**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.

Cоздание 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).
* Запушить 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

1. 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.*

1. 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.

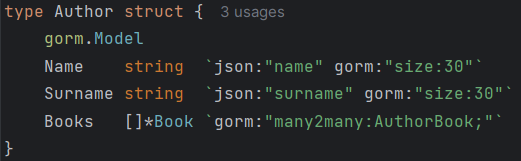


Рисунок 1 – описание модели *Author*.



Рисунок 2 – описание модели *Book*.



Рисунок 3 – описание модели *BookAtHall*.



Рисунок 4 – описание модели *LibraryHall*.

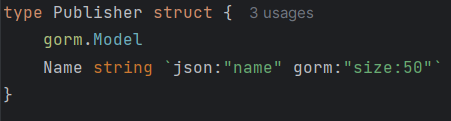


Рисунок 5 – описание модели *Publisher*.

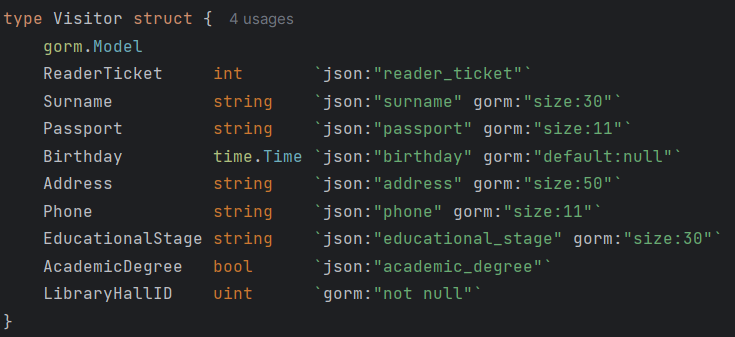


Рисунок 6 – описание модели *Visitor*.

1. 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.

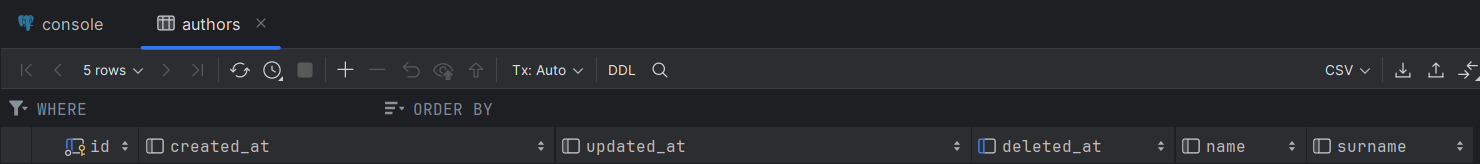


Рисунок 7 – таблица *authors.*

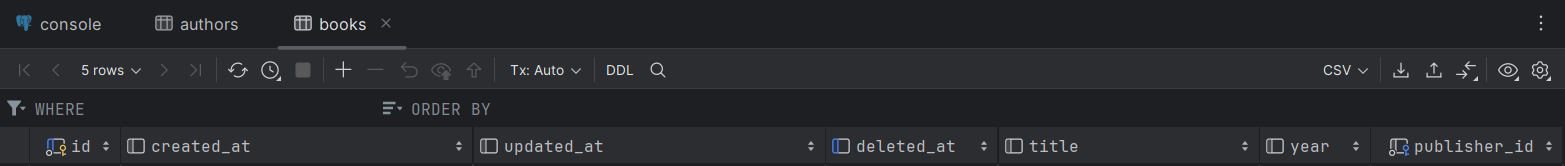


Рисунок 8 – таблица *books.*

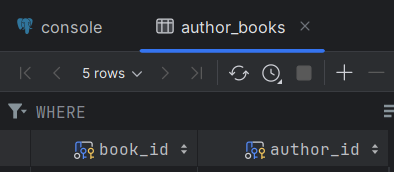


Рисунок 9 – таблица *author\_books*.

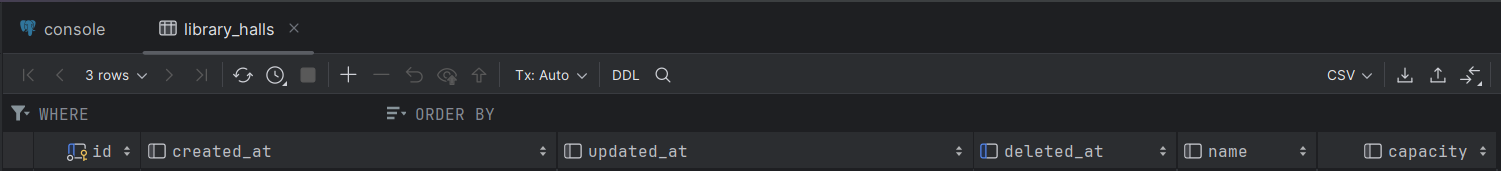


Рисунок 10 – таблица *library\_halls.*

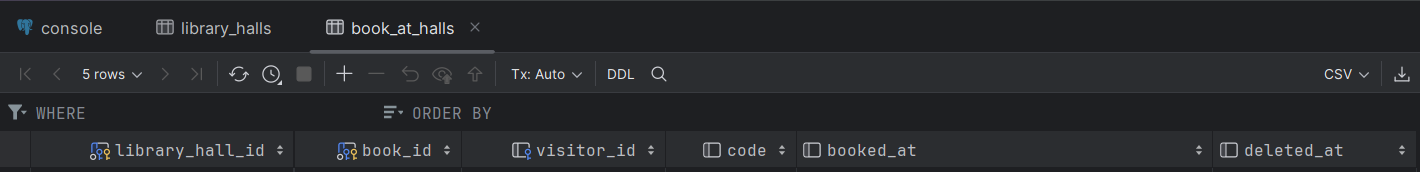


Рисунок 11 – таблица *book\_at\_halls*.

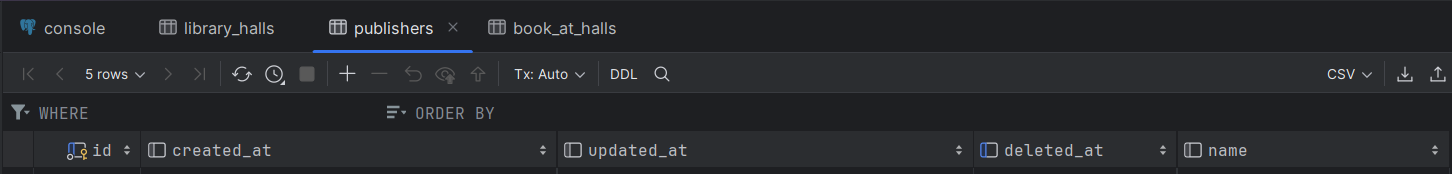


Рисунок 12 – таблица *publishers.*

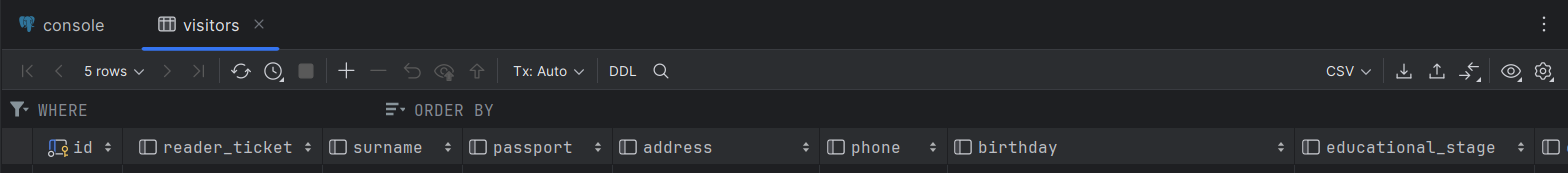


Рисунок 16 – таблица *visitors.*

1. 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.

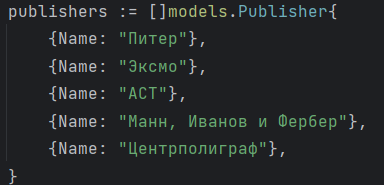


Рисунок 16 – тестовые данные для *Publisher.*

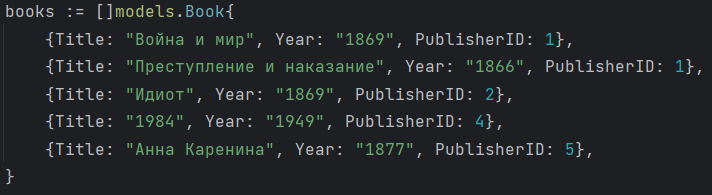


Рисунок 17 – тестовые данные для *Book.*

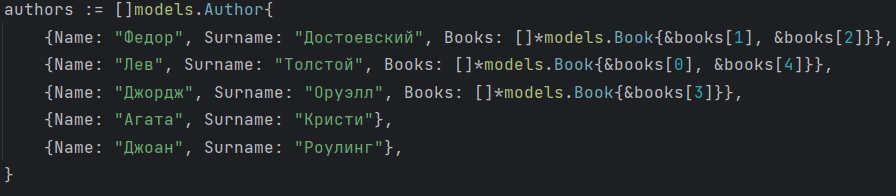
**

Рисунок 18 – тестовые данные для *Author.*

**

Рисунок 19 – тестовые данные для *Visitor.*

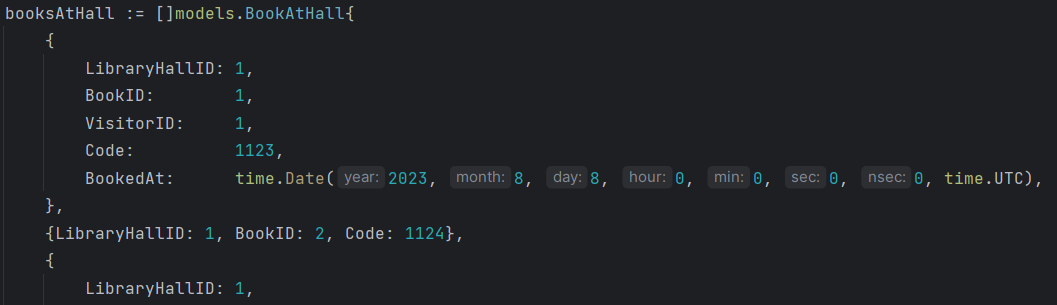
**

Рисунок 20 – тестовые данные для *BookAtHall.*

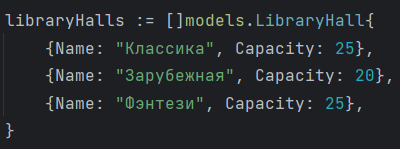
**

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.

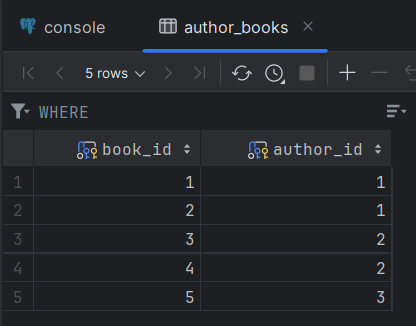


Рисунок 22 – таблица *author\_books* с данными.

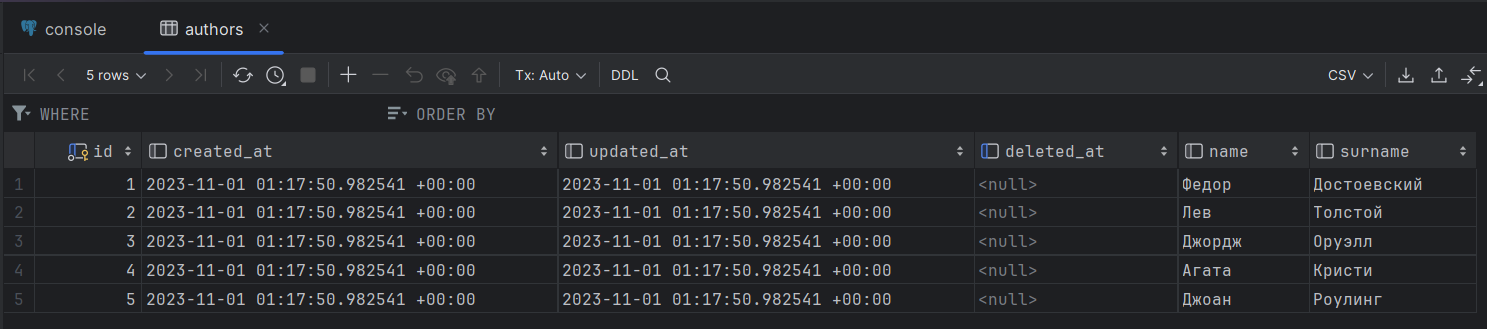


Рисунок 23 – таблица *authors* с данными.

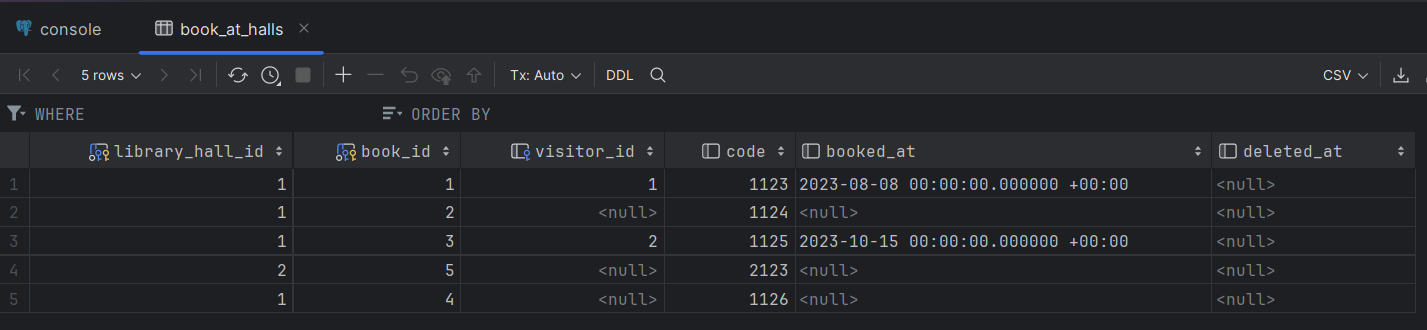


Рисунок 24 – таблица *book\_at\_halls* с данными.

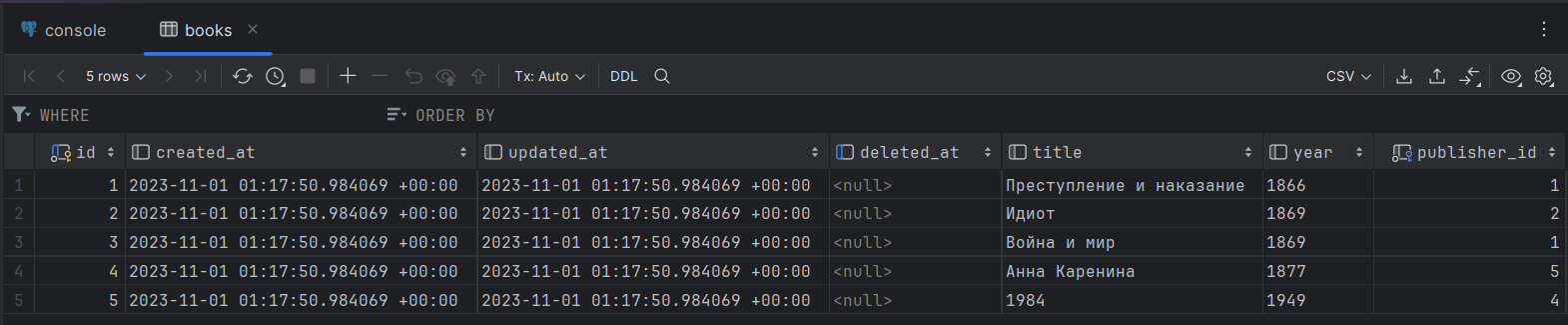


Рисунок 25 – таблица *books* с данными.

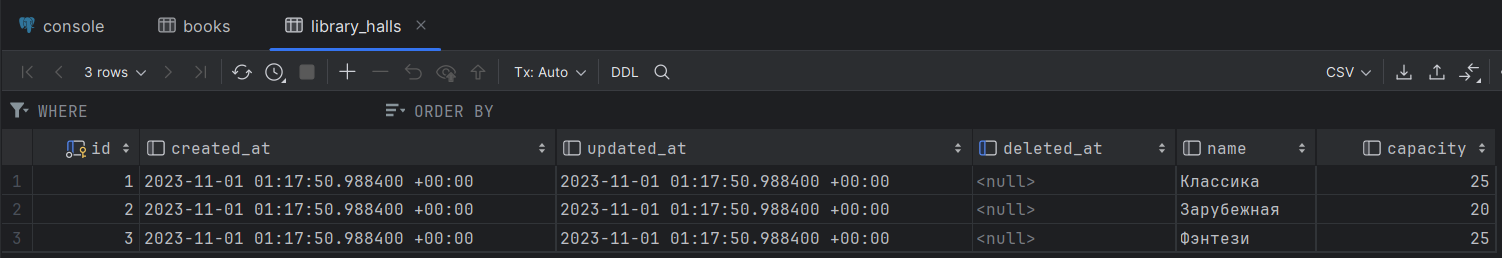


Рисунок 26 – таблица *library\_halls* с данными.

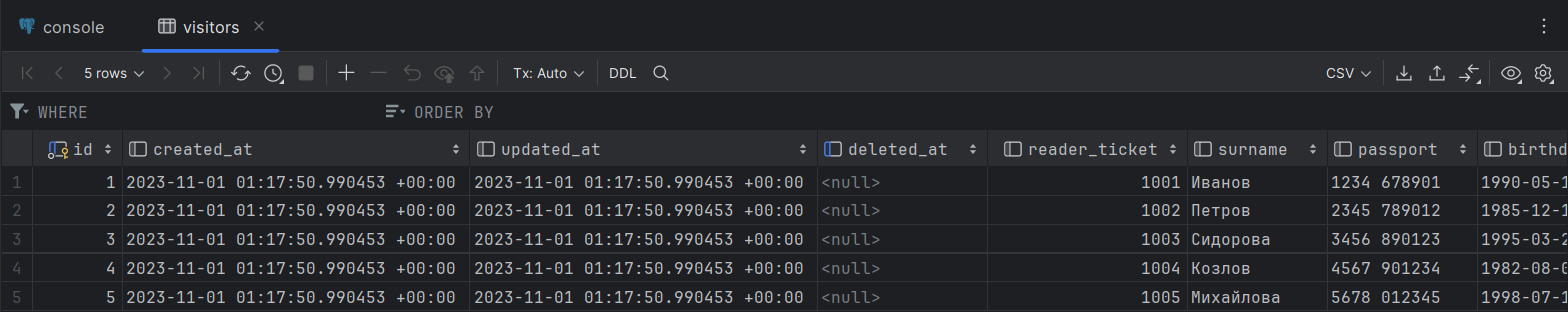


Рисунок 27 – таблица *visitors* с данными.

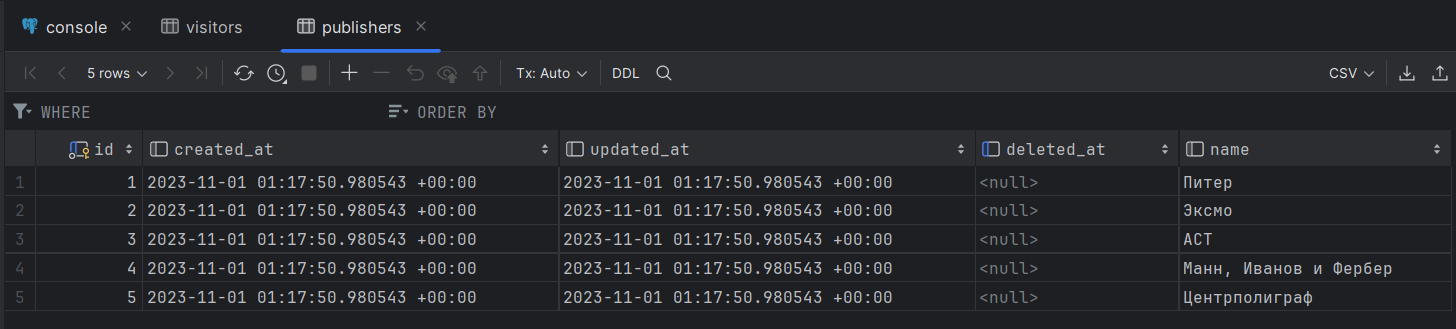


Рисунок 28 – таблица *publishers* с данными.

1. 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

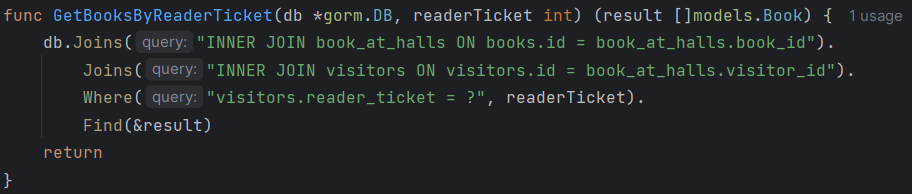


Рисунок 29 – функция с запросом по вопросу 1.

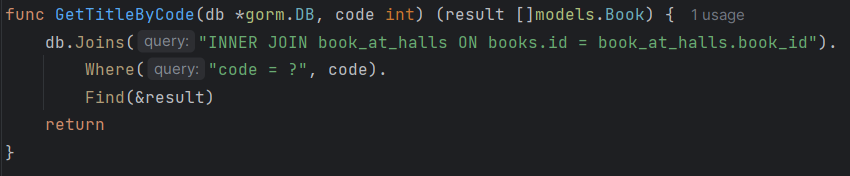


Рисунок 30 – функция с запросом по вопросу 2.

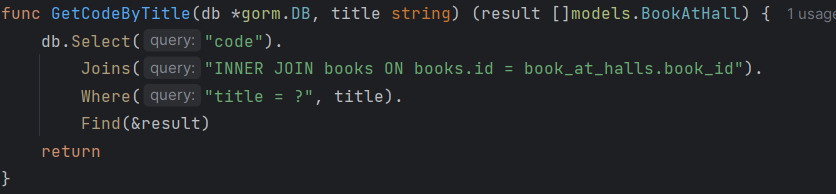


Рисунок 31 – функция с запросом по вопросу 3

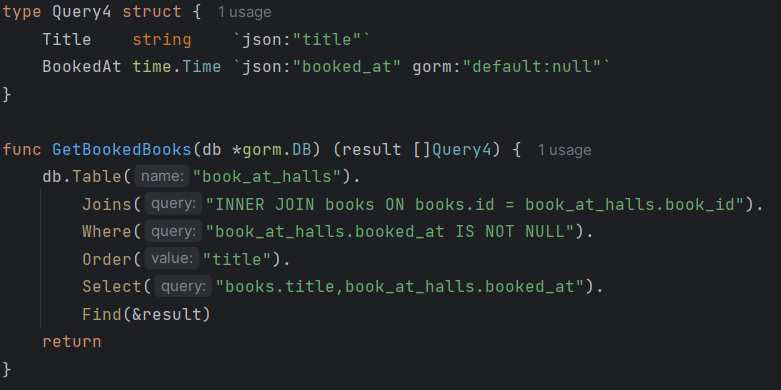


Рисунок 32 – структура результата и функция с запросом по вопросу 4

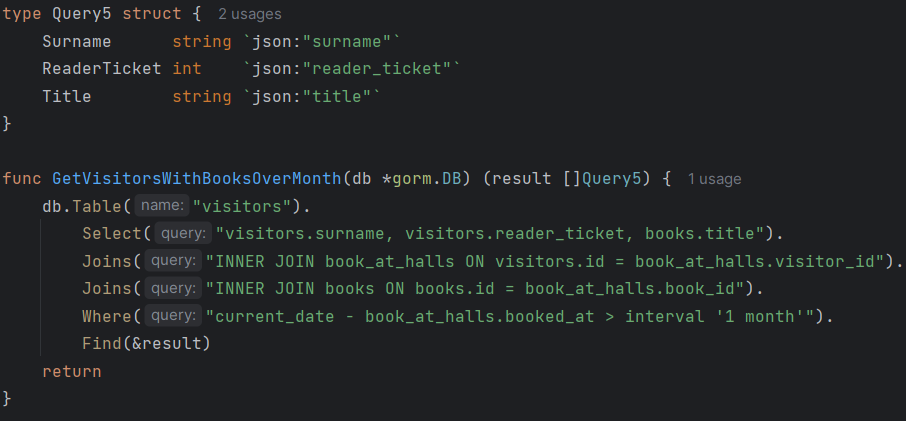


Рисунок 33 - структура результата и функция с запросом по вопросу 5

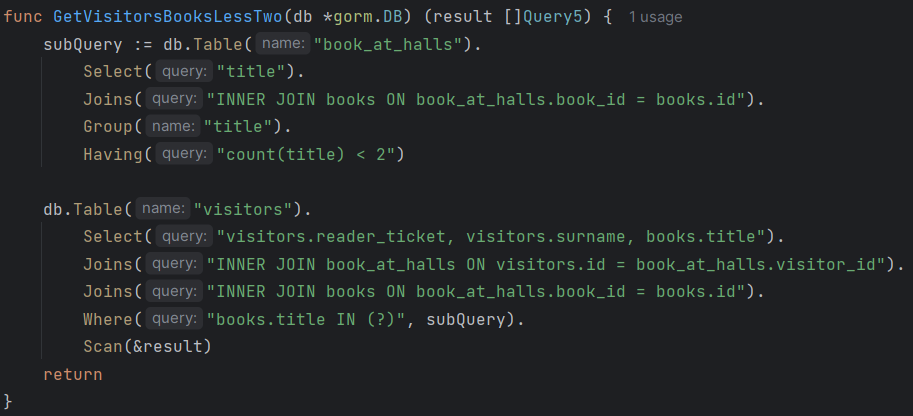


Рисунок 34 – функция с запросом по вопросу 6

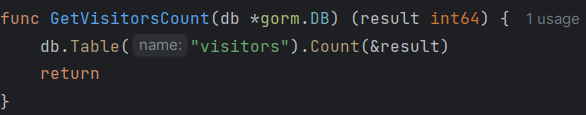


Рисунок 35 – функция с запросом по вопросу 7

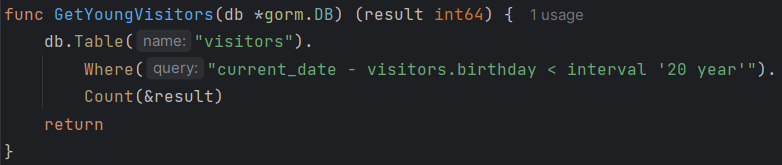


Рисунок 36 – функция с запросом по вопросу 8

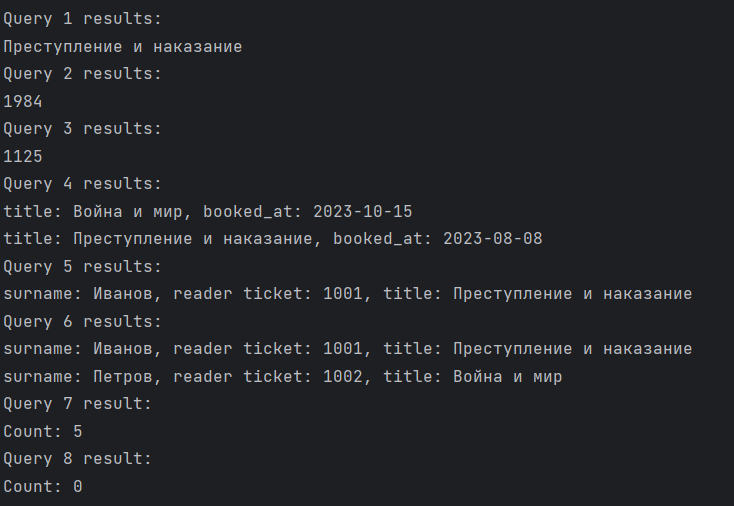


Рисунок 37 – результат выполнения данных запросов

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