-ScanList

gfORMORMBelongsTo, HasOne, HasMany, ManyToManygf

> ⚠ gf ORMGF v1.13.6

gf ORM

3-4

```
#
CREATE TABLE `user` (
  uid int(10) unsigned NOT NULL AUTO_INCREMENT,
  name varchar(45) NOT NULL,
  PRIMARY KEY (uid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

#
CREATE TABLE `user_detail` (
  uid  int(10) unsigned NOT NULL AUTO_INCREMENT,
  address varchar(45) NOT NULL,
  PRIMARY KEY (uid)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

#
CREATE TABLE `user_scores` (
  id int(10) unsigned NOT NULL AUTO_INCREMENT,
  uid int(10) unsigned NOT NULL,
  score int(10) unsigned NOT NULL,
  course varchar(45) NOT NULL,
  PRIMARY KEY (id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

1. 1:1
2. 1:N
3. N:N1:N1:N

Golang

```
//
type EntityUser struct {
    Uid  int    `orm:"uid"`
    Name string `orm:"name"`
}
//
type EntityUserDetail struct {
    Uid     int    `orm:"uid"`
    Address string `orm:"address"`
}
//
type EntityUserScores struct {
    Id     int    `orm:"id"`
    Uid    int    `orm:"uid"`
    Score  int    `orm:"score"`
    Course string `orm:"course"`
}
//
type Entity struct {
    User       *EntityUser
    UserDetail *EntityUserDetail
    UserScores []*EntityUserScores
}
```

Content Menu

EntityUser, EntityUserDetail, EntityUserScoresEntity

```
err := db.Transaction(func(tx *gdb.TX) error {
    r, err := tx.Table("user").Save(EntityUser{
        Name: "john",
    })
    if err != nil {
        return err
    }
    uid, err := r.LastInsertId()
    if err != nil {
        return err
    }
    _, err = tx.Table("user_detail").Save(EntityUserDetail{
        Uid:     int(uid),
        Address: "Beijing DongZhiMen #66",
    })
    if err != nil {
        return err
    }
    _, err = tx.Table("user_scores").Save(g.Slice{
        EntityUserScores{Uid: int(uid), Score: 100, Course: "math"},
        EntityUserScores{Uid: int(uid), Score: 99, Course: "physics"},
    })
    return err
})
```

Scan

```
//
var user Entity
//
// SELECT * FROM `user` WHERE `name`='john'
err := db.Table("user").Scan(&user.User, "name", "john")
if err != nil {
    return err
}
//
// SELECT * FROM `user_detail` WHERE `uid`=1
err := db.Table("user_detail").Scan(&user.UserDetail, "uid", user.User.Uid)
//
// SELECT * FROM `user_scores` WHERE `uid`=1
err := db.Table("user_scores").Scan(&user.UserScores, "uid", user.User.Uid)
```

ScanList

```
//
var users []Entity
//
// SELECT * FROM `user`
err := db.Table("user").ScanList(&users, "User")
//
// SELECT * FROM `user_detail` WHERE `uid` IN(1,2)
err := db.Table("user_detail").
       Where("uid", gdb.ListItemValuesUnique(users, "User", "Uid")).
       ScanList(&users, "UserDetail", "User", "uid:Uid")
//
// SELECT * FROM `user_scores` WHERE `uid` IN(1,2)
err := db.Table("user_scores").
       Where("uid", gdb.ListItemValuesUnique(users, "User", "Uid")).
       ScanList(&users, "UserScores", "User", "uid:Uid")
```

1. ScanList

```
// ScanList converts <r> to struct slice which contains other complex struct
attributes.
// Note that the parameter <listPointer> should be type of *[]struct/*[]
*struct.
// Usage example:
//
// type Entity struct {
//            User       *EntityUser
//            UserDetail *EntityUserDetail
//           UserScores []*EntityUserScores
// }
// var users []*Entity
// or
// var users []Entity
//
// ScanList(&users, "User")
// ScanList(&users, "UserDetail", "User", "uid:Uid")
// ScanList(&users, "UserScores", "User", "uid:Uid")
// The parameters "User"/"UserDetail"/"UserScores" in the example codes
specify the target attribute struct
// that current result will be bound to.
// The "uid" in the example codes is the table field name of the result, and
the "Uid" is the relational
// struct attribute name. It automatically calculates the HasOne/HasMany
relationship with given <relation>
// parameter.
// See the example or unit testing cases for clear understanding for this
function.
func (m *Model) ScanList(listPointer interface{}, attributeName string,
relation ...string) (err error)
```

- `ScanList(&users, "User")`

  `usersUser`

- `ScanList(&users, "UserDetail", "User", "uid:Uid")`

  `usersUserDetailUseruid:Uid:uid:UiduiduidUidUid`

- `ScanList(&users, "UserScores", "User", "uid:Uid")`

  `usersUserScoresUseruid:Uid:UserScores[]*EntityUserScoresUserUserScores1:N`

`nil`

2. ListItemValues/ListItemValuesUnique

```
// ListItemValues retrieves and returns the elements of all item struct
/map with key <key>.
// Note that the parameter <list> should be type of slice which contains
elements of map or struct,
// or else it returns an empty slice.
//
// The parameter <list> supports types like:
// []map[string]interface{}
// []map[string]sub-map
// []struct
// []struct:sub-struct
// Note that the sub-map/sub-struct makes sense only if the optional
parameter <subKey> is given.
func ListItemValues(list interface{}, key interface{}, subKey ...interface
{}) (values []interface{})

// ListItemValuesUnique retrieves and returns the unique elements of all
struct/map with key <key>.
// Note that the parameter <list> should be type of slice which contains
elements of map or struct,
// or else it returns an empty slice.
// See gutil.ListItemValuesUnique.
func ListItemValuesUnique(list interface{}, key string, subKey ...interface
{}) []interface{}
```

`ListItemValuesUniqueListItemValuesstruct/map/[]interface{}`

- `gdb.ListItemValuesUnique(users, "Uid")usersUid[]interface{}uidSELECT...IN...`
- `gdb.ListItemValuesUnique(users, "User", "Uid")usersUserUid[]interface{}uidSELECT...IN...`