

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ
по лабораторной работе №3
по дисциплине «Базы данных»
Тема: Реализация базы данных с использованием ORM

Студент гр. 1303

Беззубов Д.В.

Преподаватель

Заславский М.М.

Санкт-Петербург

2023

Цель работы.

Создание базы данных с использованием Object-Relational Mapping (ORM).

Задание.

Вариант 2.

В данной лабораторной работе рекомендуется использовать Sequelize (Node.js).

Вы можете использовать другой ORM по вашему выбору по согласованию с преподавателем, принимающим у вас практики.

Необходимо выполнить следующие задачи:

- Описать в виде моделей таблицы из 1-й лабораторной работы.
- Написать скрипт заполнения тестовыми данными: 5-10 строк на каждую таблицу, обязательно наличие связи между ними, данные приближены к реальности.
- Написать запросы к БД, отвечающие на вопросы из 1-й лабораторной работы с использованием **ORM**. Вывести результаты в консоль (или иной человеко-читабельный вывод).
- Запустить в репозиторий исходный код проекта, соблюсти. gitignore, убрать исходную базу из проекта (или иные нагенерированные данные бд если они есть).
- Описать процесс запуска: команды, зависимости.
- В отчете описать цель, текст задания в соответствии с вариантом, выбранную ORM, инструкцию по запуску, скриншоты (код) моделей ORM, скриншоты на каждый запрос (или группу запросов) на изменение/таблицы с выводом результатов (ответ), ссылку на PR в приложении, вывод.

Выполнение работы.

В качестве *ORM* была выбрана *GORM* для языка *Go*.

1. Установка

Для установки *GORM* и драйвера для *postgres* используем следующие команды:

```
go get -u gorm.io/gorm
go get -u gorm.io/driver/postgres
```

2. Подключение к базе данных

Подключение к базе данных PostgreSQL.

```
dsn := "host=localhost user=postgres password=1 dbname=postgres
port=5432"
```

В этой строке определена строка подключения (*Data Source Name, DSN*) для базы данных *PostgreSQL*. *DSN* содержит информацию о том, как подключиться к базе данных, включая хост (*localhost*), имя пользователя (*postgres*), пароль (*1*), имя базы данных (*postgres*) и порт (*5432*).

```
db, err := gorm.Open(postgres.Open(dsn), &gorm.Config{})
```

В этой строке выполняется попытка подключения к базе данных с использованием *GORM* и драйвера *PostgreSQL*. Функция *gorm.Open* принимает два аргумента. Первый аргумент *postgres.Open(dsn)* указывает *GORM* использовать драйвер *PostgreSQL* и передает *DSN* для подключения к базе данных. Второй аргумент *&gorm.Config{}* представляет конфигурацию *GORM* (в данном случае, конфигурация не определена, и используются значения по умолчанию). Результат этой операции, то есть подключенная база данных, сохраняется в переменной *db*, и любая ошибка сохраняется в переменной *err*.

3. Создание моделей

На основе структуры базы данных, спроектированной в лабораторной работе 1, были созданы соответствующие модели.

Модели представляют собой обычные структуры с базовыми типами *Go*, их указателями или пользовательскими типами.

Основная структура модели:

- Название структуры – название модели.
- Столбцы содержат: название поля, тип данных, теги *GORM*.

Используемые типы данных в рамках лабораторной работы:

- *uint* – беззнаковое целое число.

- *string* – строка.
- *int* – целое число.

Используемые теги *GORM*:

- *primaryKey* – указывает столбец в качестве первичного ключа.
- *autoIncrement:false* или *autoIncrement:true* – запрещает или задает автоматический инкрементный столбец.

- *size* – размер столбца.
- *not null* – задает столбцу значение *NOT NULL*.
- *foreignKey* – указывает столбец в качестве внешнего ключа.
- *default:null* – указывает значение столбца по умолчанию.

На рисунках 1 – 6 представлены описания каждой из моделей.

```
type Author struct { 3 usages
    gorm.Model
    Name      string `json:"name" gorm:"size:30"`
    Surname   string `json:"surname" gorm:"size:30"`
    Books     []*Book `gorm:"many2many:AuthorBook;"`
}
```

Рисунок 1 – описание модели *Author*.

```
type Book struct { 9 usages
    gorm.Model
    Title      string `json:"title"`
    Year       string `json:"year"`
    PublisherID uint   `json:"publisherID" gorm:"not null;on delete:cascade"`
    Publisher  Publisher `json:"publisher" gorm:"foreignkey:PublisherID"`
    Authors    []*Author `gorm:"many2many:AuthorBook;"`
}
```

Рисунок 2 – описание модели *Book*.

```

type BookAtHall struct { 3 usages
    LibraryHallID int    `json:"hall_id" gorm:"primaryKey;autoIncrement:false"`
    BookID         int    `json:"book_id" gorm:"primaryKey;autoIncrement:false"`
    VisitorID      int    `json:"visitor_id" gorm:"default:null"`
    Code           int    `json:"code"`
    BookedAt       time.Time `json:"booked_at" gorm:"default:null"`
    DeletedAt      time.Time `json:"deleted_at" gorm:"default:null"`

    LibraryHall LibraryHall `json:"- "`
    Book         Book         `json:"- "`
    Visitor      *Visitor    `json:"- "`
}

```

Рисунок 3 – описание модели *BookAtHall*.

```

type LibraryHall struct { 3 usages
    gorm.Model
    Name      string `json:"name" gorm:"size:30"`
    Capacity  int    `json:"capacity"`
    Visitors []*Visitor
}

```

Рисунок 4 – описание модели *LibraryHall*.

```

type Publisher struct { 3 usages
    gorm.Model
    Name string `json:"name" gorm:"size:50"`
}

```

Рисунок 5 – описание модели *Publisher*.

```

type Visitor struct { 4 usages
    gorm.Model
    ReaderTicket int    `json:"reader_ticket"`
    Surname       string `json:"surname" gorm:"size:30"`
    Passport      string `json:"passport" gorm:"size:11"`
    Birthday      time.Time `json:"birthday" gorm:"default:null"`
    Address       string `json:"address" gorm:"size:50"`
    Phone         string `json:"phone" gorm:"size:11"`
    EducationalStage string `json:"educational_stage" gorm:"size:30"`
    AcademicDegree bool   `json:"academic_degree"`
    LibraryHallID uint   `gorm:"not null"`
}

```

Рисунок 6 – описание модели *Visitor*.

4. Создание таблицы.

```
err = db.AutoMigrate(
```

```

        &models.Author{},
        &models.Book{},
        &models.BookAtHall{},
        &models.LibraryHall{},
        &models.Publisher{},
        &models.Visitor{},
    )

```

В предоставленном коде используется функция *AutoMigrate* из библиотеки *GORM* для автоматического создания (или обновления) таблиц в базе данных, которые соответствуют структурам данных, перечисленным в качестве аргументов функции. Эта функция создает таблицы, если их еще нет, или обновляет их, если они уже существуют, чтобы они соответствовали описанным структурам данных. Функция *AutoMigrate* анализирует структуры данных и создает таблицы в базе данных с соответствующими полями и ограничениями, как они определены в структурах.

После запуска программы в *IDE DataGrip* можно отследить создание таблиц и соответствующих полей. На рисунках 7 – 13 представлены созданные таблицы.

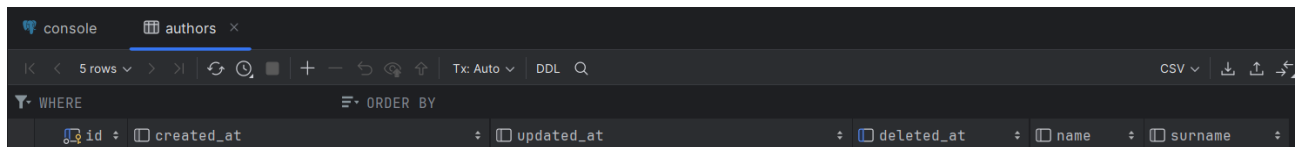


Рисунок 7 – таблица *authors*.

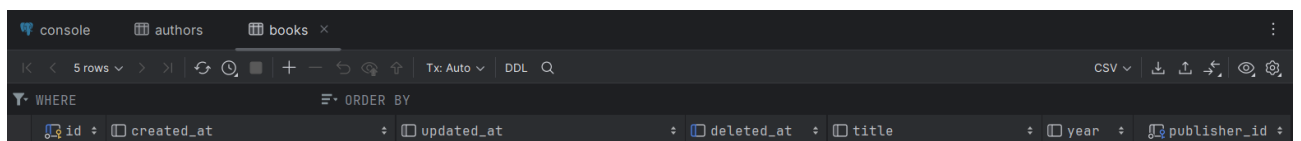


Рисунок 8 – таблица *books*.

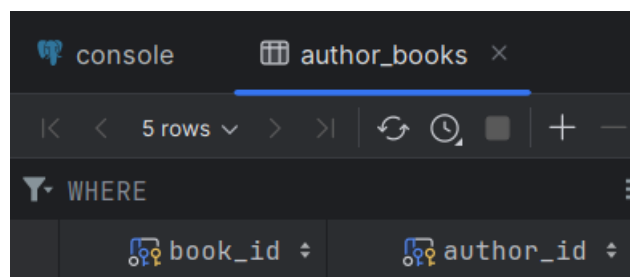


Рисунок 9 – таблица *author_books*.

The screenshot shows a database console interface with the 'library_halls' table selected. The table schema is as follows:

id	created_at	updated_at	deleted_at	name	capacity
----	------------	------------	------------	------	----------

Рисунок 10 – таблица *library_halls*.

The screenshot shows a database console interface with the 'book_at_halls' table selected. The table schema is as follows:

library_hall_id	book_id	visitor_id	code	booked_at	deleted_at
-----------------	---------	------------	------	-----------	------------

Рисунок 11 – таблица *book_at_halls*.

The screenshot shows a database console interface with the 'publishers' table selected. The table schema is as follows:

id	created_at	updated_at	deleted_at	name
----	------------	------------	------------	------

Рисунок 12 – таблица *publishers*.

The screenshot shows a database console interface with the 'visitors' table selected. The table schema is as follows:

id	reader_ticket	surname	passport	address	phone	birthday	educational_stage
----	---------------	---------	----------	---------	-------	----------	-------------------

Рисунок 16 – таблица *visitors*.

5. Добавление записей.

Создаем слайсы с объектами, которые необходимо внести в БД, часть связанных таблиц заполняются автоматически на основе указанных связей.

На рисунках 16 – 21 представлены такие переменные с тестовыми данными.

```
publishers := []models.Publisher{
    {Name: "Питер"},
    {Name: "Эксмо"},
    {Name: "АСТ"},
    {Name: "Манн, Иванов и Фербер"},
    {Name: "Центрполиграф"},
}
```

Рисунок 16 – тестовые данные для *Publisher*.

```
books := []models.Book{
    {Title: "Война и мир", Year: "1869", PublisherID: 1},
    {Title: "Преступление и наказание", Year: "1866", PublisherID: 1},
    {Title: "Идиот", Year: "1869", PublisherID: 2},
    {Title: "1984", Year: "1949", PublisherID: 4},
    {Title: "Анна Каренина", Year: "1877", PublisherID: 5},
}
```

Рисунок 17 – тестовые данные для *Book*.

```
authors := []models.Author{
    {Name: "Федор", Surname: "Достоевский", Books: []*models.Book{&books[1], &books[2]}},
    {Name: "Лев", Surname: "Толстой", Books: []*models.Book{&books[0], &books[4]}},
    {Name: "Джордж", Surname: "Оруэлл", Books: []*models.Book{&books[3]}},
    {Name: "Агата", Surname: "Кристи"},
    {Name: "Джоан", Surname: "Роулинг"},
}
```

Рисунок 18 – тестовые данные для *Author*.

```
visitors := []models.Visitor{
    {
        ReaderTicket: 1001,
        Surname: "Иванов",
        Passport: "1234 678901",
        Birthday: time.Date(year: 1990, month: 5, day: 15, hour: 0, min: 0, sec: 0, nsec: 0, time.UTC),
        Address: "ул. Ленина, 123",
        Phone: "89205678901",
        EducationalStage: "Высшее",
        AcademicDegree: true,
        LibraryHallID: 1,
    },
    {
        ReaderTicket: 1002,
        Surname: "Петров",
    },
}
```

Рисунок 19 – тестовые данные для *Visitor*.

```
booksAtHall := []models.BookAtHall{
    {
        LibraryHallID: 1,
        BookID: 1,
        VisitorID: 1,
        Code: 1123,
        BookedAt: time.Date(year: 2023, month: 8, day: 8, hour: 0, min: 0, sec: 0, nsec: 0, time.UTC),
    },
    {LibraryHallID: 1, BookID: 2, Code: 1124},
    {
        LibraryHallID: 1,
    },
}
```

Рисунок 20 – тестовые данные для *BookAtHall*.

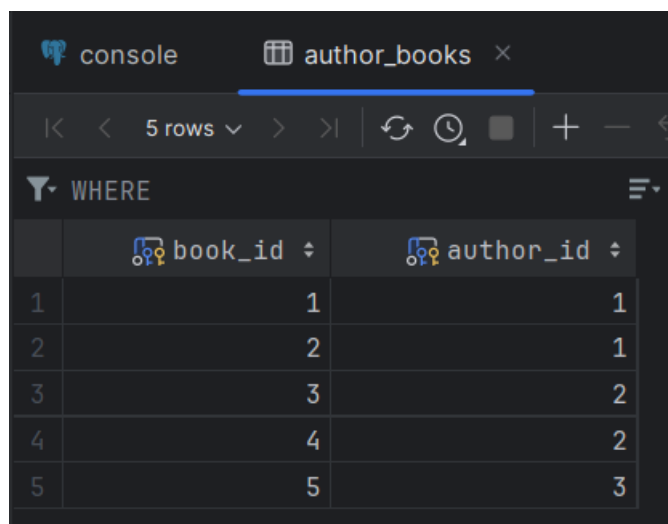

```
libraryHalls := []models.LibraryHall{
    {Name: "Классика", Capacity: 25},
    {Name: "Зарубежная", Capacity: 20},
    {Name: "Фэнтези", Capacity: 25},
}
```

Рисунок 21 – тестовые данные для *LibraryHall*

```
createRecords := func(data interface{}) {
    result := db.Create(data)
    if result.Error != nil {
        fmt.Print("Error during adding tuple")
    }
}
```

Данная функция принимает пустой интерфейс, что позволяет ей принимать на вход любой объект. Эта функция использует переданные данные для создания записей в базе данных с помощью метода *Create* объекта *db*, который является экземпляром *GORM* для взаимодействия с базой данных. Если при выполнении *Create* возникает ошибка, она логируется. Далее эта функция вызывается от переменных, которые содержат необходимые данные для добавления.

На рисунках 22 – 28 изображены итоговые таблицы вместе с данными.



The screenshot shows a database console window with the 'author_books' table selected. The table has two columns: 'book_id' and 'author_id'. The data is as follows:

	book_id	author_id
1	1	1
2	2	1
3	3	2
4	4	2
5	5	3

Рисунок 22 – таблица *author_books* с данными.

	id	created_at	updated_at	deleted_at	name	surname
1	1	2023-11-01 01:17:50.982541 +00:00	2023-11-01 01:17:50.982541 +00:00	<null>	Федор	Достоевский
2	2	2023-11-01 01:17:50.982541 +00:00	2023-11-01 01:17:50.982541 +00:00	<null>	Лев	Толстой
3	3	2023-11-01 01:17:50.982541 +00:00	2023-11-01 01:17:50.982541 +00:00	<null>	Джордж	Оруэлл
4	4	2023-11-01 01:17:50.982541 +00:00	2023-11-01 01:17:50.982541 +00:00	<null>	Агата	Кристи
5	5	2023-11-01 01:17:50.982541 +00:00	2023-11-01 01:17:50.982541 +00:00	<null>	Джоан	Роулинг

Рисунок 23 – таблица *authors* с данными.

	library_hall_id	book_id	visitor_id	code	booked_at	deleted_at
1	1	1	1	1123	2023-08-08 00:00:00.000000 +00:00	<null>
2	1	2	<null>	1124	<null>	<null>
3	1	3	2	1125	2023-10-15 00:00:00.000000 +00:00	<null>
4	2	5	<null>	2123	<null>	<null>
5	1	4	<null>	1126	<null>	<null>

Рисунок 24 – таблица *book_at_halls* с данными.

	id	created_at	updated_at	deleted_at	title	year	publisher_id
1	1	2023-11-01 01:17:50.984069 +00:00	2023-11-01 01:17:50.984069 +00:00	<null>	Преступление и наказание	1866	1
2	2	2023-11-01 01:17:50.984069 +00:00	2023-11-01 01:17:50.984069 +00:00	<null>	Идиот	1869	2
3	3	2023-11-01 01:17:50.984069 +00:00	2023-11-01 01:17:50.984069 +00:00	<null>	Война и мир	1869	1
4	4	2023-11-01 01:17:50.984069 +00:00	2023-11-01 01:17:50.984069 +00:00	<null>	Анна Каренина	1877	5
5	5	2023-11-01 01:17:50.984069 +00:00	2023-11-01 01:17:50.984069 +00:00	<null>	1984	1949	4

Рисунок 25 – таблица *books* с данными.

	id	created_at	updated_at	deleted_at	name	capacity
1	1	2023-11-01 01:17:50.988400 +00:00	2023-11-01 01:17:50.988400 +00:00	<null>	Классика	25
2	2	2023-11-01 01:17:50.988400 +00:00	2023-11-01 01:17:50.988400 +00:00	<null>	Зарубежная	20
3	3	2023-11-01 01:17:50.988400 +00:00	2023-11-01 01:17:50.988400 +00:00	<null>	Фантази	25

Рисунок 26 – таблица *library_halls* с данными.

	id	created_at	updated_at	deleted_at	reader_ticket	surname	passport	birthdate
1	1	2023-11-01 01:17:50.990453 +00:00	2023-11-01 01:17:50.990453 +00:00	<null>	1001	Иванов	1234 678901	1990-05-10
2	2	2023-11-01 01:17:50.990453 +00:00	2023-11-01 01:17:50.990453 +00:00	<null>	1002	Петров	2345 789012	1985-12-15
3	3	2023-11-01 01:17:50.990453 +00:00	2023-11-01 01:17:50.990453 +00:00	<null>	1003	Сидорова	3456 890123	1995-03-20
4	4	2023-11-01 01:17:50.990453 +00:00	2023-11-01 01:17:50.990453 +00:00	<null>	1004	Козлов	4567 901234	1982-08-05
5	5	2023-11-01 01:17:50.990453 +00:00	2023-11-01 01:17:50.990453 +00:00	<null>	1005	Михайлова	5678 012345	1998-07-18

Рисунок 27 – таблица *visitors* с данными.

	id	created_at	updated_at	deleted_at	name
1	1	2023-11-01 01:17:50.980543 +00:00	2023-11-01 01:17:50.980543 +00:00	<null>	Питер
2	2	2023-11-01 01:17:50.980543 +00:00	2023-11-01 01:17:50.980543 +00:00	<null>	Эксмо
3	3	2023-11-01 01:17:50.980543 +00:00	2023-11-01 01:17:50.980543 +00:00	<null>	АСТ
4	4	2023-11-01 01:17:50.980543 +00:00	2023-11-01 01:17:50.980543 +00:00	<null>	Манн, Иванов и Фербер
5	5	2023-11-01 01:17:50.980543 +00:00	2023-11-01 01:17:50.980543 +00:00	<null>	Центрполиграф

Рисунок 28 – таблица *publishers* с данными.

6. Написание запросов к БД, отвечающих на вопросы из первой лабораторной работы.

Для каждого запроса была написана своя функция, которая принимает в качестве параметров указатель на объект *GORM*, который представляет собой соединение с базой данных и данные, по которым необходимо сделать выборку.

Методы *GORM*, используемые для реализации запросов:

- *db.Table("...")*: Этот метод *GORM* устанавливает таблицу как источник данных для запроса.
- *db.Select(...)*: Здесь указываются столбцы, которые должны быть выбраны в результате запроса.
- *db.Joins(«INNER JOIN таблица ON по каким полям»)*: Этот метод *GORM* выполняет объединение (*join*) таблиц с использованием *PostgreSQL INNER JOIN*. Он соединяет записи в обеих таблицах, где соотносятся указываемые значения полей.
- *db.Where(...)*: Этот метод *GORM* добавляет условия для выборки данных.
- *db.Count(...)*: Этот метод *GORM* выполняет запрос к базе данных и выполняет подсчет количества записей, удовлетворяющих условиям.
- *db.Find(&result)*: Этот метод *GORM* выполняет запрос к базе данных, который соответствует условиям и полученные записи сохраняются в переменную *result*, которая представляет собой срез (список) структур определенный в зависимости от функции.

Реализованные запросы представлены на рисунках 29-36

```
func GetBooksByReaderTicket(db *gorm.DB, readerTicket int) (result []models.Book) { 1 usage
    db.Joins( query: "INNER JOIN book_at_halls ON books.id = book_at_halls.book_id").
        Joins( query: "INNER JOIN visitors ON visitors.id = book_at_halls.visitor_id").
        Where( query: "visitors.reader_ticket = ?", readerTicket).
        Find(&result)
    return
}
```

Рисунок 29 – функция с запросом по вопросу 1.

```
func GetTitleByCode(db *gorm.DB, code int) (result []models.Book) { 1 usage
    db.Joins( query: "INNER JOIN book_at_halls ON books.id = book_at_halls.book_id").
        Where( query: "code = ?", code).
        Find(&result)
    return
}
```

Рисунок 30 – функция с запросом по вопросу 2.

```
func GetCodeByTitle(db *gorm.DB, title string) (result []models.BookAtHall) { 1 usage
    db.Select( query: "code").
        Joins( query: "INNER JOIN books ON books.id = book_at_halls.book_id").
        Where( query: "title = ?", title).
        Find(&result)
    return
}
```

Рисунок 31 – функция с запросом по вопросу 3

```
type Query4 struct { 1 usage
    Title      string      `json:"title"`
    BookedAt   time.Time   `json:"booked_at" gorm:"default:null"`
}

func GetBookedBooks(db *gorm.DB) (result []Query4) { 1 usage
    db.Table( name: "book_at_halls").
        Joins( query: "INNER JOIN books ON books.id = book_at_halls.book_id").
        Where( query: "book_at_halls.booked_at IS NOT NULL").
        Order( value: "title").
        Select( query: "books.title,book_at_halls.booked_at").
        Find(&result)
    return
}
```

Рисунок 32 – структура результата и функция с запросом по вопросу 4

```

type Query5 struct { 2 usages
    Surname      string `json:"surname"`
    ReaderTicket int    `json:"reader_ticket"`
    Title        string `json:"title"`
}

func GetVisitorsWithBooksOverMonth(db *gorm.DB) (result []Query5) { 1 usage
    db.Table(name: "visitors").
        Select(query: "visitors.surname, visitors.reader_ticket, books.title").
        Joins(query: "INNER JOIN book_at_halls ON visitors.id = book_at_halls.visitor_id").
        Joins(query: "INNER JOIN books ON books.id = book_at_halls.book_id").
        Where(query: "current_date - book_at_halls.booked_at > interval '1 month'").
        Find(&result)
    return
}

```

Рисунок 33 - структура результата и функция с запросом по вопросу 5

```

func GetVisitorsBooksLessTwo(db *gorm.DB) (result []Query5) { 1 usage
    subQuery := db.Table(name: "book_at_halls").
        Select(query: "title").
        Joins(query: "INNER JOIN books ON book_at_halls.book_id = books.id").
        Group(name: "title").
        Having(query: "count(title) < 2")

    db.Table(name: "visitors").
        Select(query: "visitors.reader_ticket, visitors.surname, books.title").
        Joins(query: "INNER JOIN book_at_halls ON visitors.id = book_at_halls.visitor_id").
        Joins(query: "INNER JOIN books ON book_at_halls.book_id = books.id").
        Where(query: "books.title IN (?)", subQuery).
        Scan(&result)
    return
}

```

Рисунок 34 – функция с запросом по вопросу 6

```

func GetVisitorsCount(db *gorm.DB) (result int64) { 1 usage
    db.Table(name: "visitors").Count(&result)
    return
}

```

Рисунок 35 – функция с запросом по вопросу 7

```
func GetYoungVisitors(db *gorm.DB) (result int64) { 1 usage
    db.Table( name: "visitors").
        Where( query: "current_date - visitors.birthdate < interval '20 year'").
        Count(&result)
    return
}
```

Рисунок 36 – функция с запросом по вопросу 8

```
Query 1 results:
Преступление и наказание
Query 2 results:
1984
Query 3 results:
1125
Query 4 results:
title: Война и мир, booked_at: 2023-10-15
title: Преступление и наказание, booked_at: 2023-08-08
Query 5 results:
surname: Иванов, reader ticket: 1001, title: Преступление и наказание
Query 6 results:
surname: Иванов, reader ticket: 1001, title: Преступление и наказание
surname: Петров, reader ticket: 1002, title: Война и мир
Query 7 result:
Count: 5
Query 8 result:
Count: 0
```

Рисунок 37 – результат выполнения данных запросов

В приложении А предоставлена ссылка на PR.

Выводы.

В данной лабораторной работе освоена работа с *ORM* для *Go* – *GORM*.

Описаны в виде моделей *GORM* таблицы из 1-й лабораторной работы.

Написана функция, заполняющая все таблицы тестовыми данными.

Написаны запросы к БД, отвечающие на вопросы из 1-й лабораторной работы с использованием *ORM*.

ПРИЛОЖЕНИЕ А

ССЫЛКИ

Ссылка на PR:

<https://github.com/moevm/sql-2023-1303/pull/38>