**2 ЛАБОРАТОРНА РОБОТА 2. використання JDBC**

**2.1 Мета роботи**

В результаті виконання лабораторної роботи студент має отримати такі знання і навички:

* ознайомитися із засобами Java, що використовуються для підключення до реляційних баз даних;
* навчитися виконувати запити на створення бази даних, модифікацію даних та запити на отримання інформації з бази в класах Java;
* отримати практичні навички створення java застосунку для роботи з базою даних.

**2.2 Результат виконання роботи**

**2.2.1 Створення класу DbConnector**

Для того, щоб спростити процес з’єднання з базою даних, було створено клас, що буде реалізовувати з’єднання з нашою базою даних та надавати доступ до об’єкту Connection.

Створімо у пакеті controller клас DbConnector, код якого зображено в лістингу 2.1. Цей клас розрахований на роботу із СУБД Derby, а сама база буде називатися dbOlenchenko.

Лістинг 2.1 – Клас DbConnector

**public** **class** DbConnector {

**private** **static** String *dbFullName* = "dbOlenchenko";

**private** **static** String *url* = "jdbc:derby:" + *dbFullName*;;

**private** **static** Properties *prop* = **new** Properties();

**static** **private** Connection *conn*;

**static**{

*prop*.setProperty("user", "Olenchenko");

*prop*.setProperty("password","1212");

*prop*.setProperty("create", "true");

}

**public** **static** Connection getConnection(){

**if**(*conn* == **null**)

**try** {

//This string need when class is using by server

//If driver was download nothing to do

Class.*forName*("org.apache.derby.jdbc.EmbeddedDriver");

} **catch** (ClassNotFoundException e1) {}

**try** {

*conn* = DriverManager.*getConnection*(*url*, *prop*);

} **catch** (SQLException e) {

JOptionPane.*showMessageDialog*(**null**,e.getMessage() +

" or May by connection use another project?");

e.printStackTrace();

}

**return** *conn*;

}

**public** **static** String getDbFullName() {

**return** *dbFullName*;

}

**public** **static** **void** setDbFullName(String dbFullName) {

DbConnector.*url* = "jdbc:derby:" + dbFullName;

}

**public** **static** **void** setPropertyUser(String str) {

*prop*.setProperty("user", str);

}

**public** **static** **void** setPropertyPassword(String str) {

*prop*.setProperty("password", str);

}

**public** **static** **void** setPropertyCreate(String str) {

*prop*.setProperty("create", str);

}}

**2.2.2 Створення бази даних**

Для створення бази були виконані такі дії:

* у пакеті test створено клас CreateDB з методом main;
* у методі main написаний код (лістинг 2.2), який за допомогою класу DbConnector забезпечить створення бази даних та з’єднання з нею у вигляді об’єкта типу Connection;
* запустити на виконання створений клас і перевірити, чи було створено папку з базою даних.

Лістинг 2.2 – Метод main класу Create DB

**public** **static** **void** main(String[] args) {

Connection conn = DbConnector.*getConnection*();

System.***out***.println(conn);

}

**2.2.3 Створення таблиць бази даних**

Схема створеної бази даних зображено на рисунку 2.1.

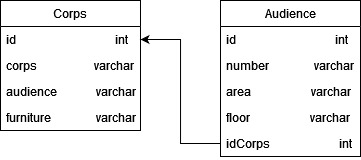


Рисунок 2.1 – Схема бази даних

2.2.3.1 SQL запит для створення таблиць бази даних

В пакеті query були створені класи з назвою QueryAudience та QueryCorps. Ці назви свідчать про те, що у класах зберігаються запити, які пов’язані з таблицею Audience та Corps. Код класу QueryAudience наведено в лістингу 2.3, а класу QueryCorps в лістингу 2.4.

Лістинг 2.3 – Клас QueryAudience

**public** **class** QueryAudience {

//Запити на створення таблиці БД

**public** **static** String queryCreate() {

String sql ="create table Audience ("

+ "ID int generated always as identity,"

+ "number varchar(30) default '' not null,"

+ "area varchar(30) default '' not null,"

+ "floor varchar(30) default '' not null,"

+ "idCorps int constraint FK\_GRAD\_CTH "

+ "references Corps on delete cascade, "

+ "primary key (ID))";

**return** sql;

}

**public** **static** String queryGetAll() {

**return** "select Audience.ID, Audience.number, Audience.area, Audience.floor, Corps.corps AS Corps, Corps.ID AS idCorps from Audience, Corps WHERE Corps.ID=Audience.idCorps ORDER BY Corps, Audience.number";

}

**public** **static** String queryAdd( Map<String, Object> map) {

**return** String.*format*("insert into Audience(number, area, floor, idCorps) values('%s', '%s', '%s', %d)", map.get("number"), map.get("area"), map.get("floor"), map.get("idCorps"));

}

**public** **static** String queryEdit( Map<String, Object> map) {

**return** String.*format*("update Audience set number = '%s', area = '%s', floor = '%s', idCorps = %d where ID = %d", map.get("number"), map.get("area"), map.get("floor"), map.get("idCorps"), map.get("id"));

}

**public** **static** String queryDelById( **int** id) {

**return** "DELETE FROM Audience WHERE ID = " + id;

}}

Лістинг 2.4 – Клас QueryCorps

**public** **class** QueryCorps {

**public** **static** String queryCreate() {

String sql = "create table Corps ("

+ "ID integer generated always as identity,"

+ "corps varchar(30) default '' not null, "

+ "audience varchar(30) default '', "

+ "furniture varchar(30) default '', "

+ "primary key (ID))";

**return** sql;

}

**public** **static** String queryGetAll() {

**return** "select \* from Corps";

}

**public** **static** String queryAdd(Map<String, Object> map) {

**return** String.*format*("insert into Corps(corps, audience, furniture) values('%s', '%s', '%s')", map.get("corps"), map.get("audience"), map.get("furniture"));

}

**public** **static** String queryEdit(Map<String, Object> map) {

**return** String.*format*("update Corps set corps = '%s', audience = '%s', furniture ='%s' where ID = %d", map.get("corps"), map.get("audience"), map.get("furniture"), map.get("id"));

}

**public** **static** String queryDelById(**int** id) {

**return** "DELETE FROM Corps WHERE id = " + id;

}

}

2.2.3.2 Створення класу CreateTable1

Було створено у пакеті test клас CreateTable1 з методом main, який забезпечить створення таблиць бази даних. В лістингу 2.5 наведено код методу, що створює таблицю Corps.

Лістинг 2.5 – Код створення таблиці Corps

**public** **class** CreateTable1 {

**public** **static** **void** main(String[] args) {

**try** {

Connection conn = DbConnector.*getConnection*();

Statement st = conn.createStatement();

String query = QueryCorps.*queryCreate*();

st.executeUpdate(query);

query = QueryAudience.*queryCreate*();

st.executeUpdate(query);

} **catch** (SQLException e) {

e.printStackTrace();

}

}

}

**2.2.4 Додавання даних до таблиці бази даних**

Для додавання записів до нашої таблиці Corps у пакеті tests було створено клас AddToTable1 з методом main. В лістингу 2.6 наведено метод, який був перероблений з урахуванням особливостей свого варіанту.

Лістинг 2.6 – Клас AddToTable1

**public** **class** AddToTable1 {

**public** **static** **void** main(String[] args) {

Connection conn = DbConnector.*getConnection*();

String query = "insert into Corps(corps, audience, furniture)"

+ " values(?,?,?)";

**try** {

PreparedStatement pst = conn.prepareStatement(query);

pst.setString(1,"1");

pst.setString(2,"123");

pst.setString(3,"66");

pst.executeUpdate();

pst.setString(1,"1");

pst.setString(2, "125");

pst.setString(3,"9");

pst.executeUpdate();

} **catch** (SQLException e) {

e.printStackTrace();

} }}

**2.2.5 Виведення вмісту таблиці в консоль**

Для того, щоб переглянути вміст таблиці у пакеті test було створено клас PrintTable1 з методом main, що зображено в лістингу 2.7 з урахуванням особливостей свого варіанта.

Лістинг 2.7 – Клас PrintTable1

**public** **class** PrintTable1 {

**public** **static** **void** main(String[] args) {

Connection conn = DbConnector.*getConnection*();

**try** {

Statement st = conn.createStatement();

String query = "select \* from Corps";

ResultSet rs = st.executeQuery(query);

**while** (rs.next())

System.***out***.println(rs.getString(1) + " "

+ rs.getString("corps") + " "

+ rs.getString("audience")+ " "

+ rs.getString("furniture"));

} **catch** (SQLException e) {

e.printStackTrace();

} }}

Після виконання коду на консолі з’явився вміст таблиці, який зображено на рисунку 2.2.

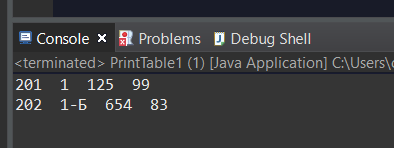


Рисунок 2.2 – Вивід вмісту таблиці після виконання класу PrintTable1

**2.2.6 Діалог та клас для реалізації пакетного запиту**

Для реалізації запиту було використано діалог DlgCorps, що було створено на попередній лабораторній роботі.

Для реалізації пакетного запиту було створено у пакеті test клас AddBatchFromDlg з методом main. Діалог повертає дані у вигляді карти. Тому рядок для запиту на додавання будемо формувати за допомогою статичного методу format класу String.

В лістингу 2.8 наведено реалізацію методу main для пакетного заповнення таблиці Corps.

Лістинг 2.8 – Пакетне заповнення таблиці Corps

**public** **class** AddBatchFromDlg {

**public** **static** **void** main(String[] args){

**try** {

Connection conn = DbConnector.*getConnection*();

String query ="insert into Corps(corps, audience, furniture) "

+ "values(?,?,?)";

PreparedStatement pst = conn.prepareStatement(query );

DlgCorps dlg = **new** DlgCorps();

**while**(**true**){

dlg.clear();

dlg.setVisible(**true**);

Map<String, Object> map = dlg.getMap();

**if**(map == **null**) **break**;

pst.setString(1,(String) map.get("corps"));

pst.setString(2,(String) map.get("audience"));

pst.setString(3,(String) map.get("furniture"));

pst.addBatch();

}

dlg.dispose();

pst.executeBatch();

} **catch** (Exception e) {

e.printStackTrace();

}

PrintTable1.*main*(**null**);

}}

**2.2.7 Виведення таблиці БД у візуальний компонент**

Для відображення таблиць бази даних було створено у пакеті view клас DbTableView, який буде успадковувати клас javax.swing.JTable. За рахунок успадкування розширимо можливості стандартного класу. Об’єкти цого класу зможуть приймати об’єкти типу List<Map<String, Object>>. Це список з картами, які відповідають рядкам таблиці бази даних і повертають карту Map<String, Object>, що відповідає вибраному рядку.

Код класуDbTableView наведено в лістингу 2.9.

Лістинг 2.9 – Клас DbTableView

**import** java.util.LinkedHashMap;

**import** java.util.List;

**import** java.util.Map;

**import** javax.swing.JOptionPane;

**import** javax.swing.JTable;

**import** javax.swing.table.DefaultTableModel;

**public** **class** DbTableView **extends** JTable {

**public** DbTableView() {

}

**public** **void** setDbTableModel(List<Map<String, Object>> mapList) {

Object[][] values = **null**;

Object[] header = **new** Object[]{"EmptyTable"};

**if**(mapList.size() > 0) {

header = mapList.get(0).keySet().toArray(header);

values = **new** Object[mapList.size()][header.length];

**int** i = 0;

**for** (Map<String, Object> map : mapList)

values[i++] = map.values().toArray();

}

setModel(**new** DefaultTableModel(values, header));

}

**public** Map<String, Object> getSelectedRowMap() {

**int** row = getSelectedRow();

**if** (row == -1) {

JOptionPane.*showMessageDialog*(**this**, "Row is not selected");

**return** **null**;

}

Map<String, Object> map = **new** LinkedHashMap<>();

**for** (**int** i = 0; i < getModel().getColumnCount(); i++) {

String key = getModel().getColumnName(i);

Object value = getModel().getValueAt(row, i);

map.put(key, value);

}

**return** map;

}

}

2.2.7.1 Метод для формування моделі таблиці БД на основі об’єкта типу ResultSet

JDBC передбачає повернення результатів запитів до бази даних у вигляді об’єктів типу ResultSet, а клас DbTableView, який створили для відображення таблиці, приймає модель таблиці бази даних у вигляді списоку карт. Тому потрібен метод, для перетворення об’єкта ResultSet у список карт.

За операції з базою даних у проєкті буде відповідати клас Controller, який створений у пакеті controller та зображено в лістингу 2.10.

Лістинг 2.10 – Клас Controller

**public** **class** Controller

{

**public** **static** List<Map<String, Object>> rsToMapList( ResultSet rs) {

List<Map<String, Object>> list = **new** ArrayList<Map<String, Object>>();

**try** {

ResultSetMetaData metadata = rs.getMetaData();

**int** columnCount = metadata.getColumnCount();

**while** (rs.next()) {

Map<String, Object> map = **new** LinkedHashMap<String, Object>();

**for** (**int** i = 1; i <= columnCount; ++i) {

String columnName = metadata.getColumnName(i);

map.put(columnName.toLowerCase(), rs.getObject(i));

}

list.add(map);

}

rs.close();

}

**catch** (SQLException e) {

e.printStackTrace();

}

**return** list;

}

2.2.7.2 Створення візуального компонента з відображенням таблиці бази даних

В якості такого компонента у пакеті view створено діалог, який дозволить відображати таблицю бази даних та вибирати потрібний запис із цієї таблиці. Вигляд діалогу в режимі дизайну зображено на рисунку 2.3.

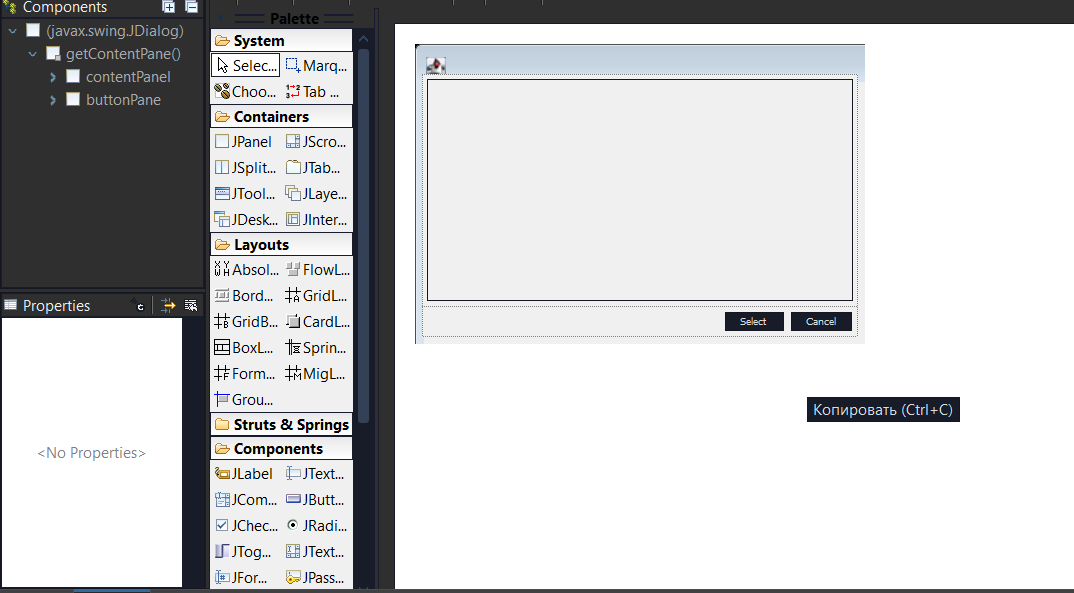


Рисунок 2.3 – Вигляд діалогу класу DlgSelect в режимі дизайну

2.2.7.3 Тестування візуального відображення таблиці БД

Для тестування створено у пакеті test клас TestSelectDialog з методом main. Для таблиці Corps код зображено в лістингу 2.11.

Лістинг 2.11 – Клас TestSelectDialog

**public** **class** TestSelectDialog {

**public** **static** **void** main(String[] args) {

Connection conn = DbConnector.*getConnection*();

**try** {

Statement st = conn.createStatement();

String sql = "select \* from Corps";

ResultSet rs = st.executeQuery(sql);

List<Map<String, Object>> list =

Controller.*rsToMapList*(rs);

DlgSelect ds = **new** DlgSelect(list);

ds.setVisible(**true**);

System.***out***.println(ds.getMap());

ds.dispose();

} **catch** (SQLException e) {

e.printStackTrace();

}}}

Скріншот результату зображено на рисунку 2.4.

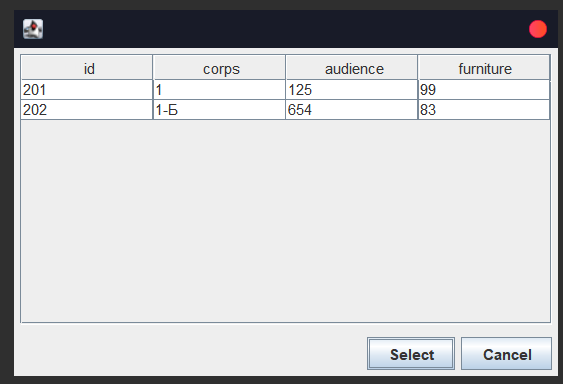


Рисунок 2.4 – Результат класу TestSelectDialog

**2.3 Результати виконання завдання самостійної роботи**

**2.3.1 Реалізація запитів до бази**

2.3.1.1 Cтандартні запити до таблиць бази

Код класу QueryCorps з реалізованими методами було зображено раніше, в лістингу 2.4. Код класу QueryAudience з реалізованими методами зображено в лістингу 2.12.

Лістинг 2.12 – Клас QueryAudience

**public** **class** QueryAudience {

//Запити на створення таблиці БД

**public** **static** String queryCreate() {

String sql ="create table Audience ("

+ "ID int generated always as identity,"

+ "number varchar(30) default '' not null,"

+ "area varchar(30) default '' not null,"

+ "floor varchar(30) default '' not null,"

+ "idCorps int constraint FK\_GRAD\_CTH "

+ "references Corps on delete cascade, "

+ "primary key (ID))";

**return** sql;

}

**public** **static** String queryGetAll() {

**return** "select Audience.ID, Audience.number, Audience.area, Audience.floor, Corps.corps AS Corps, Corps.ID AS idCorps from Audience, Corps WHERE Corps.ID=Audience.idCorps ORDER BY Corps, Audience.number";

}

**public** **static** String queryAdd( Map<String, Object> map) {

**return** String.*format*("insert into Audience(number, area, floor, idCorps) values('%s', '%s', '%s', %d)", map.get("number"), map.get("area"), map.get("floor"), map.get("idCorps"));

}

**public** **static** String queryEdit( Map<String, Object> map) {

**return** String.*format*("update Audience set number = '%s', area = '%s', floor = '%s', idCorps = %d where ID = %d", map.get("number"), map.get("area"), map.get("floor"), map.get("idCorps"), map.get("id"));

}

**public** **static** String queryDelById( **int** id) {

**return** "DELETE FROM Audience WHERE ID = " + id;

}}

2.3.1.2 Наперед визначені запити до бази

Код реалізації класу Query для таблиці Corps зображено в лістингу 2.13.

Лістинг 2.13 – Клас Query

**public** **class** Query {

**public** **static** String query1(String floor, **int** idCorps) {

**return** String.*format*("select area, number from Audience where floor = '%s' and idCorps = %d ",floor, idCorps);

}

}

**2.3.2 Розробка контролера**

2.3.2.1 Методи загального призначення

Код загальних методів, які реалізовані в класі Controller зображено в лістингу 2.14.

Лістинг 2.14 – Методи загального призначення

**public** **static** **boolean** tableExist( String tableName) {

**try** {

Connection conn = DbConnector.*getConnection*();

DatabaseMetaData md = conn.getMetaData();

String name = tableName.toUpperCase();

**return** md.getTables(**null**, **null**, name, **null**).next();

}

**catch** (SQLException e) {

e.printStackTrace();

**return** **false**;

}

}

**public** **static** List<Map<String, Object>> executeQuery( String query) {

System.***out***.println(query);

List<Map<String, Object>> list = **null**;

**try** {

Connection conn = DbConnector.*getConnection*();

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(query);

list = *rsToMapList*(rs);

}

**catch** (SQLException e) {

e.printStackTrace();

}

**return** list;

}

**public** **static** **int** executeUpdate( String query) {

System.***out***.println(query);

**try** {

Connection conn = DbConnector.*getConnection*();

Statement stmt = conn.createStatement();

**return** stmt.executeUpdate(query);

}

**catch** (SQLException e) {

e.printStackTrace();

**return** 0;

}

}

2.3.2.2 Методи для реалізації CRUD запитів до таблиць БД

Методи зображено в лістингу 2.15.

Лістинг 2.15 – Методи для реалізації CRUD запитів

**public** **static** **int** createTable( String tableName) {

**try** {

String queryClass = "query.Query" + tableName;

Class<?> clz = Class.*forName*(queryClass);

Method mtd = clz.getMethod("queryCreate", (Class<?>[])**new** Class[0]);

String sql = (String)mtd.invoke(**null**, **new** Object[0]);

**int** n = *executeUpdate*(sql);

**return** n;

}

**catch** (Exception e1) {

e1.printStackTrace();

**return** 0;

}

}

**public** **static** **void** add( String tableName, Map<String, Object> map) {

**try** {

String queryClass = "query.Query" + tableName;

Class<?> clz = Class.*forName*(queryClass);

Method mtd = clz.getMethod("queryAdd", Map.**class**);

String sql = (String)mtd.invoke(**null**, map);

*executeUpdate*(sql);

}

**catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** **void** edit( String tableName, Map<String, Object> map) {

**try** {

String queryClass = "query.Query" + tableName;

Class<?> clz = Class.*forName*(queryClass);

Method mtd = clz.getMethod("queryEdit", Map.**class**);

String sql = (String)mtd.invoke(**null**, map);

*executeUpdate*(sql);

}

**catch** (Exception e) {

e.printStackTrace();

}

}

**public** **static** **void** delete( String tableName, Map<String, Object> map) {

**try** {

String queryClass = "query.Query" + tableName;

Class<?> clz = Class.*forName*(queryClass);

Method mtd = clz.getMethod("queryDelById", Integer.***TYPE***);

**int** id = (**int**) map.get("id");

String sql = (String)mtd.invoke(**null**, id);

*executeUpdate*(sql);

}

**catch** (Exception e) {

e.printStackTrace();

}

}

}

**2.3.3 Діалоги для введення та редагування інформації в базі даних**

2.3.3.1 Розробка діалогу для таблиці другого рівня

Діалог введення та редагування інформації про аудиторію зображено на рисунку 2.5.

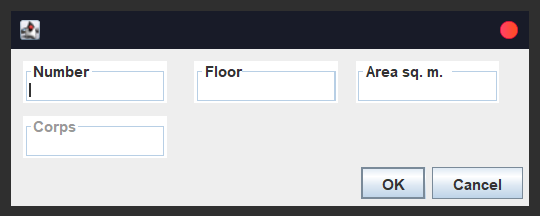


Рисунок 2.5 – Ведення інформації про аудиторію

Конструктор з параметром типу Map зображено в лістингу 2.16.

Лістинг 2.16 – Конструктор з параметром типу Map

**public** DlgAudience(Map map) {

**this**();

idAudience = (**int**) map.get("id");

//idCorps = (int) map.get("idCorps");

textField\_Number.setText(String.*valueOf*(map.get("number")));

textField\_area.setText((String) map.get("area"));

textField\_Floor.setText(String.*valueOf*(map.get("floor")));

textField\_Corps.setText((String) map.get("corps"));

}

Метод обробки кнопки ОК зображено в лістингу 2.17.

Лістинг 2.17 – Метод обробки кнопки ОК

**public** **void** actionPerformed(ActionEvent e) {

String number = textField\_Number.getText();

String area = textField\_area.getText();

String floor = textField\_Floor.getText();

(map = **new** LinkedHashMap<String, Object>()).put("id", idAudience);

map.put("number", number);

map.put("area", area);

map.put("floor", floor);

map.put("idCorps", idCorps);

setVisible(**false**);

Метод вибору id корпусу зображено в лістингу 2.18.

Лістинг 2.18 – Метод вибору id

**public** **void** mouseClicked(MouseEvent e) {

String query = QueryCorps.*queryGetAll*();

List<Map<String, Object>> listMap =

Controller.*executeQuery*(query);

DlgSelect ds = **new** DlgSelect(listMap);

ds.setTitle("QueryCorps selection");

ds.setVisible(**true**);

Map map = ds.getMap();

ds.dispose();

idCorps = (**int**) map.get("id");

textField\_Corps.setText((String) map.get("coprs"));

}

});

**2.3.4 Розробка дизайну головного вікна**

Головне вікно зображено на рисунку 2.6.

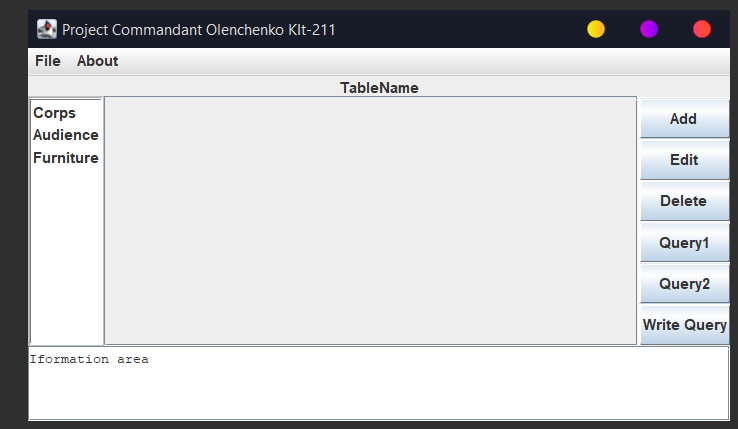


Рисунок 2.6 – Головне вікно

**2.3.5 Програмування головного вікна**

2.3.5.1 Метод реалізації запиту Query1

Метод візуальної частини, який буде викликатися після натискання кнопки Query1 наведено в лістингу 2.19.

Лістинг 2.19 – Метод кнопки Query1

**protected** **void** onBtnQuery1() {

DlgQuery1 dlg = **new** DlgQuery1();

dlg.setVisible(**true**);

String finish = dlg.getTeamName();

**int** IdFieldNumber = dlg.getIdFieldNumber();

String query = Query.*query1*(finish, IdFieldNumber);

dbTableView.setDbTableModel(Controller.*executeQuery*(query));

}

Вигляд діалогу запитів Query1 зображено на рисунку 2.7.

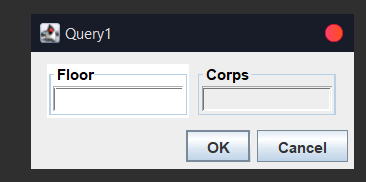


Рисунок 2.7 – Діалог Query1

2.3.5.2 Реалізація діалога для довільного запиту до БД

Зовнішній вигляд діалогу зображено на рисунку 2.8.

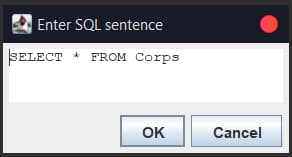


Рисунок 2.8 – Діалог DlgWriteQuery

2.3.5.3 Метод візуальної частини для реалізації довільного запиту до БД

Цей метод пов’язаний з кнопкою WriteQuery, код методу зображено в лістингу 2.20.

Лістинг 2.20 – Метод для реалізації довільного запиту до БД

**protected** **void** onWriteQuery() {

DlgWriteQuery dlg = **new** DlgWriteQuery();

dlg.setVisible(**true**);

String query = dlg.getQuery();

dlg.dispose();

**if** (query.toLowerCase().indexOf("select") >= 0) {

dbTableView.setDbTableModel(Controller.*executeQuery*(query));

textArea.setText(query);

}

**else** {

**int** res = Controller.*executeUpdate*(query);

textArea.setText("Updated " + res);

}

}

**2.4 Висновки**

В результаті виконання лабораторної роботи ми отримати такі знання і навички:

* ознайомитися із засобами Java, що використовуються для підключення до реляційних баз даних;
* навчитися виконувати запити на створення бази даних, модифікацію даних та запити на отримання інформації з бази в класах Java;
* отримати практичні навички створення java застосунку для роботи з базою даних.