**Міністерство освіти і науки України**

**Національний технічний університет**

**«Дніпровська політехніка»**

****

**ЗВІТ**

**Практична робота №6**

**з дисципліни**

# «Програмування в середовищі Java»

**Виконав:**

студент гр. 122-21-3

Кабаченко О. В.

**Прийняв:**

Доцент каф. САУ

Алексєєв О. М.

**м. Дніпро**

**2025  рік**

**Хід роботи**

1) Для початку роботи було створено базу даних, використовуючи OpenServer та PhpMyAdmin. На рисунку 1 наведено сторінку авторизації, а на рисунку 2 – структура таблиці «Students» у створеної базі даних.

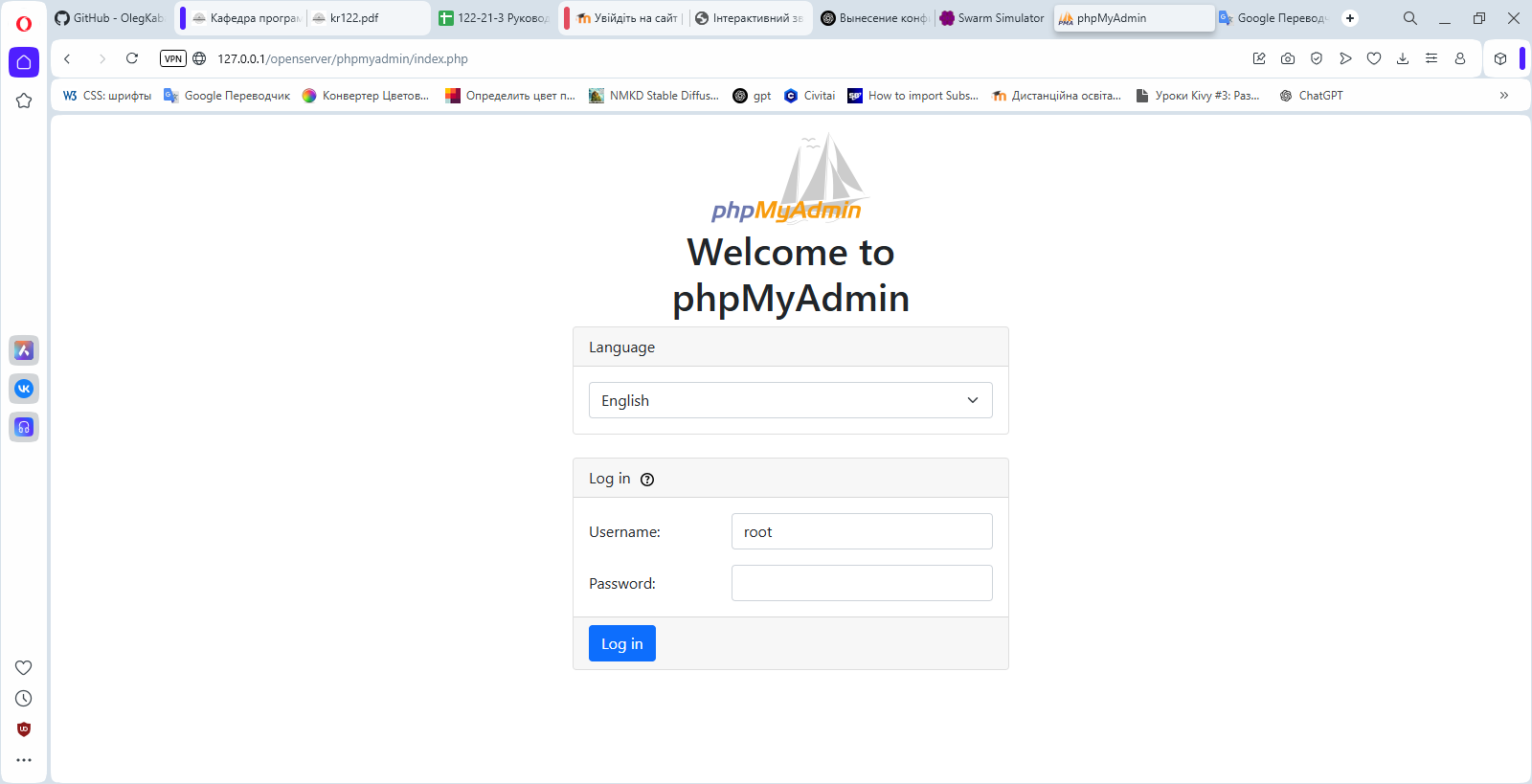


Рисунок 1 – Авторизація у PhpMyAdmin

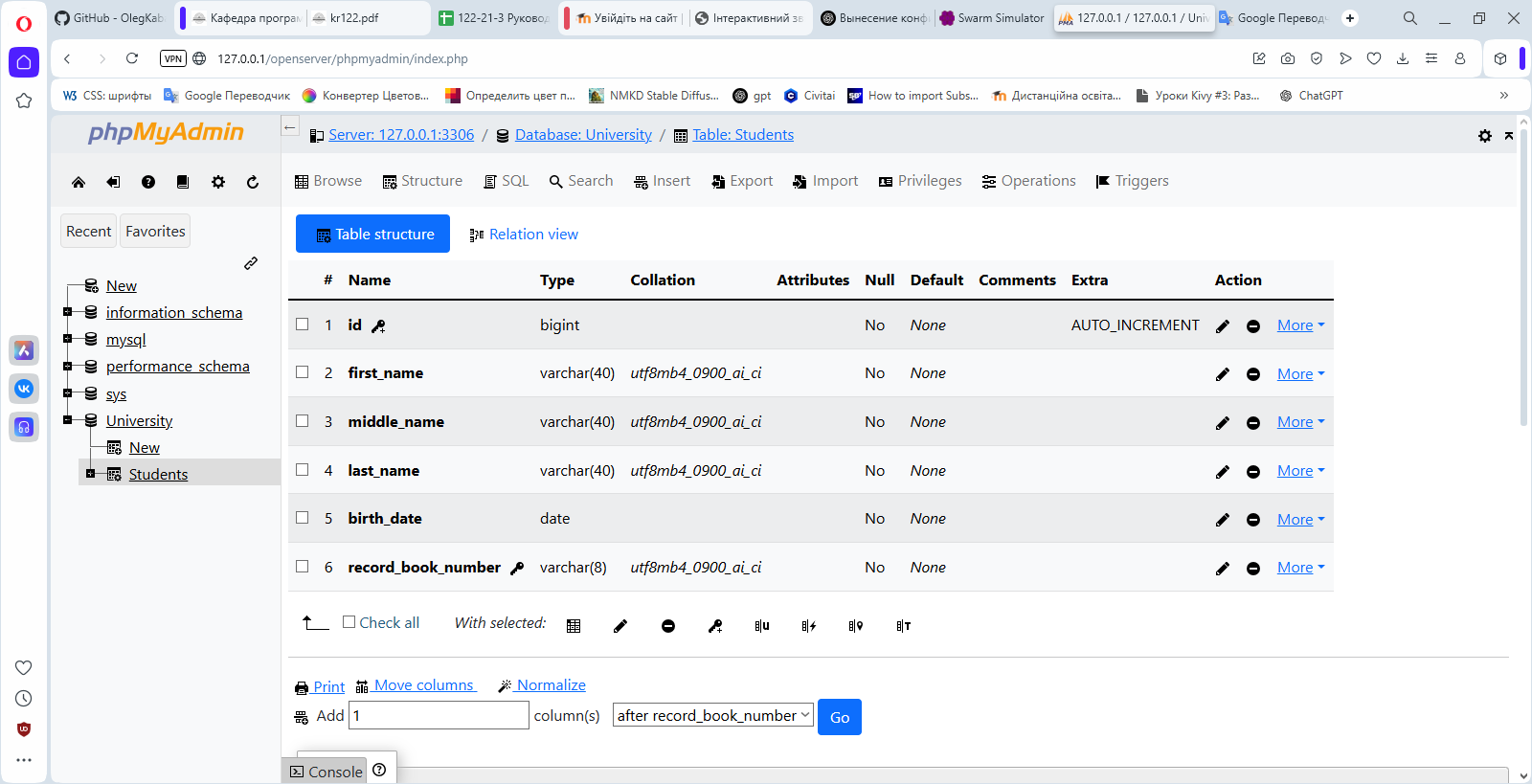


Рисунок 2 – Структура таблиці «Students»

2) Далі у вбудованій панелі користування базами даних було підключено створену базу даних з попереднього кроку, на рисунках 3-4 наведено цей процес.

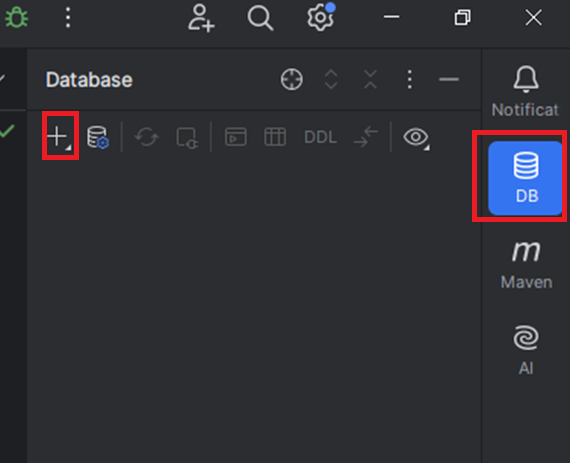


Рисунок 3 – Кнопка додання бази даних

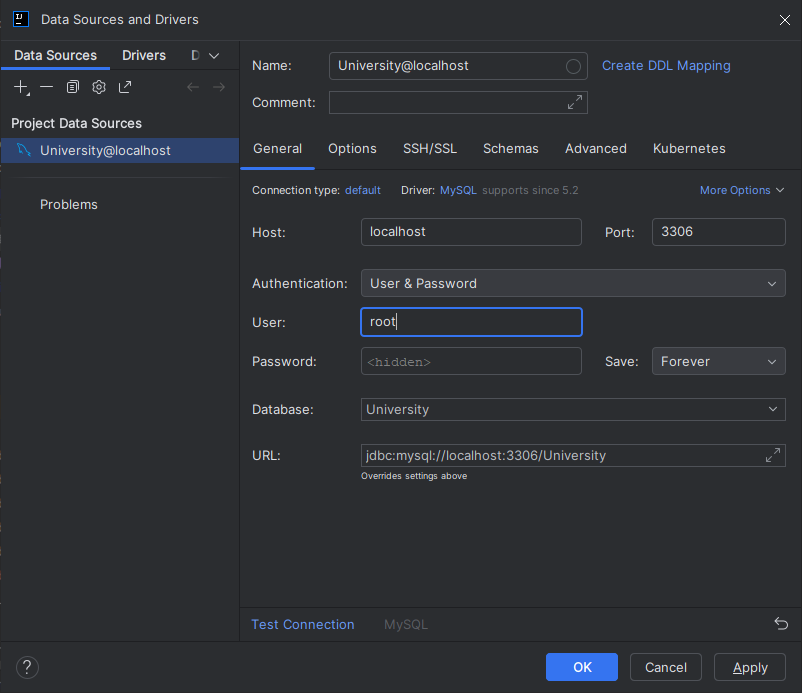


Рисунок 4 – Налаштування підключення до БД

3) На цьому кроці перевірено успішність підключення до БД, використовуючи консоль, що з’явилася після додавання бази даних. Як видно з рисунку 5, підключення відбулося успішно, оскільки в результаті виконання запиту видно стовпці таблиці.

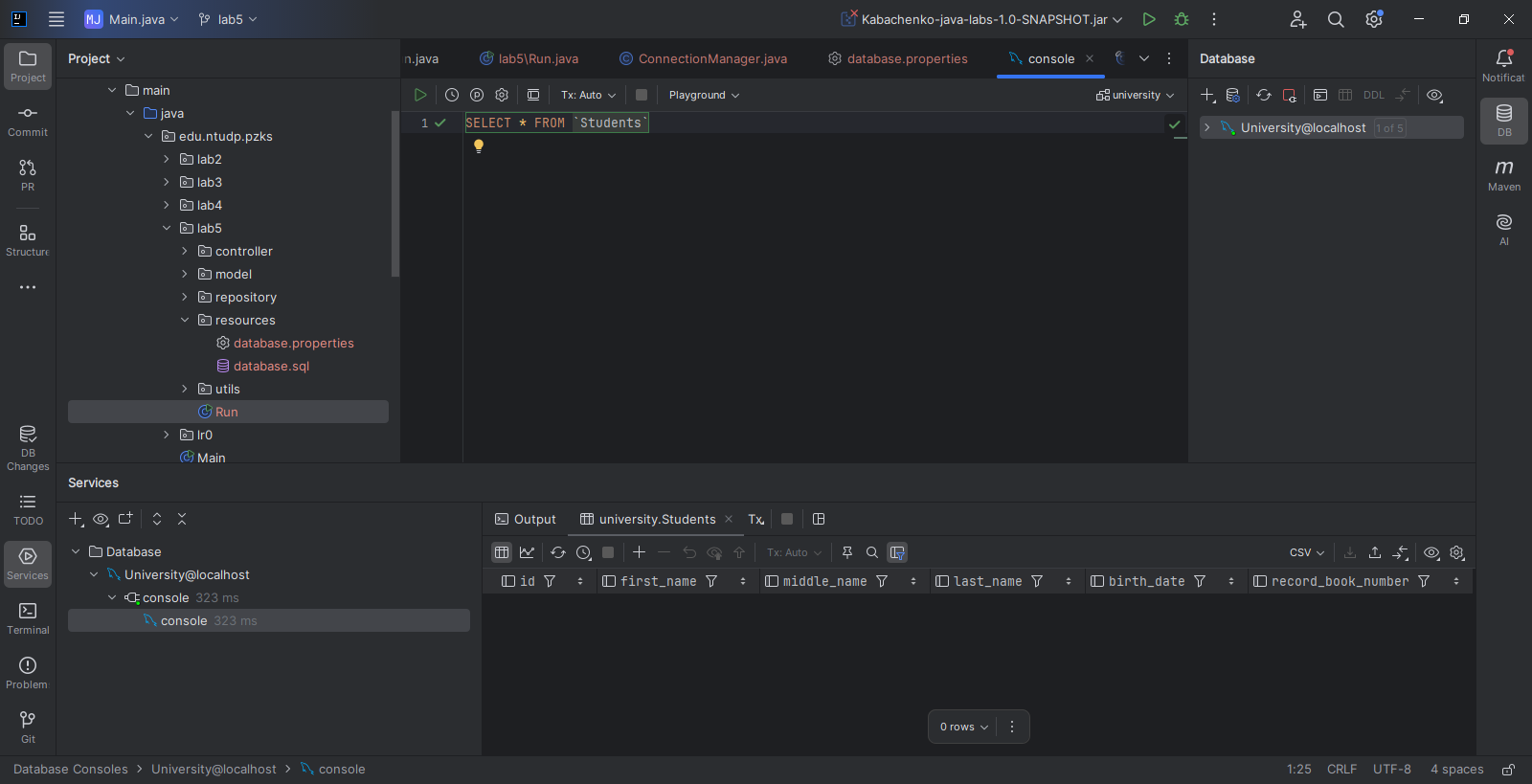


Рисунок 5 – Перевірка успішності підключення БД

4) Після попереднього кроку було підготовлено структуру програми, шляхом додавання класу Run для запуску застосунку, а також пакетів controller, model, repository, resources та utils, на рисунку 6 наведено структуру проекту.

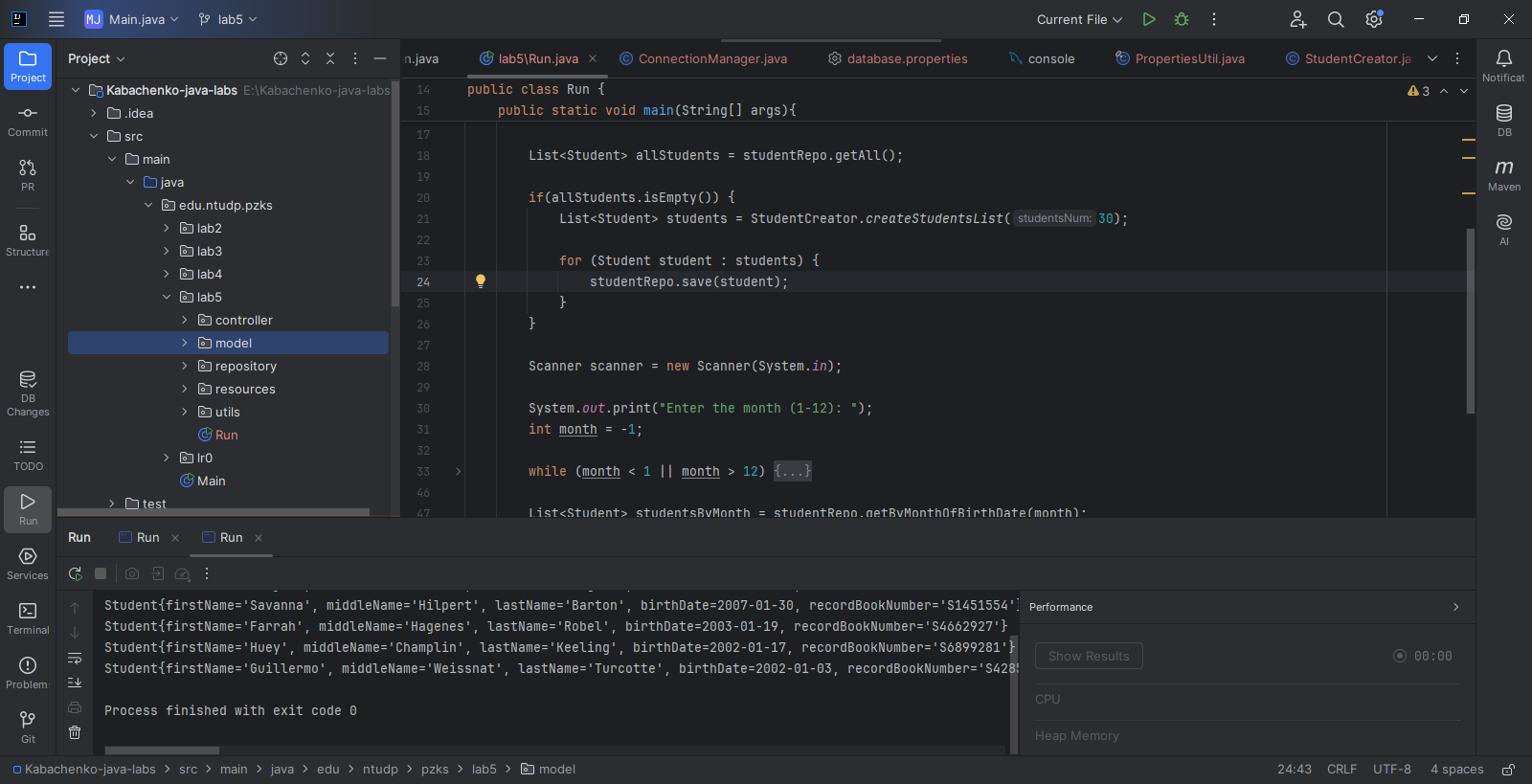


Рисунок 6 – Структура проекту

5) Далі у пакеті model створено два файла – Human.java та Student.java. Клас Human являє собою абстрактний клас, та містить такі змінні як id, firstName, middleName, lastName, birthDate, NAME\_MAX\_LENGTH, а також методи get/set, isFilled, equals, hashCode та toString. В свою чергу Student є нащадком Human, додає змінні recordBookNumber та RECORD\_BOOK\_NUM\_LENGTH та перевизначає деякі методи. Нижче наведено програмний код обох файлів.

**Програмний код файлу «Human.java»**

package edu.ntudp.pzks.lab6.model;  
  
import java.sql.Date;  
import java.util.Objects;  
import java.util.function.Consumer;  
  
public abstract class Human {  
 private long id;  
 protected String firstName;  
 protected String middleName;  
 protected String lastName;  
 protected Date birthDate;  
  
 private final int NAME\_MAX\_LENGTH = 40;  
  
 public Human() {  
 }  
  
 public Human(String firstName, String middleName, String lastName, Date birthDate) {  
 setFirstName(firstName);  
 setMiddleName(middleName);  
 setLastName(lastName);  
  
 this.birthDate = birthDate;  
 }  
  
 public boolean isFilled (){  
 return firstName != null &&  
 middleName != null &&  
 lastName != null &&  
 birthDate != null;  
 }  
  
 private void setNameField(String fieldName, String value, Consumer<String> setter) {  
 if (value.length() <= NAME\_MAX\_LENGTH) {  
 setter.accept(value);  
 } else {  
 throw new IllegalArgumentException("Error: The " + fieldName + " must be shorter than " + (NAME\_MAX\_LENGTH + 1) + " characters long.");  
 }  
 }  
 public long getId() {  
 return id;  
 }  
  
 public void setId(long id) {  
 this.id = id;  
 }  
  
 public String getFirstName() {  
 return firstName;  
 }  
  
 public void setFirstName(String firstName) {  
 setNameField("first name", firstName, value -> this.firstName = value);  
 }  
  
 public String getMiddleName() {  
 return middleName;  
 }  
  
 public void setMiddleName(String middleName) {  
 setNameField("middle name", middleName, value -> this.middleName = value);  
 }  
  
 public String getLastName() {  
 return lastName;  
 }  
  
 public void setLastName(String lastName) {  
 setNameField("last name", lastName, value -> this.lastName = value);  
 }  
  
 public Date getBirthDate() {  
 return birthDate;  
 }  
  
 public void setBirthDate(Date birthDate) {  
 this.birthDate = birthDate;  
 }  
 @Override  
 public boolean equals(Object o) {  
 if (o == null || getClass() != o.getClass()) return false;  
 Human human = (Human) o;  
 return Objects.*equals*(id, human.id) && Objects.*equals*(firstName, human.firstName) && Objects.*equals*(middleName, human.middleName) && Objects.*equals*(lastName, human.lastName) && Objects.*equals*(birthDate, human.birthDate);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(id, firstName, middleName, lastName, birthDate);  
 }  
 @Override  
 public String toString() {  
 return "Human{" +  
 "birthDate=" + birthDate +  
 ", lastName='" + lastName + '\'' +  
 ", middleName='" + middleName + '\'' +  
 ", firstName='" + firstName + '\'' +  
 ", id=" + id +  
 '}';  
 }  
}

**Програмний код файлу «Student.java»**

package edu.ntudp.pzks.lab6.model;  
  
import java.sql.Date;  
import java.util.Objects;  
  
public class Student extends Human {  
 private String recordBookNumber;  
 private final int RECORD\_BOOK\_NUM\_LENGTH = 8;  
  
 public Student() {  
 }  
  
 public Student(String firstName, String middleName, String lastName, String recordBookNumber, Date birthDate) {  
 super(firstName, middleName, lastName, birthDate);  
 setRecordBookNumber(recordBookNumber);  
 }  
  
 @Override  
 public boolean isFilled() {  
 return super.isFilled() && recordBookNumber != null;  
 }  
  
 public String getRecordBookNumber() {  
 return recordBookNumber;  
 }  
  
 public void setRecordBookNumber(String recordBookNumber) {  
 if (recordBookNumber.length() == RECORD\_BOOK\_NUM\_LENGTH){  
 this.recordBookNumber = recordBookNumber;  
 }  
 else {  
 throw new IllegalArgumentException("Error: The record book number must be exactly " + RECORD\_BOOK\_NUM\_LENGTH + " characters long.");  
 }  
 }  
  
 public int getRECORD\_BOOK\_NUM\_LENGTH() {  
 return RECORD\_BOOK\_NUM\_LENGTH;  
 }  
  
 @Override  
 public boolean equals(Object o) {  
 if (o == null || getClass() != o.getClass()) return false;  
 if (!super.equals(o)) return false;  
 Student student = (Student) o;  
 return Objects.*equals*(recordBookNumber, student.recordBookNumber);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(super.hashCode(), recordBookNumber);  
 }  
  
 @Override  
 public String toString() {  
 return "Student{" +  
 "firstName='" + firstName + '\'' +  
 ", middleName='" + middleName + '\'' +  
 ", lastName='" + lastName + '\'' +  
 ", birthDate=" + birthDate +  
 ", recordBookNumber='" + recordBookNumber + '\'' +  
 '}';  
 }  
}

6) На цьому кроці у пакеті controller створено два файла – HumanCreator.java та StudentCreator.java. Клас HumanCreator є абстрактним, та містить в собі змінні random і faker, а також методи generateRandomFullName і createValidBirthDate. У свою чергу StudentCreator є нащадком класу HumanCreator та реалізує три різних метода для створення студентів. Нижче наведено програмний код обох файлів.

**Програмний код файлу «HumanCreator.java»**

package edu.ntudp.pzks.lab6.controller;  
  
import com.github.javafaker.Faker;  
  
import java.sql.Date;  
import java.util.Calendar;  
import java.util.Random;  
  
public abstract class HumanCreator {  
 public static final Random *random* = new Random();  
 public static final Faker *faker* = new Faker();  
  
 protected static String[] generateRandomFullName() {  
 return new String[]{  
 *faker*.name().firstName(),  
 *faker*.name().lastName(),  
 *faker*.name().lastName()  
 };  
 }  
 public static Date createValidBirthDate() {  
 Date birthDate = null;  
 boolean validBirthDate = false;  
  
 while (!validBirthDate) {  
 birthDate = new Date(*faker*.date().birthday().getTime());  
  
 Calendar calendar = Calendar.*getInstance*();  
 calendar.setTime(birthDate);  
  
 int year = calendar.get(Calendar.*YEAR*);  
 int month = calendar.get(Calendar.*MONTH*);  
  
 if (year >= 2002 && month != Calendar.*JUNE*) {  
 validBirthDate = true;  
 }  
 }  
 return birthDate;  
 }  
  
}

**Програмний код файлу «StudentCreator.java»**

package edu.ntudp.pzks.lab6.controller;  
  
import edu.ntudp.pzks.lab6.model.Student;  
  
import java.sql.Date;  
import java.util.ArrayList;  
import java.util.HashSet;  
import java.util.List;  
import java.util.Set;  
  
public class StudentCreator extends HumanCreator {  
  
 public static String generateRandomRecordBookNumber(){  
 StringBuilder recordBookNumber = new StringBuilder("S");  
  
 for (int i = 0; i < 7; i++) {  
 int digit = *random*.nextInt(10);  
 recordBookNumber.append(digit);  
 }  
  
 return recordBookNumber.toString();  
 }  
  
 public static List<Student> createStudentsList(int studentsNum){  
 List<Student> students = new ArrayList<>();  
 Set<String> usedRecordBookNumbers = new HashSet<>();  
  
 for (int i = 0; i < studentsNum; i++) {  
 Student student = createTypicalStudent();  
 String recordBookNumber = student.getRecordBookNumber();  
  
 while (usedRecordBookNumbers.contains(recordBookNumber)) {  
 recordBookNumber = generateRandomRecordBookNumber();  
 student.setRecordBookNumber(recordBookNumber);  
 }  
  
 usedRecordBookNumbers.add(recordBookNumber);  
 students.add(student);  
 }  
  
 return students;  
 }  
  
 public static Student createTypicalStudent() {  
 String[] names = generateRandomFullName();  
 String recordBookNumber = generateRandomRecordBookNumber();  
 Date birthDate = createValidBirthDate();  
  
 return new Student(names[0], names[1], names[2], recordBookNumber, birthDate);  
 }  
  
 public static Student createStudent(String firstName, String middleName, String lastName, String recordBookNumber, Date birthDate) {  
 return new Student (firstName, middleName, lastName, recordBookNumber, birthDate);  
 }  
}

7) Далі у пакеті repository створено два файла – HumanRepository.java та StudentRepository.java. HumanRepository.java містить інтерфейс HumanRepository, який описує загальні операції для роботи з об'єктами типу T. Він містить методи для збереження (save), отримання всіх об'єктів (getAll), фільтрації за місяцем народження (getByMonthOfBirthDate) та видалення (delete). У свою чергу клас StudentRepository реалізує цей інтерфейс. Слід зазначити що SQL запит формується згідно запросу який зробив користувач і вже сервер управління баз даних вирішує, які самі студенти народилися в тому чи іншому місяці. Такий підхід дозволяє оптимізує продуктивність, оскільки передається та оброблюється менше даних, що зменшує використання пам’яті в Java. До недоліків такого підходу можливо віднести більш складний алгоритм формування SQL запиту, а також залежність від можливостей СУБД (не всі підтримують, наприклад, MONTH() у запитах). Нижче наведено програмний код обох файлів.

**Програмний код файлу «HumanRepository.java»**

package edu.ntudp.pzks.lab6.repository;  
  
import java.util.List;  
  
public interface HumanRepository <T> {  
  
 void save (T t);  
  
 List<T> getAll();  
  
 List<T> getByMonthOfBirthDate(int month);  
  
 void delete (T t);  
}

**Програмний код файлу «StudentRepository.java»**

package edu.ntudp.pzks.lab6.repository;  
  
import edu.ntudp.pzks.lab6.model.Student;  
import edu.ntudp.pzks.lab6.utils.ConnectionManager;  
  
import java.sql.\*;  
import java.util.ArrayList;  
import java.util.Calendar;  
import java.util.List;  
  
public class StudentRepository implements HumanRepository<Student>{  
 private List<Student> extractStudents(ResultSet resultSet) {  
 List<Student> students = new ArrayList<>();  
 while (true) {  
 try {  
 if (!resultSet.next()) break;  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 Student student = new Student();  
  
 try {  
 student.setId(resultSet.getLong("id"));  
 student.setFirstName(resultSet.getString("first\_name"));  
 student.setMiddleName(resultSet.getString("middle\_name"));  
 student.setLastName(resultSet.getString("last\_name"));  
 student.setBirthDate(resultSet.getDate("birth\_date"));  
 student.setRecordBookNumber(resultSet.getString("record\_book\_number"));  
  
 students.add(student);  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 return students;  
 }  
  
 @Override  
 public void save(Student student) {  
 if (student.isFilled()){  
 String query = "INSERT INTO Students (first\_name, middle\_name, last\_name, birth\_date, record\_book\_number) " +  
 "VALUES (?, ?, ?, ?, ?)";  
  
 try (Connection connection = ConnectionManager.*open*();  
 PreparedStatement preparedStatement = connection.prepareStatement(query)){  
 preparedStatement.setString(1, student.getFirstName());  
 preparedStatement.setString(2, student.getMiddleName());  
 preparedStatement.setString(3, student.getLastName());  
 preparedStatement.setDate(4, student.getBirthDate());  
 preparedStatement.setString(5, student.getRecordBookNumber());  
  
 preparedStatement.executeUpdate();  
 } catch (SQLException e) {  
 throw new RuntimeException("Error during student save: ", e);  
 }  
 }  
 else{  
 throw new IllegalArgumentException("Error: Student have null data!");  
 }  
 }  
  
 @Override  
 public List<Student> getAll() {  
 String query = "SELECT \* FROM `Students`";  
 List<Student> students;  
  
 try (Connection connection = ConnectionManager.open();  
 PreparedStatement preparedStatement = connection.prepareStatement(query);  
 ResultSet resultSet = preparedStatement.executeQuery()) {  
 students = extractStudents(resultSet);  
  
 } catch (SQLException e) {  
 throw new RuntimeException("Error during student getAll: ", e);  
 }  
 return students;  
 }  
  
 @Override  
 public List<Student> getByMonthOfBirthDate(int month) {  
 String query = "SELECT \* FROM `Students` WHERE MONTH(birth\_date) = ?";  
 List<Student> students;  
  
 try (Connection connection = ConnectionManager.open();  
 PreparedStatement preparedStatement = connection.prepareStatement(query)) {  
 preparedStatement.setInt(1, month);  
  
 try (ResultSet resultSet = preparedStatement.executeQuery()) {  
 students = extractStudents(resultSet);  
 }  
 } catch (SQLException e) {  
 throw new RuntimeException("Error during student getByMonthOfBirthDate: ", e);  
 }  
 return students;  
 }  
  
 @Override  
 public void delete(Student student) {  
 String query = "DELETE FROM `Students` WHERE `id` = ?";  
  
 try (Connection connection = ConnectionManager.open();  
 PreparedStatement preparedStatement = connection.prepareStatement(query)) {  
  
 preparedStatement.setLong(1, student.getId());  
  
 int rowsAffected = preparedStatement.executeUpdate();  
  
 if (rowsAffected == 0) {  
 throw new RuntimeException("Error: Student with this ID not found!");  
 }  
 } catch (SQLException e) {  
 throw new RuntimeException("Error during student deleting: ", e);  
 }  
 }  
  
}

8) Після попереднього кроку в пакеті resourcеs створено файл database.properties, що містить налаштування для підключення до бд, а також файл database.sql, який містить код створення БД. Нижче наведено програмний код обох файлів.

**Програмний код файлу «database.properties»**

db.url=jdbc:mysql://localhost:3306/University  
db.user=root  
db.password=

**Програмний код файлу «database.sql»**

-- phpMyAdmin SQL Dump  
-- version 5.2.0  
-- https://www.phpmyadmin.net/  
--  
-- Host: 127.0.0.1:3306  
-- Generation Time: Mar 05, 2025 at 08:36 PM  
-- Server version: 8.0.30  
-- PHP Version: 7.2.34  
  
SET SQL\_MODE = "NO\_AUTO\_VALUE\_ON\_ZERO";  
START TRANSACTION;  
SET time\_zone = "+00:00";  
  
  
/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET @OLD\_CHARACTER\_SET\_RESULTS=@@CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET @OLD\_COLLATION\_CONNECTION=@@COLLATION\_CONNECTION \*/;  
/\*!40101 SET NAMES utf8mb4 \*/;  
  
--  
-- Database: `University`  
--  
  
-- --------------------------------------------------------  
  
--  
-- Table structure for table `Students`  
--  
  
CREATE TABLE `Students` (  
 `id` bigint NOT NULL,  
 `first\_name` varchar(40) NOT NULL,  
 `middle\_name` varchar(40) NOT NULL,  
 `last\_name` varchar(40) NOT NULL,  
 `birth\_date` date NOT NULL,  
 `record\_book\_number` varchar(8) NOT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4\_0900\_ai\_ci;  
  
--  
-- Indexes for dumped tables  
--  
  
--  
-- Indexes for table `Students`  
--  
ALTER TABLE `Students`  
 ADD PRIMARY KEY (`id`),  
 ADD UNIQUE KEY `record\_book\_number` (`record\_book\_number`);  
  
--  
-- AUTO\_INCREMENT for dumped tables  
--  
  
--  
-- AUTO\_INCREMENT for table `Students`  
--  
ALTER TABLE `Students`  
 MODIFY `id` bigint NOT NULL AUTO\_INCREMENT;  
COMMIT;  
  
/\*!40101 SET CHARACTER\_SET\_CLIENT=@OLD\_CHARACTER\_SET\_CLIENT \*/;  
/\*!40101 SET CHARACTER\_SET\_RESULTS=@OLD\_CHARACTER\_SET\_RESULTS \*/;  
/\*!40101 SET COLLATION\_CONNECTION=@OLD\_COLLATION\_CONNECTION \*/;

9) Далі у пакеті utils створено два файла – ConnectionManager.java та PropertiesUtil.java. Клас ConnectionManager відповідальний за створення підключення до БД та повернення його у місто виклику функції, у той же час PropertiesUtil займається завантаженням налаштувань підключення до БД. Нижче наведено програмний код обох файлів.

**Програмний код файлу «ConnectionManager.sql»**

package edu.ntudp.pzks.lab6.utils;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
public class ConnectionManager {  
  
 private static final String *URL* = PropertiesUtil.*get*("db.url");  
 private static final String *USER* = PropertiesUtil.*get*("db.user");  
 private static final String *PASSWORD* = PropertiesUtil.*get*("db.password");  
  
 public static Connection open() {  
 try {  
 return DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

**Програмний код файлу «PropertiesUtil.sql»**

package edu.ntudp.pzks.lab6.utils;  
  
import java.io.FileInputStream;  
import java.io.IOException;  
import java.io.InputStream;  
import java.util.Properties;  
  
public final class PropertiesUtil {  
  
 private static final Properties *PROPERTIES* = new Properties();  
 private static final String *PATH* = "src/main/java/edu/ntudp/pzks/lab6/resources/database.properties";  
  
 static {  
 *loadProperties*();  
 }  
  
 private PropertiesUtil(){  
 }  
  
 public static String get(String key){  
 return *PROPERTIES*.getProperty(key);  
 }  
  
 private static void loadProperties(){  
 try(InputStream resourceAsStream = new FileInputStream(*PATH*)){  
 *PROPERTIES*.load(resourceAsStream);  
 } catch (IOException e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

10) Далі налаштовано клас Run, який буде надавати користувачу можливість знайти студентів з необхідним місяцем народження та створює у базі даних 30 студентів, якщо БД пуста. Нижче наведено код файлу Run.java.

package edu.ntudp.pzks.lab6;  
import edu.ntudp.pzks.lab6.controller.StudentCreator;  
import edu.ntudp.pzks.lab6.model.Student;  
import edu.ntudp.pzks.lab6.repository.StudentRepository;  
  
import java.util.List;  
import java.util.Scanner;  
  
public class Run {  
 public static void main(String[] args){  
 StudentRepository studentRepo = new StudentRepository();  
  
 List<Student> allStudents = studentRepo.getAll();  
  
 if(allStudents.isEmpty()) {  
 List<Student> students = StudentCreator.*createStudentsList*(30);  
  
 for (Student student : students) {  
 studentRepo.save(student);  
 }  
 }  
  
 Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.print("Enter the month (1-12): ");  
 int month = -1;  
  
 while (month < 1 || month > 12) {  
 System.*out*.print("Enter the month (1-12): ");  
 if (scanner.hasNextInt()) {  
 month = scanner.nextInt();  
  
 if (month < 1 || month > 12) {  
 System.*out*.println("Invalid month! Please enter a month between 1 and 12.");  
 }  
 } else {  
 System.*out*.println("Invalid input! Please enter a number between 1 and 12.");  
 scanner.next();  
 }  
 }  
  
 List<Student> studentsByMonth = studentRepo.getByMonthOfBirthDate(month);  
 for (Student student : studentsByMonth) {  
 System.*out*.println(student);  
 }  
 }  
}

11) Після попереднього кроку програму було запущено на виконання. На рисунку 7 зображено результат роботи програми, а на рисунку 8 – перевірка вмісту БД.

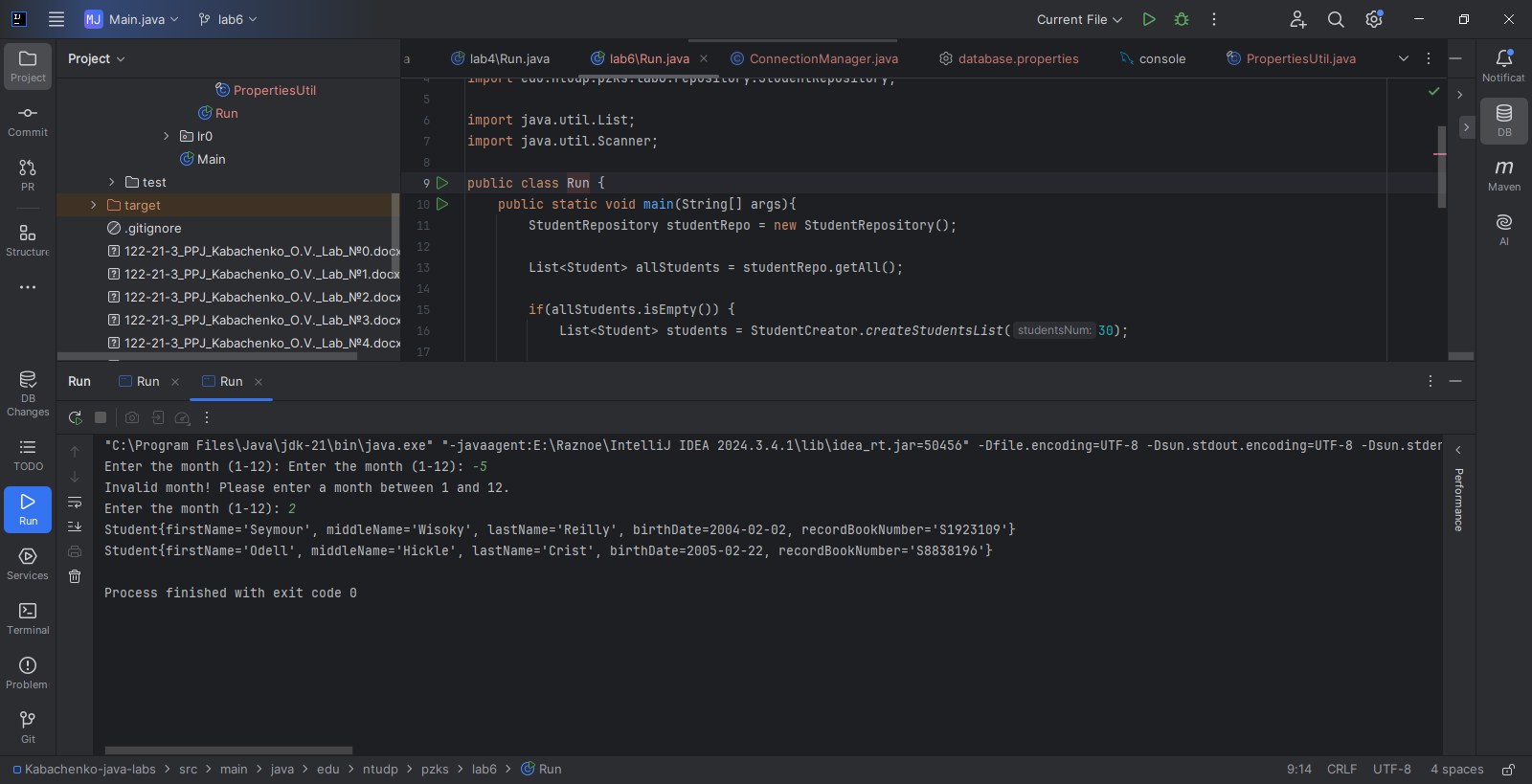


Рисунок 7 – Результат роботи програми

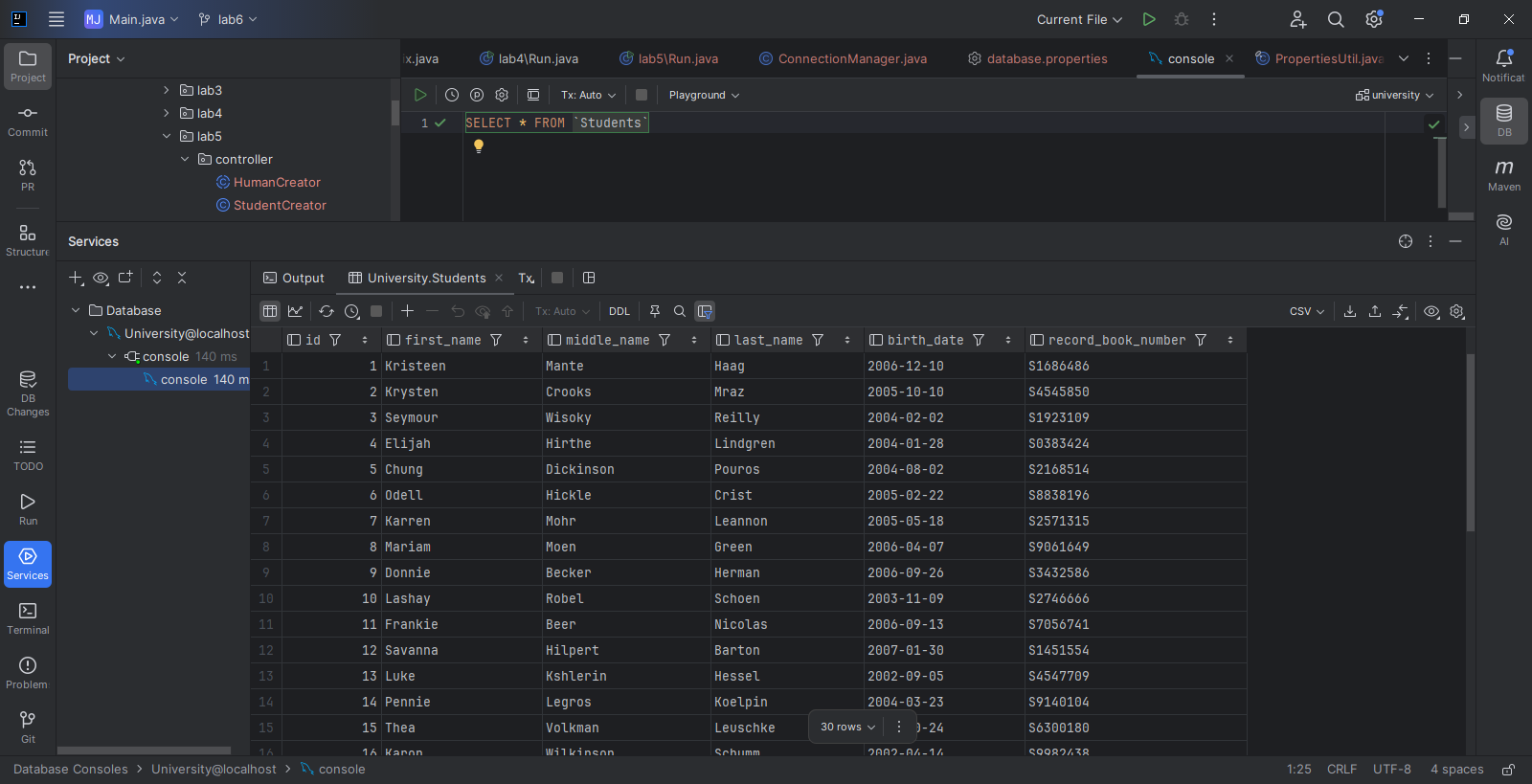


Рисунок 8 – Вміст таблиці «Students» у БД

**Висновки**

У ході виконання лабораторної роботи було створено програму, яка дозволяє виводити на екран інформацію про студентів які народилися в тому чи іншому місяці року. Для виконання цього завдання застосовано принцип, коли SQL запит формується згідно запиту, який зробив користувач і вже сервер управління баз даних вирішує, які самі студенти народилися в тому чи іншому місяці. Такий підхід дозволяє оптимізує продуктивність, оскільки передається та оброблюється менше даних, що зменшує використання пам’яті в Java. До недоліків такого підходу можливо віднести більш складний алгоритм формування SQL запиту, а також залежність від можливостей СУБД (не всі підтримують, наприклад, MONTH() у запитах).