

ORM в Go: границы применимости и шаблоны изоляции слоя данных

Александр Грунин, разрабатываю на Go @ **Gett**

Границы применимости ORM - за что ORM любят?

- Ассоциации и Lazy Load
- Хуки
- Активные записи
- Простотой апдейт сложных объектов
- Просто делать запросы
- Маппинг

Границы применимости ORM - Ассоциации и Lazy Load

- Проблема 1 + N -
Некоторые реализации
непредсказуемы вместе с
циклами
- Неконсистентность
получаемых данных



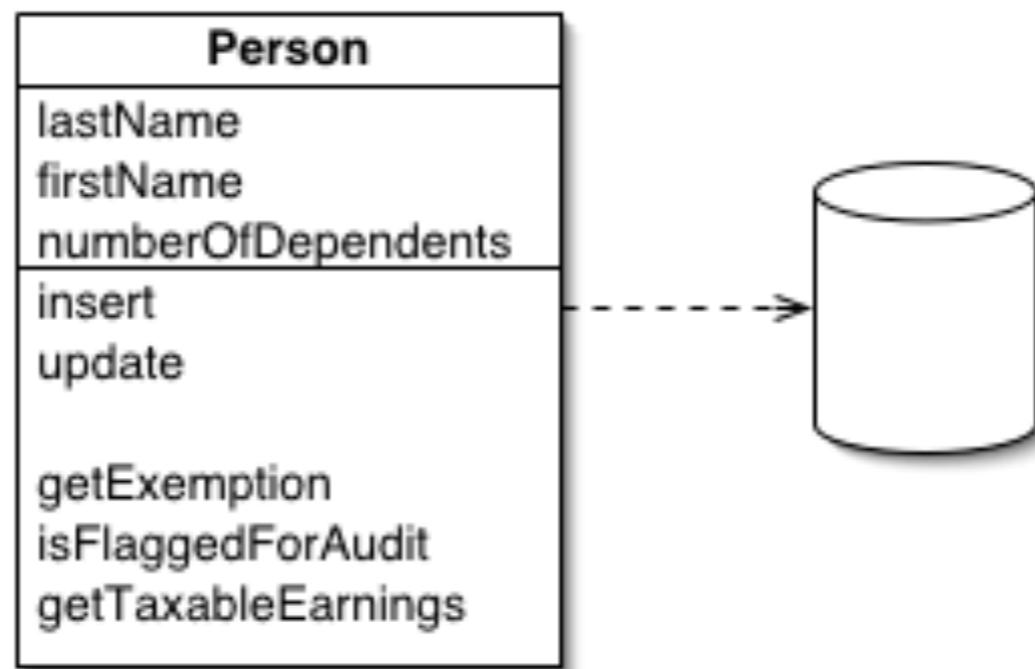
Границы применимости ORM - Хуки

- Триггерная логика - трудно отлаживать
- Неожиданные побочные эффекты



Александр Грунин, **Gett**

Границы применимости ORM - Активные Записи

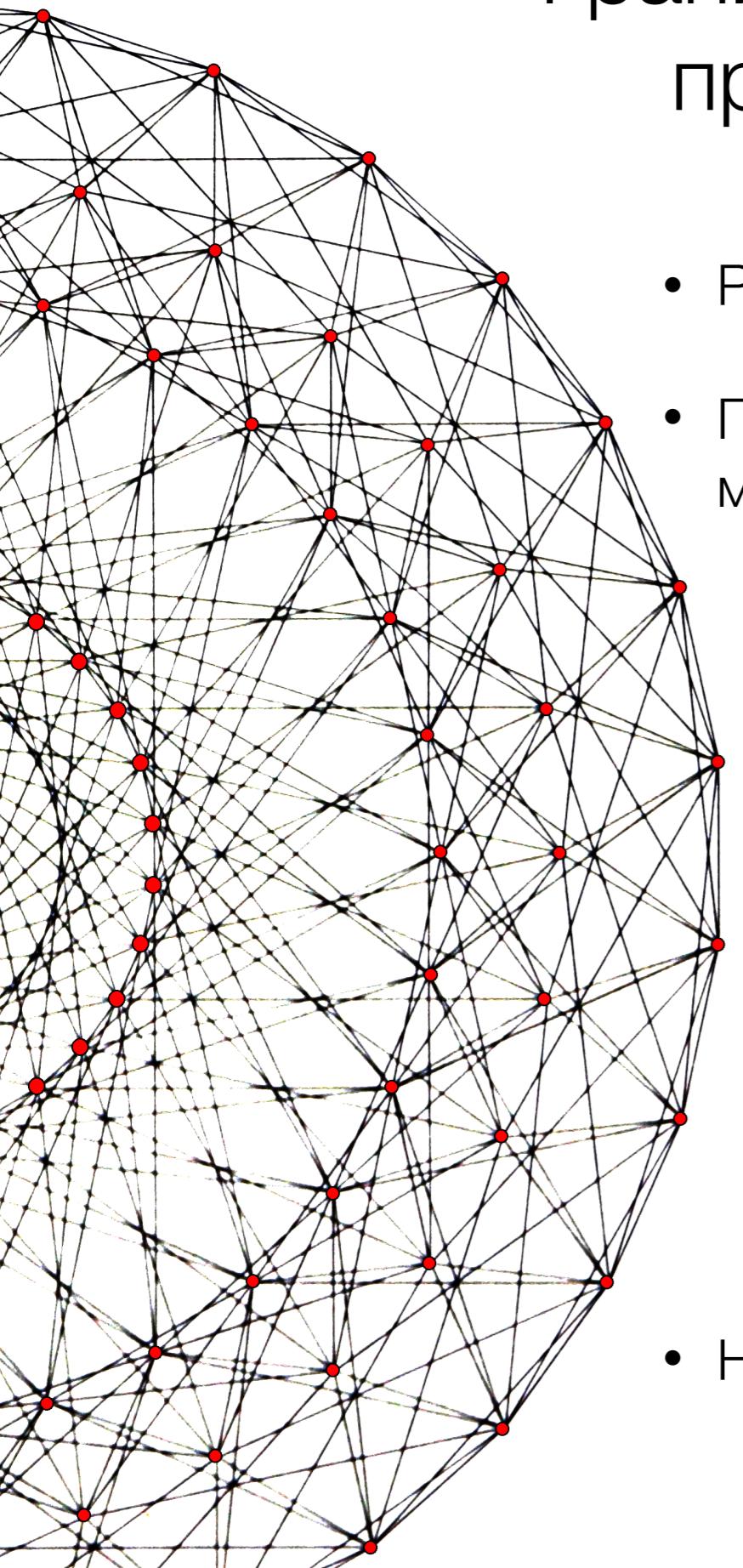


- Преобразование 1 в 1 -> Либо плохая БД либо плохие объекты.
- Превращает объекты в Property Bag
- Активные записи чрезмерно ответственные.

Границы применимости ORM - Обобщенные апдейты

- Превращает ваши объекты в Property Bag
- Сложно описать все валидации, сложно тестировать
- Неприлично много побочных эффектов для простых операций





Границы применимости ORM - простота использования

- Разрыв слоя доступа к данным -> нетестируемый код
- Пользователи ORM забывают (а иногда и не знают) о мощных инструментах РСУБД:
 - Database Views + array_agg + json_agg + xmllagg ...
 - Оконные функции (аналитические запросы)
 - (вредный совет для быстрых) Пишите батчи быстрее с помощью prepared statement.
 - (очень вредный совет для совсем быстрых)
Экономьте раунд-трипы с помощью хранимых функций
- Некоторые ORM просто генерируют странный sql

Границы применимости ORM - Маппинг

- С маппингом обычно все хорошо.
- Впрочем, с маппингом справится даже db/sql
- Не прозевайте неявную смену типа данных в колонке



Что делать?

- Как только в вашем запросе появляется join/CTE/подзапрос - пишите sql вместо вызова методов ORM
- Выносите сложные запросы в Database View и работайте с ними как с ReadOnly таблицами.
- Изолируйте слой доступа к данным с помощью Repository или Command- и Query-объектов

Что НЕ делать?

- Не используйте хуки для бизнес-логики
- Избегайте ORM предоставляющие Активную Запись
- Избегайте Lazy Load
- Не используйте generic update без явной необходимости

Каким должен быть слой данных?

Где хранить слой данных?

Что использовать для доступа к данным?

Каким должен быть слой данных?

- Быть тестируемым - давать возможность мокнуть любые объекты слоя
- Быть изолируемым - обеспечивать изоляцию Бд от бизнес-логики приложения

Где хранить слой данных?

В отдельном модуле

- Слои моделей и доступа данных становятся зависимостью любого модуля
- Нет проблем с code-sharing
- Модели и коллекции получится хранить либо вместе со слоем данных, либо в модуле `models`

Ad-hoc

- Модули автономны
- Есть смысл выделить отдельный модуль для `shared code`.
- Если два модуля используют одну и ту же таблицу - выделяйте еще один модуль-зависимость (или миритесь с дубликацией).

Где хранить слой данных?

В отдельном
модуле

```
.  
|   └── storages  
|       |   └── book_repo.go  
|       |   └── person_repo.go  
|       └── repo_utils.go  
└── models  
    |   └── book.go  
    └── person.go  
└── books  
    └── other files...  
└── persons  
    └── other files...
```

Ad-hoc

```
.  
|   └── books  
|       |   └── book.go  
|       |   └── book_repo.go  
|       └── other files...  
└── persons  
    |   └── person_repo.go  
    |   └── person.go  
    └── other files...  
└── storages  
    └── repo_utils.go
```

Что использовать для доступа к данным?

- Repository
- Commands & Querries

Что использовать для доступа к данным?

- Repository
- Commands & Querries

Что использовать для доступа к данным?

Repository

```
type BookRepo interface {
    func Get(id int) (Book, error)
    func GetAll(criteria Criteria) (BookCollection, error)
    func Save(book Book) (int, error)
    func Destroy(id int) error
}
```

Что использовать для доступа к данным?

Repository

```
type BookRepo interface {  
    func Get(id int) (Book, error)  
    func GetAll(criteria Criteria) (BookCollection, error)  
    func Save(book Book) (int, error)  
    func Destroy(id int) error  
}
```

Repository без generic update и GetAll

```
type BookRepo interface{} {  
    func Get(id int) (Book, error)  
    func GetByTitle(title string) (BookCollection, error)  
    func GetByISBN(isbn string) (BookCollection, error)  
    func MarkAsAvailable(id int) error  
    func UpdateQty(id int, qty int) error  
    func Destroy(id int) error  
}
```

Что использовать для доступа к данным?

Commands and Queries

```
type BookQuery interface{} {
    func Get(id int) (Book, error)
}

type BooksByISBNQuery interface{} {
    func GetByISBN(isbn string) (BookCollection, error)
}

type BooksByTitleQuery interface{} {
    func GetByTitle(title string) (BookCollection, error)
}

type BooksMarkAvailableCmd interface{} {
    func MarkAsAvailable(id int) error
}

type BooksUpdateQtyCmd interface{} {
    func UpdateQty(id int, qty int) error
}

type BooksDestroyCmd interface{} {
    func Destroy(id int) error
}
```

Каким должен быть слой данных?

Быть трестируемым и изолируемым

Где хранить слой данных?

*В отдельном пакете,
либо ad-hoc в моделях с бизнес-логикой*

Что использовать для доступа к данным?

*Паттерн Repository,
либо Command- и Query-объекты*

Спасибо!

Александр Грунин, **Gett**