МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО”

Факультет прикладної математики

Кафедра системного програмування і спеціалізованих комп’ютерних систем

**Лабораторна робота №** 2

з дисципліни «Бази даних і засоби управління»

«**Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL**»

Виконав: Скиба Ярослав

Студент групи КВ-93

Перевірив: Павловський В.І.

**Київ 2021**

*Метою роботи* є здобуття вмінь програмування прикладних додатків баз даних PostgreSQL.

*Загальне завдання* роботи полягає у наступному:

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

**Інформація про програму**

Посилання на репозиторій у GitHub з вихідним кодом програми та звітом: https://github.com/yaroslavskybadev/DB\_Lab2

Використана мова програмування: Java 11

Використані бібліотеки: postgresql, commons-dbcp2, commons-lang3, commons-math3

**Відомості про обрану предметну галузь з лабораторної роботи №1**

ER-діаграма, предметної області “Бібліотека”.

Модель «сутність-зв’язок» намальована у draw.io з використанням нотації Crow’s foot.

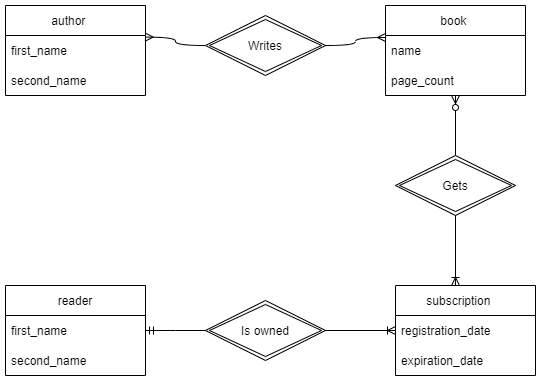


Рисунок 1 - Модель «сутність-зв’язок»

**Опис зв’язків**

У автора може бути багато книг, але обов’язково повинна бути одна. В однієї книги може бути багато авторів, але обов’язково повинен бути один. Книга може використовуватись в багатьох абонементах або може бути відсутня у всіх абонементах. Абонемент може містити багато книг, але принаймні повинна бути одна. Абонемент повинен належати виключно одному читачеві. В одного читача може бути багато абонементів, але обов’язково повинен бути хоча б один.

**Перетворення моделі у схему бази даних**

Таблиця book – відповідає за сутність книги. Таблиця author – відповідає за сутність автора. Ці дві таблиці пов’язані між собою відношення many-to-many, тому потрібно ввести ще одну таблицю book\_author. Таблиця subscription – відповідає за сутність підписки. Таблиці book & subscription пов’язані між собою відношення many-to-many, тому потрібно ввести ще одну таблицю book\_subscription. Таблиця reader – відповідає за сутність читача. Таблиці subscription & reader пов’язані між собою відношення many-to-one, тому потрібно ввести foreign key у таблиці subscription.

Схема «сутність-зв’язок» намальована у draw.io з використанням нотації Crow’s foot.

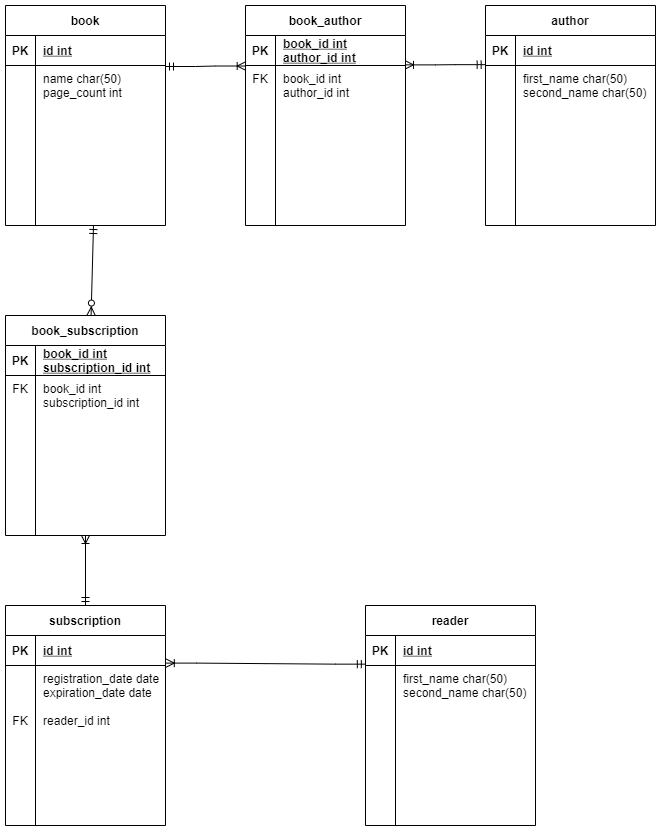
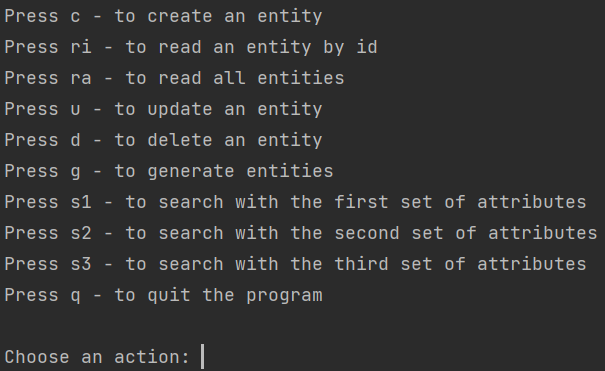


Рисунок 2 - Схема «сутність-зв’язок»

**Схема меню користувача**

****

На знімку екрана видно головне меню, яке відображається після запуску програми.

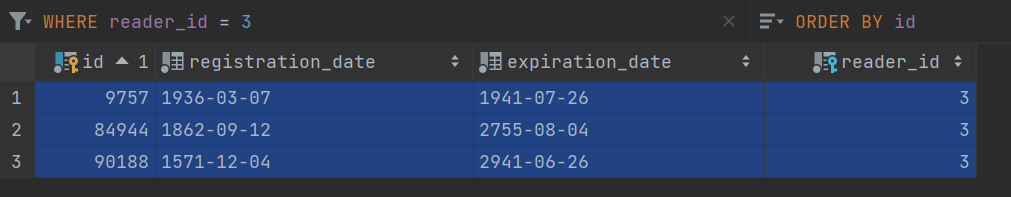
Опис команд (всі команди вводяться англійською мовою):

1. Нажміть “c” – для створення сутності
2. Нажміть “ri” – для того, щоб знайти сутність за її ідентифікатором
3. Нажміть “ra” – для того, щоб знайти всі сутності
4. Нажміть “u” — щоб оновити поля сутності
5. Нажміть “d” – щоб видалити сутність
6. Нажміть “g” – щоб згенерувати 100 000 сутностей
7. Нажміть “s1” – щоб знайти сутності за атрибутами first\_name, second\_name, registration\_date
8. Нажміть “s2” – щоб знайти сутності за атрибутами first\_name, second\_name, expiration\_date
9. Нажміть “s3” – щоб знайти сутності за атрибутами second\_name, registration\_date, expiration\_date
10. Нажміть q – щоб вийти с програми

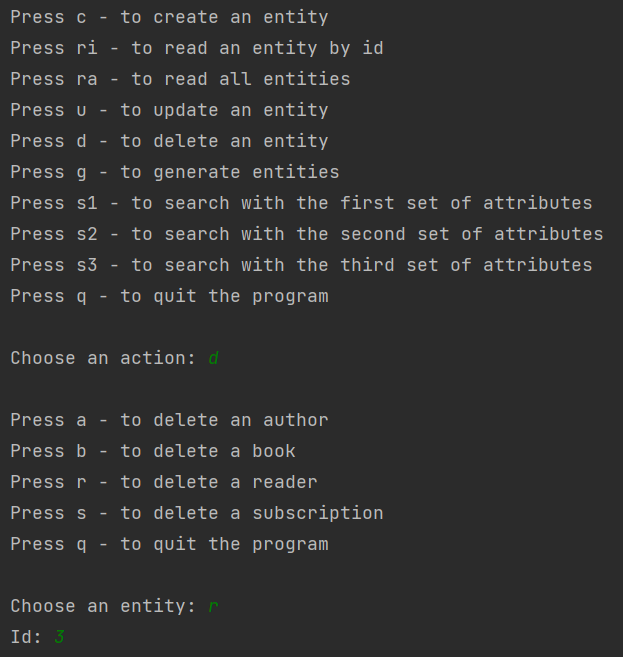
**Завдання 1**

Для перевірки роботи розглянемо лістинги та скріншоти результатів виконання операції вилучення запису з ідентифікатором 2 батьківської таблиці reader та виведення вмісту дочірньої таблиці subscription після цього вилучення.

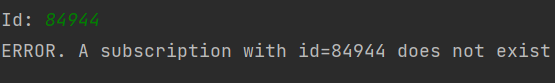
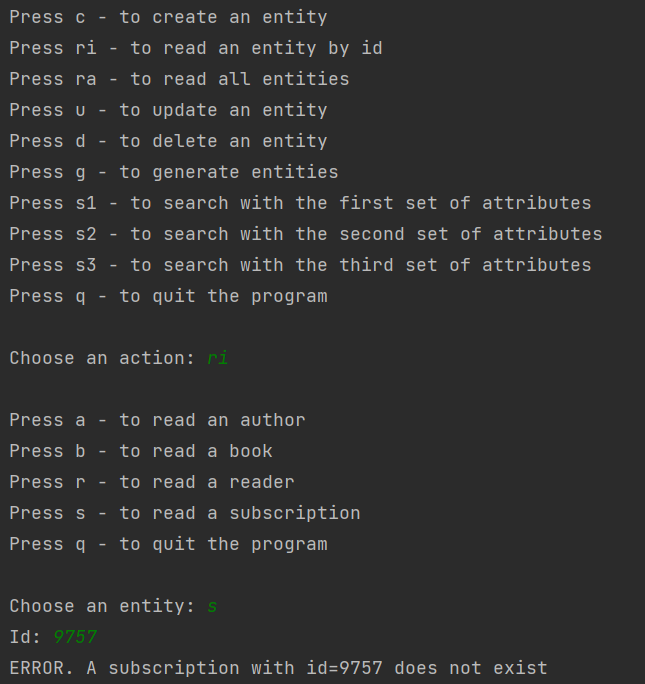
Інформація про subscription до видалення запису reader з ідентифікатором 3

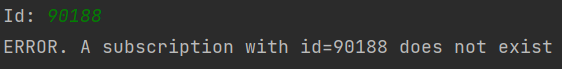


Операція видалення запису reader з ідентифікатором 3



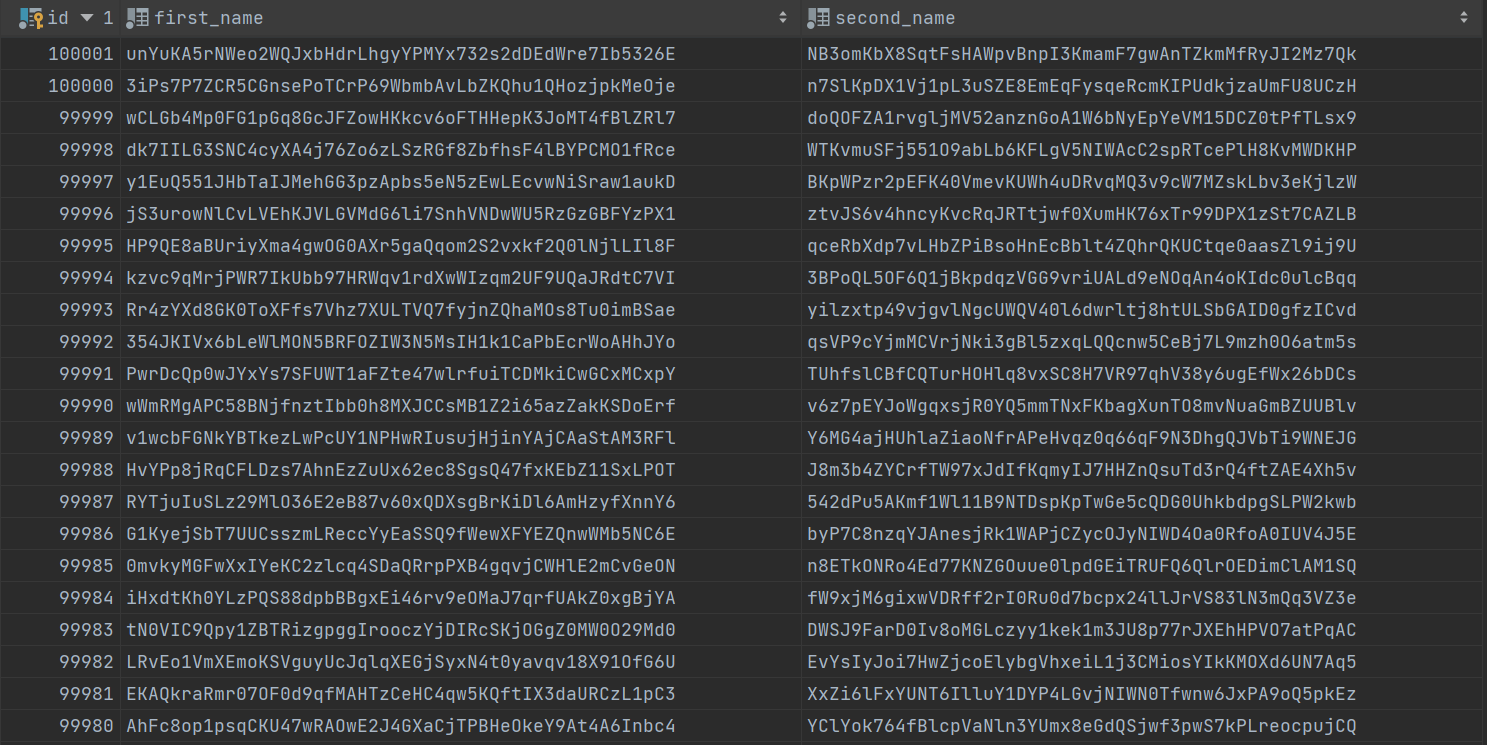
Спроба отримати видалені записи із таблиці subscription, які містили foreign keys із ідентифікатором 3 таблиці reader



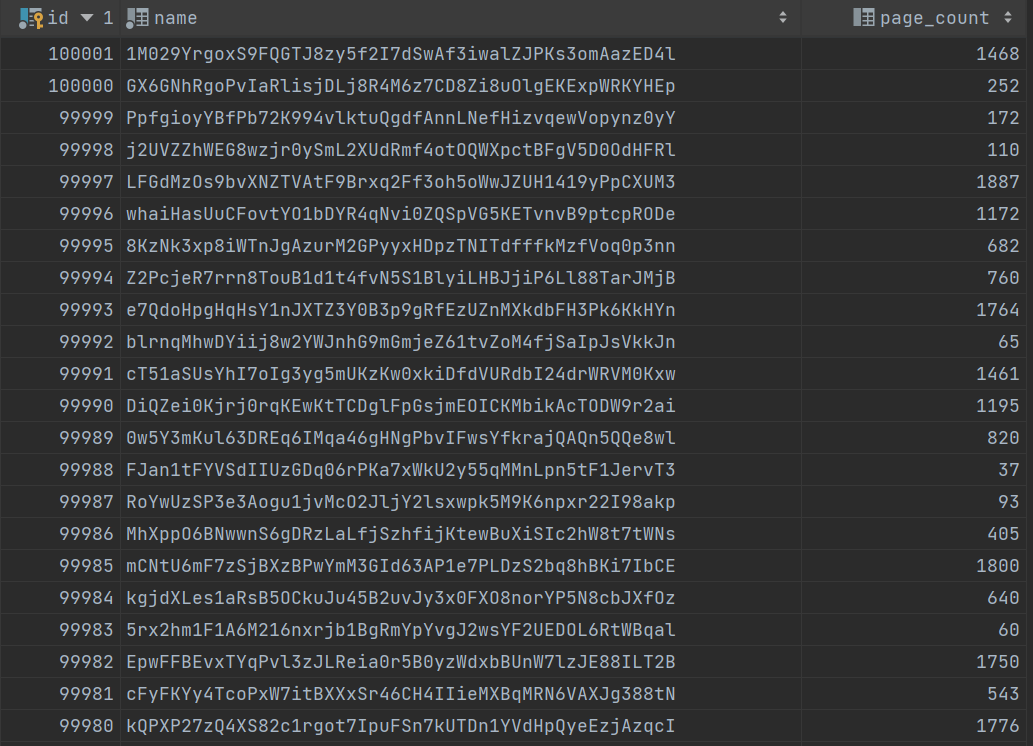


**Завдання 2**

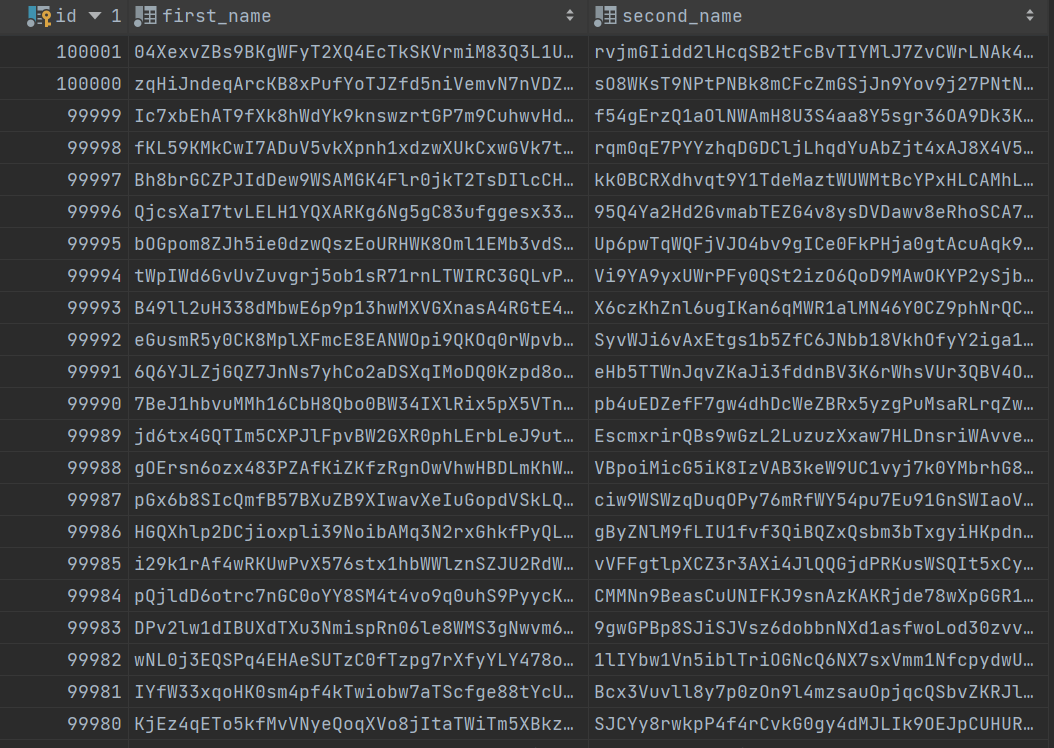
Таблиця author

****

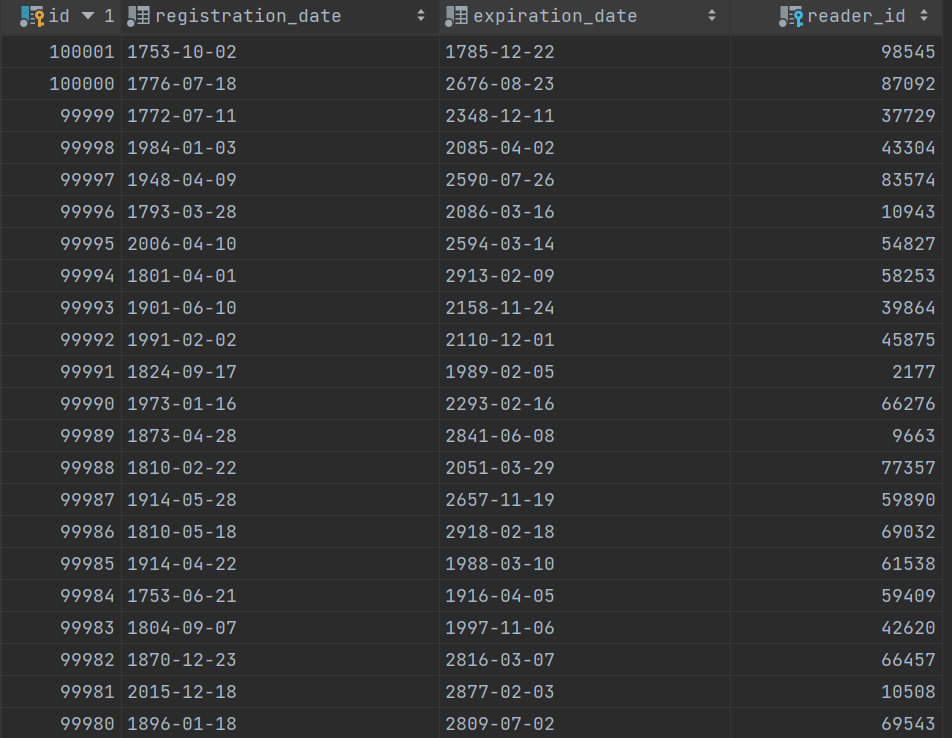
Таблиця book

****

Таблиця reader

****

Таблиця subscription

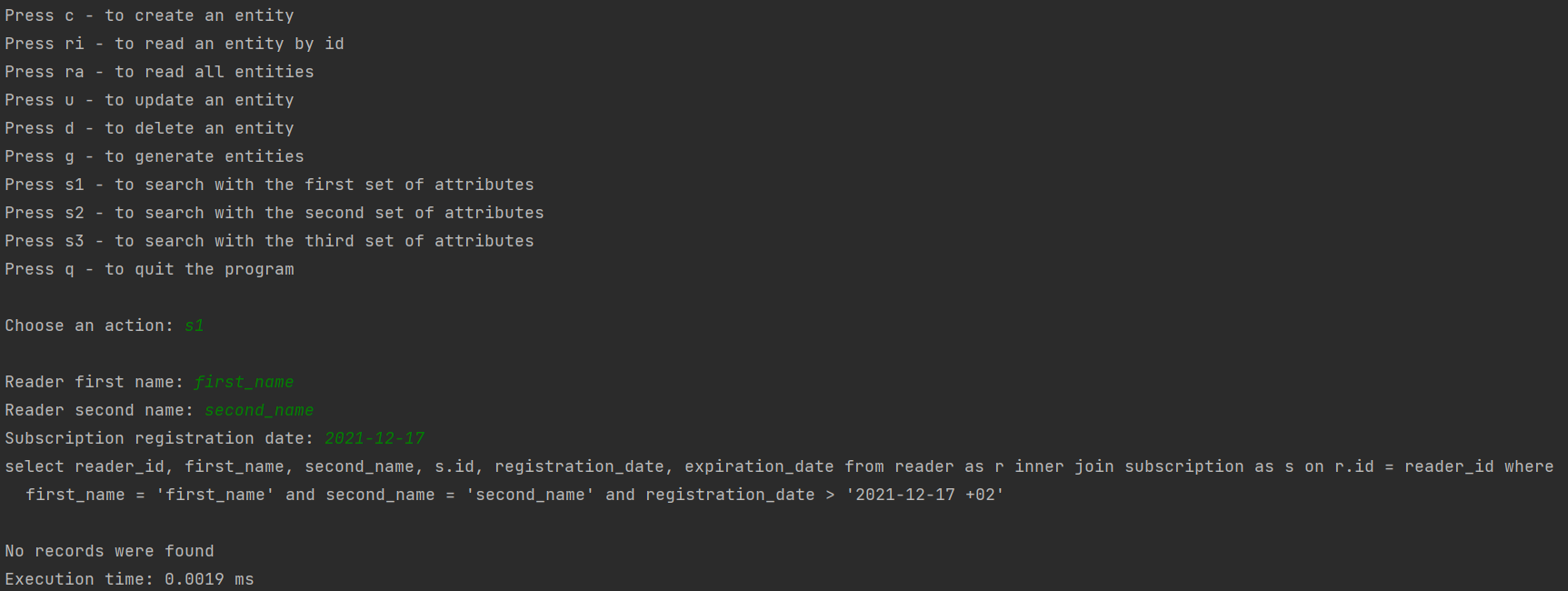
****

**Завдання 3**

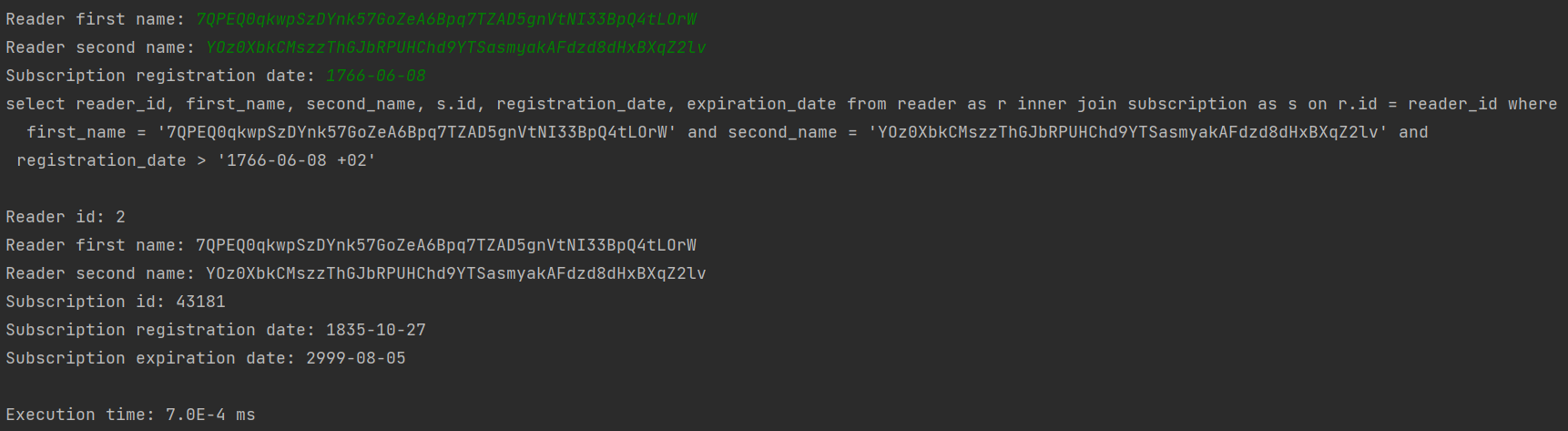
Для трьох пошукових запитів використовуються таблиці reader та subscription з різними варіаціями їх атрибутів.

*Запити з використанням атрибутів first\_name, second\_name і registration\_date*

Запит, при якому не знайдемо ні одного запису

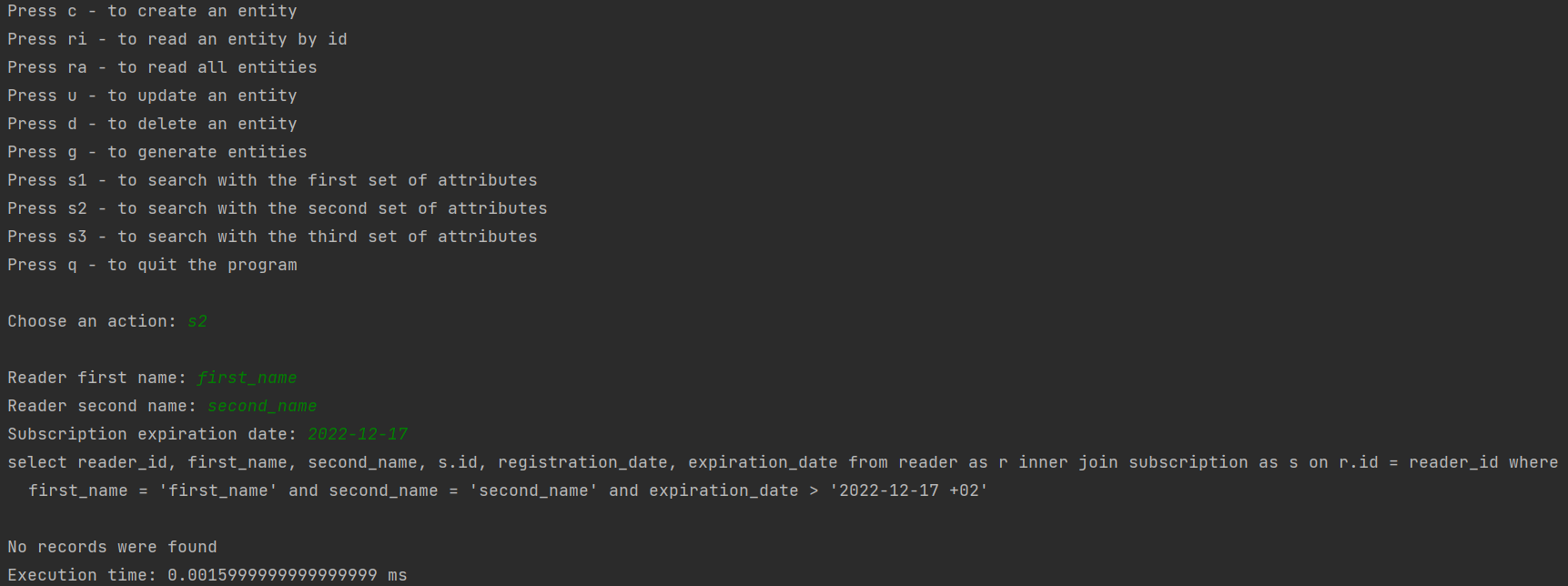
****

Запит, при якому знайдемо запис(и)

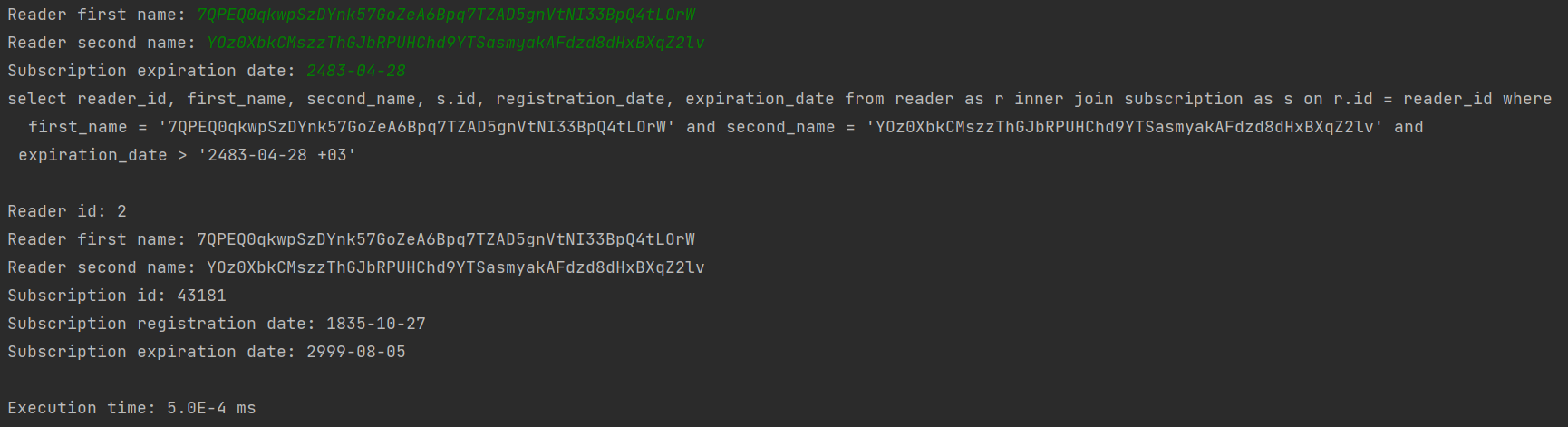
****

*Запити з використанням атрибутів first\_name, second\_name і expiration\_date*

Запит, при якому не знайдемо ні одного запису

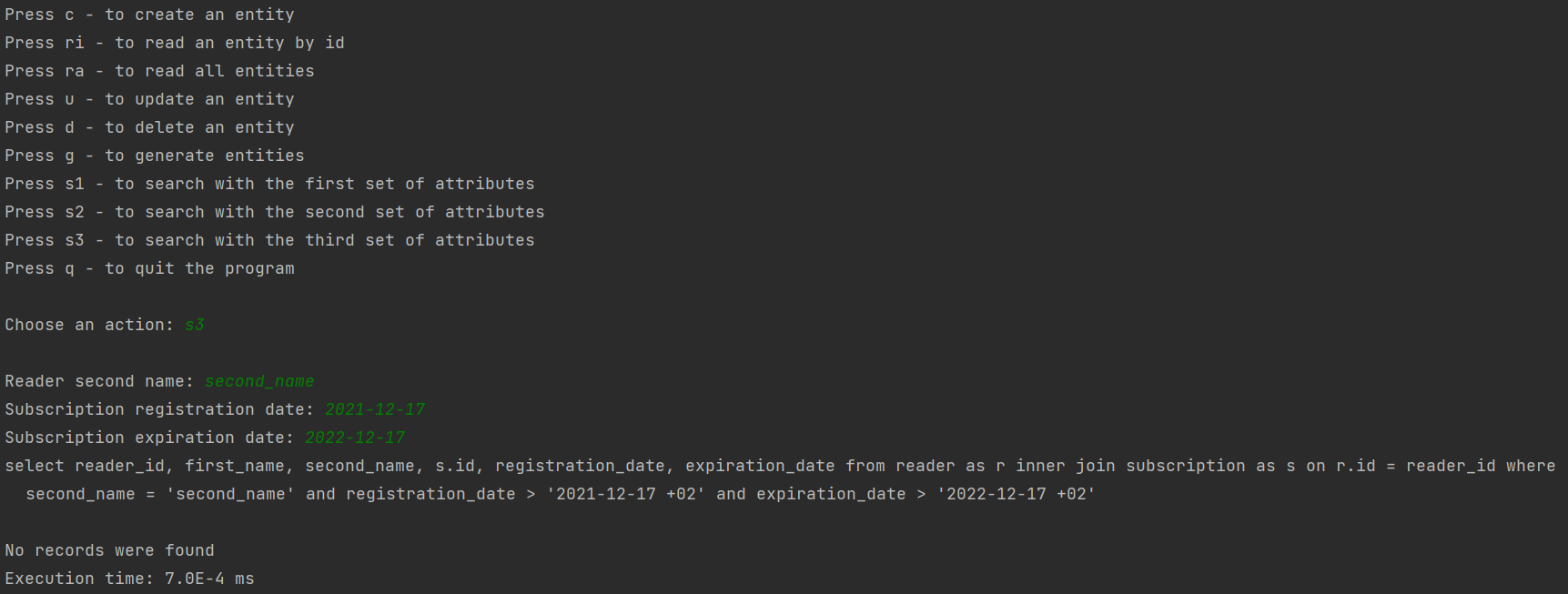
******

Запит, при якому знайдемо запис(и)

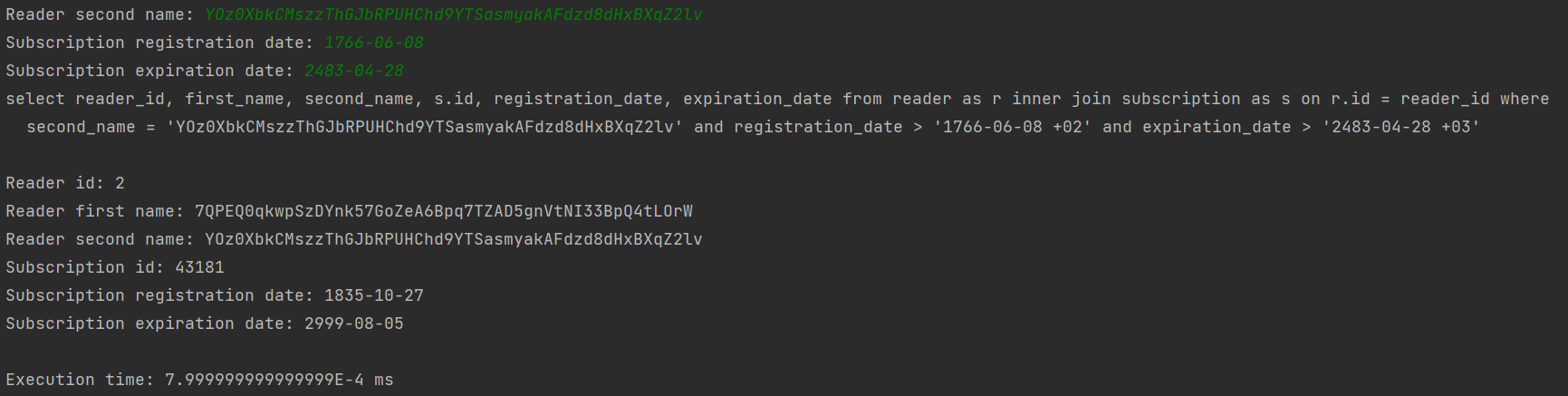
******

*Запити з використанням атрибутів second\_name, registration\_date і expiration\_date*

Запит, при якому не знайдемо ні одного запису

******

Запит, при якому знайдемо запис(и)

******

**Завдання 4**

*Dao.java*

package com.github.yaroslavskybadev.dao;import java.util.List;public interface Dao<T> { void create(T entity); void update(T entity); void remove(T entity); List<T> findAll(); T findById(Long id); void generateEntities();}

Документанція:

1. void create(T entity) - абстрактний метод, який приймає дженерік аргумент в якості об’єкта, який потрібно додати до бази даних як запис. Цей метод є контактом для класів, які будуть реалізовувати інтерфейс Dao.
2. void update(T entity) - абстрактний метод, який приймає дженерік аргумент в якості об’єкта, який є оновленням для відповідного запису із бази даних. Цей метод є контактом для класів, які будуть реалізовувати інтерфейс Dao.
3. void remove(T entity) - абстрактний метод, який приймає дженерік аргумент в якості об’єкта, запису якого — потрібно видалити із бази даних. Цей метод є контактом для класів, які будуть реалізовувати інтерфейс Dao.
4. List<T> findAll() - абстрактний метод, який повертає список дженеріків, які є записами із відповідної таблиці. Цей метод є контактом для класів, які будуть реалізовувати інтерфейс Dao.
5. T findById(Long id) – абстрактний метод, який приймає ідентифікатор запису із відповідної таблиці і повертає дженерік об’єкт. Цей метод є контактом для класів, які будуть реалізовувати інтерфейс Dao.
6. void generateEntities() - абстракний метод, який генерує 100 000 записів з випадковими значеннями атрибутів для відповідної таблиці. Цей метод є контактом для класів, які будуть реалізовувати інтерфейс Dao.

*AbstractDao.java*

package com.github.yaroslavskybadev.dao;import com.github.yaroslavskybadev.ConnectionManager;import java.sql.Connection;import java.sql.PreparedStatement;import java.sql.ResultSet;import java.sql.SQLException;import java.util.ArrayList;import java.util.List;import java.util.function.BiConsumer;import java.util.function.Supplier;public abstract class AbstractDao<T> implements Dao<T> { protected abstract T getEntity(ResultSet resultSet) throws SQLException; protected abstract String getSqlForCreate(); protected abstract String getSqlForUpdate(); protected abstract String getSqlForRemove(); protected abstract String getSqlForFindAll(); protected abstract String getSqlForFindById(); protected abstract void setValuesForCreate(PreparedStatement preparedStatement, T e); protected abstract void setValuesForUpdate(PreparedStatement preparedStatement, T e); protected abstract void setValuesForRemove(PreparedStatement preparedStatement, T e); protected abstract T createRandomizedEntity(); @Override public void create(T entity) { modify(entity, this::getSqlForCreate, this::setValuesForCreate); } @Override public void update(T entity) { modify(entity, this::getSqlForUpdate, this::setValuesForUpdate); } @Override public void remove(T entity) { modify(entity, this::getSqlForRemove, this::setValuesForRemove); } @Override public List<T> findAll() { final List<T> entityList = new ArrayList<>(); try (Connection connection = ConnectionManager.*getConnection*(); PreparedStatement preparedStatement = connection.prepareStatement(getSqlForFindAll()); ResultSet resultSet = preparedStatement.executeQuery()) { while (resultSet.next()) { entityList.add(getEntity(resultSet)); } } catch (SQLException exception) { throw new IllegalArgumentException("Some errors occurred while connecting", exception); } return entityList; } @Override public T findById(Long id) { try (Connection connection = ConnectionManager.*getConnection*(); PreparedStatement preparedStatement = connection.prepareStatement(getSqlForFindById())) { preparedStatement.setLong(1, id); try (ResultSet resultSet = preparedStatement.executeQuery()) { if (!resultSet.next()) { throw new IllegalStateException("No rows were found"); } return getEntity(resultSet); } } catch (SQLException exception) { throw new IllegalArgumentException("Some errors occurred while connecting", exception); } } @Override public void generateEntities() { for (int i = 0; i < 100000; i++) { create(createRandomizedEntity()); } } protected void modify(T entity, Supplier<String> sqlSupplier, BiConsumer<PreparedStatement, T> valuesBiConsumer) { try (Connection connection = ConnectionManager.*getConnection*()) { try (PreparedStatement preparedStatement = connection.prepareStatement(sqlSupplier.get())) { valuesBiConsumer.accept(preparedStatement, entity); if (preparedStatement.executeUpdate() == 0) { throw new IllegalStateException("Rows to modify were not found"); } } } catch (SQLException exception) { throw new IllegalArgumentException("Some errors occurred while connecting", exception); } }}

Документація

1. protected abstract T getEntity(ResultSet resultSet) throws SQLException — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
2. protected abstract String getSqlForCreate() — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
3. protected abstract String getSqlForUpdate() — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
4. protected abstract String getSqlForRemove() — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
5. protected abstract String getSqlForFindAll() — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
6. protected abstract String getSqlForFindById() — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
7. protected abstract void setValuesForCreate(PreparedStatement preparedStatement, T e) — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
8. protected abstract void setValuesForUpdate(PreparedStatement preparedStatement, T e) — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
9. protected abstract void setValuesForRemove(PreparedStatement preparedStatement, T e) — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
10. protected abstract T createRandomizedEntity() — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.
11. protected void modify(T entity, Supplier<String> sqlSupplier, BiConsumer<PreparedStatement, T> valuesBiConsumer) — абстрактний метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.

*AuthorDao.java*

package com.github.yaroslavskybadev.dao.impl;import com.github.yaroslavskybadev.ConnectionManager;import com.github.yaroslavskybadev.dao.AbstractDao;import com.github.yaroslavskybadev.dto.Author;import com.github.yaroslavskybadev.dto.Book;import org.apache.commons.lang3.RandomStringUtils;import java.sql.Connection;import java.sql.PreparedStatement;import java.sql.ResultSet;import java.sql.SQLException;public class AuthorDao extends AbstractDao<Author> { private static final BookDao *BOOK\_DAO* = new BookDao(); private long lastId = findAll().stream().mapToLong(Author::getId).max().getAsLong(); @Override protected Author getEntity(ResultSet resultSet) throws SQLException { final Author author = new Author(); author.setId(resultSet.getLong("id")); author.setFirstName(resultSet.getString("first\_name")); author.setSecondName(resultSet.getString("second\_name")); try (Connection connection = ConnectionManager.*getConnection*(); PreparedStatement preparedStatement = connection.prepareStatement("select book\_id from book\_author where author\_id = ?")) { preparedStatement.setLong(1, author.getId()); try (ResultSet bookResultSet = preparedStatement.executeQuery()) { while (bookResultSet.next()) { author.addBook(*BOOK\_DAO*.findById(bookResultSet.getLong("book\_id"))); } } } return author; } @Override protected String getSqlForCreate() { return "insert into author (id, first\_name, second\_name) values (?, ?, ?)"; } @Override protected String getSqlForUpdate() { return "update author set id = ?, first\_name = ?, second\_name = ? where id = ?"; } @Override protected String getSqlForRemove() { return "delete from author where id = ?"; } @Override protected String getSqlForFindAll() { return "select id, first\_name, second\_name from author"; } @Override protected String getSqlForFindById() { return "select id, first\_name, second\_name from author where id = ?"; } @Override protected void setValuesForCreate(PreparedStatement preparedStatement, Author e) { try { preparedStatement.setLong(1, e.getId()); setSpecificValuesForCreate(preparedStatement, e); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected void setValuesForUpdate(PreparedStatement preparedStatement, Author e) { final Long id = e.getId(); try { preparedStatement.setLong(1, id); setSpecificValuesForCreate(preparedStatement, e); preparedStatement.setLong(4, id); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected void setValuesForRemove(PreparedStatement preparedStatement, Author e) { try { preparedStatement.setLong(1, e.getId()); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected Author createRandomizedEntity() { final Author author = new Author(); author.setId(++lastId); final int stringLength = 50; author.setFirstName(RandomStringUtils.*randomAlphanumeric*(stringLength)); author.setSecondName(RandomStringUtils.*randomAlphanumeric*(stringLength)); return author; } public void addBooks(Author author) { try (Connection connection = ConnectionManager.*getConnection*(); PreparedStatement preparedStatement = connection.prepareStatement("insert into book\_author values (?, ?)")) { for (Book book : author.getBookList()) { preparedStatement.setLong(1, book.getId()); preparedStatement.setLong(2, author.getId()); preparedStatement.addBatch(); } preparedStatement.executeBatch(); } catch (SQLException exception) { throw new IllegalArgumentException("Some errors occurred while connecting", exception); } } private void setSpecificValuesForCreate(PreparedStatement preparedStatement, Author e) throws SQLException { preparedStatement.setString(2, e.getFirstName()); preparedStatement.setString(3, e.getSecondName()); }}

*BookDao.java*

package com.github.yaroslavskybadev.dao.impl;import com.github.yaroslavskybadev.ConnectionManager;import com.github.yaroslavskybadev.dao.AbstractDao;import com.github.yaroslavskybadev.dto.Author;import com.github.yaroslavskybadev.dto.Book;import com.github.yaroslavskybadev.dto.Subscription;import org.apache.commons.lang3.RandomStringUtils;import java.sql.Connection;import java.sql.PreparedStatement;import java.sql.ResultSet;import java.sql.SQLException;import java.util.Random;import java.util.function.LongSupplier;public class BookDao extends AbstractDao<Book> { private static final Random *RANDOM* = new Random(); private long lastId = findAll().stream().mapToLong(Book::getId).max().getAsLong(); @Override protected Book getEntity(ResultSet resultSet) throws SQLException { final Book book = new Book(); book.setId(resultSet.getLong("id")); book.setName(resultSet.getString("name")); book.setPageCount(resultSet.getInt("page\_count")); return book; } @Override protected String getSqlForCreate() { return "insert into book (id, name, page\_count) values (?, ?, ?)"; } @Override protected String getSqlForUpdate() { return "update book set id = ?, name = ?, page\_count = ? where id = ?"; } @Override protected String getSqlForRemove() { return "delete from book where id = ?"; } @Override protected String getSqlForFindAll() { return "select id, name, page\_count from book"; } @Override protected String getSqlForFindById() { return "select id, name, page\_count from book where id = ?"; } @Override protected void setValuesForCreate(PreparedStatement preparedStatement, Book e) { try { preparedStatement.setLong(1, e.getId()); setSpecificValuesForCreate(preparedStatement, e); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected void setValuesForUpdate(PreparedStatement preparedStatement, Book e) { final Long id = e.getId(); try { preparedStatement.setLong(1, id); setSpecificValuesForCreate(preparedStatement, e); preparedStatement.setLong(4, id); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected void setValuesForRemove(PreparedStatement preparedStatement, Book e) { try { preparedStatement.setLong(1, e.getId()); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected Book createRandomizedEntity() { final Book book = new Book(); book.setId(++lastId); book.setName(RandomStringUtils.*randomAlphanumeric*(50)); book.setPageCount(*RANDOM*.nextInt(2000)); return book; } public void addAuthors(Book book) { try (Connection connection = ConnectionManager.*getConnection*(); PreparedStatement preparedStatement = connection.prepareStatement("insert into book\_author values (?, ?)")) { for (Author author : book.getAuthorList()) { addBatch(preparedStatement, book, author::getId); } preparedStatement.executeBatch(); } catch (SQLException exception) { throw new IllegalArgumentException("Some errors occurred while connecting", exception); } } public void addSubscriptions(Book book) { try (Connection connection = ConnectionManager.*getConnection*(); PreparedStatement preparedStatement = connection.prepareStatement("insert into book\_subscription values (?, ?)")) { for (Subscription subscription : book.getSubscriptionList()) { addBatch(preparedStatement, book, subscription::getId); } preparedStatement.executeBatch(); } catch (SQLException exception) { throw new IllegalArgumentException("Some errors occurred while connecting", exception); } } private void addBatch(PreparedStatement preparedStatement, Book book, LongSupplier idSupplier) throws SQLException { preparedStatement.setLong(1, book.getId()); preparedStatement.setLong(2, idSupplier.getAsLong()); preparedStatement.addBatch(); } private void setSpecificValuesForCreate(PreparedStatement preparedStatement, Book book) throws SQLException { preparedStatement.setString(2, book.getName()); preparedStatement.setInt(3, book.getPageCount()); }}

*ReaderDao.java*

package com.github.yaroslavskybadev.dao.impl;import com.github.yaroslavskybadev.dao.AbstractDao;import com.github.yaroslavskybadev.dto.Reader;import org.apache.commons.lang3.RandomStringUtils;import java.sql.PreparedStatement;import java.sql.ResultSet;import java.sql.SQLException;public class ReaderDao extends AbstractDao<Reader> { private long lastId = findAll().stream().mapToLong(Reader::getId).max().getAsLong(); @Override protected Reader getEntity(ResultSet resultSet) throws SQLException { final Reader reader = new Reader(); reader.setId(resultSet.getLong("id")); reader.setFirstName(resultSet.getString("first\_name")); reader.setSecondName(resultSet.getString("second\_name")); return reader; } @Override protected String getSqlForCreate() { return "insert into reader (id, first\_name, second\_name) values (?, ?, ?)"; } @Override protected String getSqlForUpdate() { return "update reader set id = ?, first\_name = ?, second\_name = ? where id = ?"; } @Override protected String getSqlForRemove() { return "delete from reader where id = ?"; } @Override protected String getSqlForFindAll() { return "select id, first\_name, second\_name from reader"; } @Override protected String getSqlForFindById() { return "select id, first\_name, second\_name from reader where id = ?"; } @Override protected void setValuesForCreate(PreparedStatement preparedStatement, Reader e) { try { preparedStatement.setLong(1, e.getId()); setSpecificValuesForCreate(preparedStatement, e); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected void setValuesForUpdate(PreparedStatement preparedStatement, Reader e) { final Long id = e.getId(); try { preparedStatement.setLong(1, id); setSpecificValuesForCreate(preparedStatement, e); preparedStatement.setLong(4, id); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected void setValuesForRemove(PreparedStatement preparedStatement, Reader e) { try { preparedStatement.setLong(1, e.getId()); } catch (SQLException ex) { throw new IllegalStateException(ex); } } @Override protected Reader createRandomizedEntity() { final Reader reader = new Reader(); reader.setId(++lastId); final int stringLength = 50; reader.setFirstName(RandomStringUtils.*randomAlphanumeric*(stringLength)); reader.setSecondName(RandomStringUtils.*randomAlphanumeric*(stringLength)); return reader; } private void setSpecificValuesForCreate(PreparedStatement preparedStatement, Reader e) throws SQLException { preparedStatement.setString(2, e.getFirstName()); preparedStatement.setString(3, e.getSecondName()); }}

*SubscriptionDao.java*

package com.github.yaroslavskybadev.dao.impl;  
  
import com.github.yaroslavskybadev.ConnectionManager;  
import com.github.yaroslavskybadev.dao.AbstractDao;  
import com.github.yaroslavskybadev.dto.Book;  
import com.github.yaroslavskybadev.dto.Reader;  
import com.github.yaroslavskybadev.dto.Subscription;  
import org.apache.commons.math3.random.RandomDataGenerator;  
  
import java.sql.Connection;  
import java.sql.Date;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.time.Instant;  
import java.util.concurrent.ThreadLocalRandom;  
  
public class SubscriptionDao extends AbstractDao<Subscription> {  
 private static final RandomDataGenerator *RANDOM\_DATA\_GENERATOR* = new RandomDataGenerator();  
 private static final long *LAST\_READER\_ID* = new ReaderDao().findAll().stream().mapToLong(Reader::getId).max().getAsLong();  
  
 private long lastId = findAll().stream().mapToLong(Subscription::getId).max().getAsLong();  
  
 @Override  
 protected Subscription getEntity(ResultSet resultSet) throws SQLException {  
 final Subscription subscription = new Subscription();  
 subscription.setId(resultSet.getLong("id"));  
 subscription.setReaderId(resultSet.getLong("reader\_id"));  
 subscription.setRegistrationDate(resultSet.getDate("registration\_date"));  
 subscription.setExpirationDate(resultSet.getDate("expiration\_date"));  
  
 return subscription;  
 }  
  
 @Override  
 protected String getSqlForCreate() {  
 return "insert into subscription (id, reader\_id, registration\_date, expiration\_date) values (?, ?, ?, ?)";  
 }  
  
 @Override  
 protected String getSqlForUpdate() {  
 return "update subscription set id = ?, reader\_id = ?, registration\_date = ?, expiration\_date = ? where id = ?";  
 }  
  
 @Override  
 protected String getSqlForRemove() {  
 return "delete from subscription where id = ?";  
 }  
  
 @Override  
 protected String getSqlForFindAll() {  
 return "select id, reader\_id, registration\_date, expiration\_date from subscription";  
 }  
  
 @Override  
 protected String getSqlForFindById() {  
 return "select id, reader\_id, registration\_date, expiration\_date from subscription where id = ?";  
 }  
  
 @Override  
 protected void setValuesForCreate(PreparedStatement preparedStatement, Subscription e) {  
 try {  
 preparedStatement.setLong(1, e.getId());  
 setSpecificValuesForCreate(preparedStatement, e);  
 } catch (SQLException ex) {  
 throw new IllegalStateException(ex);  
 }  
 }  
  
 @Override  
 protected void setValuesForUpdate(PreparedStatement preparedStatement, Subscription e) {  
 final Long id = e.getId();  
  
 try {  
 preparedStatement.setLong(1, id);  
 setSpecificValuesForCreate(preparedStatement, e);  
 preparedStatement.setLong(5, id);  
 } catch (SQLException ex) {  
 throw new IllegalStateException(ex);  
 }  
 }  
  
 @Override  
 protected void setValuesForRemove(PreparedStatement preparedStatement, Subscription e) {  
 try {  
 preparedStatement.setLong(1, e.getId());  
 } catch (SQLException ex) {  
 throw new IllegalStateException(ex);  
 }  
 }  
  
 @Override  
 protected Subscription createRandomizedEntity() {  
 final Subscription subscription = new Subscription();  
 subscription.setId(++lastId);  
 subscription.setReaderId(*RANDOM\_DATA\_GENERATOR*.nextLong(1, *LAST\_READER\_ID*));  
  
 final ThreadLocalRandom current = ThreadLocalRandom.*current*();  
 final long registrationTime = current.nextLong(Date.*valueOf*("1750-1-1").getTime(), new java.util.Date().getTime());  
  
 subscription.setRegistrationDate(new Date(registrationTime));  
 subscription.setExpirationDate(new Date(current.nextLong(registrationTime, Date.*valueOf*("3000-1-1").getTime())));  
  
 return subscription;  
 }  
  
 public void addBooks(Subscription subscription) {  
 try (Connection connection = ConnectionManager.*getConnection*();  
 PreparedStatement preparedStatement = connection.prepareStatement("insert into book\_subscription values (?, ?)")) {  
  
 for (Book book : subscription.getBookList()) {  
 preparedStatement.setLong(1, book.getId());  
 preparedStatement.setLong(2, subscription.getId());  
  
 preparedStatement.addBatch();  
 }  
  
 preparedStatement.executeBatch();  
 } catch (SQLException exception) {  
 throw new IllegalArgumentException("Some errors occurred while connecting", exception);  
 }  
 }  
  
 private void setSpecificValuesForCreate(PreparedStatement preparedStatement, Subscription e) throws SQLException {  
 preparedStatement.setLong(2, e.getReaderId());  
 preparedStatement.setDate(3, e.getRegistrationDate());  
 preparedStatement.setDate(4, e.getExpirationDate());  
 }  
}

*SearcherDao.java*

package com.github.yaroslavskybadev.dao;import com.github.yaroslavskybadev.ConnectionManager;import java.sql.Connection;import java.sql.Date;import java.sql.PreparedStatement;import java.sql.ResultSet;import java.sql.SQLException;import java.util.ArrayList;import java.util.List;import java.util.Map;public class Searcher { public List<Map<String, Object>> searchWithFirstNameAndSecondNameAndRegistrationDate(String firstName, String secondName, Date registrationDate) { return searchWithReaderAndSubscription( "select reader\_id, first\_name, second\_name, s.id, registration\_date, expiration\_date from reader as r " + "inner join subscription as s on r.id = reader\_id " + "where first\_name = ? and second\_name = ? and registration\_date > ?", new Object[]{firstName, secondName, registrationDate} ); } public List<Map<String, Object>> searchWithFirstNameAndSecondNameAndExpirationDate(String firstName, String secondName, Date expirationDate) { return searchWithReaderAndSubscription( "select reader\_id, first\_name, second\_name, s.id, registration\_date, expiration\_date from reader as r " + "inner join subscription as s on r.id = reader\_id " + "where first\_name = ? and second\_name = ? and expiration\_date > ?", new Object[]{firstName, secondName, expirationDate} ); } public List<Map<String, Object>> searchWithSecondNameAndRegistrationDateAndExpirationDate(String secondName, Date registrationDate, Date expirationDate) { return searchWithReaderAndSubscription( "select reader\_id, first\_name, second\_name, s.id, registration\_date, expiration\_date from reader as r " + "inner join subscription as s on r.id = reader\_id " + "where second\_name = ? and registration\_date > ? and expiration\_date > ?", new Object[]{secondName, registrationDate, expirationDate} ); } private List<Map<String, Object>> searchWithReaderAndSubscription(String sql, Object[] attributeValues) { try (Connection connection = ConnectionManager.*getConnection*(); PreparedStatement preparedStatement = connection.prepareStatement(sql)) { for (int i = 1; i <= attributeValues.length; i++) { preparedStatement.setObject(i, attributeValues[i - 1]); } final List<Map<String, Object>> resultList = new ArrayList<>(); final String readerIdKey = "reader\_id"; final String firstNameKey = "first\_name"; final String secondNameKey = "second\_name"; final String subscriptionIdKey = "id"; final String registrationDateKey = "registration\_date"; final String expirationDateKey = "expiration\_date"; try (ResultSet resultSet = preparedStatement.executeQuery()) { while (resultSet.next()) { resultList.add(Map.*of*( readerIdKey, resultSet.getLong(readerIdKey), firstNameKey, resultSet.getString(firstNameKey), secondNameKey, resultSet.getString(secondNameKey), subscriptionIdKey, resultSet.getLong(subscriptionIdKey), registrationDateKey, resultSet.getDate(registrationDateKey), expirationDateKey, resultSet.getDate(expirationDateKey) )); } } System.*out*.println(preparedStatement); return resultList; } catch (SQLException e) { throw new IllegalArgumentException("Some errors occurred while connecting", e); } }}

Документація:

1. public List<Map<String, Object>> searchWithFirstNameAndSecondNameAndRegistrationDate(String firstName, String secondName, Date registrationDate) — метод, який виконує пошук записів із таблиць reader і subscription. В якості параметрів приймає значення атрибутів first\_name, second\_name та registration\_date, і повертає список мап з іменем атрибута в якості ключа та значенням цього атрибута в якості значення.
2. public List<Map<String, Object>> searchWithFirstNameAndSecondNameAndExpirationDate(String firstName, String secondName, Date expirationDate) — метод, який виконує пошук записів із таблиць reader і subscription. В якості параметрів приймає значення атрибутів first\_name, second\_name та expiration\_date, і повертає список мап з іменем атрибута в якості ключа та значенням цього атрибута в якості значення.
3. public List<Map<String, Object>> searchWithSecondNameAndRegistrationDateAndExpirationDate(String secondName, Date registrationDate, Date expirationDate) — метод, який виконує пошук записів із таблиць reader і subscription. В якості параметрів приймає значення атрибутів second\_name, registration\_date та expiration\_date, і повертає список мап з іменем атрибута в якості ключа та значенням цього атрибута в якості значення.
4. private List<Map<String, Object>> searchWithReaderAndSubscription(String sql, Object[] attributeValues) - метод, який не є частиною публічного API і використовується, щоб зменшити кількість дублікату в коді.