МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЯ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

# Отчет по лабораторной работе №9

Специальность ПО3

Выполнила
Гаврилюк Р. И., студентка
группы ПО3

Проверил
Крощенко А. А., ст. преп.
Кафедры ИИТ

Брест 2021

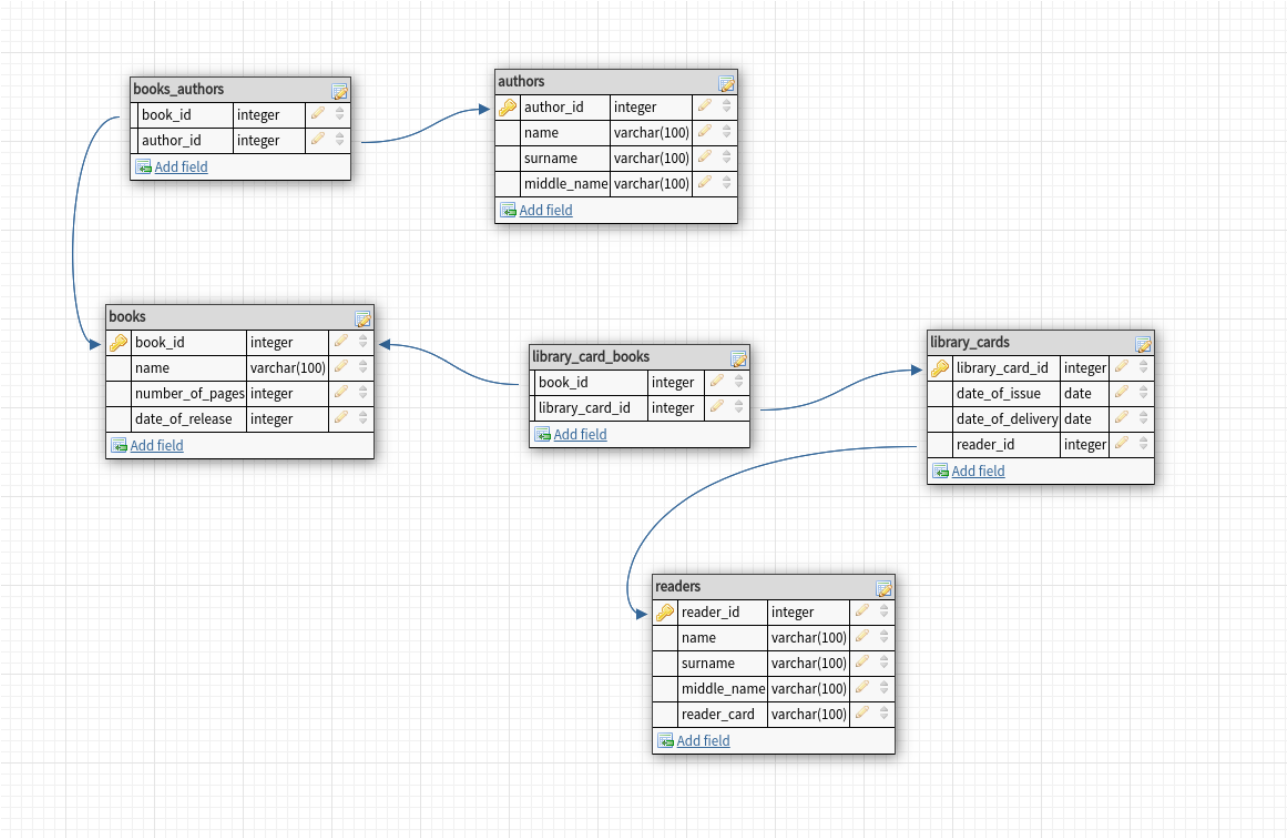
Вариант 6

Цель работы: приобрести практические навыки разработки баз данных и начальной интеграции БД с кодом Java с помощью JDBC.

Задание:
Реализовать базу данных из не менее 5 таблиц на заданную тематику. При реализации продумать типизацию полей и внешние ключи в таблицах. Визуализировать разработанную БД с помощью схемы, на которой отображены все таблицы и связи между ними. На языке Java с использованием JDBC реализовать подключение к БД и выполнить основные типы запросов, продемонстрировать результаты преподавателю и включить тексты составленных запросов в отчет. Основные типы запросов – 1. На выборку/на выборку с упорядочиванием (SELECT); 2. На добавление (INSERT INTO); 3. На удаление (DELETE FROM); 4. На модификацию (UPDATE). Базу данные можно реализовать в любой СУБД (MySQL, PostgreSQL, SQLite и др.)

6) База данных «Библиотека»

Схема базы данных:




Текст программы:
TableConfiguration (основные функции для работы с бд)

```
CREATE FUNCTION insert_into_readers(VARCHAR(100), VARCHAR(100), VARCHAR(100),
VARCHAR(100)) RETURNS VOID AS $$
    INSERT INTO readers (name,surname,middle_name, reader_card) VALUES($1, $2, $3, $4);
```

```
$$ LANGUAGE SQL;
CREATE FUNCTION update_readers(INT, VARCHAR(100), VARCHAR(100), VARCHAR(100),
VARCHAR(100)) RETURNS VOID AS $$
    UPDATE readers SET name=$2, surname=$3, middle_name=$4, reader_card=$5 WHERE
reader_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION delete_from_readers(INT) RETURNS VOID AS $$
    DELETE FROM readers WHERE reader_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION select_all_from_readers() RETURNS SETOF readers AS $$
    SELECT * FROM readers;
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_id_from_readers(INT) RETURNS SETOF readers AS $$
    SELECT * FROM readers WHERE reader_id=$1;
$$ LANGUAGE SQL;


CREATE FUNCTION insert_into_library_cards(DATE, DATE, INT) RETURNS VOID AS $$
    INSERT INTO library_cards (date_of_issue,date_of_delivery, reader_id) VALUES($1, $2, $3);
$$ LANGUAGE SQL;
CREATE FUNCTION update_library_cards(INT, DATE, DATE) RETURNS VOID AS $$
    UPDATE library_cards SET date_of_issue=$2, date_of_delivery=$3 WHERE
library_card_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION delete_from_library_cards(INT) RETURNS VOID AS $$
    DELETE from library_cards WHERE library_card_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION select_all_from_library_cards() RETURNS SETOF library_cards AS $$
    SELECT * from library_cards;
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_id_from_library_cards(INT) RETURNS SETOF library_cards AS $$
    SELECT * from library_cards WHERE reader_id=$1;
$$ LANGUAGE SQL;


CREATE FUNCTION insert_into_authors(VARCHAR(100), VARCHAR(100), VARCHAR(100))
RETURNS VOID AS $$
    INSERT INTO authors (name,surname,middle_name) VALUES($1, $2, $3);
$$ LANGUAGE SQL;
CREATE FUNCTION update_authors(INT, VARCHAR(100), VARCHAR(100), VARCHAR(100))
RETURNS VOID AS $$
    UPDATE authors SET name=$2, surname=$3, middle_name=$4 WHERE author_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION delete_from_authors(INT) RETURNS VOID AS $$
    DELETE FROM authors WHERE author_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION select_all_from_authors() RETURNS SETOF authors AS $$
    SELECT * FROM authors;
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_id_from_authors(INT) RETURNS SETOF authors AS $$
    SELECT * FROM authors WHERE author_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION get_author_id_from_authors(VARCHAR(100), VARCHAR(100), VARCHAR(100))
RETURNS INT AS $$
    SELECT author_id FROM authors WHERE name=$1 AND surname=$2 AND middle_name=$3;
$$ LANGUAGE SQL;


CREATE FUNCTION insert_into_books(VARCHAR(100), INT, INT) RETURNS VOID AS $$
    INSERT INTO books (name,number_of_pages,date_of_release) VALUES($1, $2, $3);
$$ LANGUAGE SQL;
CREATE FUNCTION update_books(INT, VARCHAR(100), INT, INT) RETURNS VOID AS $$
    UPDATE books SET name=$2, number_of_pages=$3, date_of_release=$4 WHERE
book_id=$1;
```

```
$$ LANGUAGE SQL;
CREATE FUNCTION delete_from_books(INT) RETURNS VOID AS $$
    DELETE FROM books WHERE book_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION select_all_from_books() RETURNS SETOF books AS $$
    SELECT * FROM books;
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_id_from_books(INT) RETURNS SETOF books AS $$
    SELECT * FROM books WHERE book_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION get_book_id_from_books(VARCHAR(100), INT, INT) RETURNS INT AS $$
    SELECT book_id FROM books WHERE name=$1 AND number_of_pages=$2 AND
date_of_release=$3;
$$ LANGUAGE SQL;


CREATE FUNCTION insert_into_library_card_books(INT, INT) RETURNS VOID AS $$
    INSERT INTO library_card_books (book_id,library_card_id) VALUES($1, $2);
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_book_id_from_card_books(INT) RETURNS SETOF
library_card_books AS $$
    SELECT * FROM library_card_books WHERE book_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_library_card_id_from_card_books(INT) RETURNS SETOF
library_card_books AS $$
    SELECT * FROM library_card_books WHERE library_card_id=$1;
$$ LANGUAGE SQL;


CREATE FUNCTION insert_into_book_authors(INT, INT) RETURNS VOID AS $$
    INSERT INTO book_authors (book_id,author_id) VALUES($1, $2);
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_book_id_from_book_authors(INT) RETURNS SETOF book_authors
AS $$
    SELECT * FROM book_authors WHERE book_id=$1;
$$ LANGUAGE SQL;
CREATE FUNCTION select_by_author_id_from_book_authors(INT) RETURNS SETOF book_authors
AS $$
    SELECT * FROM book_authors WHERE author_id=$1;
$$ LANGUAGE SQL;
```

Пакет connection

PgsqlFactory.java (реализация подключения к postgresql)

```
public class PgsqlFactory {
    static String DB_DRIVER;

    public static void createDataConnection() throws IOException {
        Properties properties = new Properties();
        properties.load(new FileReader("src/properties/database.properties"));

        DB_DRIVER = (String) properties.get("db.driver");

        try {
            Class.forName(DB_DRIVER);
        } catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

WrapperConnection.java (реализация установки соединения с бд Библиотека)

```
public class WrapperConnection {
    private static String DB_URL;
    private static String DB_USER;
    private static String DB_PASSWORD;
    private static Connection connection = null;

    static {
        try {
            Properties properties = new Properties();
            properties.load(new FileReader("src/properties/database.properties"));

            DB_URL = (String) properties.get("db.url");
            DB_USER = (String) properties.get("db.user");
            DB_PASSWORD = (String) properties.get("db.password");
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private WrapperConnection() {}

    public static Connection createConnection() throws IOException {
        try {
            connection = DriverManager.getConnection(DB_URL, DB_USER, DB_PASSWORD);
            return connection;
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }

        return connection;
    }

    public static Statement getStatement() throws SQLException {
        if (connection != null) {
            Statement statement = connection.createStatement();
            if (statement != null) {
                return statement;
            }
        }
        throw new SQLException("connection or statement is null");
    }

    public static void closeStatement(Statement statement) throws SQLException {
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                System.err.println("statement is null " + e);
            }
        }
    }
    public static PreparedStatement getPreparedStatement(String sql) throws SQLException {
        if (connection != null) {
            PreparedStatement statement = connection.prepareStatement(sql);
            if (statement != null) {
                return statement;
```

```
            }
        }
        throw new SQLException("connection or Prepared statement is null");
    }

    public static void closePreparedStatement(PreparedStatement statement) throws
SQLException {
        if (statement != null) {
            try {
                statement.close();
            } catch (SQLException e) {
                System.err.println("Prepared statement is null " + e);
            }
        }
    }

    public static void closeConnection() throws SQLException {
        if (connection != null) {
            try {
                connection.close();
            } catch (SQLException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}
```

Пакет entity

Entity.java (абстрактный класс, от которого будет наследоваться каждая программная  реализация сущности таблиц)

```
public abstract class Entity {
    private int id;

    public Entity() { }

    public Entity(ResultSet rs) {}

    public Entity(int id) {
        this.id = id;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public abstract void setResultSet(ResultSet rs) throws SQLException;
}
```

Book.java (программная реализация таблицы books, остальные ущности анаогично)

```
public class Book extends Entity{
    private String name;
```

```
private int dataOfRelease;
private int numberOfPages;

public Book() { }

public Book(int id, String name, int dataOfRelease, int numberOfPages) {
    super(id);
    this.name = name;
    this.dataOfRelease = dataOfRelease;
    this.numberOfPages = numberOfPages;
}

public Book(String name, int dataOfRelease, int numberOfPages) {
    this.name = name;
    this.dataOfRelease = dataOfRelease;
    this.numberOfPages = numberOfPages;
}

public void setResultSet(ResultSet rs) throws SQLException {
    try {
        this.setId(rs.getInt("book_id"));
        this.setName(rs.getString("name"));
        this.setNumberOfPages(rs.getInt("number_of_pages"));
        this.setDataOfRelease(rs.getInt("date_of_release"));
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public int getDataOfRelease() {
    return dataOfRelease;
}

public void setDataOfRelease(int dataOfRelease) {
    this.dataOfRelease = dataOfRelease;
}

public int getNumberOfPages() {
    return numberOfPages;
}

public void setNumberOfPages(int numberOfPages) {
    this.numberOfPages = numberOfPages;
}

@Override
public String toString() {
    return "Book{" +
            "id='" + super.getId() + '\'' +
            "name='" + name + '\'' +
```

```
            ", dataOfRelease='" + dataOfRelease + '\'' +
            ", numberOfPages=" + numberOfPages +
            '}';
    }
}
```

Пакет db

Dao.java (абстрактный класс, обеспечивающий интерфейс для работы с бд для сущностей)

```
public abstract class DAO <T extends Entity> {
    protected Connection connection;
    protected PreparedStatement prStatement;

    protected final Constructor<? extends T> ctor;

    public DAO(Connection connection, Class<? extends T> ctor) throws
NoSuchMethodException {
        this.ctor = ctor.getConstructor();
        try {
            this.connection = WrapperConnection.createConnection();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public Connection getConnection() {
        return connection;
    }

    public void setConnection(Connection connection) {
        this.connection = connection;
    }

    public Statement getStatement() throws SQLException {
        return WrapperConnection.getStatement();
    }

    public void closeStatement(Statement st) throws SQLException {
        WrapperConnection.closeStatement(st);
    }
}
```

AbstractDao.java (абстрактный класс, обеспечивающий интерфейс для работы с бд для сущностей)

```
public abstract class AbstractDAO <T extends Entity> extends DAO<T> {
    protected String sqlSelectAll;
    protected String sqlSelectById;
    protected String sqlDeleteById;
    protected String sqlUpdateById;
    protected String sqlInsert;

    public AbstractDAO(Connection connection, Class<? extends T> ctor) throws
NoSuchMethodException {
        super(connection, ctor);
    }
```

```
public List<T> findAll() throws DAOExtension {
    List<T> entities = new ArrayList<>();
    try {
        prStatement = WrapperConnection.getPreparedStatement(sqlSelectAll);
        ResultSet rs = prStatement.executeQuery();
        while (rs.next()) {
            T entity = ctor.newInstance();
            entity.setResultSet(rs);
            entities.add(entity);
        }
        rs.close();
        prStatement.close();
        return entities;
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    }
    return null;
}

public T findEntityById(int id) throws DAOExtension {
    T entity = null;
    try {
        prStatement = WrapperConnection.getPreparedStatement(sqlSelectById);
        prStatement.setInt(1, id);
        ResultSet rs = prStatement.executeQuery();
        while (rs.next()) {
            entity = ctor.newInstance();
            entity.setResultSet(rs);
        }
        rs.close();
        prStatement.close();
        return entity;
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    }
    return null;
}

public boolean deleteEntityById(int id) throws DAOExtension {
    try {
        prStatement = WrapperConnection.getPreparedStatement(sqlDeleteById);
        prStatement.setInt(1, id);
        prStatement.execute();
        prStatement.close();
        return true;
    } catch (SQLException throwables) {
        throwables.printStackTrace();
    }
    return false;
```

```
    }

    public boolean deleteEntity(T entity) throws DAOExtension {
        try {
            prStatement = WrapperConnection.getPreparedStatement(sqlDeleteById);
            prStatement.setInt(1, entity.getId());
            prStatement.execute();
            prStatement.close();
            return true;
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return false;
    }

    public boolean createEntity(T entity) throws DAOExtension {
        try {
            prStatement = WrapperConnection.getPreparedStatement(sqlInsert);
            this.setEntityCreateParams(entity, prStatement);
            prStatement.execute();
            prStatement.close();
            return true;
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return false;
    }

    public boolean updateEntity(T entity) throws DAOExtension {
        try {
            prStatement = WrapperConnection.getPreparedStatement(sqlUpdateById);
            this.setEntityUpdateParams(entity, prStatement);
            prStatement.execute();
            prStatement.close();
            return true;
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return false;
    }

    public abstract void setEntityCreateParams(T entity, PreparedStatement pr) throws
SQLException;
    public abstract void setEntityUpdateParams(T entity, PreparedStatement pr) throws
SQLException;
}
```

BookDao.java (пример реализации интерфейса для сущности Book для работы с таблицей, остальные аналогично)

```
public class BookDAO extends AbstractDAO<Book>{
    private static final Class<? extends Book> ctor = Book.class;
    private String sqlSelectBookId;

    public BookDAO(Connection connection) throws NoSuchMethodException {
        super(connection, ctor);
        sqlSelectAll = "SELECT * FROM select_all_from_books();";
        sqlSelectById = "SELECT * FROM select_by_id_from_books(?);";
        sqlInsert = "SELECT insert_into_books(?, ?, ?);";
        sqlUpdateById = "SELECT update_books(?, ?, ?, ?);";
```

```
        sqlDeleteById = "SELECT delete_from_books(?);";
        sqlSelectBookId = "SELECT get_book_id_from_books(?, ?, ?)";
    }

    public int findBookId(String name, int dateOfRelease, int numOfPages) throws DAOExtension
{
        int bookId = -1;
        try {
            prStatement = WrapperConnection.getPreparedStatement(sqlSelectBookId);
            prStatement.setString(1, name);
            prStatement.setInt(2, numOfPages);
            prStatement.setInt(3, dateOfRelease);
            ResultSet rs = prStatement.executeQuery();
            while (rs.next()) {
                bookId = rs.getInt(1);
            }
            rs.close();
            prStatement.close();
            return bookId;
        } catch (SQLException throwables) {
            throwables.printStackTrace();
        }
        return bookId;
    }

    @Override
    public void setEntityCreateParams(Book entity, PreparedStatement pr) throws SQLException
{
        prStatement.setString(1, entity.getName());
        prStatement.setInt(2, entity.getNumberOfPages());
        prStatement.setInt(3, entity.getDataOfRelease());
    }

    @Override
    public void setEntityUpdateParams(Book entity, PreparedStatement pr) throws SQLException
{
        prStatement.setInt(1, entity.getId());
        prStatement.setString(2, entity.getName());
        prStatement.setInt(3, entity.getNumberOfPages());
        prStatement.setInt(4, entity.getDataOfRelease());
    }
}
```

Пример работы с бд:

```
PgsqlFactory.createDataConnection();
Connection conn = WrapperConnection.createConnection();

// 1. init DAO
AuthorDAO authorDAO = new AuthorDAO(conn);
Book_AuthorsDAO bookAuthorsDAO = new Book_AuthorsDAO(conn);

// 2. transaction initialization for DAO objects
EntityTransaction transaction = new EntityTransaction();
transaction.initTransaction(authorDAO, bookAuthorsDAO);

int authorId = 0;
```

```
List<Author> authors;
// 3. query execution
try {
    authors = authorDAO.findAll();
    authorId = bookAuthorsDAO.findEntitiesByBookId(book.getId()).get(0).getAuthorId();
} catch (DAOExtension e) {
    transaction.rollback();
    throw new Exception(e);
} finally {
    // 4. transaction closing
    transaction.endTransaction();
}

conn.close();
```

Тестирование

```
library=# \dt
                 Список отношений
 Схема  |        Имя          |   Тип   | Владелец
--------+---------------------+---------+----------
 public | authors             | таблица | postgres
 public | book_authors        | таблица | postgres
 public | books               | таблица | postgres
 public | library_card_books  | таблица | postgres
 public | library_cards       | таблица | postgres
 public | readers             | таблица | postgres
(6 строк)

library=# select * from readers;
 reader_id |   name   | surname  | middle_name | reader_card
-----------+----------+----------+-------------+-------------
         1 | Рената   | Гаврилюк | Игоревна    | 1
         2 | Кристина | Дубина   | Игоревна    | 2
(2 строки)

library=# select * from library_cards;
 library_card_id | date_of_issue | date_of_delivery | reader_id
-----------------+---------------+------------------+-----------
               1 | 2021-02-04    | 2021-02-19       |         1
               8 | 2021-02-05    | 2021-02-22       |         2
               9 | 2021-02-06    | 2021-02-07       |         2
              10 | 2021-02-02    | 2021-02-28       |         1
               3 | 2021-01-01    | 2021-02-25       |         1
(5 строк)

library=#
```

Вывод: приобрела практические навыки разработки баз данных и начальной интеграции БД с кодом Java с помощью JDBC.