ORM CountFields

Sookie Justin ORM-

Content Menu

1

```
func (u *userApi) Test(r *ghttp.Request) {
   model := dao.User.Ctx(r.Context()).Fields("id, nickname, status").Where
("id > ?", 1)
   //
   total, err := model.Count()
   if err != nil {
      tresponse.Json(r, tresponse.WithCode(500), tresponse.WithMessage(err.
Error()), tresponse.WithExit())
   }
   //
   result, err := model.Offset(0).Limit(10).All()
   if err != nil {
      return
   }
   tresponse.Json(r, tresponse.WithData(g.Map{
      "list": result.List(),
      "total": total,
   }))
}
```

2

```
Error 1064: You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near
'nickname,status) FROM `user` WHERE id \u003e ?' at line 1, SELECT COUNT
(id,nickname,status) FROM `user` WHERE id \u003e 1\n
```

3

sqlCount()

1Fields

```
func (u *userApi) Test(r *ghttp.Request) {
   model := dao.User.Ctx(r.Context()).Where("id > ?", 1)
   //
   total, err := model.Count()
   if err != nil {
      tresponse.Json(r, tresponse.WithCode(500), tresponse.WithMessage(err.
Error()), tresponse.WithExit())
   }
   //
   result, err := model.Offset(0).Limit(10).All()
   if err != nil {
      return
   }
   tresponse.Json(r, tresponse.WithData(g.Map{
      "list": result.List(),
      "total": total,
   }))
}
```

2Count

```
// Count does "SELECT COUNT(x) FROM ..." statement for the model.
// The optional parameter `where` is the same as the parameter of Model.
Where function,
// see Model.Where.
func (m *Model) Count(where ...interface{}) (int, error) {
        //m.fields = ""
        if len(where) > 0 {
                return m.Where(where[0], where[1:]...).Count()
        }
        var (
                sqlWithHolder, holderArgs = m.getFormattedSqlAndArgs
(queryTypeCount, false)  // sql
                list, err                 = m.doGetAllBySql(sqlWithHolder,
holderArgs...)
        )
        if err != nil {
                return 0, err
        }
        if len(list) > 0 {
                for _, v := range list[0] {
                        return v.Int(), nil
                }
        }
        return 0, nil
}
```

fieldscount

```
func (m *Model) getFormattedSqlAndArgs(queryType int, limit1 bool)
(sqlWithHolder string, holderArgs []interface{}) {
        switch queryType {
        case queryTypeCount:
                countFields := "COUNT(1)"
                if m.fields != "" && m.fields != "*" {
                        // DO NOT quote the m.fields here, in case of
fields like:
                        // DISTINCT t.user_id uid
                        countFields = fmt.Sprintf(`COUNT(%s%s)`, m.
distinct, m.fields)
                }
                // Raw SQL Model.
                if m.rawSql != "" {
                        sqlWithHolder = fmt.Sprintf("SELECT %s FROM (%s)
AS T", countFields, m.rawSql)
                        return sqlWithHolder, nil
                }
                conditionWhere, conditionExtra, conditionArgs := m.
formatCondition(false, true)
                sqlWithHolder = fmt.Sprintf("SELECT %s FROM %s%s",
countFields, m.tables, conditionWhere+conditionExtra)
                if len(m.groupBy) > 0 {
                        sqlWithHolder = fmt.Sprintf("SELECT COUNT(1) FROM
(%s) count_alias", sqlWithHolder)
                }
                return sqlWithHolder, conditionArgs

        default:
                conditionWhere, conditionExtra, conditionArgs := m.
formatCondition(limit1, false)
                // Raw SQL Model, especially for UNION/UNION ALL featured
SQL.
                if m.rawSql != "" {
                        sqlWithHolder = fmt.Sprintf(
                                "%s%s",
                                m.rawSql,
                                conditionWhere+conditionExtra,
                        )
                        return sqlWithHolder, conditionArgs
                }
                // DO NOT quote the m.fields where, in case of fields like:
                // DISTINCT t.user_id uid
                sqlWithHolder = fmt.Sprintf(
                        "SELECT %s%s FROM %s%s",
                        m.distinct,
                        m.getFieldsFiltered(),
                        m.tables,
                        conditionWhere+conditionExtra,
                )
                return sqlWithHolder, conditionArgs
        }
```

Fields

```
func (u *userApi) Test(r *ghttp.Request) {
   model := dao.User.Ctx(r.Context()).Fields("id, nickname, status").Where
("id > ?", 1)
   //
   countModel := model.Clone()
   countModel = countModel.Fields("*")
   total, err := countModel.Count()
   if err != nil {
      tresponse.Json(r, tresponse.WithCode(500), tresponse.WithMessage(err.
Error()), tresponse.WithExit())
   }
   //
   result, err := model.Offset(0).Limit(10).All()
   if err != nil {
      return
   }
   tresponse.Json(r, tresponse.WithData(g.Map{
      "list": result.List(),
      "total": total,
   }))
}
```

count

```
Error 1064: You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near
'nickname,status,*) FROM `user` WHERE id \u003e ?' at line 1, SELECT COUNT
(id,nickname,status,*) FROM `user` WHERE id \u003e 1\n
```

Fields()

```
// Fields appends `fieldNamesOrMapStruct` to the operation fields of the
model, multiple fields joined using char ','.
// The parameter `fieldNamesOrMapStruct` can be type of string/map/*map
/struct/*struct.
func (m *Model) Fields(fieldNamesOrMapStruct ...interface{}) *Model {
        length := len(fieldNamesOrMapStruct)
        if length == 0 {
                return m
        }
        switch {
        // String slice.
        case length >= 2:
                return m.appendFieldsByStr(gstr.Join(
                        m.mappingAndFilterToTableFields(gconv.Strings
(fieldNamesOrMapStruct), true),
                        ",",
                ))
        // It needs type asserting.
        case length == 1:
                switch r := fieldNamesOrMapStruct[0].(type) {
                case string:
                        return m.appendFieldsByStr(gstr.Join(
                                m.mappingAndFilterToTableFields([]string
{r}, false), ",",
                        ))
                case []string:
                        return m.appendFieldsByStr(gstr.Join(
                                m.mappingAndFilterToTableFields(r, true),
",",
                        ))
                default:
                        return m.appendFieldsByStr(gstr.Join(
                                m.mappingAndFilterToTableFields(gutil.Keys
(r), true), ",",
                        ))
                }
        }
        return m
}
```

Fields()append

```
// mappingAndFilterToTableFields mappings and changes given field name to
really table field name.
// Eg:
// ID        -> id
// NICK_Name -> nickname
func (m *Model) mappingAndFilterToTableFields(fields []string, filter
bool) []string {
        //   SHOW FULL COLUMNS FROM `table`
        fieldsMap, err := m.TableFields(m.tablesInit)
        if err != nil || len(fieldsMap) == 0 {
                return fields
        }
        var (
                inputFieldsArray  = gstr.SplitAndTrim(gstr.Join(fields,
","), ",")
                outputFieldsArray = make([]string, 0, len
(inputFieldsArray))
        )
        fieldsKeyMap := make(map[string]interface{}, len(fieldsMap))
        for k, _ := range fieldsMap {
                fieldsKeyMap[k] = nil
        }
        for _, field := range inputFieldsArray {
                if _, ok := fieldsKeyMap[field]; !ok {  //
                        if !gregex.IsMatchString
(regularFieldNameWithoutDotRegPattern, field) {  //
                                // Eg: user.id, user.name
                                outputFieldsArray = append
(outputFieldsArray, field)  // append
                                continue
                        } else {
                                // Eg: id, name
                                if foundKey, _ := gutil.
MapPossibleItemByKey(fieldsKeyMap, field); foundKey != "" {
                                        outputFieldsArray = append
(outputFieldsArray, foundKey) // append
                                } else if !filter {
                                        outputFieldsArray = append
(outputFieldsArray, field)
                                }
                        }
                } else {
                        outputFieldsArray = append(outputFieldsArray,
field)
                }
        }
        return outputFieldsArray
}
```

Model.fields

1Count()Model.fields

Count()Sum(), Avg()

```
// Count does "SELECT COUNT(x) FROM ..." statement for the model.
// The optional parameter `where` is the same as the parameter of Model.
Where function,
// see Model.Where.
func (m *Model) Count(where ...interface{}) (int, error) {
        m.fields = ""
        if len(where) > 0 {
                return m.Where(where[0], where[1:]...).Count()
        }
        var (
                sqlWithHolder, holderArgs = m.getFormattedSqlAndArgs
(queryTypeCount, false)  // sql
                list, err                 = m.doGetAllBySql(sqlWithHolder,
holderArgs...)
        )
        if err != nil {
                return 0, err
        }
        if len(list) > 0 {
                for _, v := range list[0] {
                        return v.Int(), nil
                }
        }
        return 0, nil
}
```

1Count()

```
func (u *userApiForDao) Test(r *ghttp.Request) {
        model := dao.User.Ctx(r.Context()).Fields("id, nickname, status").
Where("id > ?", 1)
        //
        countModel := model.Clone()
        total, err := countModel.Count()
        if err != nil {
                tresponse.Json(r, tresponse.WithCode(500), tresponse.
WithMessage(err.Error()), tresponse.WithExit())
        }
        //
        result, err := model.Offset(0).Limit(10).All()
        if err != nil {
                return
        }
        tresponse.Json(r, tresponse.WithData(g.Map{
                "list": result.List(),
                "total": total,
        }))
}
```

CloneModelFields()

```
func (u *userApiForDao) Test(r *ghttp.Request) {
        model := dao.User.Ctx(r.Context()).Fields("id, nickname, status").
Where("id > ?", 1)
        //
        //countModel := model.Clone()
        //total, err := countModel.Count()
        total, err := model.Fields("*").Count()


        if err != nil {
                tresponse.Json(r, tresponse.WithCode(500), tresponse.
WithMessage(err.Error()), tresponse.WithExit())
        }
        //
        result, err := model.Offset(0).Limit(10).All()
        if err != nil {
                return
        }
        tresponse.Json(r, tresponse.WithData(g.Map{
                "list": result.List(),
                "total": total,
        }))
}
```

2Model.fieldsmodel.fields

gdb_model_fields.goFieldsReset

```
// FieldsReset Reset the model fields property
func (m *Model) FieldsReset(fieldNamesOrMapStruct ...interface{}) *Model {
   model := m.getModel()
   model.fields = ""
   return model.Fields(fieldNamesOrMapStruct...)
}
```

1

```
func (u *userApiForDao) Test(r *ghttp.Request) {
        model := dao.User.Ctx(r.Context()).Fields("id, nickname, status").
Where("id > ?", 1)
        //
        countModel := model.Clone()
        total, err := countModel.FieldsReset("id").Count()
        if err != nil {
                tresponse.Json(r, tresponse.WithCode(500), tresponse.
WithMessage(err.Error()), tresponse.WithExit())
        }
        //
        result, err := model.Offset(0).Limit(10).All()
        if err != nil {
                return
        }
        tresponse.Json(r, tresponse.WithData(g.Map{
                "list": result.List(),
                "total": total,
        }))
}
```

2

```
func (u *userApiForDao) Test(r *ghttp.Request) {
        model := dao.User.Ctx(r.Context()).Fields("id, nickname, status").
Where("id > ?", 1)
        //
        total, err := model.FieldsReset("").CountColumn("id")
        if err != nil {
                tresponse.Json(r, tresponse.WithCode(500), tresponse.
WithMessage(err.Error()), tresponse.WithExit())
        }
        //
        result, err := model.Offset(0).Limit(10).All()
        if err != nil {
                return
        }
        tresponse.Json(r, tresponse.WithData(g.Map{
                "list": result.List(),
                "total": total,
        }))
}
```

3fieldsAll()fieldsorm

```
func (u *userApiForDao) Test(r *ghttp.Request) {
        fields := "id, nickname, status"
        model := dao.User.Ctx(r.Context()).Where("id > ?", 1)
        //
        total, err := model.Count()
        if err != nil {
                tresponse.Json(r, tresponse.WithCode(500), tresponse.
WithMessage(err.Error()), tresponse.WithExit())
        }
        //
        result, err := model.Fields(fields).Offset(0).Limit(10).All()
        if err != nil {
                return
        }
        tresponse.Json(r, tresponse.WithData(g.Map{
                "list": result.List(),
                "total": total,
        }))
}
```