GOPHER CHINA 2020

中国 上海 / 2020-11.21-22

GORM 剖析与最佳实践

张金柱 @ 字节跳动

Jinzhu / 张金柱

- Github： @jinzhu
- 字节跳动： @zhangjinzhu
- GORM 官网： https://gorm.io

  https://gorm.cn （国内 CDN）

GORM 剖析与最佳实践 - 大纲

- GORM 五分钟快速入门
- SQL 是怎么生成的（Locking，Optimizer Hints，Batch Upsert）
- 插件是怎么工作的（读写分离，Open Tracing，压测平台）
- 最佳实践 & FAQ

GORM 五分钟快速入门 - 初始化

```go
import "gorm.io/gorm"
import "gorm.io/driver/mysql"

// 模型定义
type Product struct {
  ID        uint
  Code      string
  Price     uint
  CreatedAt time.Time
  UpdatedAt time.Time
}

// 连接数据库
dsn := "gorm:gorm@tcp(localhost:9910)/gorm?charset=utf8&parseTime=True&loc=Local"
db, err := gorm.Open(mysql.Open(dsn), &gorm.Config{
  Logger: logger.Default.LogMode(logger.Silent),
  // ... https://gorm.io/docs/gorm_config.html
})

// 操作数据库
db.AutoMigrate(&Product{})
db.Migrator().CreateTable(&Product{})
// ... https://gorm.io/docs/migration.html
```

GORM 五分钟快速入门 - 常见用法

```go
// 创建
db.Create(&Product{Code: "L1212", Price: 1000})

// 读取
var product Product
db.First(&product, 1)                      // 查询id为1的product
db.First(&product, "code = ?", "L1212") // 查询code为L1212的product

// 更新某个字段
db.Model(&product).Update("Price", 2000)
db.Model(&product).UpdateColumn("Price", 2000)

// 更新多个字段
db.Model(&product).Updates(Product{Price: 2000, Code: "L1212"})
db.Model(&product).Updates(map[string]interface{}{"Price": 2000, "Code": "L1212"})

// 批量更新
db.Model(&Product{}).Where("price < ?", 2000).Updates(map[string]interface{}{"Price": 2000})

// 删除 - 删除product
db.Delete(&product)
```

SQL是怎么生成的 (SQL Statement & Clauses)

```sql
SELECT name, age, employee_number

FROM users

WHERE
  role <> 'manager' AND
  age > 35

ORDER BY age DESC

LIMIT 10 OFFSET 0

FOR UPDATE
```

SQL是怎么生成的 (SQL Statement & Clauses)

SELECT Clause
FROM Clause
WHERE Clause
ORDER BY Clause
LIMIT Clause
FOR Clause

```
SELECT name, age, employee_number
FROM users
WHERE
  role <> 'manager' AND
  age > 35
ORDER BY age DESC
LIMIT 10 OFFSET 0
FOR UPDATE
```

expression

expression

SQL STATEMENT

SQL是怎么生成的 (GORM Statement & Clauses)

```go
db.Select("name", "age").Limit(10).Order("age").Where("role <> ?", "manager").Where("age > ?", 35).Find(&users)
```

```go
db.Clauses(
  clause.Select{
    Columns: []string{"name", "age"},
  },
  clause.Limit{
    Limit: 10, Offset: 0,
  },
  clause.OrderBy{
    Columns: {Column: "age"},
  }
  clause.Where{
    Exprs: []clause.Expression{
      clause.Neq{Column: "role", Value: "manager"},
      clause.Gt{Column: "age", Value: 35},
    },
  }
).Find(&users) // clause.From{Tables: []clause.Table{"users"}}
```

SELECT Clause

LIMIT Clause

ORDER BY Clause

WHERE Clause

FROM Clause

STATEMENT

SQL是怎么生成的 - 使用 Clause / 修改 Clause Builder - Locking

```go
// 查询时包含 Locking Clause
db.Clauses(clause.Locking{Strength: "UPDATE"}).Find(&users)
// SELECT * FROM `users` FOR UPDATE
```

SELECT SQL + Locking Clause

```go
// 不同的数据库对于相同的 Clause 生成的 SQL 不同
db.Clauses(clause.Locking{
  Strength: "SHARE",
  Table: clause.Table{Name: clause.CurrentTable},
}).Find(&users)
// SELECT * FROM `users` LOCK IN SHARE MODE  // MySQL < 8
// SELECT * FROM `users` FOR SHARE OF `users` // Others
```

Locking Clause 生成不同 SQL

```go
// gorm.io/driver/mysql/mysql.go
if dialector.Config.DontSupportForShareClause {
  db.ClauseBuilders["FOR"] = func(c clause.Clause, builder clause.Builder) {
    if values, ok := c.Expression.(clause.Locking); ok &&
      strings.EqualFold(values.Strength, "SHARE") {
      builder.WriteString("LOCK IN SHARE MODE")
      return
    }
    c.Build(builder)
  }
}
```

SQL是怎么生成的 - Clause 钩子 - Hints

```go
import "gorm.io/hints"

db.Clauses(hints.New("MRR(idx1)")).Find(&User{})
// SELECT /*+ MRR(idx1) */ * FROM `users`

db.Clauses(hints.UseIndex("idx_user_name")).Find(&User{})
// SELECT * FROM `users` USE INDEX (`idx_user_name`)

db.Clauses(hints.ForceIndex("idx_user_name", "idx_user_id").ForJoin()).Find(&User{})
// SELECT * FROM `users` FORCE INDEX FOR JOIN (`idx_user_name`,`idx_user_id`)"

db.Clauses(hints.Comment("select", "master")).Find(&User{})
// SELECT /*master*/ * FROM `users`;

db.Clauses(hints.CommentBefore("insert", "node2")).Create(&user)
// /*node2*/ INSERT INTO `users` ...;

db.Clauses(hints.CommentBefore("select", "node2")).Find(&user)
// /*node2*/ SELECT * FROM `users`;

db.Clauses(hints.CommentAfter("where", "hint")).Find(&User{}, "id = ?", 1)
// SELECT * FROM `users` WHERE id = ? /* hint */
```

添加到 SELECT Clause 后

添加到 FORM Clause 后

Before, After Name, After 某 Clause

SQL是怎么生成的 - 修改 Clause / 重新组装 Clause - Batch Upsert

```
var users = []User{{Name: "jinzhu1"}, {Name: "jinzhu2"}, {Name: "jinzhu3"}}
db.Create(&users) // user.ID => 1,2,3
```

批量插入

```
db.Clauses(clause.OnConflict{DoNothing: true}).Create(&users)
// INSERT INTO `users` *** ON DUPLICATE KEY DO NOTHING; // PostgreSQL
// INSERT INTO `users` *** ON DUPLICATE KEY UPDATE `id`=`id`; // MySQL
```

批量插入 & 忽略错误

```
// 批量更新数据的某些字段
db.Clauses(clause.OnConflict{
  Columns:   []clause.Column{{Name: "id"}},
  DoUpdates: clause.AssignmentColumns([]string{"name", "age"}),
}).Create(&users)
// INSERT INTO `users` *** ON DUPLICATE KEY UPDATE `name`=VALUES(name),`age=VALUES(age); // MySQL
// INSERT INTO "users" *** ON CONFLICT ("id") DO UPDATE SET *** RETURNING "id"; // PostgreSQL
```

批量插入 || 更新现有数据

```
db.Clauses(clause.OnConflict{
  Columns:   []clause.Column{{Name: "id"}},
  DoUpdates: clause.Assignments(map[string]interface{}{"deleted_at": nil}),
}).Create(&users)
// INSERT INTO `users` *** ON DUPLICATE KEY UPDATE `deleted_at`=NULL;

// SQL Server
// MERGE INTO "users" USING *** WHEN NOT MATCHED THEN INSERT *** WHEN MATCHED THEN UPDATE SET ***
```

批量插入 || 更新现有数据

插件是怎么工作的 - GORM Callbacks

```go
// create callbacks
db.callbacks.creates
// update callbacks
db.callbacks.updates
// delete callbacks
db.callbacks.deletes
// query callbacks
db.callbacks.queries
// row/rows sql query callbacks
db.callbacks.row
// raw sql query callbacks
db.callbacks.raw
```

GORM Callbacks 支持 Create, Update, Delete, Query, Row, Raw 六种 Callbacks

```go
// 创建对象
db.Create(&Product{Code: "L1212", Price: 1000})
```

```go
func Create(data interface{}) error {
  // ... 伪代码
  for _, f := range db.callbacks.creates {
    f()
  }
}
```

执行 Create 的过程

取出注册的 Create Callbacks 并调用

插件是怎么工作的 - GORM Callbacks

```go
db.Callback().Create().Register("gorm:begin_transaction", BeginTransaction)
db.Callback().Create().Register("gorm:before_create", BeforeCreate)
db.Callback().Create().Register("gorm:save_before_associations", SaveBeforeAssociations)
db.Callback().Create().Register("gorm:create", Create)
db.Callback().Create().Register("gorm:save_after_associations", SaveAfterAssociations)
db.Callback().Create().Register("gorm:after_create", AfterCreate)
db.Callback().Create().Register("gorm:commit_or_rollback_transaction", CommitOrRollback)
```

GORM 默认 Create Callbacks

```go
// 注册新Callback
db.Callback().Create().Register("my_plugin:new_callback", func(*gorm.DB) {})

// 删除Callback
db.Callback().Create().Remove("gorm:begin_transaction")

// 替换Callback
db.Callback().Create().Replace("gorm:before_create", func(*gorm.DB) {})
```

GORM Callbacks API

```go
// 指定Callback顺序
db.Callback().Create().Before("gorm:before_create").After("my_plugin:new_callback").
  Register("my_plugin:new_callback2", func(*gorm.DB) {})

// 注册到所有服务之前
db.Callback().Create().Before("*").Register("my_plugin:new_callback", func(*gorm.DB) {})
```

指定 Callbacks 顺序

插件是怎么工作的 - GORM Callbacks - OpenTracing

```go
func before(db *gorm.DB) {
  span, _ := opentracing.StartSpanFromContext(db.Statement.Context, "gorm")
  db.InstanceSet("gorm_span_key", span)
  return
}

func after(db *gorm.DB) {
  if _span, ok := db.InstanceGet("gorm_span_key"); ok {
    if span, ok := _span.(opentracing.Span); ok {
      defer span.Finish()
      if db.Error != nil {
        span.LogFields(tracerLog.Error(db.Error))
      }
      span.LogFields(
        tracerLog.String("sql", db.Dialector.Explain(db.Statement.SQL.String(), db.Statement.Vars...))
      )
    }
  }
}

db.Callback().Create().Before("gorm:before_create").Register("opentracing:before", before)
db.Callback().Create().After("gorm:after_create").Register("opentracing:after", after)
// ...
```

插件是怎么工作的 - GORM Callbacks - 多数据库/读写分离

```go
DB, err := gorm.Open(mysql.Open("db1_dsn"), &gorm.Config{})

DB.Use(dbresolver.Register(dbresolver.Config{
  Sources:  []gorm.Dialector{ mysql.Open("db2_dsn") },
  Replicas: []gorm.Dialector{ mysql.Open("db3_dsn"), mysql.Open("db4_dsn") },
  Policy: dbresolver.RandomPolicy{},
}).Register(dbresolver.Config{
  Replicas: []gorm.Dialector{ mysql.Open("db5_dsn") },
}, &User{}, &Address{}).Register(dbresolver.Config{
  Sources:  []gorm.Dialector{ mysql.Open("db6_dsn"), mysql.Open("db7_dsn") },
  Replicas: []gorm.Dialector{ mysql.Open("db8_dsn") },
}, "orders", &Product{}, "secondary"))
```

注册 GORM Callbacks

db2 为主数据库（全局）

db3, db4 为从数据库（全局）

db5 为从数据库，默认连接 db1 为主数据库 (User，Address)

```go
// 使用 Write 模式：从 sources db `db1` 读取 user
DB.Clauses(dbresolver.Write).First(&user)
```

指定写模式

```go
// 指定 Resolver：从 `secondary` 的 replicas db `db8` 读取 user
DB.Clauses(dbresolver.Use("secondary")).First(&user)
```

指定策略组 secondary

```go
// 指定 Resolver 和 Write 模式：使用 secondary 的 sources db db6 或 db7
DB.Clauses(dbresolver.Use("secondary"), dbresolver.Write).First(&user)
```

指定写模式 + 策略组

插件是怎么工作的 - Callbacks/Hooks API

```go
func myPlugin(db *gorm.DB) { // func (user User) BeforeCreate(db *gorm.DB) error
  db.Statement.Value // 当前操作的对象
  db.Statement.ReflectValue // 当前操作的对象的 reflect value
  db.Statement.Table // 表名

  // 当前 Model 的所有字段
  db.Statement.Schema.Fields
  // 当前 Model 的所有主键字段
  db.Statement.Schema.PrimaryFields
  // 优先主键字段：带有 db 名为 `id` 或定义的第一个主键字段。
  db.Statement.Schema.PrioritizedPrimaryField
  // 当前 Model 的所有关系
  db.Statement.Schema.Relationships

  // 根据 db 名或字段名查找字段
  field := db.Statement.Schema.LookUpField("Name")
  field.Name / field.DBName / ...

  for _, field := range db.Statement.Schema.Fields {
    switch db.Statement.ReflectValue.Kind() {
    case reflect.Slice, reflect.Array:
      for i := 0; i < db.Statement.ReflectValue.Len(); i++ {
        // 从字段获取值
        fieldValue, isZero := field.ValueOf(db.Statement.ReflectValue.Index(i))
      }
    case reflect.Struct:
      // 从字段获取值
      fieldValue, isZero := field.ValueOf(db.Statement.ReflectValue)
      // 设置字段值
      err := field.Set(db.Statement.ReflectValue, "newValue")
    }
  }
}
```

插件是怎么工作的 - Callbacks/Hooks API

```go
func (user User) BeforeUpdate(db *gorm.DB) error { // func myPlugin(db *gorm.DB)
  // 通过 tx.Statement 修改当前操作，例如：
  tx.Statement.Select("Name", "Age")
  tx.Statement.AddClause(clause.OnConflict{DoNothing: true})

  // 基于 tx 的操作会在同一个事务中，但不会带上任何当前的条件 (Hooks: tx.Statement.DB)
  var role Role
  err := tx.First(&role, "name = ?", user.Role).Error
  // SELECT * FROM roles WHERE name = "admin"

  // 如果 Role 字段有变更 (BeforeUpdate)
  if tx.Statement.Changed("Role") {
    return errors.New("role not allowed to change")
  }

  if tx.Statement.Changed("Name", "Admin") { // Name 或 Role 变更 (BeforeUpdate)
    tx.Statement.SetColumn("Age", 18)
  }

  // 如果任意字段有变更
  if tx.Statement.Changed() {
    tx.Statement.SetColumn("RefreshedAt", time.Now())
  }
  return err
}

db.Model(&User{ID: 1, Name: "jinzhu"}).Updates(User{Name: "jinzhu2"})
// Changed("Name") => true
db.Model(&User{ID: 1, Name: "jinzhu"}).Updates(map[string]interface{}{"Name": "jinzhu"})
// Changed("Name") => false, 因为 `Name` 没有变更
db.Model(&User{ID: 1, Name: "jinzhu"}).Select("Admin").Updates(User{Name: "jinzhu2"})
// Changed("Name") => false, 因为 `Name` 没有被 Select 选中并更新
```

最佳实践 & FAQ - 更新零值问题

```go
// 使用 struct 只更新非零值字段
db.Model(&user).Updates(User{Name: "hello", Age: 18, Active: false})
// UPDATE users SET name='hello', age=18, updated_at = '2013-11-17 21:34:10' WHERE id = 1;

db.Select("*").Updates(&product)                  // 更新全部字段 (包含零值字段)
db.Select("Code", "Price").Updates(&product)      // 只更新 Code, Price (包含零值字段)
db.Omit("Code").Updates(&product)                 // 除 Code 外的非零值字段
db.Select("*").Omit("Code").Updates(&product)     // 除 Code 外的全部字段 (包含零值字段)
```

方法 1

```go
// 更新全部字段 (包含零值字段)
db.Model(&product).Updates(map[string]interface{"Price": 200, "Name": "T-Shirt", "Age": 0})
// 更新选择的字段
db.Model(&product).Select("Price").Updates(map[string]interface{"Price": 200, "Name": "T-Shirt"})
```

方法 2

```go
// 会更新为零值
db.Model(&user).Update("age", 0)
```

方法 3

```go
// 定义为指针或 Scanner/Valuer
type User struct {
  Age  *int
  Age  sql.NullInt64
}
```

方法 4

最佳实践 & FAQ - 使用 SQL 表达式创建/更新数据

```go
db.Model(User{}).Create(map[string]interface{}{
  "Name": "jinzhu",
  "Location": clause.Expr{SQL: "ST_PointFromText(?)", Vars: []interface{}{"POINT(100 100)"}},
})
// INSERT INTO `users` (`name`,`point`) VALUES ("jinzhu",ST_PointFromText("POINT(100 100)"));
```

通过 Map 使用 SQL 表达式创建

```go
// 使用 GORMValuer 使用 SQL 表达式
type Location struct {
    X, Y int
}
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`,`point`) VALUES ("jinzhu",ST_PointFromText("POINT(100 100)"))
```

通过 struct 用 SQL 表达式

最佳实践 & FAQ - 自定义数据类型 / 使用 SQL 表达式查询数据

```go
type User struct {
  // ...
  Attributes datatypes.JSON // 自定义数据格式实现接口 Scanner, Valuer
}

db.Create(&User{
  Attributes: datatypes.JSON(`{"name":"jinzhu","age":18,"tags":["tag1","tag2"],"org":{"name":"org"}}`),
})

// 自定义查询 SQL 实现接口 clause.Expression
type Expression interface {
  Build(builder Builder)
}

// 检查主键
db.Find(&user, datatypes.JSONQuery("attributes").HasKey("role"))
db.Clauses(datatypes.JSONQuery("attributes").HasKey("org", "name")).Find(&user)
// SELECT * FROM `users` WHERE JSON_EXTRACT(`attributes`, '$.role') IS NOT NULL

// 根据值过滤
db.First(&user, datatypes.JSONQuery("attributes").Equals("jinzhu", "name"))
db.Clauses(datatypes.JSONQuery("attributes").Equals("name1", "org", "name")).First(&user)
// SELECT * FROM `user` WHERE JSON_EXTRACT(`attributes`, '$.name') = "jinzhu"
```

自定义类型

SQL 表达式查询

最佳实践 & FAQ - 共享代码

```go
func Paginate(r *http.Request) func(db *gorm.DB) *gorm.DB {
  return func (db *gorm.DB) *gorm.DB {
    page, _ := strconv.Atoi(r.Query("page"))
    if page == 0 {
      page = 1
    }

    pageSize, _ := strconv.Atoi(r.Query("page_size"))
    switch {
    case pageSize > 100:
      pageSize = 100
    case pageSize <= 0:
      pageSize = 10
    }

    offset := (page - 1) * pageSize
    return db.Offset(offset).Limit(pageSize)
  }
}

// 代码共享
db.Scopes(Paginate(r)).Find(&users)
db.Scopes(Paginate(r)).Find(&articles)
```

共享分页逻辑代码

最佳实践 & FAQ - 数据库事务

```go
// 全局禁用 (仍然可以使用 db.Begin, db.Transaction)
db, err := gorm.Open(sqlite.Open("gorm.db"), &gorm.Config{
  SkipDefaultTransaction: true,
})

// 持续会话模式
tx := db.Session(&Session{SkipDefaultTransaction: true})
tx.Create(&User{Name: "jinzhu"})
tx.Model(&user).Update("Age", 18)

// 推荐使用 Transaction 而不是 Begin
db.Transaction(func(tx *gorm.DB) error {
  tx.Create(&Animal{Name: "Giraffe"})

  tx.Transaction(func(tx2 *gorm.DB) error {
    return errors.New("rollback tx2") // 回滚 tx2
  })

  if random {
    panic("failed") // 全部回滚，并继续抛出 panic
  }
  return nil // 返回 nil 提交事务
})
```

关闭全局默认事务

会话关闭默认事务

嵌套事务

事务代码块

最佳实践 & FAQ - 字段权限设置

```go
type User struct {
  Name1 string `gorm:"<-:create"` // 允许读和创建
  Name2 string `gorm:"<-:update"` // 允许读和更新
  Name3 string `gorm:"<-"`        // 允许读和写（创建和更新）
  Name4 string `gorm:"<-:false"`  // 允许读，禁止写
  Name5 string `gorm:"->"`        // 只读
  Name6 string `gorm:"->;<-:create"` // 允许读和写
  Name7 string `gorm:"->:false;<-:create"` // 仅创建（禁止从 db 读）
  Name8 string `gorm:"-"`  // 读写操作均会忽略该字段

  Manager    User        `gorm:"->"` // 只读
  CreditCard CreditCard `gorm:"->:false;<-:create"`
}

db.Create(&User{
  Name1: "user1",
  Manager: User{Name: "manager"}, // 忽略
  CreditCard: CreditCard{Number: "411111111111"}, // 创建
})

db.Find(&user, "name1 = ?", "user1")
user.Manager //=> User{Name: ""}
user.CreditCard //=> CreditCard{Number: ""}
```

字段读、创建、更新权限

无创建权限

未创建

无读取权限

最佳实践 & FAQ - 关联模型定义

User 拥有一个 Account (has one), 拥有多个 Pets (has many), 多个 Toys (多态 has many)

属于某 Company (belongs to) 属于某 Manager (单表 belongs to) 管理 Team (单表 has many)

会多种 Languages (many to many) 拥有很多 Friends (单表 many to many)

并且他的 Pet 也有一个玩具 Toy (多态 has one)

```go
type User struct {
  gorm.Model
  Name      string // go 基本类型
  Account   Account
  Pets      []*Pet
  Toys      []Toy `gorm:"polymorphic:Owner"`
  CompanyID *int
  Company   Company
  ManagerID *uint
  Manager   *User
  Team      []User     `gorm:"foreignkey:ManagerID"`
  Languages []Language `gorm:"many2many:UserSpeak;"`
  Friends   []*User    `gorm:"many2many:user_friends;"`
}
```

```go
type Pet struct {
  gorm.Model
  UserID *uint
  Toy    Toy `gorm:"polymorphic:Owner;"`
}

type Toy struct {
  ID        uint
  Name      string
  OwnerID   string
  OwnerType string
  CreatedAt time.Time
}
```

最佳实践 & FAQ - 关联模式

```
// 保存用户及其关联 (Upsert)
db.Create(&User{
  Name:       "jinzhu",
  Languages: []Language{{Name: "zh-CN"}, {Name: "en-US"}},
})

// 关联模式
langAssociation := db.Model(&user).Association("Languages")
// 查询关联
langAssociation.Find(&languages)
// 将汉语，英语语添加到用户掌握的语言中
langAssociation.Append([]Language{languageZH, languageEN})
// 把用户掌握的语言替换为汉语，德语
langAssociation.Replace([]Language{languageZH, languageDE})
// 删除用户掌握的两个语言
langAssociation.Delete(languageZH, languageEN)
// 删除用户所有掌握的语言
langAssociation.Clear()
// 返回用户所掌握的语言的数量
langAssociation.Count()
```

关联模式常见操作

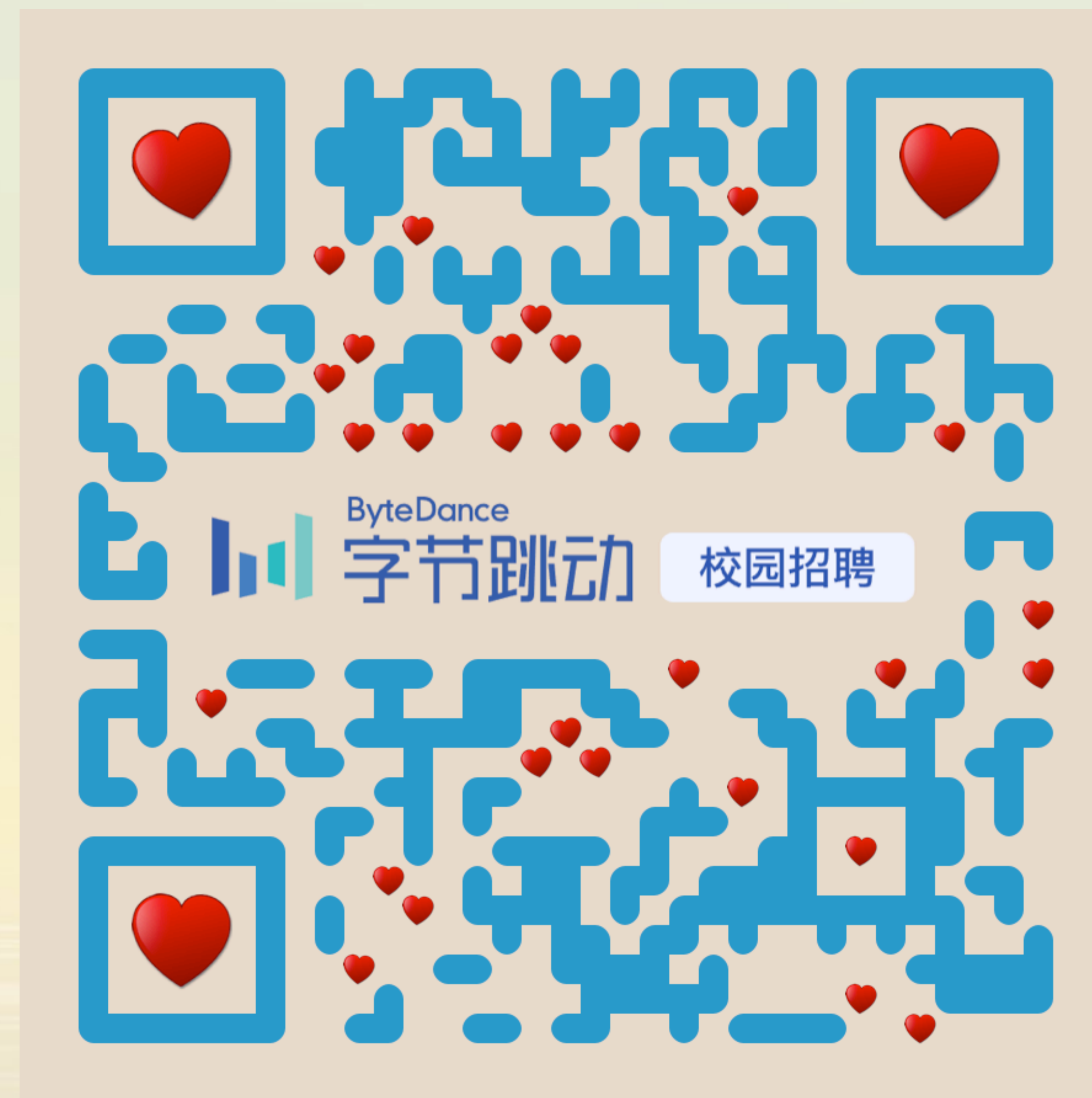
最佳实践 & FAQ - 关联模式预加载

```
// 使用 Join SQL 加载 (单条 JOIN SQL)
db.Joins("Company").Joins("Manager").First(&user, 1)

// 查询用户的时候并找出其订单，个人信息 (1+1 条SQL)
db.Preload("Orders").Preload("Profile").Find(&users)
// SELECT * FROM users;
// SELECT * FROM orders WHERE user_id IN (1,2,3,4); // 一对多
// SELECT * FROM profiles WHERE user_id IN (1,2,3,4); // 一对一

// 预加载全部关联 (只加载一级关联)
db.Preload(clause.Associations).Find(&users)

// 多级预加载
db.Preload("Orders.OrderItems.Product").Find(&users)
db.Preload("Orders.OrderItems.Product").Preload(clause.Associations).Find(&users)

// 查询用户的时候找出其未取消的订单
db.Preload("Orders", "state NOT IN (?)", "cancelled").Find(&users)
db.Preload("Orders", "state = ?", "paid").Preload("Orders.OrderItems").Find(&users)

db.Preload("Orders", func(db *gorm.DB) *gorm.DB {
  return db.Order("orders.amount DESC")
}).Find(&users)
```

最佳实践 & FAQ - To Be Continue

- 关联模式批量操作 / 关联的级联删除
- 数据库连接参数
- 分库分表
- CreateInBatches
- Find To Map
- Sub Query / From Sub Query / SQL Builder
- Group Conditions
- Iteration / Find In Batches
- Prepared Statements
- 默认值问题 / 虚拟字段
- Soft Delete
- gorm.Config
- 修改默认命名策略
- Logger
- New Driver Support
- Session Mode
- Time Tracking (Unix Seconds/Milli/Nano Seconds)
- Hooks
- DryRun Mode
- 复合主键
- Indexes / Checker
- Prometheus
- Performance
- …

ByteDance
字节跳动
内推


ByteDance
字节跳动
校园招聘

Q&A

Thank You!