Introduction to Beego’s ORM

Michael Van Sickle

@vansimke

Introduction

Setup

Model Definition

- Tables
- Joins
- Indexes

CRUD Operations

Queries

Transactions

Supported Databases

MySQL®

PostgreSQL
the world's most advanced open source database

SQLite

Connect to Database

```
package main

import “github.com/astaxie/beego/orm”

func init() {

    orm.RegisterDriver(“mysql”, orm.DRMySQL)

    orm.RegisterDatabase(“default”, “mysql”, “root:root@orm_test”)

}

func main() {

    o := orm.NewOrm()

    o.Using(“default”) //name given to DB when registered

}
```

Defining Models

```
type User struct {
    Id int  //ID would create field “i_d” in database
    Name string
}

func init() {
    orm.RegisterModel(new(User))
}
```

Specifying Table Names

```
type User struct {
   Id int
   Name string
}

func (u *User) TableName() string {
   return “custom_user”
}
```

Customizing Field Definition

```
type Product struct {
    Id              int        `orm:"pk;auto"`
    Name            string     `orm:"index"`
    Description     string     `orm:"column(product_desc)"`
    SerialNumber    int        `orm:"size(15)"`
    Value           float32    `orm:"digits(10);decimals(2)"`
    Inventory       int        `orm:"-"`
    LastOrdered     time.Time  `orm:"auto_now_add;type(date)"`
}
```

Relationships

```
type User struct {
    Id          int
    Address     *Address       `orm:"rel(one);on_delete(null)"`
}
type Address struct {
    Id      int
    ...
    User    *User              `orm:"reverse(one)"`
}
```

`` `orm:"rel(one)"` ``	◀ One-to-one / one-to-many (owner)
`` `orm:"reverse(one)"` ``	◀ One-to-one (owned)
`` `orm:"rel(fk)"` ``	◀ One-to-one with foreign key (owner)
`` `orm:"reverse(many)"` ``	◀ One-to-many / many-to-many (owned)
`` `orm:"rel(m2m)"` ``	◀ Many-to-many (owner)

Specifying Indexes

```
type Product struct {
    Id              int
    SerialNumber    int          `orm:”index”`
    LastOrdered     *time.Time
    …
}
```

Specifying Indexes

```
type Product struct {
    Id                  int
    SerialNumber        int
    LastOrdered         *time.Time
    …
}
func (p *Product) TableIndex() [][]string {
    return [][]string{
        []string{“SerialNumber, “LastOrdered”},
    }
}
```

Generating Database Schema

```
func init() {
    orm.RegisterDriver(“mysql”, orm.DRMySQL)
    orm.RegisterDatabase(“default”, “mysql”, “...”)
    orm.RegisterModel(&User{})

    orm.RunSyncdb(“default”, true /*force*/,
        true /*verbose*/)
}
```

Creating Data

```
type User struct {
    Id     int         `orm:”pk;auto”`
    Name   string
}
o := orm.NewOrm()
o.Using(“default”)
user := &User{Name:”Fred”}
id, err := o.Insert(user)  //user’s Id field is populated
```

Updating Data

```
o := orm.NewOrm()
user := &User{Name:”Fred”}
id, err := o.Insert(user)  //user’s Id field is populated

user.Name = “Barney”
id, err = o.Update(user)   //update record with user’s Id
```

Updating Data

```
user.Name = “Barney”
id, err = o.Update(user)   //update record with user’s Id

user.Name = “Wilma”
created, id, err = o.ReadOrCreate(user)
```

Deleting Data

```
user := &User{Name:”Fred”}
id, err := o.Insert(user)

o.Delete(user)              //delete specific user
o.Delete(&User{Id:1})       //delete user by Id
```

Queries

```
type User struct {
    Id      int
    Name    string
}
user := User{Id:1})
err := o.Read(&user)            //find user by Id

user = User{Name:”Fred”}
err := o.Read(&user, “Name”)   //find user by other field
```

Query Setters

```
qs := o.QueryTable(“user”)  //use table name, or a reference object

qs.Filter(“id”, 1)                  // WHERE id = 1

qs.Exclude(“name”, “Fred”)          // WHERE NOT name = ‘Fred’

qs.Limit(10)                        // retrieve at most 10 records

qs.Offset(5)                        // start at the fifth result

qs.GroupBy(“name”)                  // GROUP BY name

qs.OrderBy(“id”, “-name”)           // ORDER BY id, name desc

qs.Distinct()                       // SELECT DISTINCT


o.QueryTable(“user”).Filter(“name”, “Fred”).Limit(10).Distinct()
```

Query Setter - Results

```
num, err     := qs.All(&users)
err          := qs.One(&user)
num, err     := qs.Update(
                    orm.Params{“field_name”:”new_value”})
num, err     := qs.Delete()
```

Query Builder

```
qb, err := orm.NewQueryBuilder(“mysql”)

qb.Select(“user.id, user.name, account.is_active”).

    From(“user”).

    InnerJoin(“account”).On(“user.id = profile.fk_user”).

    Where(“profile.age > ?”).

    OrderBy(“name”).Desc().

    Limit(10).Offset(0)

sql := qb.String()

orm.NewOrm().Raw(sql, 20)
```

Transactions

```
o := orm.NewOrm()
err := o.Begin()
// do stuff inside of the transaction
...
if SomeError {
   err = o.Rollback()
} else {
   err = o.Commit()
}
```

Summary

Setup

Model Definition

- Tables
- Joins
- Indexes

CRUD Operations

Queries

Transactions

Beego

View	Controller	Model	Persistence
🙂	🙂	😑	😀