НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ
Кафедра системного програмування та спеціалізованих комп'ютерних систем

Лабораторна робота №2
з дисципліни
«Бази даних і засоби управління»

Виконав студент III курсу
ФПМ групи КП-73
Заяць Євген Євгенович
Перевірив: Петрашенко А.В.

Київ – 2019

Ознайомлення з базовими операціями СУБД PostgreSQL

Завдання роботи полягає у наступному:

1. Виконати нормалізацію бази даних, яка була створена у лабораторній роботі №1, до третьої нормальної форми (3НФ);
2. Реалізувати функціональні вимоги, наведені нижче.

Функціональні вимоги

1. Реалізувати внесення, редагування та вилучення даних у базі засобами консольного інтерфейсу;
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі;
3. Забезпечити реалізацію пошуку за двома-трьома атрибутами з двох сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як перелічення, для логічного типу – значення True/False, для дат – у рамках діапазону дат;
4. Забезпечити реалізацію повнотекстового пошуку за будь-яким текстовим атрибутом бази даних засобами PostgreSQL з виділенням знайденого фрагменту.

Вимоги до інтерфейсу користувача

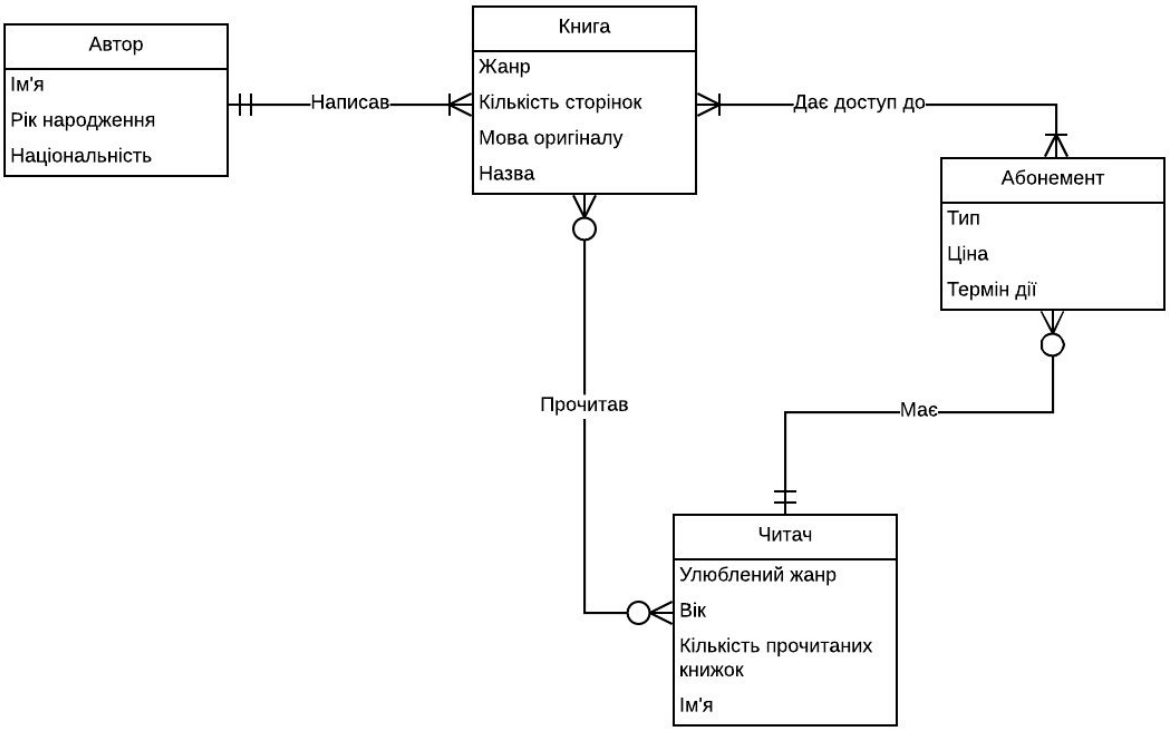
1. Використовувати консольний інтерфейс користувача.

Варіант 9

Пошук за атрибутами – діапазон чисел та логічний тип.

Повнотекстовий пошук – слово не входить та обов’язкове входження слова.

Нормалізована модель даних БД «Сервіс запитання та відповіді»



Усі таблиці (відношення) знаходяться в 3 НФ тому, що у кожній із них:

1. Всі атрибути є атомарними та відсутні повторення рядків (1НФ);
2. Первинний ключ складається лише з одного атрибуту (2НФ);
3. Кожний не первинний атрибут не є транзитивно залежним від первинного ключа (3НФ).

Опис програми

Програма створена для управління базою даних за допомогою базових операцій СУБД PostgreSQL та реалізовує функціональні вимоги, що наведені у завданні. Програма складається з 5 модулів:

1. Main.java – точка входу до програми, містить засоби обробки виключень та повідомлення помилок, викликає функцію головного меню із Controller.java;
2. DAO.java – клас DAO, який містить методи для управління даними програми та БД а також виконує підключення до БД;
3. View.java – клас View, який містить методи для виводу результатів роботи Model на екран;
4. Controller.java – клас Controller, який містить методи для контролю даних введених користувачем та контролю викликів методів Model.

Структура меню програми

Меню програми можна розглядати як її концептуальну модель

Лістинг модуля Main.java

```
package com.lab2;

import com.lab2.controller.Controller;
import com.lab2.dao.DAO;
import com.lab2.view.View;

import java.io.FileInputStream;
import java.io.IOException;
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;

public class Main {

    public static void main(String[] args) {

        Properties props = new Properties();
        Logger lgr = Logger.getLogger(Main.class.getName());

        try {
            FileInputStream in = new FileInputStream(System.getProperty("user.dir") + "/db.properties");
            props.load(in);
            in.close();
        } catch (IOException ex) {
            lgr.log(Level.SEVERE, ex.getMessage(), ex);
            return;
        }

        String url = props.getProperty("db.url");
        String username = props.getProperty("db.user");
        String password = props.getProperty("db.password");

        try {
            View view = new View();
            DAO dao = new DAO();
            dao.connect(url, username, password);

            Controller controller = new Controller(view, dao);
            controller.mainMenu();
        } catch (Exception ex) {
            lgr.log(Level.SEVERE, ex.getMessage(), ex);
        }
    }
}
```

Лістинг модуля Controller.java

```
package com.lab2.controller;

import com.lab2.dao.IDAO;
import com.lab2.model.*;
import com.lab2.view.View;

import java.security.SecureRandom;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;
import java.util.List;

public class Controller {

  private View view;
  private IDAO dao;

  public Controller(View view, IDAO dao) {
      this.view = view;
      this.dao = dao;
  }

  public void mainMenu() throws SQLException, IllegalAccessException {
      boolean exit = false;
      while (!exit) {
          view.clearConsole();
          view.printMenu();
          int operation = view.getOption();
          int entity = 0;
          switch (operation) {
              case 0: {
                  view.clearConsole();
                  view.printTables();
                  entity = view.getOption();
                  view.clearConsole();
                  randomize(entity);
                  break;
              }
              case 1: {
                  view.clearConsole();
                  view.printTables();
                  entity = view.getOption();
                  view.clearConsole();
                  selectAll(entity);
                  break;
              }
              case 2: {
                  view.clearConsole();
                  view.printTables();
                  entity = view.getOption();
                  view.clearConsole();
```

```
                view.askId();
                long id = view.getOption();
                selectById(id, entity);
                break;
            }
            case 3: {
                view.clearConsole();
                view.printTables();
                entity = view.getOption();
                view.clearConsole();
                insert(entity);
                break;
            }
            case 4: {
                view.clearConsole();
                view.printTables();
                entity = view.getOption();
                view.clearConsole();
                update(entity);
                break;
            }
            case 5: {
                view.clearConsole();
                view.printTables();
                entity = view.getOption();
                view.clearConsole();
                view.askId();
                long id = view.getOption();
                delete(id, entity);
                break;
            }
            case 6: {
                view.clearConsole();
                List<Book> books = dao.searchWord(
                        view.getStringOption("word"),
                        view.getBooleanOption("including(true/false)"));
                view.printBooks(books);
                break;
            }
            case 7: {
                view.clearConsole();
                ResultSet resultSet = dao.joinedBookSearch(
                        view.getIntOption("pages count start"),
                        view.getIntOption("pages count end"),
                        view.getBooleanOption("author alive(true/false)")
                );
                view.joinedSearchResult(resultSet);
                break;
            }
            case 8: {
                exit = true;
                break;
            }
        }
```

```
        }
    }

    private void randomize(int entity) throws SQLException, IllegalAccessException {
        switch (entity) {
            case 1: {
                Author author = new Author(
                        null,
                        generateRandomString(15),
                        new Date(),
                        generateRandomString(10),
                        generateRandomBoolean()
                );
                Author insertedAuthor = dao.insertAuthor(author);
                view.clearConsole();
                view.printAuthorInfo(insertedAuthor);
                break;
            }
            case 2: {
                Book book = new Book(
                        null,
                        generateRandomString(15),
                        generateRandomInt(),
                        generateRandomString(10),
                        generateRandomString(5),
                        1L
                );
                Book insertedBook = dao.insertBook(book);
                view.clearConsole();
                view.printBookInfo(insertedBook);
                break;
            }
            case 3: {
                Reader reader = new Reader(
                        null,
                        generateRandomString(10),
                        generateRandomString(5),
                        generateRandomInt(),
                        generateRandomInt());
                Reader insertedReader = dao.insertReader(reader);
                view.clearConsole();
                view.printReaderInfo(insertedReader);
                break;
            }
            case 4: {
                Subscription subscription = new Subscription(
                        null,
                        generateRandomString(10),
                        generateRandomInt(),
                        generateRandomString(15),
                        1L);
                Subscription insertedSubscription = dao.insertSubscription(subscription);
                view.clearConsole();
                view.printSubscriptionInfo(insertedSubscription);
```

```
            break;
        }
    }
}

void selectAll(int entity) throws SQLException {
    switch (entity) {
        case 1: {
            List<Author> authors = dao.getAuthorList();
            view.clearConsole();
            view.printAuthors(authors);
            break;
        }
        case 2: {
            List<Book> books = dao.getBookList();
            view.clearConsole();
            view.printBooks(books);
            break;
        }
        case 3: {
            List<Reader> readers = dao.getReaderList();
            view.clearConsole();
            view.printReaders(readers);
            break;
        }
        case 4: {
            List<Subscription> subscriptions = dao.getSubscriptionList();
            view.clearConsole();
            view.printSubscriptions(subscriptions);
            break;
        }
    }
}
void selectById(Long id, int entity) throws SQLException {
    switch (entity) {
        case 1: {
            Author author = dao.getAuthor(id);
            view.clearConsole();
            view.printAuthorInfo(author);
            break;
        }
        case 2: {
            Book book = dao.getBook(id);
            view.clearConsole();
            view.printBookInfo(book);
            break;
        }
        case 3: {
            Reader reader = dao.getReader(id);
            view.clearConsole();
            view.printReaderInfo(reader);
            break;
        }
        case 4: {
```

```
            Subscription subscription = dao.getSubscription(id);
            view.clearConsole();
            view.printSubscriptionInfo(subscription);
            break;
        }
    }
}

void delete(Long id, int entity) throws SQLException {
    view.showDeleted();
    switch (entity) {
        case 1: {
            Author author = dao.deleteAuthor(id);
            view.clearConsole();
            view.printAuthorInfo(author);
            break;
        }
        case 2: {
            Book book = dao.deleteBook(id);
            view.clearConsole();
            view.printBookInfo(book);
            break;
        }
        case 3: {
            Reader reader = dao.deleteReader(id);
            view.clearConsole();
            view.printReaderInfo(reader);
            break;
        }
        case 4: {
            Subscription subscription = dao.deleteSubscription(id);
            view.clearConsole();
            view.printSubscriptionInfo(subscription);
            break;
        }
    }
}
void insert(int entity) throws SQLException, IllegalAccessException {
    switch (entity) {
        case 1: {
            Author author = new Author(
                    null,
                    view.getStringOption("username"),
                    new Date(),
                    view.getStringOption("nationality"),
                    view.getBooleanOption("is author alive")
            );

            Author insertedAuthor = dao.insertAuthor(author);
            view.clearConsole();
            view.printAuthorInfo(insertedAuthor);
            break;
        }
        case 2: {
```

```
            Book book = new Book(null, view.getStringOption("title"), view.getIntOption("pages count"),
                    view.getStringOption("genre"), view.getStringOption("original language"),
                    view.getLongOption("author"));
            Book insertedBook = dao.insertBook(book);
            view.clearConsole();
            view.printBookInfo(insertedBook);
            break;
        }
        case 3: {
            Reader reader = new Reader(null, view.getStringOption("name"), view.getStringOption("favourite genre"),
                    view.getIntOption("age"), view.getIntOption("finished books"));
            Reader insertReader = dao.insertReader(reader);
            view.clearConsole();
            view.printReaderInfo(insertReader);
            break;
        }
        case 4: {
            Subscription subscription = new Subscription(null, view.getStringOption("type"), view.getIntOption("price"),
                    view.getStringOption("validity"), view.getLongOption("owner"));
            Subscription insertedSubscription = dao.insertSubscription(subscription);
            view.clearConsole();
            view.printSubscriptionInfo(insertedSubscription);
            break;
        }
    }
  }
  void update(int entity) throws SQLException, IllegalAccessException {
    switch (entity) {
        case 1: {
            Author author = new Author(
                    view.getLongOption("id"),
                    view.getStringOption("username"),
                    new Date(),
                    view.getStringOption("nationality"),
                    view.getBooleanOption("is author alive")
            );

            Author updatedAuthor = dao.updateAuthor(author);
            view.clearConsole();
            view.printAuthorInfo(updatedAuthor);
            break;
        }
        case 2: {
            Book book = new Book(view.getLongOption("id"), view.getStringOption("title"), view.getIntOption("pages count"),
                    view.getStringOption("genre"), view.getStringOption("original language"),
                    view.getLongOption("author"));
            Book updatedBook = dao.updateBook(book);
            view.clearConsole();
            view.printBookInfo(updatedBook);
            break;
        }
        case 3: {
            Reader reader = new Reader(view.getLongOption("id"), view.getStringOption("name"),
view.getStringOption("favourite genre"),
```

```
                    view.getIntOption("age"), view.getIntOption("finished books"));
            Reader updatedReader = dao.updateReader(reader);
            view.clearConsole();
            view.printReaderInfo(updatedReader);
            break;
        }
        case 4: {
            Subscription subscription = new Subscription(view.getLongOption("id"), view.getStringOption("type"),
view.getIntOption("price"),
                    view.getStringOption("validity"), view.getLongOption("owner"));
            Subscription updatedSubscription = dao.updateSubscription(subscription);
            view.clearConsole();
            view.printSubscriptionInfo(updatedSubscription);
            break;
        }
    }
}

private static final String CHAR_LOWER = "abcdefghijklmnopqrstuvwxyz";
private static final String CHAR_UPPER = CHAR_LOWER.toUpperCase();
private static final String NUMBER = "0123456789";

private static final String DATA_FOR_RANDOM_STRING = CHAR_LOWER + CHAR_UPPER + NUMBER;
private static SecureRandom random = new SecureRandom();



private static Boolean generateRandomBoolean() {
    return random.nextBoolean();
}

private static int generateRandomInt() {
    return random.nextInt();
}

private static String generateRandomString(int length) {
    if (length < 1) throw new IllegalArgumentException();

    StringBuilder sb = new StringBuilder(length);
    for (int i = 0; i < length; i++) {

        // 0-62 (exclusive), random returns 0-61
        int rndCharAt = random.nextInt(DATA_FOR_RANDOM_STRING.length());
        char rndChar = DATA_FOR_RANDOM_STRING.charAt(rndCharAt);

        sb.append(rndChar);
    }
    return sb.toString();
}


}
```

Лістинг модуля View.py

```
package com.lab2.view;

import com.lab2.model.*;

import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.util.Scanner;
import java.util.List;

public class View {
    public View() {}

    public void printMenu() {
        System.out.println("Welcome to lab2! Choose on of options below:");
        System.out.println("0. Randomize");
        System.out.println("1. Get list of entities from table");
        System.out.println("2. Get entity by id from table");
        System.out.println("3. Insert into");
        System.out.println("4. Update in");
        System.out.println("5. Delete from");
        System.out.println("6. FTS");
        System.out.println("7. Search by some input");
        System.out.println("8. Exit");
    }

    public void askId() {
        System.out.println("Enter id: ");
    }

    public void printTables() {
        System.out.println("1. Author");
        System.out.println("2. Book");
        System.out.println("3. Reader");
        System.out.println("4. Subscription");
    }

    public void showDeleted() {
        System.out.println("Deleted entity");
    }

    public void clearConsole() {
        try {
            Runtime.getRuntime().exec("clear");
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }

    public int getOption() {
        Scanner in = new Scanner(System.in);
        return in.nextInt();
    }

    public String getStringOption(String name) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter " + name + ": ");
```

```
        return sc.nextLine();
    }

    public Long getLongOption(String name) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter " + name + ": ");
        return sc.nextLong();
    }

    public int getIntOption(String name) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter " + name + ": ");
        return sc.nextInt();
    }

    public boolean getBooleanOption(String name) {
        Scanner n = new Scanner(System.in);
        System.out.println("Enter " + name + ": ");
        return n.nextBoolean();
    }

    private void printResultSet(ResultSet resultSet) throws SQLException {
        ResultSetMetaData rsmd = resultSet.getMetaData();
        int columnsNumber = rsmd.getColumnCount();
        while (resultSet.next()) {
            for (int i = 1; i <= columnsNumber; i++) {
                String columnValue = resultSet.getString(i);
                System.out.println(rsmd.getColumnName(i) + ": " + columnValue);
            }
            System.out.println("----------");
        }
    }

    public void joinedSearchResult(ResultSet resultSet) throws SQLException {
        System.out.println("Joinded Search result: ");
        printResultSet(resultSet);
    }

    private void printHR() {
        System.out.println("--------------");
    }

    public void printAuthorInfo(Author a) {
        printHR();
        System.out.println("ID: " + a.getAid());
        System.out.println("Name: " + a.getName());
        System.out.println("Nationality: " + a.getNationality());
        System.out.println("Birth date: " + a.getBirth_date().toString());
    }

    public void printBookInfo(Book b) {
        printHR();
        System.out.println("ID: " + b.getBid());
        System.out.println("Title: " + b.getTitle());
        System.out.println("Author: " + b.getAuthor());
        System.out.println("Genre: " + b.getGenre());
        System.out.println("Original language: " + b.getOriginal_language());
        System.out.println("Pages count: " + b.getPages_count());
    }

    public void printReaderInfo(Reader r) {
```

```
        printHR();
        System.out.println("ID: " + r.getRid());
        System.out.println("Name: " + r.getName());
        System.out.println("Age: " + r.getAge());
        System.out.println("Favourite genre: " + r.getFavourite_genre());
        System.out.println("Read books count: " + r.getFinished_books());
    }

    public void printSubscriptionInfo(Subscription s) {
        printHR();
        System.out.println("ID: " + s.getSid());
        System.out.println("Type: " + s.getType());
        System.out.println("Validity: " + s.getValidity());
        System.out.println("Owner: " + s.getOwner());
        System.out.println("Price: " + s.getPrice());
    }

    public void printAuthors(List<Author> authors) {
        for (Author author : authors) {
            printAuthorInfo(author);
        }
    }

    public void printBooks(List<Book> books) {
        for (Book book : books) {
            printBookInfo(book);
        }
    }

    public void printReaders(List<Reader> readers) {
        for (Reader reader: readers) {
            printReaderInfo(reader);
        }
    }

    public void printSubscriptions(List<Subscription> subscriptions) {
        for (Subscription subscription : subscriptions) {
            printSubscriptionInfo(subscription);
        }
    }

}
```

Лістинг модуля DAO.java

```
package com.lab2.dao;

import com.lab2.model.Author;
import com.lab2.model.Book;
import com.lab2.model.Reader;
import com.lab2.model.Subscription;

import java.sql.*;
import java.util.List;

public class DAO implements IDAO {

    private IDAOImpl<Author> authorsDAOImpl;
    private IDAOImpl<Book> booksDAOImpl;
```

```
    private IDAOImpl<Reader> readersDAOImpl;
    private IDAOImpl<Subscription> subscriptionsDAOImpl;
    private Connection connection;

    public void connect(String url, String user, String password) throws SQLException {
        connection = DriverManager.getConnection(url, user, password);
        authorsDAOImpl = new DAOImpl<>(Author.class, connection);
        booksDAOImpl = new DAOImpl<>(Book.class, connection);
        readersDAOImpl = new DAOImpl<>(Reader.class, connection);
        subscriptionsDAOImpl = new DAOImpl<>(Subscription.class, connection);
    }

    @Override
    public Author getAuthor(Long id) throws SQLException {
        return authorsDAOImpl.getEntity(id);
    }

    @Override
    public Author deleteAuthor(Long id) throws SQLException {
        return authorsDAOImpl.deleteEntity(id);
    }

    @Override
    public Author updateAuthor(Author a) throws SQLException, IllegalAccessException {
        return authorsDAOImpl.updateEntity(a);
    }

    @Override
    public Author insertAuthor(Author a) throws SQLException, IllegalAccessException {
        return authorsDAOImpl.insertEntity(a);
    }

    @Override
    public List<Author> getAuthorList() throws SQLException {
        return authorsDAOImpl.getEntityList();
    }

    @Override
    public Book getBook(Long id) throws SQLException {
        return booksDAOImpl.getEntity(id);
    }

    @Override
    public Book deleteBook(Long id) throws SQLException {
        return booksDAOImpl.deleteEntity(id);
    }

    @Override
    public Book updateBook(Book b) throws SQLException, IllegalAccessException {
        return booksDAOImpl.updateEntity(b);
    }

    @Override
    public Book insertBook(Book b) throws SQLException, IllegalAccessException {
        return booksDAOImpl.insertEntity(b);
    }

    @Override
    public List<Book> getBookList() throws SQLException {
        return booksDAOImpl.getEntityList();
    }
```

```
    @Override
    public Reader getReader(Long id) throws SQLException {
        return readersDAOImpl.getEntity(id);
    }

    @Override
    public Reader deleteReader(Long id) throws SQLException {
        return readersDAOImpl.deleteEntity(id);
    }

    @Override
    public Reader updateReader(Reader r) throws SQLException, IllegalAccessException {
        return readersDAOImpl.updateEntity(r);
    }

    @Override
    public Reader insertReader(Reader r) throws SQLException, IllegalAccessException {
        return readersDAOImpl.insertEntity(r);
    }

    @Override
    public List<Reader> getReaderList() throws SQLException {
        return readersDAOImpl.getEntityList();
    }

    @Override
    public Subscription getSubscription(Long id) throws SQLException {
        return subscriptionsDAOImpl.getEntity(id);
    }

    @Override
    public Subscription deleteSubscription(Long id) throws SQLException {
        return subscriptionsDAOImpl.deleteEntity(id);
    }

    @Override
    public Subscription updateSubscription(Subscription s) throws SQLException,
IllegalAccessException {
        return subscriptionsDAOImpl.updateEntity(s);
    }

    @Override
    public Subscription insertSubscription(Subscription s) throws SQLException,
IllegalAccessException {
        return subscriptionsDAOImpl.insertEntity(s);
    }

    @Override
    public List<Subscription> getSubscriptionList() throws SQLException {
        return subscriptionsDAOImpl.getEntityList();
    }

    @Override
    public List<Book> searchWord(String word, boolean including) throws SQLException {
        String sql = "SELECT bid, pages_count, title, genre, original_language, author,"
                + " ts_headline(title, q, 'StartSel=<!>, StopSel=<!>') as title,"
                + " ts_headline(genre, q, 'StartSel=<!>, StopSel=<!>') as genre"
                + " FROM public.book , to_tsquery(?) as q"
                + " WHERE " + (including ? "" : "not")
                + " to_tsvector(title) || to_tsvector(genre) @@ q";
```

```
        PreparedStatement preparedStatement = connection.prepareStatement(
                sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY
        );
        preparedStatement.setString(1, word);
        ResultSet resultSet = preparedStatement.executeQuery();

        return booksDAOImpl.resultSetToList(resultSet);
    }

    @Override
    public ResultSet joinedBookSearch(int pagesStart, int pagesEnd, boolean isAuthorAlive)
throws SQLException {
        String sql = "SELECT * FROM public.book INNER JOIN public.author ON author.aid =
book.author "
                + "WHERE (Ω.isalive = ? AND pages_count BETWEEN ? AND ?)";

        PreparedStatement preparedStatement = connection.prepareStatement(
                sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY
        );
        preparedStatement.setBoolean(1, isAuthorAlive);
        preparedStatement.setInt(2, pagesStart);
        preparedStatement.setInt(3, pagesEnd);

        return preparedStatement.executeQuery();
    }
}
```

Лістинг модуля DAOImpl.java

```
package com.lab2.dao;

import com.lab2.annotations.PrimaryKey;
import com.lab2.annotations.TableName;

import java.lang.reflect.Field;
import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.List;
import java.util.Arrays;
import java.util.Date;

public class DAOImpl<T> implements IDAOImpl<T> {

    private Class<T> clazz;
    private Connection connection;

    public DAOImpl(Class<T> clazz, Connection connection) {
        this.clazz = clazz;
        this.connection = connection;
    }

    private void getAllFields(List<Field> fields, Class<?> type) {
        fields.addAll(Arrays.asList(type.getDeclaredFields()));
```

```
        if (type.getSuperclass() != null) {
            getAllFields(fields, type.getSuperclass());
        }

    }

    private T createEntity(ResultSet resultSet, List<Field> fields) {
        T entity;
        try {
            entity = clazz.getConstructor().newInstance();
            for (Field field: fields) {
                String name = field.getName();
                String value = resultSet.getString(name);
                Class type = field.getType();
                if (value != null) {
                    if (type == boolean.class) {
                        field.set(entity, resultSet.getBoolean(name));
                    } else if (type == java.util.Date.class) {
                        field.set(entity, resultSet.getDate(name));
                    } else {
                        field.set(entity, type.getConstructor(String.class).newInstance(value));
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            entity = null;
        }
        return entity;
    }

    public List<T> resultSetToList(ResultSet resultSet) throws SQLException {
        List<Field> fields = new ArrayList<>();
        getAllFields(fields, clazz);

        fields.forEach(field -> field.setAccessible(true));

        List<T> list = new ArrayList<>();
        resultSet.beforeFirst();
        while (resultSet.next()) {
            T entity = createEntity(resultSet, fields);
            list.add(entity);
        }
        return list;
    }

    @Override
    public T getEntity(Long id) throws SQLException {
        T entity;
        TableName tableAnnotation = clazz.getAnnotation(TableName.class);
        Field primary = getPrimaryField();

        String sql = String.format("SELECT * FROM public.%s WHERE %s = ?", tableAnnotation.name(), primary.getName());
        PreparedStatement preparedStatement = connection.prepareStatement(
```

```
                sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY
        );
        preparedStatement.setLong(1, id);
        ResultSet resultSet = preparedStatement.executeQuery();

        try {
            entity = resultSetToList(resultSet).get(0);
        } catch (IndexOutOfBoundsException ex) {
            entity = null;
        }

        return entity;
    }

    @Override
    public List<T> getEntityList() throws SQLException {
        TableName tableAnnotation = clazz.getAnnotation(TableName.class);

        String sql = String.format("SELECT * FROM public.%s", tableAnnotation.name());
        PreparedStatement preparedStatement = connection.prepareStatement(
                sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
        ResultSet resultSet = preparedStatement.executeQuery();

        return resultSetToList(resultSet);
    }

    @Override
    public T deleteEntity(Long id) throws SQLException {
        TableName tableAnnotation = clazz.getAnnotation(TableName.class);
        Field primary = getPrimaryField();

        String sql = String.format("DELETE FROM public.%s WHERE %s = ? RETURNING *;",
                tableAnnotation.name(), primary.getName());

        PreparedStatement preparedStatement = connection.prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
        preparedStatement.setLong(1, id);
        ResultSet resultSet = preparedStatement.executeQuery();

        T deletedEntity;
        try {
            deletedEntity = resultSetToList(resultSet).get(0);
        } catch (IndexOutOfBoundsException ex) {
            deletedEntity = null;
        }

        return deletedEntity;
    }

    @Override
    public T updateEntity(T entity) throws SQLException, IllegalAccessException {
        TableName tableAnnotation = clazz.getAnnotation(TableName.class);
        Field primary = getPrimaryField();
```

```
        String sql = String.format("UPDATE public.%s SET %s WHERE %s = ? RETURNING *;",
                tableAnnotation.name(), getFieldSqlString(entity), primary.getName());

        PreparedStatement preparedStatement = connection.prepareStatement(sql, ResultSet.TYPE_SCROLL_SENSITIVE,
ResultSet.CONCUR_READ_ONLY);
        preparedStatement.setLong(1, (Long) primary.get(entity));
        ResultSet resultSet = preparedStatement.executeQuery();

        T updatedEntity;
        try {
            updatedEntity = resultSetToList(resultSet).get(0);
        } catch (IndexOutOfBoundsException ex) {
            updatedEntity = null;
        }

        return updatedEntity;
    }

    private Field getPrimaryField() {
        List<Field> fields = new ArrayList<>();
        getAllFields(fields, clazz);

        Field primaryField = null;
        for(Field field: fields) {
            field.setAccessible(true);
            if (field.isAnnotationPresent(PrimaryKey.class)) {
                primaryField = field;
            }
        }
        return primaryField;
    }

    private String getRowsString(T entity) throws IllegalAccessException {
        List<Field> fields = new ArrayList<>();
        getAllFields(fields, clazz);

        List<String> rows = new ArrayList<>();
        for (Field field : fields) {
            field.setAccessible(true);
            Object value = field.get(entity);
            if (value != null) {
                rows.add(field.getName());
            }
        }

        return String.join(", ", rows);
    }

    private String getValuesPrepareString(T entity) throws IllegalAccessException {
        List<Field> fields = new ArrayList<>();
        getAllFields(fields, clazz);

        List<String> values = new ArrayList<>();
        for (Field field : fields) {
```

```
            field.setAccessible(true);
            Object value = field.get(entity);
            if (value != null) {
                values.add("?");
            }
        }

        return String.join(", ", values);
    }


    @Override
    public T insertEntity(T entity) throws SQLException, IllegalAccessException {
        TableName tableAnnotation = clazz.getAnnotation(TableName.class);
        List<Field> fields = new ArrayList<>();
        getAllFields(fields, clazz);

        String sql = String.format("INSERT INTO public.%s (%s) VALUES (%s) RETURNING *;",
                tableAnnotation.name(), getRowsString(entity), getValuesPrepareString(entity));

        PreparedStatement preparedStatement = connection.prepareStatement(
                sql, ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY
        );


        int parameterIndex = 1;
        for (Field field : fields) {
            field.setAccessible(true);
            Class type = field.getType();
            Object value = field.get(entity);

            if (value != null) {
                if (type == Long.class) {
                    preparedStatement.setLong(parameterIndex, (Long) field.get(entity));
                } else if (type == boolean.class) {
                    preparedStatement.setBoolean(parameterIndex, (boolean) field.get(entity));
                } else if (type == java.util.Date.class) {
                    java.sql.Date date = new java.sql.Date(((java.util.Date) field.get(entity)).getTime());
                    preparedStatement.setDate(parameterIndex, date);
                } else if (type == Integer.class) {
                    preparedStatement.setInt(parameterIndex, (Integer) field.get(entity));
                } else {
                    preparedStatement.setString(parameterIndex, String.valueOf(field.get(entity)));
                }
                parameterIndex += 1;
            }
        }

        ResultSet resultSet = preparedStatement.executeQuery();
        T insertedEntity;
        try {
            insertedEntity = resultSetToList(resultSet).get(0);
        } catch (IndexOutOfBoundsException ex) {
            insertedEntity = null;
```

```
        }

        return insertedEntity;
    }

    private String getFieldSqlString(T entity) throws IllegalAccessException {
        List<Field> fields = new ArrayList<>();
        getAllFields(fields, clazz);

        StringBuilder sql = new StringBuilder();
        for(int fieldId = 0; fieldId < fields.size(); fieldId++) {
            Field field = fields.get(fieldId);
            field.setAccessible(true);
            sql.append(String.format("%s = '%s'", field.getName(), field.get(entity)));
            if(fieldId != fields.size() - 1) {
                sql.append(", ");
            }
        }

        return sql.toString();
    }

}
```

Результати роботи програми

- Запуск програми:

```
Welcome to lab2! Choose on of options below:
0. Randomize
1. Get list of entities from table
2. Get entity by id from table
3. Insert into
4. Update in
5. Delete from
6. FTS
7. Search by some input
8. Exit
```

- Завдання 1

1. Додавання даних до БД (Model.insert()):

```
3
1. Author
2. Book
3. Reader
4. Subscription
3
Enter name:
Eugene
Enter favourite genre:
Fantasy
Enter age:
20
Enter finished books:
6
```

Результат:

```
---------------
ID: 2
Name: Eugene
Age: 20
Favourite genre: Fantasy
Read books count: 6
```

2. Редагування даних (Model.update()):

```
2
Enter id:
1
Enter title:
The Man in the High Castle
Enter pages count:
240
Enter genre:
Fantasy, Dystopian, Political
Enter original language:
American
Enter author:
1
```

Результат:

```
----------------
ID: 1
Title: The Man in the High Castle
Author: 1
Genre: Fantasy, Dystopian, Political
Original language: American
Pages count: 240
```

3. Видалення даних (Model.delete()):

```
5
1. Author
2. Book
3. Reader
4. Subscription
2
Enter id:
3
```

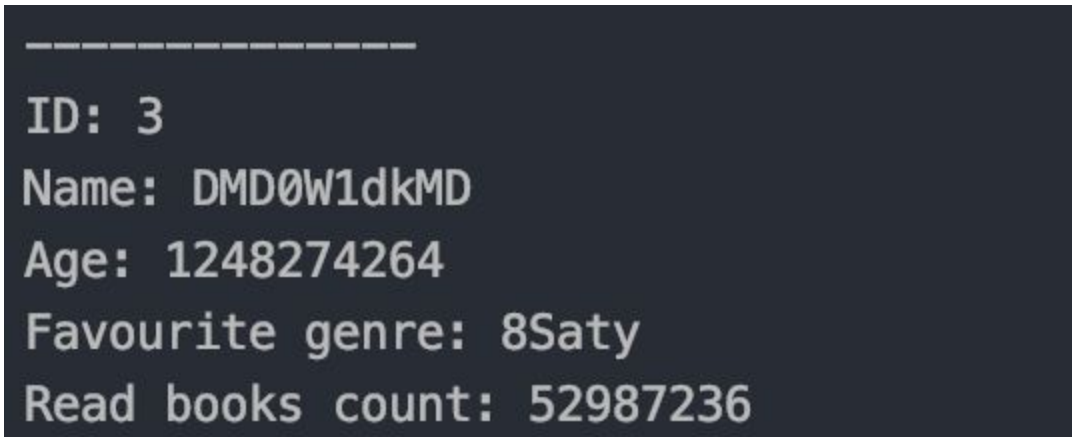
Результат (таблиця без сутності з ID 3):

```
--------------
ID: 1
Title: The Man in the High Castle
Author: 1
Genre: Fantasy, Dystopian, Political
Original language: English
Pages count: 240
--------------
ID: 2
Title: We
Author: 2
Genre: Dystopian
Original language: Russian
Pages count: 226
```

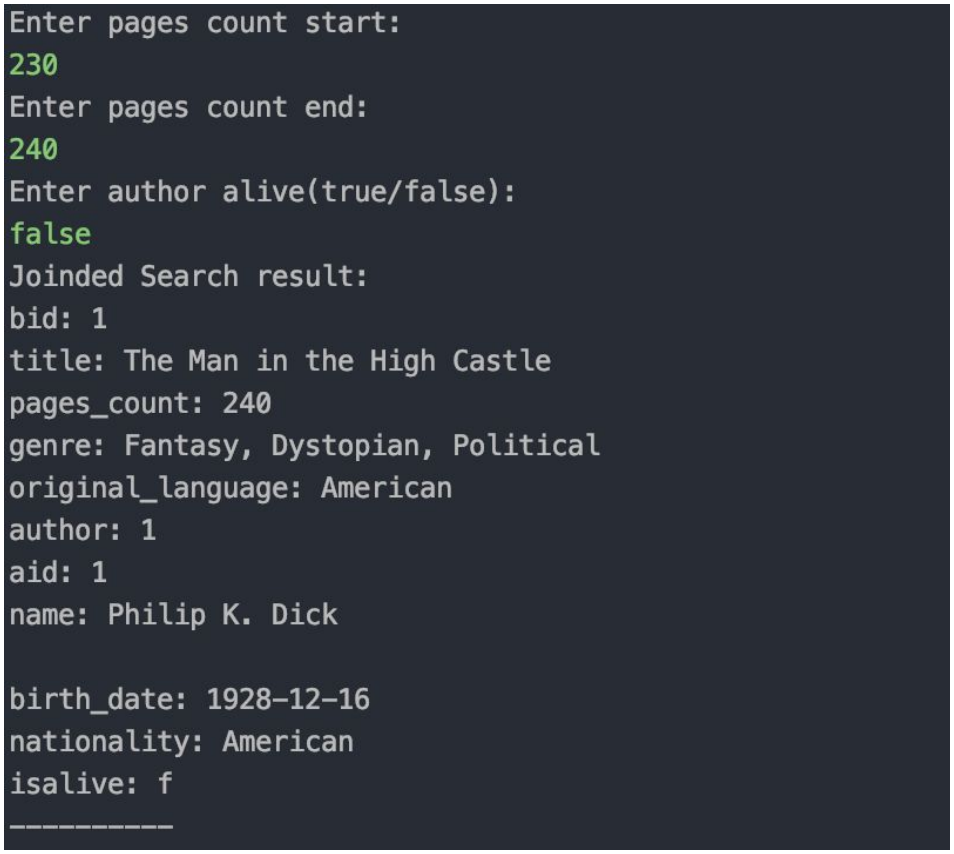
- Завдання 2. Пакетне генерування даних в таблиці Client (Model.random()):

```
0
1. Author
2. Book
3. Reader
4. Subscription
3
```

Результат:

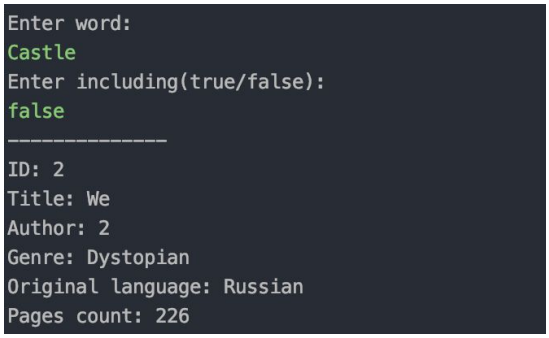


- Завдання 3. Пошук за двома-трьома атрибутами з декількох сутностей одночасно за варіантом (DAO.joinedBookSearch()):



- Завдання 4. Повнотекстовий пошук за варіантом (DAO.searchWord()):

1. Слово не входить:



2. Обов`язкове входження слова: