Работа 1

# 

# Рисунок 1 – создание таблицы Client

# 

# Рисунок 2 – создание таблицы item\_category

# 

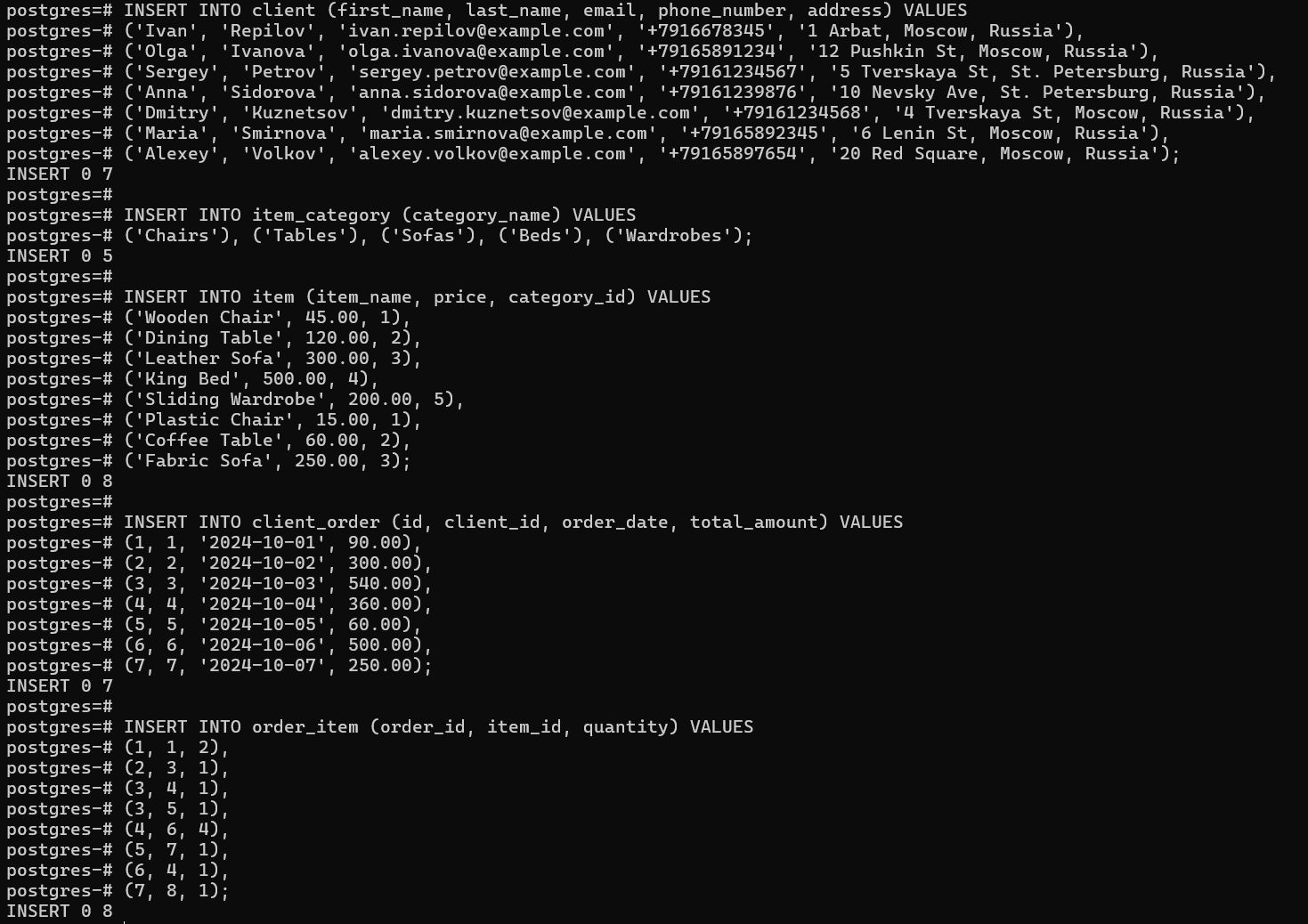
# Рисунок 3 – создание таблицы item

# 

# Рисунок 4 – создание таблицы client\_order

# 

# Рисунок 5 – создание таблицы order\_item



# Рисунок 6 – заполнение таблиц

Работа 2

# 

# Рисунок 7 – применение к выборке оператора ORDER BY

# 

# Рисунок 8 – применение к выборке оператора WHERE c VARCHAR

# 

# Рисунок 9 – применение к выборке оператора WHERE c INT

# 

# Рисунок 10 – применение к выборке оператора WHERE cо знаком «Больше»

# 

# Рисунок 11 – применение к выборке оператора WHERE c LIKE

Работа 3

1

# 

# Рисунок 12 – ERD-диаграммы базы данных

2

# 

# Рисунок 13 – создание дампа БД

# 

# Рисунок 14 – копирование дампа БД по SSH на сервер с Ubuntu

# 

# Рисунок 15 – IP-адрес сервера с Ubuntu

# 

# Рисунок 16 – восстановление БД из дампа

# 

# Рисунок 17 – восстановленные таблицы

3

# 

# Рисунок 18 – изменение данных

4

# 

# Рисунок 19 –выборка данных

# 

# Рисунок 20 – выборка данных с использованием WHERE

# 

# Рисунок 21 – операция соединения

# 

# Рисунок 22 – операция пересечения

# 

# Рисунок 23 – операция разности

# 

# Рисунок 24 – операция группировки

# 

# Рисунок 25 – операция сортировки

# 

# Рисунок 26 – операция деления

5

# 

# Рисунок 27 – процедура добавления нового клиента

## 

# Рисунок 28 – функция для получения полного имени клиента по id

# 

# Рисунок 29 – триггер для логирования изменений client

Работа 4

# 

# Рисунок 30 – простая оконная функция

# 

# Рисунок 31 – оконная функция с использованием SUM

# 

# Рисунок 32 – оконная функция с использованием MIN

# 

# Рисунок 33 – оконная функция с использованием MAX

# 

# Рисунок 34 – оконная функция с использованием MIN

# 

# Рисунок 35 – оконная функция с использованием ROW\_NUMBER

# 

# Рисунок 36 – оконная функция с использованием DENSE\_RANK

# 

# Рисунок 37 – оконная функция с использованием ROW\_NUMBER

# 

# Рисунок 38 – оконная функция с использованием LAG

# 

# Рисунок 39 – оконная функция с использованием LEAD

# 

# Рисунок 40 – оконная функция с использованием FIRST\_VALUE

# 

# Рисунок 40 – оконная функция с использованием LAST\_VALUE

Итоговый проект

furniture-rest – это RESTful приложение, написанное на языке Java с использованием Spring REST, Spring Boot, JDBC API с подключением к базе данных PostgreSQL. Данное приложение предоставляет полный доступ к БД, разрабатываемой в процессе курса через REST API. С помощью этого приложения вы можете производить CRUD-операции на всех сущностях БД. Например, добавить клиента, присвоить клиенту заказ, добавить товары и т.п.

Конфигурация осуществляется с помощью Spring Boot, приложение общается с пользователем с помощью HTTP-запросов на порту 8080 используя REST API. В проекте задействована СУБД PostgreSQL, для формирования запросов используется JDBC API, предоставляющий удобные и гибкие методы для написания запросов, а также предотвращает атаки с помощью SQL-инъекций благодаря механизму Prepared Statement, запросы реализованы в Data Access Object (DAO) классах. Для отображения структуры таблицы на класс-сущность были созданы классы-сопоставители (RowMapper’ы). HTTP-запросами управляют классы Controller’ы для обработки запросов, также в них реализована обработка исключений через дополнительные классы-обработчики исключений (ExceptionHandler’ы). Для улучшения расширяемости проекта, передача данных из и в контроллеры были реализованы специальные классы для передачи информации о сущностях (DataTransferObject’ы).

Исходный код приложения доступен на [GitHub](https://github.com/flynt-3650/furniture-rest).

Для взаимодействия с хранящимися в БД данными был использован паттерн проектирования DAO (Data Access Object).

Например, класс ClientDao:

|  |
| --- |
| package ru.flynt3650.project.furniture.dao;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.dao.EmptyResultDataAccessException;import org.springframework.jdbc.core.JdbcTemplate;import org.springframework.stereotype.Component;import ru.flynt3650.project.furniture.mappers.ClientRowMapper;import ru.flynt3650.project.furniture.models.Client;import ru.flynt3650.project.furniture.util.exceptions.ClientNotFoundException;import java.util.List;import java.util.Map;@Componentpublic class ClientDao implements MyCrudDao<Client, Integer> {private final JdbcTemplate jdbcTemplate;@Autowiredpublic ClientDao(JdbcTemplate jdbcTemplate) {this.jdbcTemplate = jdbcTemplate;}@Overridepublic void create(Client client) {String sql = "INSERT INTO client(first\_name, last\_name, email, phone\_number, address) VALUES (?, ?, ?, ?, ?)";jdbcTemplate.update(sql,client.getFirstName(),client.getLastName(),client.getEmail(),client.getPhoneNumber(),client.getAddress());}@Overridepublic List<Client> readAll() {String sql = "SELECT \* FROM client";return jdbcTemplate.query(sql, new ClientRowMapper());}@Overridepublic Client readOne(Integer id) {String sql = "SELECT \* FROM client WHERE id=?";try {return jdbcTemplate.queryForObject(sql, new ClientRowMapper(), id);} catch (EmptyResultDataAccessException e) {throw new ClientNotFoundException("Client with id '" + id + "' was not found");}}public void update(Integer id, Client client) {String sql = "UPDATE client SET first\_name=?, last\_name=?, email=?, phone\_number=?, address=? WHERE id=?";jdbcTemplate.update(sql,client.getFirstName(),client.getLastName(),client.getEmail(),client.getPhoneNumber(),client.getAddress(),id);}public void delete(Integer id) {String sql = "DELETE FROM client where id=?";jdbcTemplate.update(sql, id);}public List<Map<String, Object>> getClientOrderInfo() {String sql = "SELECT " +"co.id AS order\_id, " +"c.id AS client\_id, " +"c.first\_name, " +"c.last\_name, " +"c.address, " +"oi.quantity, " +"i.item\_name, " +"i.price, " +"ic.category\_name " +"FROM " +"client c " +"JOIN client\_order co ON c.id = co.client\_id " +"JOIN order\_item oi ON co.id = oi.order\_id " +"JOIN item i ON oi.item\_id = i.id " +"JOIN item\_category ic ON i.category\_id = ic.id";return jdbcTemplate.queryForList(sql);}} |

# Листинг 1 – класс ClientDao

|  |
| --- |
| package ru.flynt3650.project.furniture.controllers;  import jakarta.validation.Valid; import org.modelmapper.ModelMapper; import org.springframework.beans.factory.annotation.Autowired; import org.springframework.validation.BindingResult; import org.springframework.validation.FieldError; import org.springframework.web.bind.annotation.\*; import ru.flynt3650.project.furniture.dao.ClientDao; import ru.flynt3650.project.furniture.dto.ClientDto; import ru.flynt3650.project.furniture.models.Client; import ru.flynt3650.project.furniture.util.exceptions.ClientNotPostedException;  import java.util.List; import java.util.Map; import java.util.stream.Collectors;  @RestController @RequestMapping("/clients") public class ClientController {   private final ClientDao clientDao;   private final ModelMapper modelMapper;   @Autowired  public ClientController(ClientDao clientDao, ModelMapper modelMapper) {  this.clientDao = clientDao;  this.modelMapper = modelMapper;  }   @PostMapping("/add")  public void postClient(@RequestBody @Valid ClientDto clientDto,  BindingResult bindingResult) {   if (bindingResult.hasErrors()) {  StringBuilder errorMessage = new StringBuilder();   List<FieldError> errors = bindingResult.getFieldErrors();  for (var error : errors)  errorMessage  .append(error.getField())  .append(" - ")  .append(error.getDefaultMessage())  .append(";");   throw new ClientNotPostedException(errorMessage.toString());  }   clientDao.create(toClient(clientDto));  }   @GetMapping()  public List<ClientDto> getAllClients() {  return clientDao  .readAll()  .stream()  .map(this::toClientDto)  .collect(Collectors.toList());  }   @GetMapping("/{id}")  public ClientDto getOneClient(@PathVariable("id") Integer id) {    return toClientDto(clientDao.readOne(id));  }   @PatchMapping("/update/{id}")  public void patchOneClient(@PathVariable("id") Integer id, @RequestBody @Valid ClientDto clientDto,  BindingResult bindingResult) {   if (bindingResult.hasErrors()) {  StringBuilder errorMessage = new StringBuilder();   List<FieldError> errors = bindingResult.getFieldErrors();  for (var error : errors)  errorMessage  .append(error.getField())  .append(" - ")  .append(error.getDefaultMessage())  .append(";");   throw new ClientNotPostedException(errorMessage.toString());  }   clientDao.update(id, toClient(clientDto));  }   @DeleteMapping("/delete/{id}")  public void deleteOneClient(@PathVariable("id") Integer id) {  clientDao.delete(id);  }   @GetMapping("/clientOrder")  public List<Map<String, Object>> getClientOrder() {  return clientDao.getClientOrderInfo();  }   private Client toClient(ClientDto clientDto) {  return modelMapper.map(clientDto, Client.class);  }   private ClientDto toClientDto(Client client) {  return modelMapper.map(client, ClientDto.class);  } } |

# Листинг 2 – класс ClientController

|  |
| --- |
| package ru.flynt3650.project.furniture.exception\_handlers;  import org.springframework.http.HttpStatus; import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.ControllerAdvice; import org.springframework.web.bind.annotation.ExceptionHandler; import org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler; import ru.flynt3650.project.furniture.util.ClientExceptionResponse; import ru.flynt3650.project.furniture.util.exceptions.ClientNotPostedException; import ru.flynt3650.project.furniture.util.exceptions.ClientNotFoundException; import ru.flynt3650.project.furniture.util.exceptions.ClientNotPatchedException;  @ControllerAdvice public class ClientExceptionHandler extends ResponseEntityExceptionHandler {   @ExceptionHandler(ClientNotPostedException.class)  public ResponseEntity<ClientExceptionResponse> handleClientNotAddedException(ClientNotPostedException ex) {  ClientExceptionResponse response = new ClientExceptionResponse(  ex.getMessage(), System.currentTimeMillis()  );  return new ResponseEntity<>(response, HttpStatus.BAD\_REQUEST);  }   @ExceptionHandler(ClientNotFoundException.class)  public ResponseEntity<ClientExceptionResponse> handleClientNotFoundException(ClientNotFoundException ex) {  ClientExceptionResponse response = new ClientExceptionResponse(  ex.getMessage(), System.currentTimeMillis()  );  return new ResponseEntity<>(response, HttpStatus.NOT\_FOUND);  }   @ExceptionHandler(ClientNotPatchedException.class)  public ResponseEntity<ClientExceptionResponse> handleClientNotPatchedException(ClientNotPatchedException ex) {  ClientExceptionResponse response = new ClientExceptionResponse(  ex.getMessage(), System.currentTimeMillis()  );  return new ResponseEntity<>(response, HttpStatus.BAD\_REQUEST);  } } |

# Листинг 3 – класс ClientExceptionHandler

|  |
| --- |
| package ru.flynt3650.project.furniture.mappers;   import org.springframework.jdbc.core.RowMapper; import ru.flynt3650.project.furniture.models.Client;  import java.sql.ResultSet; import java.sql.SQLException;  public class ClientRowMapper implements RowMapper<Client> {   @Override  public Client mapRow(ResultSet rs, int rowNum) throws SQLException {  Client client = new Client();  client.setId(rs.getInt("id"));  client.setFirstName(rs.getString("first\_name"));  client.setLastName(rs.getString("last\_name"));  client.setEmail(rs.getString("email"));  client.setPhoneNumber(rs.getString("phone\_number"));  client.setAddress(rs.getString("address"));  return client;  } } |

# Листинг 4 – класс ClientRowMapper

Рассмотрим функции приложения на примере работы с таблицей Client

# 

# Рисунок 41 – получение всех клиентов в формате JSON

# 

# Рисунок 42 – добавление нового клиента

# 

# Рисунок 43 – получение клиента по id

# 

# Рисунок 44 – получение таблицы заказов всех клиентов

# 

# Рисунок 45 – изменение данных клиента

# 

# Рисунок 46 – удаление клиента