将该类型接收、保存到数据库