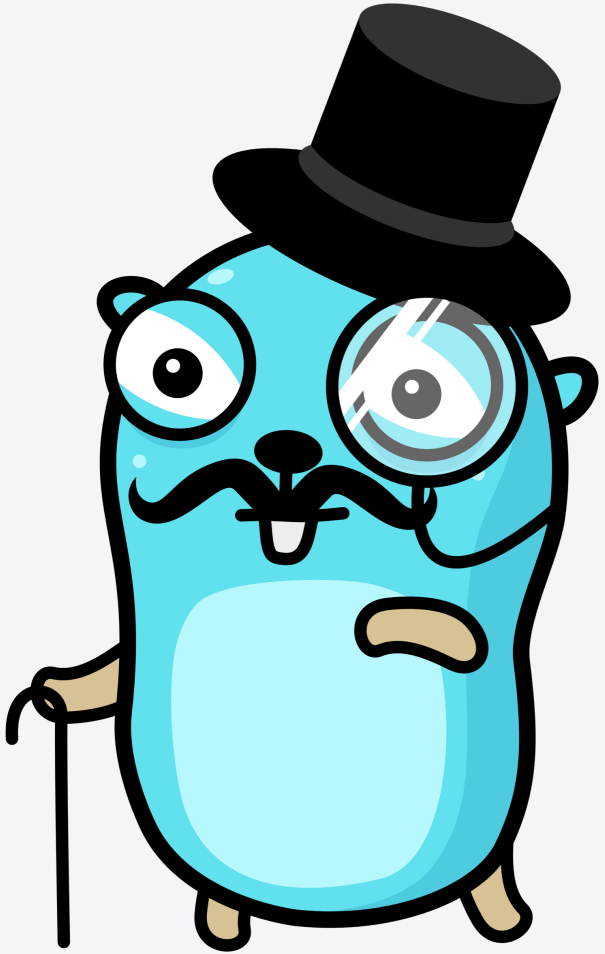
ТИНЬКОФФ

Базы, деньги и Go Way

Когда ORM – не говнокод, а отличное бизнес решение



Ушков Николай
Ведущий инженер

Студенты

Ваня

Берем готовое!
Будут проблемы
будем решать

Goway! DDD!
Чистый код!
Архитектура!

Петя

Выбор технологий

Бизнес хочет: Быстро, дешево, про высокое качество и нагрузку разговоров не было

Ваня

GORM

PostgreSQL

Петя

jackc/pgx

Ваня. Пишем код: модель

```go
package model

import ...

const TableNameProduct = "products"  1 usage  new *

// Product mapped from table <products>
type Product struct {  11 usages  new *
    ID          int32     `gorm:"column:id;type:integer;primaryKey;autoIncrement:true" json:"id"`
    CreatedAt   time.Time `gorm:"column:created_at;type:timestamp without time zone;not null" json:"created_at"`
    UpdatedAt   time.Time `gorm:"column:updated_at;type:timestamp without time zone;not null" json:"updated_at"`
    PublishedAt time.Time `gorm:"column:published_at;type:timestamp without time zone;not null" json:"published_at"`
    CategoryID  int32     `gorm:"column:category_id;type:integer;not null" json:"category_id"`
    SellerID    int32     `gorm:"column:seller_id;type:integer;not null" json:"seller_id"`
    Name        string    `gorm:"column:name;type:character varying(50);not null" json:"name"`

    Category *Category `gorm:"foreignKey:CategoryID" json:"category"`
    Seller   *Seller   `gorm:"foreignKey:SellerID" json:"seller"`
}

// TableName Product's table name
func (*Product) TableName() string { return TableNameProduct }
```

GORM

А Ваня точно тут все сам писал?

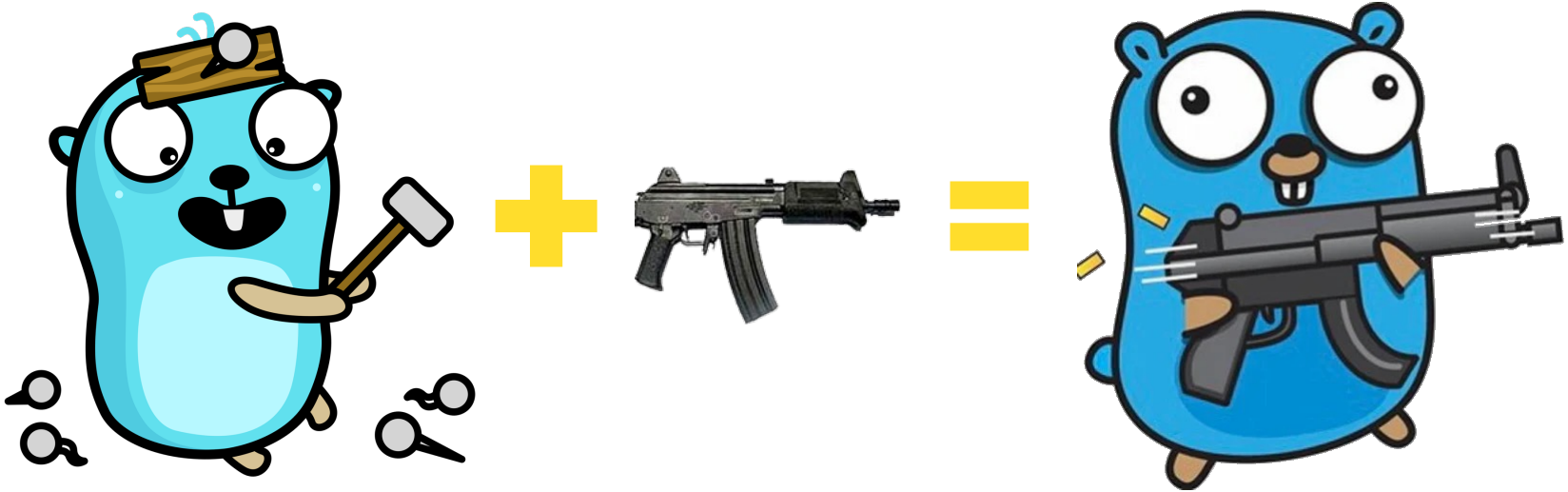
Ваня. Пишем код: сторадж

```go
package storage

import ...

type Product struct {...}

func (c *Product) Create(ctx context.Context, product *model.Product) (*model.Product, error) {
    if err := c.getDB(ctx).Create(product).Error; err != nil {
        return nil, fmt.Errorf( format: "storage.Product Create: %w", err)
    }

    return product, nil
}

func (c *Product) List(ctx context.Context, categoryId int, limit, offset int) ([]*model.Product, error) {
    var categories []*model.Product

    err := c.getDB(ctx).
        Where( query: "category_id = ? and published_at <= ?", categoryId, time.Now()).
        Order( value: "published_at DESC").
        Offset(offset).Limit(limit).
        Find(&categories).Error
    if err != nil { return nil, fmt.Errorf( format: "storage.Product List: %w", err) }

    return categories, nil
}

func (c *Product) Update(ctx context.Context, product *model.Product) (*model.Product, error) {...}

func (c *Product) Delete(ctx context.Context, product *model.Product) error {...}
```

GORM

Тут сам писал

Ваня: новый навык



GORM

А ты помнишь название доклада?

Петя. Пишем код: модель

```go
package model

import ...

const TableNameProduct = "products"  no usages  new *

type Product struct {  9 usages  new *
    ID          int32     `json:"id"`
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`
    PublishedAt time.Time `json:"published_at"`
    CategoryID  int32     `json:"category_id"`
    SellerID    int32     `json:"seller_id"`
    Name        string    `json:"name"`

    Category *Category `json:"category"`
    Seller   *Seller   `json:"seller"`
}

```

Без тегов модель выглядит проще

jackc/pgx

Петя. Пишем код: сторадж

```go
package storage

import ...

type Product struct {...}

func (c *Product) Create(ctx context.Context, product *model.Product) (*model.Product, error) {
    product.CreatedAt, product.UpdatedAt = time.Now(), time.Now()

    sql := `INSERT INTO products(created_at, updated_at, published_at, category_id, seller_id, "name")
VALUES(@created_at, @updated_at, @published_at, @category_id, @seller_id, @name) RETURNING id;`
    args := pgx.NamedArgs{
        "created_at":   product.CreatedAt,
        "updated_at":   product.UpdatedAt,
        "published_at": product.PublishedAt,
        "category_id":  product.CategoryID,
        "seller_id":    product.SellerID,
        "name":         product.Name,
    }

    if err := c.getDB(ctx).QueryRow(ctx, sql, args).Scan(&product.ID); err != nil {
        return nil, fmt.Errorf( format: "storage.Product Create: %w", err)
    }

    return product, nil
}

func (c *Product) List(ctx context.Context, categoryId int, limit, offset int) ([]*model.Product, error) {...}

func (c *Product) Update(ctx context.Context, product *model.Product) (*model.Product, error) {...}

func (c *Product) Delete(ctx context.Context, product *model.Product) error {...}
```

jackc/pgx

В экран не убирается

Петя. Пишем код: сторадж, метод списка

```go
func (c *Product) List(ctx context.Context, categoryId int, limit, offset int) ([]*model.Product, error) {
    sql := `SELECT p.* FROM products p LEFT JOIN sellers s on p.seller_id = s.id
                WHERE category_id = @category_id AND published_at <= @published_at
                ORDER BY published_at DESC LIMIT @limit OFFSET @offset;`
    args := pgx.NamedArgs{
        "category_id":  categoryId,
        "published_at": time.Now(),
        "limit":        limit,
        "offset":       offset,
    }

    rows, err := c.getDB(ctx).Query(ctx, sql, args)
    if err != nil { return nil, fmt.Errorf( format: "storage.Product List query: %w", err) }

    defer rows.Close()

    items := make([]*model.Product, 0, limit)

    for rows.Next() {
        item := new(model.Product)

        err := rows.Scan(&item.ID, &item.CreatedAt, &item.UpdatedAt, &item.PublishedAt,
            &item.CategoryID, &item.SellerID, &item.Name,
        )
        if err != nil { return nil, fmt.Errorf( format: "storage.Product List scan: %w", err) }

        items = append(items, item)
    }

    return items, nil

}
```

jackc/pgx

Еле влезло

Петя. Петя: новый навык

jackc/pgx

Кто понял причем тут пушки, не спойлерить концовку!

Запустились, пора делать фичи!

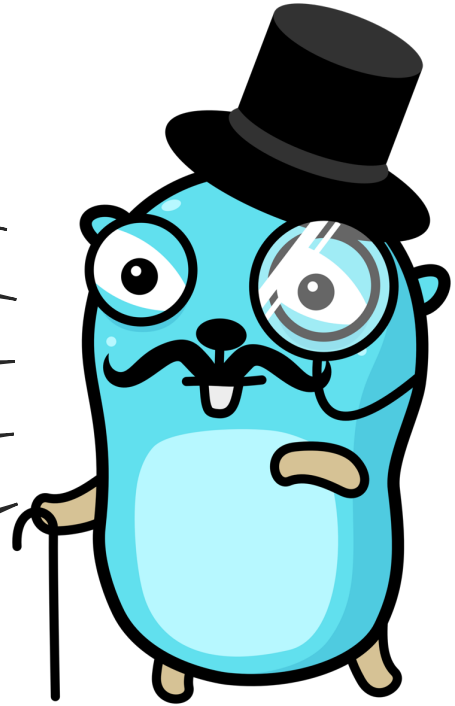
Давай сделаем вот так

Фигня какая-то, может так?

Нее опять не то - верни все как было

Я придумал давай это добавим

А помнишь первую фичу, ту что удалили? Давай вернем её



Системный аналитик бы повесился

Ваня. Новая фича: добавление поля

```go
package model

const TableNameSeller = "sellers"

// Seller mapped from table <sellers>
type Seller struct {
    ID      int32  `gorm:"column:id;type:integer;primaryKey;autoIncrement:true" json:"id"`
    Name    string `gorm:"column:name;type:character varying(50);not null" json:"name"`
    Special bool   `gorm:"column:special;type:boolean;not null" json:"special"`
}

// TableName Seller's table name
func (*Seller) TableName() string {
    return TableNameSeller
}
```

GORM

Фич: 6

И все)

Петя. Новая фича: добавление поля

```
16  func (c *Seller) Create(ctx context.Context, seller *model.Seller) (*model.Seller, error) {
17      sql := `INSERT INTO sellers("name",special) VALUES(@name, @special ) RETURNING id;`
18      args := pgx.NamedArgs{
19          "name":    seller.Name,
20          "special": seller.Special,
21      }
22
23      if err := c.getDB(ctx).QueryRow(ctx, sql, args).Scan(&seller.ID); err != nil {
24          return nil, fmt.Errorf("storage.Seller Create: %w", err)
    *Seller.List(ctx context.Context, limit int, offset int) ([]*model.Seller, error)
43      items := make([]*model.Seller, 0, limit)
44
45      for rows.Next() {
46          item := new(model.Seller)
47          if err := rows.Scan(&item.ID, &item.Name, &item.Special); err != nil {
48              return nil, fmt.Errorf("storage.Seller List scan: %w", err)
49          }
50
51          items = append(items, item)
54      return items, nil
55  }
56
57  func (c *Seller) Update(ctx context.Context, seller *model.Seller) (*model.Seller, error) {
58      sql := `UPDATE sellers SET name = @name, special = @special WHERE id = @id;`
59      args := pgx.NamedArgs{
60          "id":      seller.ID,
61          "name":    seller.Name,
62          "special": seller.Special,
63      }
```

jackc/pgx
фич: 1

Опять все в экран не лезет(

Ваня. Новая фича: вытащить связь

```
24  func (c *Product) List(ctx context.Context, categoryId int, limit, offset int) ([]*model.Product, error) {
25      var categories []*model.Product
26
27      err := c.getDB(ctx).
28          Joins("Seller").
29          Where("category_id = ? and published_at <= ?", categoryId, time.Now()).
30          Order("published_at DESC").
31          Offset(offset).Limit(limit).
32          Find(&categories).Error
```

GORM

Фич: 12

Блин, со следующим слайдом опять возиться...

Петя. Новая фича: вытащить связь

```
36  func (c *Product) List(ctx context.Context, categoryId int, limit, offset int) ([]*model.Product, error) {
37      sql := `SELECT p.*, s.* FROM products p LEFT JOIN sellers s on p.seller_id = s.id
38                  WHERE category_id = @category_id AND published_at <= @published_at
39                  ORDER BY published_at DESC LIMIT @limit OFFSET @offset;`
40      args := pgx.NamedArgs{
41          "category_id":  categoryId,

54      items := make([]*model.Product, 0, limit)
55
56      for rows.Next() {
57          item := new(model.Product)
58          item.Seller = new(model.Seller)
59
60          err := rows.Scan(&item.ID, &item.CreatedAt, &item.UpdatedAt, &item.PublishedAt,
61              &item.CategoryID, &item.SellerID, &item.Name,
62              &item.Seller.ID, &item.Seller.Name,
63          )
64          if err != nil {
65              return nil, fmt.Errorf("storage.Product List scan: %w", err)
66          }
67
68          items = append(items, item)
69      }
```

jackc/pgx

фич: 2

А не... нормально влезло

Ваня. Новая фича: связь many to many

```
22  func (c *Product) List(ctx context.Context, categoryId int, limit, offset int) ([]*model.Product, error) {
23      var categories []*model.Product
24
25      err := c.getDB(ctx).
26          Joins("Seller").
27          Preload("Tags").
28          Where("category_id = ? AND published_at <= ?", categoryId, time.Now(), time.Now()).
29          Order("published_at DESC").
30          Offset(offset).Limit(limit).
31          Find(&categories).Error
32      if err != nil {
33          return nil, fmt.Errorf("storage.Product List: %w", err)
34      }
35
36      return categories, nil
37  }
```

GORM

Фич: 18

Если связи в модели нет – добавить еще 1 строку в модель

Петя. Новая фича: связь many to many

```
    items := make([]*model.Product, 0, limit)
    productIds := make([]interface{}, 0, limit)
    productMap := make(map[int32]*model.Product, limit)

    for rows.Next() {
        item := new(model.Product)
        item.Seller = new(model.Seller)

            return nil, fmt.Errorf("storage.Product List scan: %w", err)
        }

        items = append(items, item)
        productIds = append(productIds, item.ID)
        productMap[item.ID] = item
    }

    if len(productIds) == 0 {
        return items, nil
    }

    var params string

    for i, _ := range productIds {
        params += `$` + strconv.Itoa(i+1) + `,`
    }

    params = params[:len(params)-1] // remove last ","

    sql = `SELECT  l.product_id, t.* FROM products_2_tags l INNER JOIN tags t ON l.tag_id = t.id
                WHERE l.product_id IN (` + params + `);`

    rows, err = c.getDB(ctx).Query(ctx, sql, productIds...)
    if err != nil {
        return nil, fmt.Errorf("storage.Product List tags query: %w", err)
    }

    defer rows.Close()

    for rows.Next() {
        item := new(model.Tag)
        productID := int32(0)
        err := rows.Scan(&productID, &item.ID, &item.Name)
        if err != nil {
            return nil, fmt.Errorf("storage.Product List tags scan: %w", err)
        }

        product := productMap[productID]
        product.Tags = append(product.Tags, item)
    }

    return items, nil
```

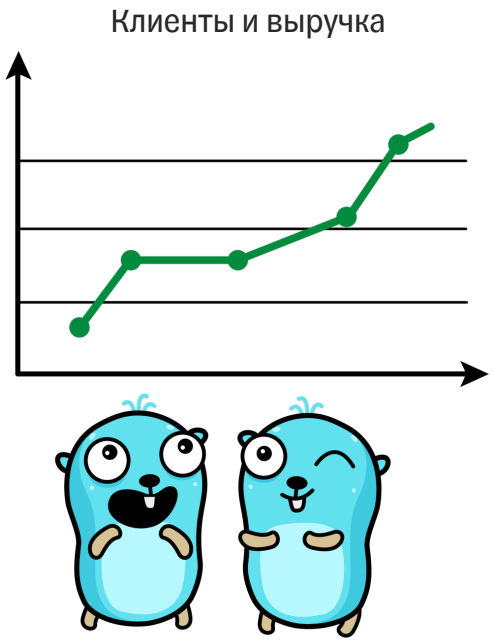
jackc/pgx
фич: 3

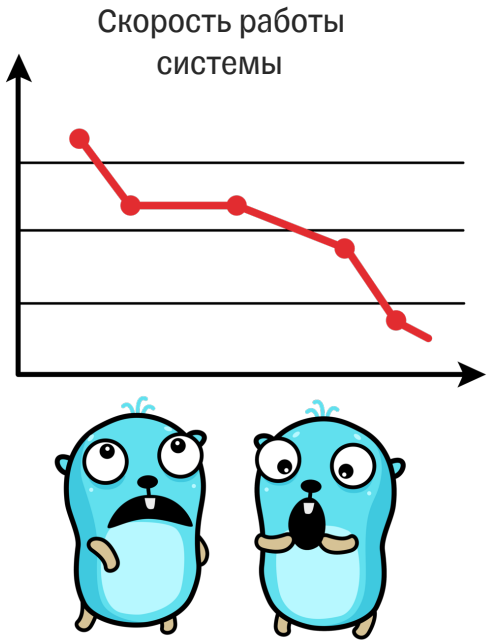
К нашему методу списка элементов добавляем:

1) Объявление массива ID и мапу с результатами
2) В цикле со сканом заполняем массив и мапу
3) Формируем запрос для связи используя массив ID
4) Исполняем и сканим результаты запроса
5) Добавляем нашу связь в результат используя мапу

С меня хватит – больше никакого кода!

Проблемы с производительностью

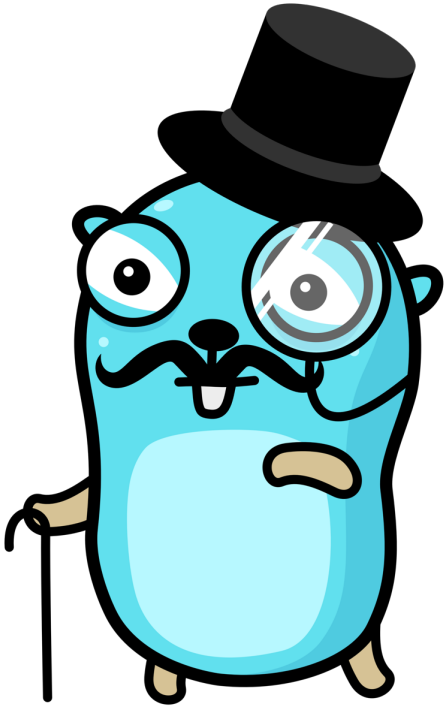




Как же хорошо: слайд в который не надо всматриваться

Проблемы с производительностью

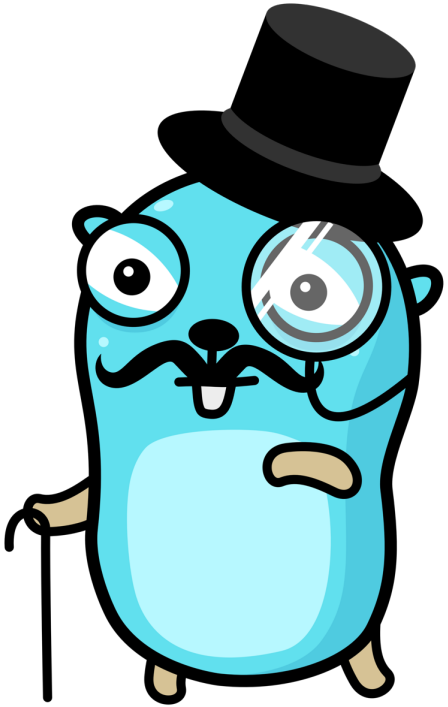
Оптимизировать код	Купить еще сервер
▪ Затраты ежемесячно: 0 р ▪ Затраты разово: 750 000 р ▪ Срок: 3 месяца ▪ Результат: +100% но это не точно	▪ Затраты ежемесячно: 50 000 р ▪ Затраты разово: 62 500 р ▪ Срок: 1 неделя ▪ Результат: +100%



Правильный ответ: 1 год и 2 месяца

Проблемы с производительностью

Оптимизировать код	Купить еще сервер
▪ Затраты ежемесячно: 0 р ▪ Затраты разово: 750 000 р ▪ Срок: 3 месяца ▪ Результат: +100% но это не точно	▪ Затраты ежемесячно: 50 000 р ▪ Затраты разово: 62 500 р ▪ Срок: 1 неделя ▪ Результат: +100%
▪ Новые фичи на паузе 3 месяца ▪ Выручка от клиентов стагнирует ▪ Теряем конкурентное преимущество	▪ Новые фичи разрабатываются ▪ Выручка от клиентов растет ▪ Наращиваем конкурентное преимущество
Потеря денег	Получение денег



Говнокодим дальше

Бабло побеждает зло

Много это ведь хорошо?

Ваня. Бабло побеждает зло

```go
package factory

import (
    "log"

    "gorm.io/driver/postgres"
    "gorm.io/gorm"
    "gorm.io/plugin/dbresolver"
)

func NewGORM(masterDSN string, replicaDSN ...string) *gorm.DB {
    cfg := &gorm.Config{
        PrepareStmt:            true,
        SkipDefaultTransaction: true,
    }
    db, err := gorm.Open(postgres.Open(masterDSN), cfg)
    if err != nil {
        log.Fatalf("gorm.Open: %s", err.Error())
    }

    replicas := make([]gorm.Dialector, 0, len(replicaDSN))

    for _, dsn := range replicaDSN {
        replicas = append(replicas, postgres.Open(dsn))
    }

    resolver := dbresolver.Register(dbresolver.Config{
        Replicas: replicas,
        Policy:   dbresolver.RandomPolicy{},
    })

    if err = db.Use(resolver); err != nil {
        log.Fatalf("пичалка: %s", err.Error())
    }

    return db
}
```

GORM

Фич: 24

Это конфиг, а не код!

Петя. Бабло побеждает зло

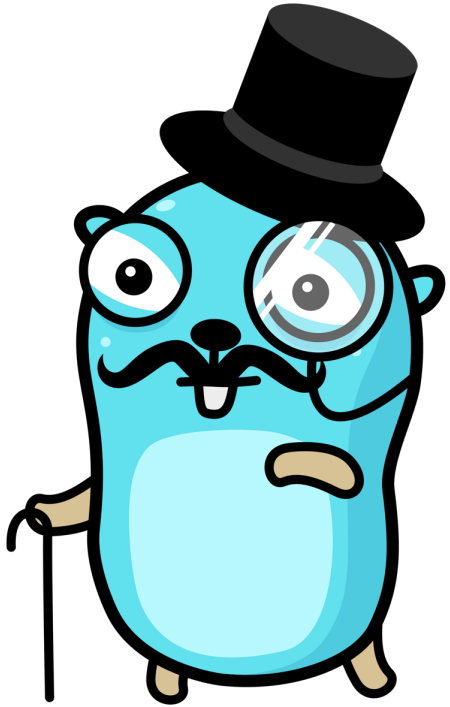
1) Ищет готовый резолвер БД
2) Отчаивается
3) Пишет свой резолвер
4) Правит все стораджи для работы с новым резолвером
5) Вспоминает что есть еще транзакции
6) Правит ошибки в резолвере

jackc/pgx
Фич: 4

Я же писал: «Больше никакого кода»

Время идет, нагрузка растет, а прибыль...

Затраты на сервера	Выручка от клиентов	Прибыль на 100 затрат
100	200	100
200	350	75
400	650	62.5
800	1 200	50
1 600	2 200	37.5
3 200	3 800	18,75
6 400	5 500	- 14



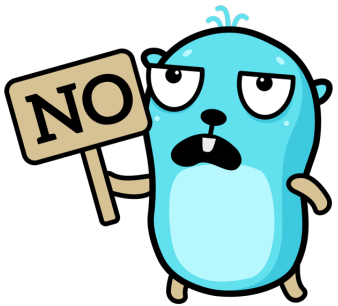
Аренда сервера + обслуживание = затраты на сервер

Начинаем оптимизацию

Опять деньги считать

Петя. Оптимизация: индексы (было)

```
CREATE TABLE "пользователи"(
    id          serial,
    добавлен    timestamp NOT NULL,
    фамилия     varchar(50) NOT NULL,
    имя         varchar(50) NOT NULL,
    отчество    varchar(50) NOT NULL,
    рожден      date NOT NULL
);
```



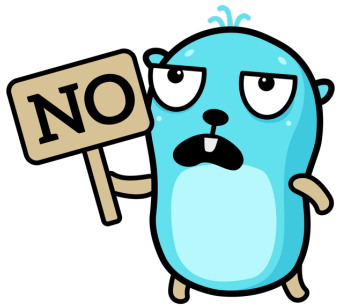

```
SELECT * FROM пользователи WHERE функция(фамилия, имя, отчество, рожден)
        ILIKE функция('Иванов', 'Иван', 'Иваныч', '2001-03-05')
            ORDER BY добавлен;
```

Это не код, а про SQL я ничего не говорил)

Петя. Оптимизация: индексы (было)

```
CREATE INDEX пользователи_джин ON пользователи USING gin(
            функция(фамилия,имя,отчество,рожден) gin_trgm_ops
        );
```

```
SELECT * FROM пользователи WHERE функция(фамилия, имя, отчество, рожден)
        ILIKE функция('Иванов', 'Иван', 'Иваныч', '2001-03-05')
            ORDER BY добавлен;
```



jackc/pgx

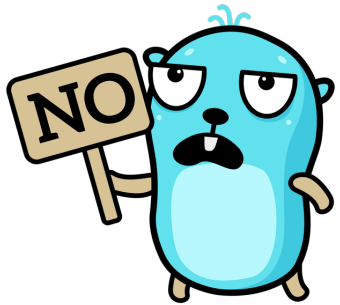
Фич: 5

Запрос исполняется 3 секунды на бою

Петя. Оптимизация: индексы (было)

```
CREATE INDEX пользователи_джин ON пользователи USING gin(
            функция(фамилия,имя,отчество,рожден) gin_trgm_ops
        );
```

```
SELECT * FROM пользователи WHERE функция(фамилия, имя, отчество, рожден)
        ILIKE функция('Иванов', 'Иван', 'Иваныч', '2001-03-05')
            ORDER BY добавлен;
```




```
Sort  (cost=4'736.96..4'739.46 rows=999 width=76)
  Sort Key: "добавлен"
  ->  Bitmap Heap Scan on "пользователи"  (cost=692.48..4'687.19 rows=999 width=76)
        Recheck Cond: (("функция"("фамилия", "имя", "отчество", "рожден"))::text ~~* 'иванов:иван:иваныч:2001-03-05'::text)
        ->  Bitmap Index Scan on "пользователи_джин"  (cost=0.00..692.23 rows=999 width=0)
              Index Cond: (("функция"("фамилия", "имя", "отчество", "рожден"))::text ~~* 'иванов:иван:иваныч:2001-03-05'::text)
```

jackc/pgx

Фич: 5

Тензор, респект вам!

Петя. Оптимизация: индексы (стало)

```
CREATE INDEX пользователи_бтри ON пользователи (
            фамилия, имя, отчество, рожден, добавлен
        );
```

```
SELECT * FROM пользователи WHERE фамилия = 'Иванов' AND имя = 'Иван'
        AND отчество = 'Иваныч' AND рожден = '2001-03-05'
    ORDER BY добавлен;
```


```
Index Scan using "пользователи_бтри" on "пользователи"  (cost=0.56..8.58 rows=1 width=76)
  Index Cond: ((("фамилия")::text = 'Иванов'::text) AND (("имя")::text = 'Иван'::text)
AND (("отчество")::text = 'Иваныч'::text) AND ("рожден" = '2001-03-05'::date))
```

jackc/pgx

Фич: 5

Запрос исполняется 0.5 секунды на бою

Ваня. Оптимизация: индексы (в процессе)

```
CREATE INDEX пользователи_хеш ON пользователи USING hash (
        функция(фамилия , имя , отчество , рожден)
        );
```

```
SELECT * FROM пользователи WHERE функция(фамилия, имя, отчество, рожден)
    = функция('Иванов', 'Иван', 'Иваныч', '2001-03-05')
      ORDER BY добавлен;
```

```
Sort  (cost=8.03..8.03 rows=1 width=76)
  Sort Key: "добавлен"
  ->  Index Scan using "пользователи_хеш" on "пользователи"  (cost=0.00..8.02 rows=1 width=76)
        Index Cond: (("функция"("фамилия", "имя", "отчество", "рожден"))::text = 'Иван...'::text)
```

GORM

Фич: 30

Даёшь эксперименты на бою!

Ваня. Оптимизация: индексы (в процессе)

```
Sort  (cost=8.03..8.03 rows=11 width=76) (actual time=12.602..36.106 rows=11 loops=1)
  Sort Key: "добавлен"
  Sort Method: quicksort  Memory: 25kB
  Buffers: shared hit=3 read=2
  I/O Timings: shared read=36.105
  ->  Index Scan using "пользователи_хеш" on "пользователи"  (cost=0.00..8.02 rows=11
width=76) (actual time=12.456..12.458 rows=11 loops=1)
        Index Cond: (("функция"("фамилия", "имя", "отчество", "рожден"))::text =
'Иванов:Иван:Иваныч:2001-03-05'::text)
        Buffers: shared hit=2 read=1
        I/O Timings: shared read=12.294
Planning:
  Buffers: shared hit=1
Planning Time: 0.641 ms
Execution Time: 36.126 ms
```

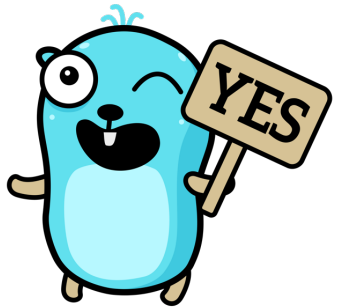
GORM

Фич: 30

А ты умеешь считать сложность алгоритма в больших системах?)

Ваня. Оптимизация: индексы (стало)

```
ALTER TABLE пользователи ADD COLUMN "хеш" uuid;

CREATE INDEX пользователи_бтри_умнее ON пользователи(хеш, добавлен);
```




```
SELECT * FROM пользователи
    WHERE хеш = 'd66fa7f4-cdb8-4a54-896d-04fabebabe0a'
        ORDER BY добавлен;
```

```
Index Scan using "пользователи_бтри_умнее" on "пользователи"  (cost=0.42..8.44 rows=11 width=66)
  Index Cond: ("хеш" = 'd66fa7f4-cdb8-4a54-896d-04fabebabe0a'::uuid)
```

GORM

Фич: 30

Сорян, нормальный эксплейн на слайд не влезает

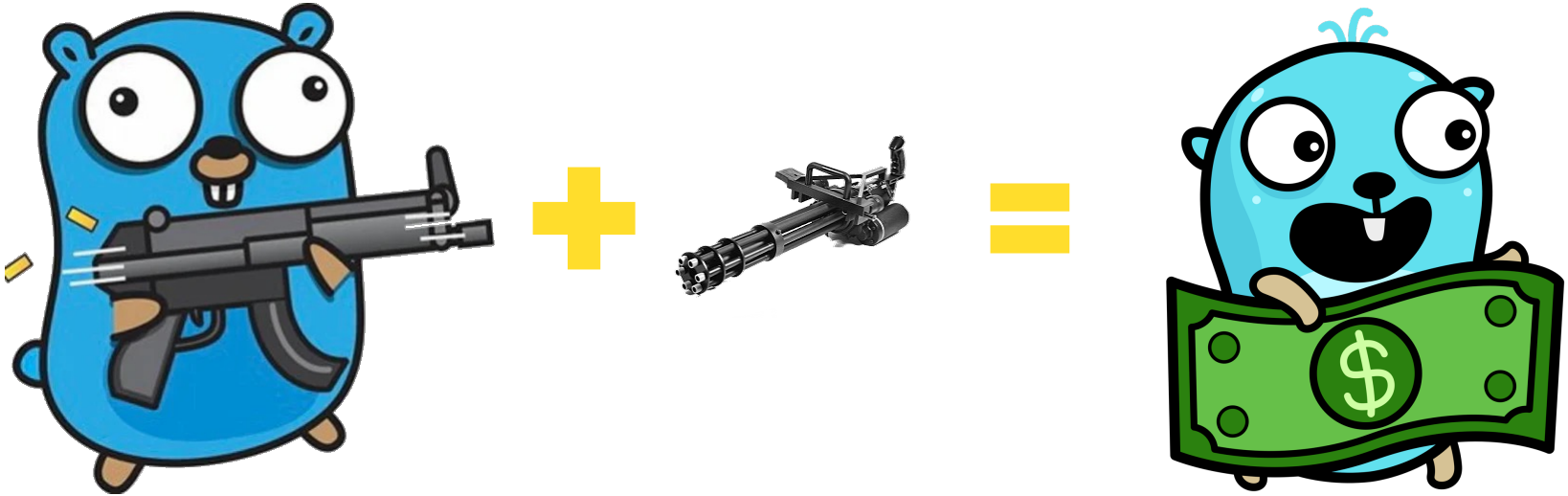
Оптимизация: индексы (закрепим)

```
SELECT
    pg_size_pretty(pg_table_size('пользователи')) таблица,
    pg_size_pretty(pg_table_size('пользователи_джин')) индекс_джин,
    pg_size_pretty(pg_table_size('пользователи_бтри'))  индекс_бтри,
    pg_size_pretty(pg_table_size('пользователи_хеш'))  индекс_хеш,
    pg_size_pretty(pg_table_size('пользователи_бтри_умнее'))  индекс_бтри_умнее
;
```

таблица	индекс_джин	индекс_бтри	индекс_хеш	индекс_бтри_умнее
1098 MB	1243 MB	800 MB	256 MB	387 MB

Теория считает, что она не отличается от практики, но у практики другое мнение

Ваня: новый навык



GORM

Фич: 30

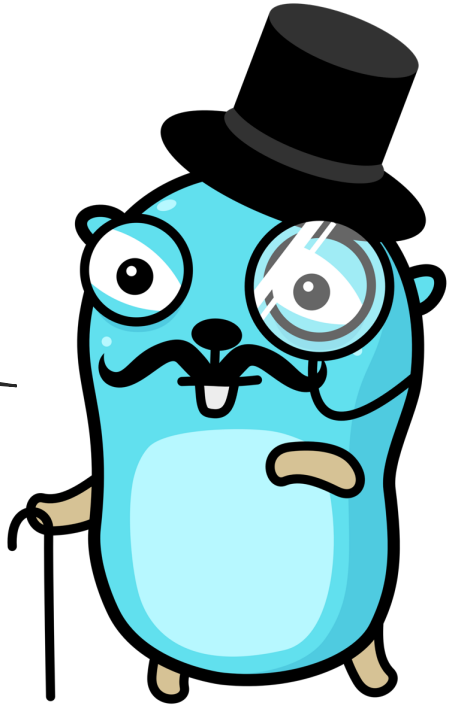
В 2х руках оружие не нарисуешь(

Ваня. Оптимизация: понимаем бизнес

Нас хостер обманывает!

Найми еще человека, БД настроить, а то хостеру не выгодно что бы у нас быстро все работало

Отлично! Я стрясу с него денег и заставлю исправить!



Ваши расходы это чья – то прибыль

Ваня. Оптимизация: мелочи

А она оказывается не плоха

```
rows, err := db.Raw(
    sql: "SELECT id, name FROM products WHERE seller_id = @seller_id",
    sql.Named( name: "seller_id", sellerID)).
    Rows()

defer rows.Close()

for rows.Next() {
    err = rows.Scan(&id, &name)
```

Да знаю я что это код, но деваться не куда

Оптимизация: итог

GORM
Фич: 30

А я в 10

В сравнении со мной, ты ничего не знаешь

Я снизил потребление ресурсов в 2 раза

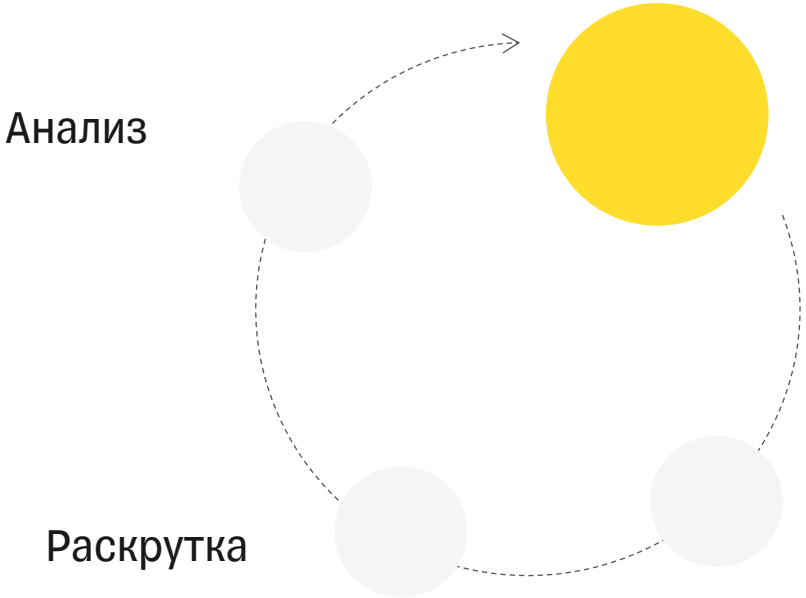
Я прокачался в БД

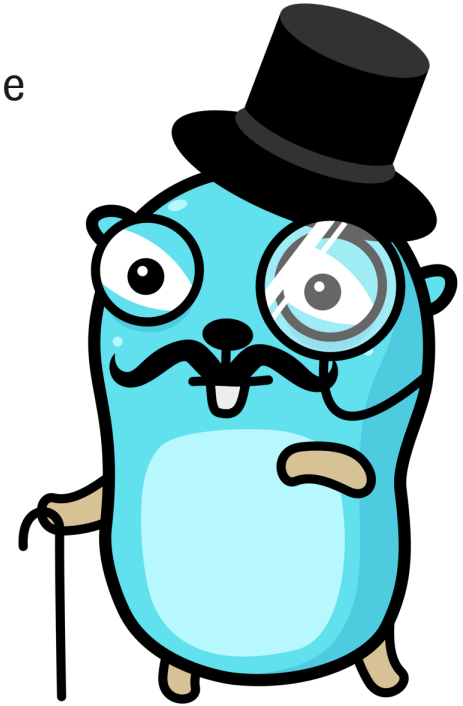
Так у тебя было в 6 раз больше времени!

jackc/pgx
Фич: 5

А ты обратил внимание когда у Пети покраснели глаза? Нет не сейчас)

Ваня. Откуда 6 месяцев на оптимизацию





И что Ванин проект имел круг в 6 месяцев?

Ваня. Откуда 6 месяцев на оптимизацию

Реализовал фичу	Оптимизируй!
Опт. архитектуру	Раскручивал
Опт. базу данных	Анализировал
Опт. код	Ждал результата
Готово	
	Планирование

GORM

Фич: 30

Зачем наняли ИБшника? Что-то будет?

Ваня. Откуда 6 месяцев на оптимизацию

Реализовал фичу

Опт. архитектуру

Опт. базу данных

Опт. код

Готово

Оптимизируй!

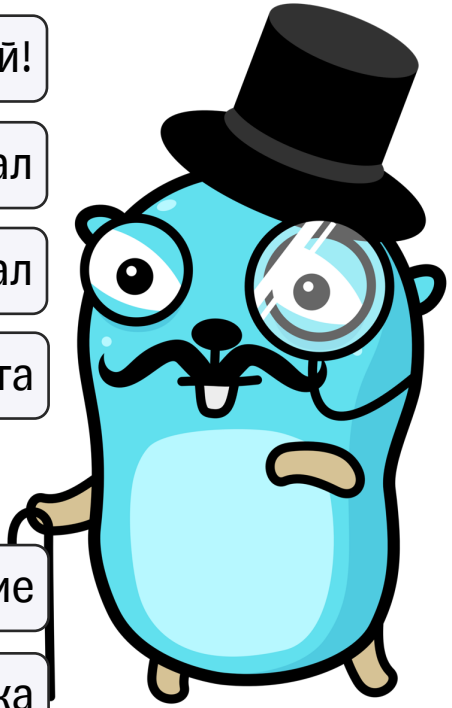
Раскручивал

Анализировал

Ждал результата

Планирование

Раскрутка



GORM

Фич: 30

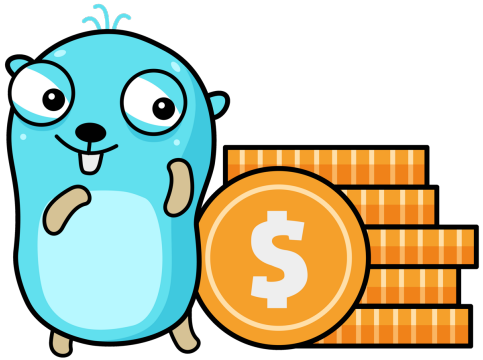
Аааа вот оно что!

Петя. Финал

jackc/pgx
Фич: 10

Жизнь боль, или нет?

Ваня. Финал

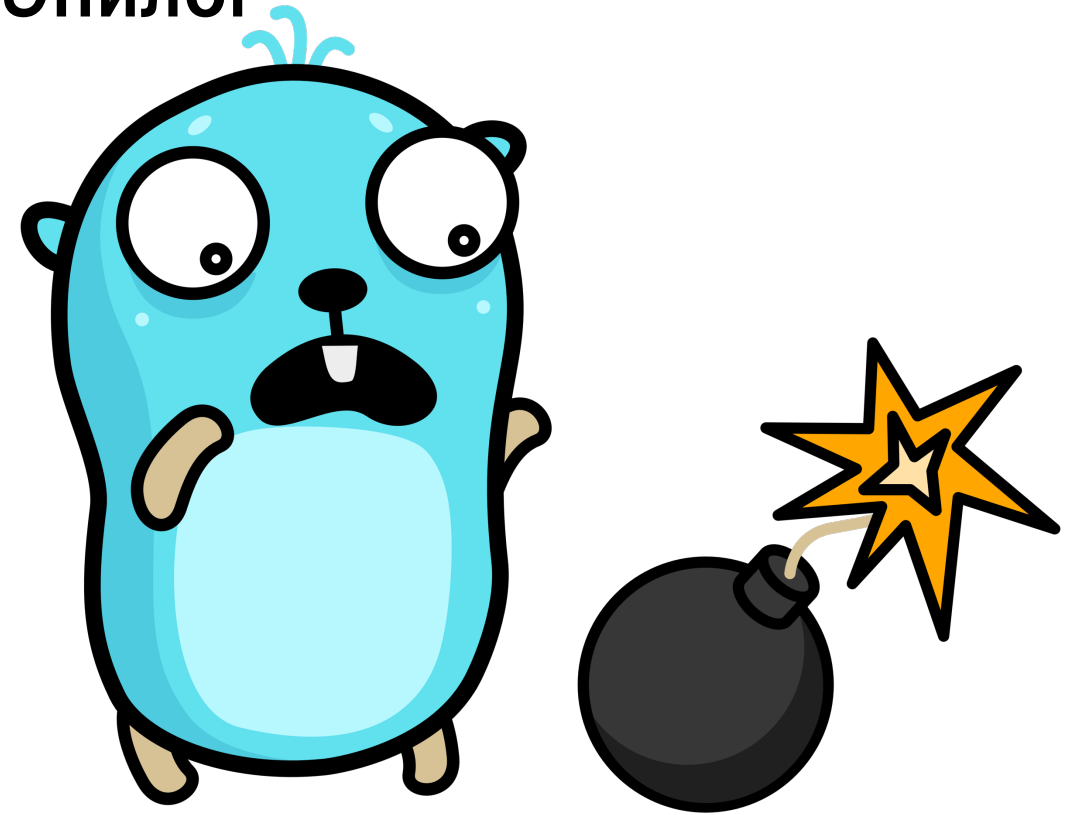


GORM

Фич: 30

Вот теперь можешь в чатике всем говорить про фильм

Ваня. Эпилог



GORM

Фич: 30

Это был просто охренительный доклад!

Конец

Вань, если ты ошибся в выборе решения - не переживай, если ошибся не сильно - простят. Если ошибся на пару миллиардов, то конечно дело другое:

Обновляй резюме, расписывай в нем какие охренительные ты выбрал технологии и инструменты. Как привел свой стартап к вершинам, ~~как потом накосячил и обанкротил компанию~~ получил опыт работы с высоконагруженными системами.
А сейчас тебе перестала нравится старая компания и ты решил сменить место работы, что бы получить новый опыт.

Повторять пока не станешь архитектором.

Или иди работать в Тинькофф – такую махину просто так не обанкротишь)