

БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 6
Тема: «Создание приложения для базы данных»

Выполнила:
студент группы 150502
Альхимович Н.Г.

Проверил:
Игнатович А.О.

Минск
2024

1 ИСХОДНОЕ ЗАДАНИЕ

Создать прикладную программу для работы с базой данных и выполняющую заданные транзакции. Можно использовать любую среду и язык программирования.

Вариант задания: организация «Туристическое агентство».

2 ОПИСАНИЕ ПРИЛОЖЕНИЯ

Основные характеристики приложения приведены в таблице 2.1.

Таблица 2.1 – Характеристики приложения

Вид приложения	Desktop
Язык программирования	Java
Среда разработки	IntelliJ IDEA
Используемая БД	PostgreSQL

Используемые библиотеки и пакеты:

- Java Swing (для создания графического пользовательского интерфейса);
- Java AWT (для работы с графическими элементами и обработки событий);
- Java SQL (для взаимодействия с базой данных).

Доступный функционал:

- просмотр таблиц базы данных;
- сортировка данных таблицы по содержимому столбца по возрастанию/убыванию;
- добавление строк данных в таблицу;
- удаление строк данных из таблицы;
- редактирование полей;
- поиск данных по значению любого столбца.

3 ОПИСАНИЕ КОДА ПРОГРАММЫ

Основу программы составляет класс `DatabaseViewer`, которые включает поля и методы для реализации функционала, описанного в разделе 2.

Поля класса и их описание:

1. `tables_list` – объект типа «выпадающий список», хранящий названия таблиц, доступных для просмотра;
2. `table_db` – объект-таблица, который используется для отображения нужной таблицы в окне приложения;

3. `scroll_panel` – панель с полосой прокрутки, на которой располагаются все основные графические элементы;

4. `input_fields` – поля для ввода данных при добавлении новой строки в таблицу;

5. `is_ascending` – флаг, отражающий текущий вид сортировки таблицы;

6. `columns_list` – объект типа «выпадающий список», хранящий названия столбцов текущей таблицы;

7. `search_field` – поле для ввода ключевого слова (фразы) для поиска данных в таблице;

8. `connection` – объект, определяющий текущий сеанс соединения с базой данных.

Методы класса и их описание:

1. `DatabaseViewer()` – метод, создающий окно приложения и его основные графические элементы (строку меню с кнопками добавления/удаления данных, меню выбора таблиц, саму таблицу, меню выбора столбца текущей таблицы, поисковую панель, кнопку «Search»), содержащий слушатели событий взаимодействия с элементами (нажатий кнопок, щелчков мыши), а также вызывающий методы подключения к базе данных и загрузки таблиц из нее для дальнейшей работы;

2. `connectToDatabase()` – метод, осуществляющий подключение к базе данных;

3. `loadTables()` – метод, получающий список таблиц в составе базы данных и вносящий их в меню выбора таблиц (`tables_list`);

4. `displayTable()` – метод, принимающий в качестве аргументов название таблицы и параметр сортировки (необязательно) и выполняющий запрос к базе данных для получения данных требуемой таблицы (`SELECT`), создающий ее модель и отображающий заполненную таблицу в окне приложения;

5. `addNewRow()` – метод, вызываемый при выборе пункта меню `Edit->Add data...`, выводящий диалоговое окно с полями для ввода значений при добавлении новой строки данных, выполняющий соответствующий запрос в базу данных (`INSERT`) и обновляющий представленную в окне таблицу;

6. `deleteSelectedRow()` – метод, вызываемый при выделении подлежащей удалению строки данных и последующем выборе пункта меню `Edit->Delete data` и выполняющий соответствующий запрос к базе данных (`DELETE`);

7. `editCellValue()` – метод, вызываемые щелчком ПКМ по нужной ячейке, принимающий в качестве аргументов номер строки и столбца, содержимое ячейки на пересечении которых нужно изменить; выводящий диалоговое окно с полем для ввода нового значения, выполняющий соответствующий запрос к базе данных (`UPDATE`, `SET`) и обновляющий таблицу в окне приложения;

8. `searchRows()` – метод, вызываемый при нажатии кнопки «Search», выполняющий запрос к базе данных (`SELECT`) для выбора данных из таблицы, для которых значение (набирается в текстовом поле) столбца (выбирается из выпадающего списка) соответствует поисковому запросу; и выводящий таблицу с результатами поиска;

9. `isInteger()` – вспомогательный метод, для определения типа данных, введенным пользователем;

10. `buildTableModel()` – метод, принимающий в качестве аргумента результат выполнения запроса в методе `searchRows()` и выводящий полученную таблицу в окно приложения;

11. `main()` – основной метод класса, отображающий графический интерфейс пользователю.

Осуществление всех видов взаимодействия с базой данных, в том числе SQL-запросов, подвергается проверке на возникновение исключительных ситуаций. Если это происходит, то пользователю выводится соответствующее сообщение об ошибке.

4 ПРИМЕРЫ РАБОТЫ ПРОГРАММЫ

4.1 Начальное состояние после запуска

Главное окно приложения приведено на рисунке 4.1.1.

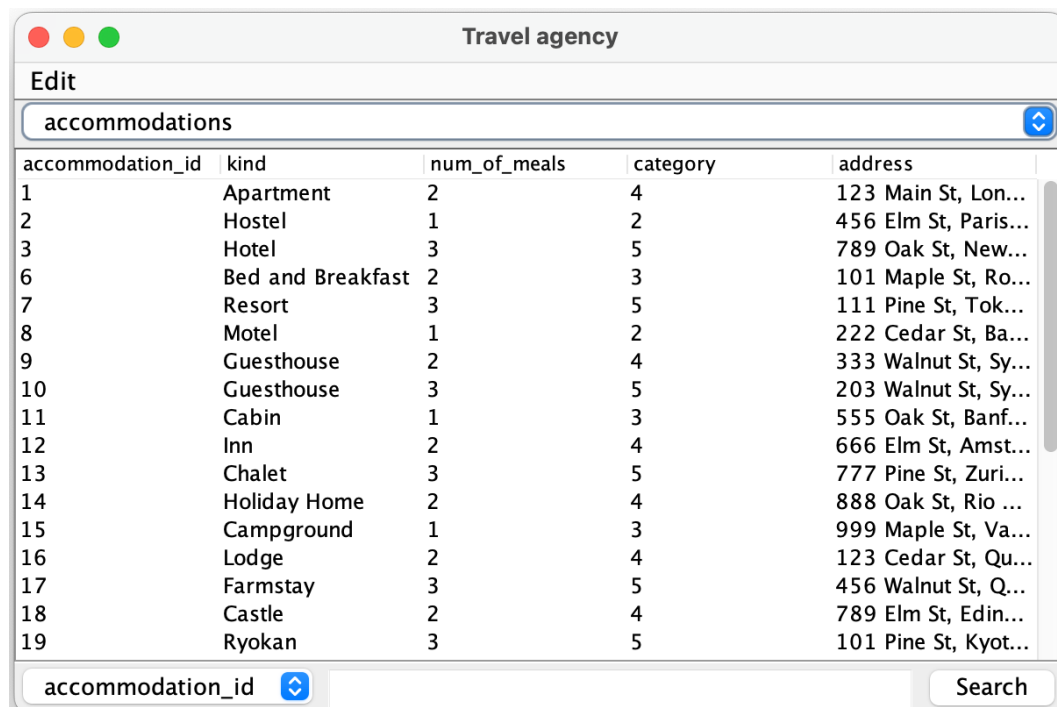


Рисунок 4.1.1 – Главное окно приложения

Основные элементы окна (сверху вниз, слева направо):

- меню Edit, имеющее пункты Add data... (добавить строку данных) и Delete data (удалить строку данных);
 - список таблиц базы данных;
 - выбранная для просмотра таблица;
 - список столбцов текущей таблицы;
 - поле для ввода параметра для поиска;
 - кнопка поиска данных в текущей таблице.
- Некоторые из них приведены на рисунках 4.1.2, 4.1.3 и 4.1.4.

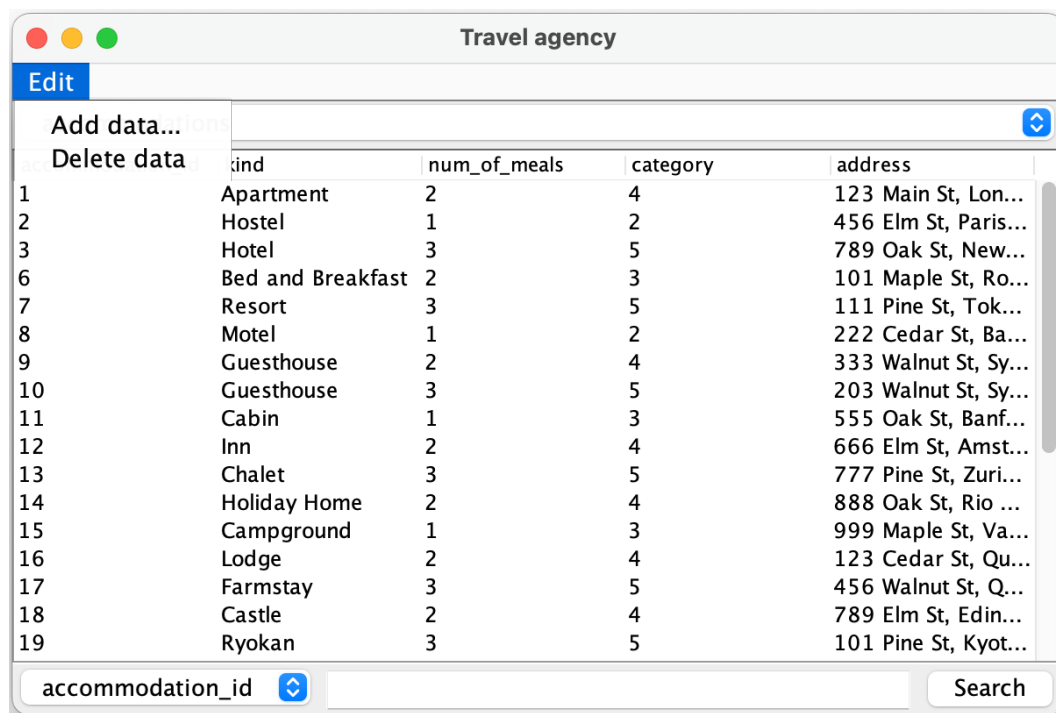


Рисунок 4.1.2 – Меню Edit

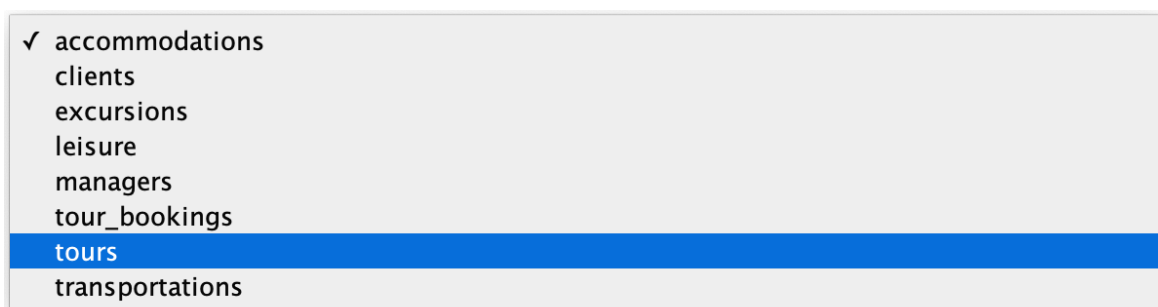


Рисунок 4.1.3 – Список таблиц

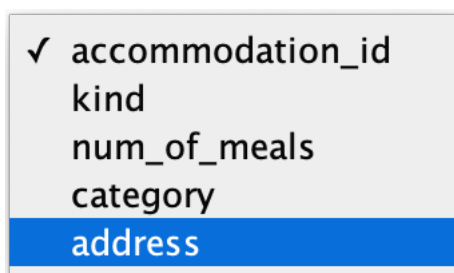


Рисунок 4.1.4 – Список столбцов

4.2 Добавление данных

Для добавления новой строки данных необходимо:

1. Выбрать нужную таблицу из списка (см. рисунок 4.1.3);
2. Выбрать пункт меню Edit->Add data... (см. рисунок 4.1.2);
3. Ввести нужные данные в поля для ввода, соответствующие столбцам выбранной таблицы (см. рисунок 4.2.1);
4. Нажать кнопку ОК.

accommodation_id	31
kind	Hostel
num_of_meals	1
category	3
address	52, Herzog Odilo

Рисунок 4.2.1 – Диалоговое окно для ввода данных

При несоответствии типа вводимых данных выводится сообщение об ошибке (см. рисунок 4.2.2).

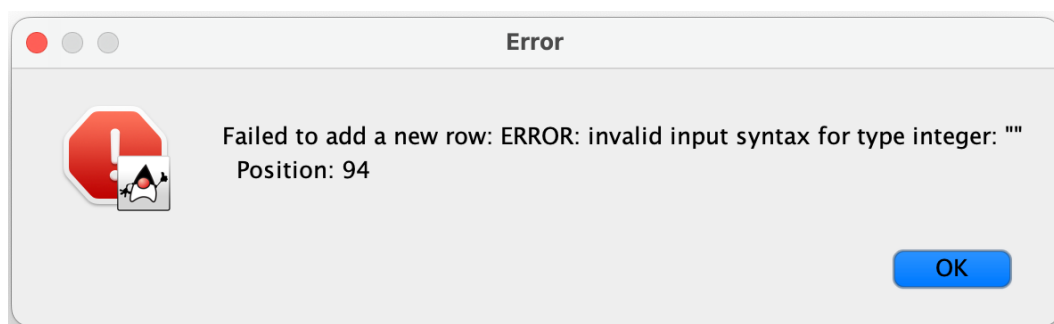


Рисунок 4.2.2 – Сообщение об ошибке при вводе данных

4.3 Удаление данных

Для удаления строки данных необходимо:

1. Выбрать нужную таблицу из списка (см. рисунок 4.1.3);
 2. Выбрать строку данных, подлежащую удалению, щелчком ЛКМ (см. рисунок 4.3.1);
 3. Выбрать пункт меню Edit->Delete data (см. рисунок 4.1.2);
- Результат работы функции отражен на рисунках 4.3.1 и 4.3.2.

accommodation_id	kind	num_of_meals	category	address
31	Hostel	1	3	52, Herzog Odilo...
30	Glamping	2	4	456 Walnut St, M...

Рисунок 4.3.1 – Таблица до удаления

accommodation_id	kind	num_of_meals	category	address
30	Glamping	2	4	456 Walnut St, M...
29	Spa Resort	3	5	123 Cedar St, Ba...

Рисунок 4.3.2 – Таблица после удаления строки данных

При попытке удалить данные без выбора строки выводится сообщение об ошибке (см. рисунок 4.3.3).

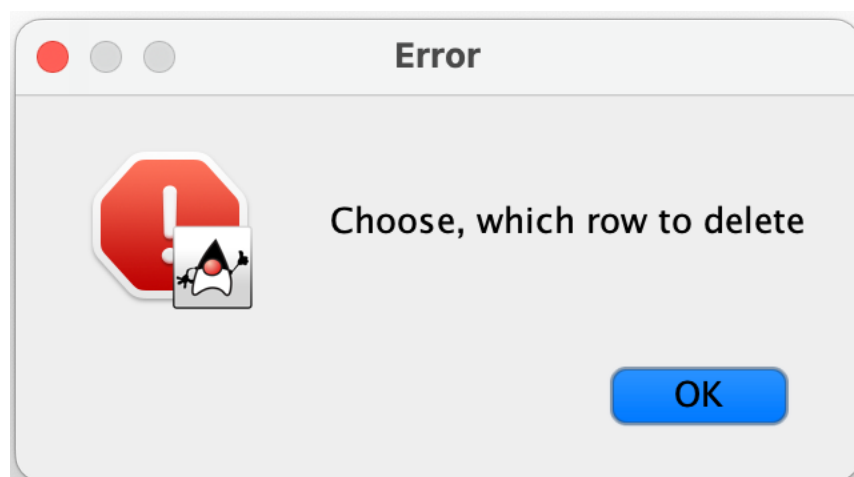


Рисунок 4.3.3 – Сообщение об ошибке при удалении данных

4.4 Редактирование данных

Для редактирования данных необходимо:

1. Щелкнуть ПКМ по ячейке таблицы, данные которой требуется изменить;
2. В появившемся диалоговом окне ввести новое значение ячейки (см. рисунок 4.4.1);
3. Нажать кнопку ОК;

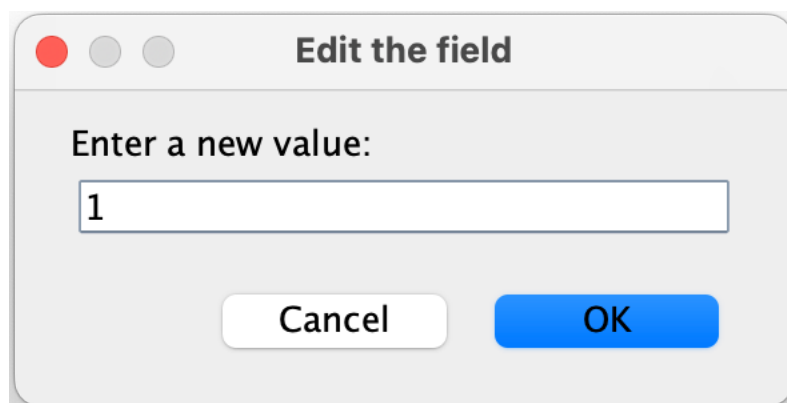


Рисунок 4.4.1 – Диалоговое окно для ввода нового значения

При несоответствии типа изменяемых данных выводится сообщение об ошибке (см. рисунок 4.4.2).

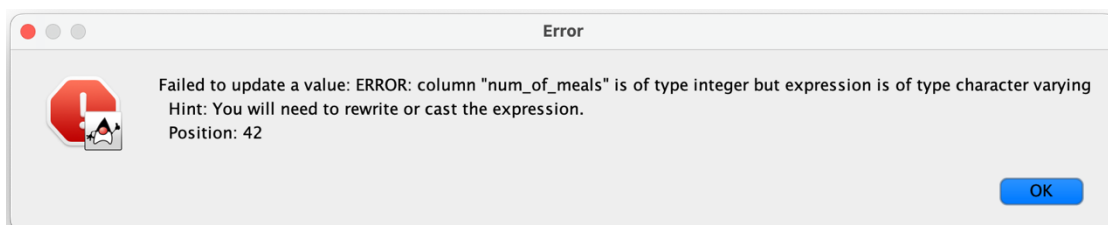


Рисунок 4.4.2 – Сообщение об ошибке при изменении данных

4.5 Поиск данных

Для выборки данных по параметру поиска необходимо:

1. В списке столбцов (см. рисунок 4.1.4) выбрать столбец, по значению которого требуется произвести поиск;
2. В поле для ввода под таблицей ввести текст поискового запроса. На рисунке 4.5.1 приведен пример поиска вариантов проживания в Сиднее;
3. Нажать кнопку Search.

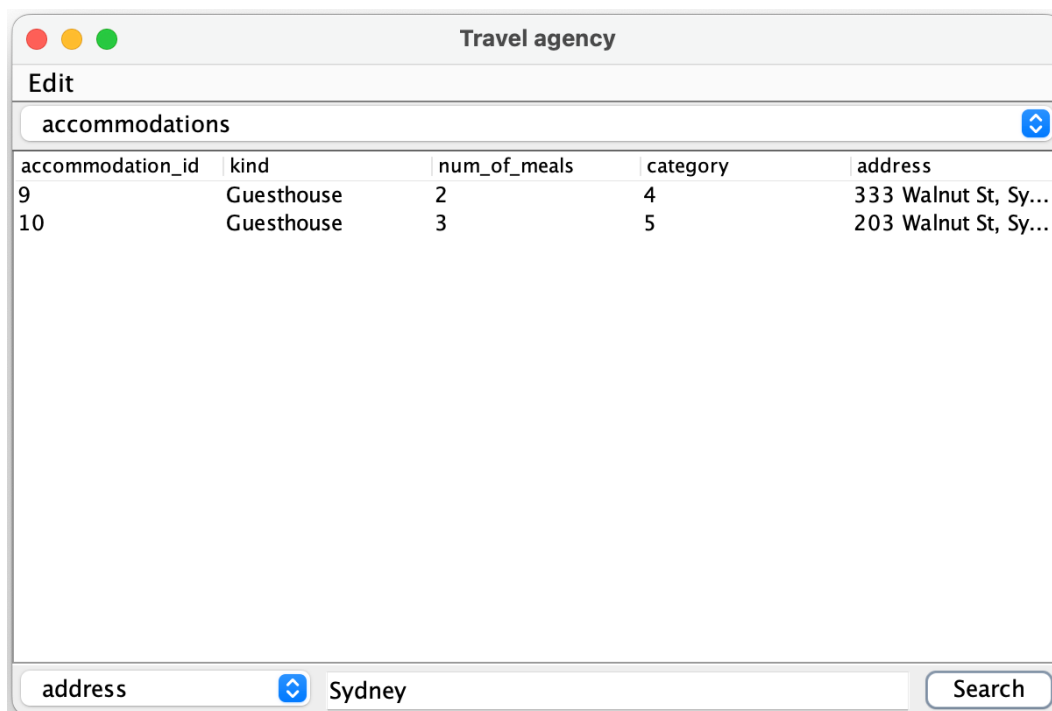


Рисунок 4.5.1 – Пример поиска данных

При попытке найти данные без указания параметра для поиска выводится сообщение об ошибке (см. рисунок 4.5.2).

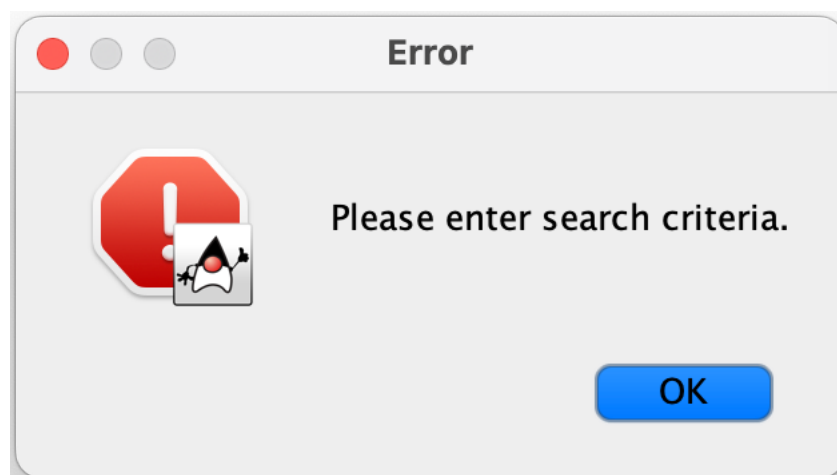


Рисунок 4.5.2 – Сообщение об ошибке при отсутствии критерия

При несоответствии типа искомых данных также выводится сообщение об ошибке (см. рисунок 4.5.3).

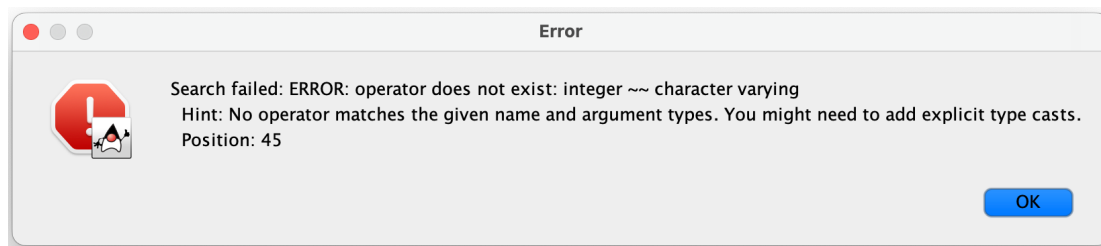
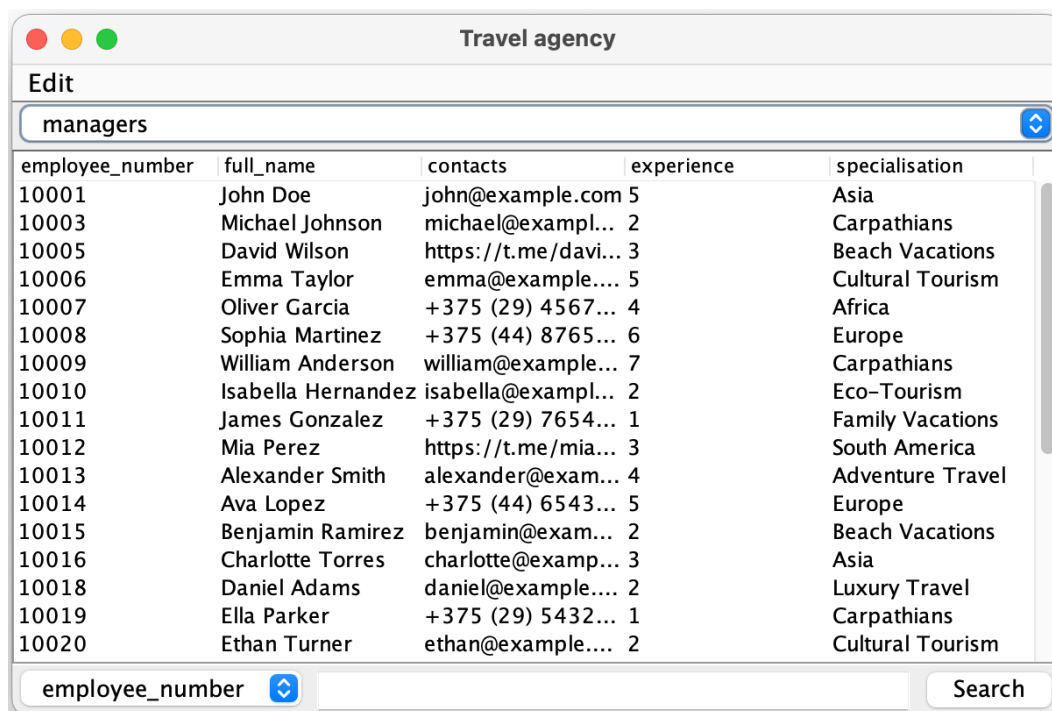


Рисунок 4.5.3 – Сообщение об ошибке при поиске данных

4.6 Сортировка данных

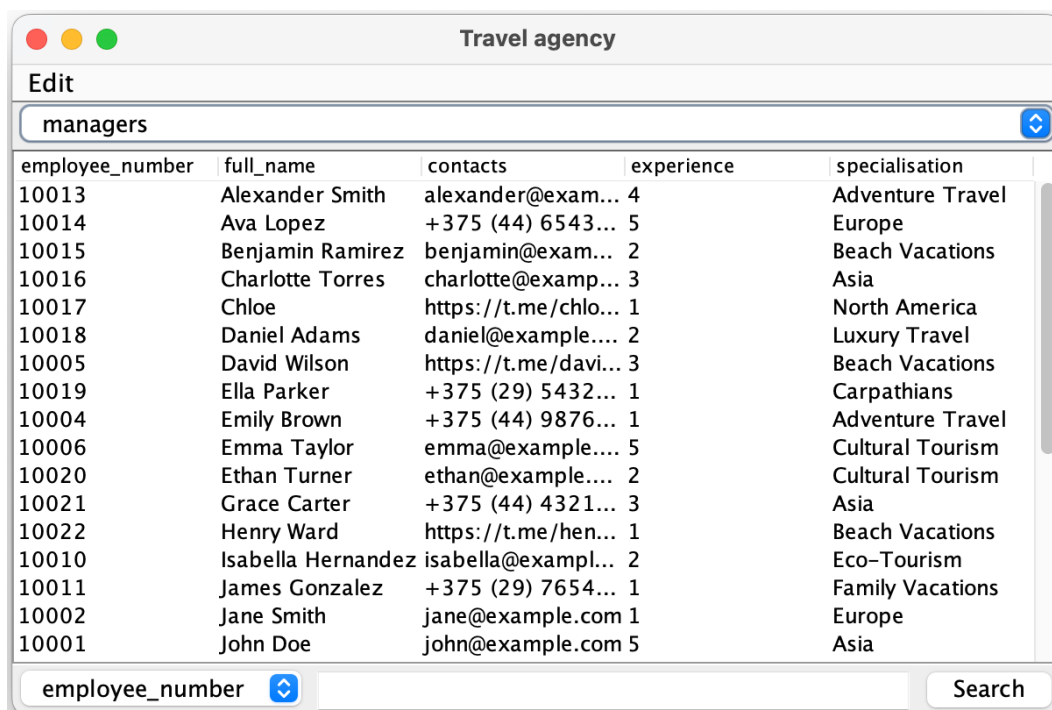
Для сортировки данных по значениям любого столбца необходимо щелкнуть ЛКМ по названию столбца один раз после загрузки таблицы для сортировки по возрастанию, еще раз – для сортировки по убыванию.

На рисунках 4.6.1 и 4.6.2 представлен пример сортировки таблицы менеджеров по их именам в алфавитном порядке:



employee_number	full_name	contacts	experience	specialisation
10001	John Doe	john@example.com	5	Asia
10003	Michael Johnson	michael@exampl...	2	Carpathians
10005	David Wilson	https://t.me/davi...	3	Beach Vacations
10006	Emma Taylor	emma@example....	5	Cultural Tourism
10007	Oliver Garcia	+375 (29) 4567...	4	Africa
10008	Sophia Martinez	+375 (44) 8765...	6	Europe
10009	William Anderson	william@example...	7	Carpathians
10010	Isabella Hernandez	isabella@exampl...	2	Eco-Tourism
10011	James Gonzalez	+375 (29) 7654...	1	Family Vacations
10012	Mia Perez	https://t.me/mia...	3	South America
10013	Alexander Smith	alexander@exam...	4	Adventure Travel
10014	Ava Lopez	+375 (44) 6543...	5	Europe
10015	Benjamin Ramirez	benjamin@exam...	2	Beach Vacations
10016	Charlotte Torres	charlotte@examp...	3	Asia
10018	Daniel Adams	daniel@example....	2	Luxury Travel
10019	Ella Parker	+375 (29) 5432...	1	Carpathians
10020	Ethan Turner	ethan@example....	2	Cultural Tourism

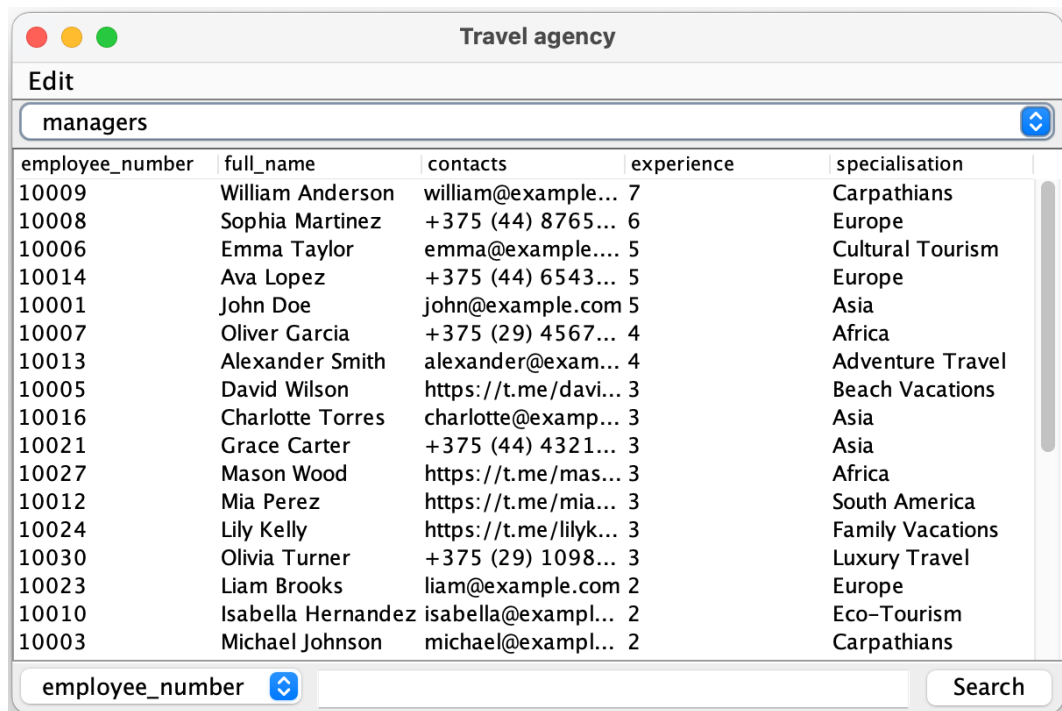
Рисунок 4.6.1 – Таблица до сортировки данных по именам



employee_number	full_name	contacts	experience	specialisation
10013	Alexander Smith	alexander@exam...	4	Adventure Travel
10014	Ava Lopez	+375 (44) 6543...	5	Europe
10015	Benjamin Ramirez	benjamin@exam...	2	Beach Vacations
10016	Charlotte Torres	charlotte@examp...	3	Asia
10017	Chloe	https://t.me/chlo...	1	North America
10018	Daniel Adams	daniel@example....	2	Luxury Travel
10005	David Wilson	https://t.me/davi...	3	Beach Vacations
10019	Ella Parker	+375 (29) 5432...	1	Carpathians
10004	Emily Brown	+375 (44) 9876...	1	Adventure Travel
10006	Emma Taylor	emma@example....	5	Cultural Tourism
10020	Ethan Turner	ethan@example....	2	Cultural Tourism
10021	Grace Carter	+375 (44) 4321...	3	Asia
10022	Henry Ward	https://t.me/hen...	1	Beach Vacations
10010	Isabella Hernandez	isabella@exampl...	2	Eco-Tourism
10011	James Gonzalez	+375 (29) 7654...	1	Family Vacations
10002	Jane Smith	jane@example.com	1	Europe
10001	John Doe	john@example.com	5	Asia

Рисунок 4.6.2 – Таблица после сортировки данных по именам

На рисунке 4.6.3 представлен пример сортировки таблицы менеджеров по стажу от самых опытных до новичков в сфере:



employee_number	full_name	contacts	experience	specialisation
10009	William Anderson	william@example...	7	Carpathians
10008	Sophia Martinez	+375 (44) 8765...	6	Europe
10006	Emma Taylor	emma@example....	5	Cultural Tourism
10014	Ava Lopez	+375 (44) 6543...	5	Europe
10001	John Doe	john@example.com	5	Asia
10007	Oliver Garcia	+375 (29) 4567...	4	Africa
10013	Alexander Smith	alexander@exam...	4	Adventure Travel
10005	David Wilson	https://t.me/davi...	3	Beach Vacations
10016	Charlotte Torres	charlotte@examp...	3	Asia
10021	Grace Carter	+375 (44) 4321...	3	Asia
10027	Mason Wood	https://t.me/mas...	3	Africa
10012	Mia Perez	https://t.me/mia...	3	South America
10024	Lily Kelly	https://t.me/lilyk...	3	Family Vacations
10030	Olivia Turner	+375 (29) 1098...	3	Luxury Travel
10023	Liam Brooks	liam@example.com	2	Europe
10010	Isabella Hernandez	isabella@exempl...	2	Eco-Tourism
10003	Michael Johnson	michael@exempl...	2	Carpathians

Рисунок 4.6.3 – Таблица после сортировки данных по опыту работы

5 ЛИСТИНГ КОДА

```
package org.example;

import javax.swing.*.*;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableModel;
import java.awt.*.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.sql.*;
import java.util.Vector;

public class DatabaseViewer extends JFrame {
    private JComboBox<String> tables_list;
    private JTable table_db;
    private JScrollPane scroll_panel;
    private JTextField[] input_fields;
    private boolean is_ascending = true;
    private JComboBox<String> columns_list;
    private JTextField search_field;
    private Connection connection;

    public DatabaseViewer() {
        setTitle("Travel agency");
        setSize(600, 400);
    }
}
```

```

setLayout(new BorderLayout());

setDefaultCloseOperation(EXIT_ON_CLOSE);

JMenuBar menu_bar = new JMenuBar();
setJMenuBar(menu_bar);

JMenu file_menu = new JMenu("Edit");
menu_bar.add(file_menu);

JMenuItem add_item = new JMenuItem("Add data...");
add_item.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        addNewRow();
    }
});
file_menu.add(add_item);

JMenuItem delete_item = new JMenuItem("Delete data");
delete_item.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        deleteSelectedRow();
    }
});
file_menu.add(delete_item);

tables_list = new JComboBox<>();
tables_list.addActionListener(e -> {
    String selected = (String)
tables_list.getSelectedItem();
    if (selected != null)
        displayTable(selected);
});
add(tables_list, BorderLayout.NORTH);

table_db = new JTable();
table_db.setAutoCreateRowSorter(true);
table_db.getTableHeader().addMouseListener(new
MouseListener() {
    @Override
    public void mouseClicked(MouseEvent e) {
        int column =
table_db.columnAtPoint(e.getPoint());
        if (column != -1) {
            String table_name = (String)
tables_list.getSelectedItem();
            String column_name =
table_db.getColumnName(column);

            is_ascending = !is_ascending;

```

```

        displayTable(table_name, column_name);
    }
}
});

columns_list = new JComboBox<>();
search_field = new JTextField();
JButton search_button = new JButton("Search");
search_button.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e) {
        searchRows();
    }
});

JPanel search_panel = new JPanel(new BorderLayout());
search_panel.add(columns_list, BorderLayout.WEST);
search_panel.add(search_field, BorderLayout.CENTER);
search_panel.add(search_button, BorderLayout.EAST);
add(search_panel, BorderLayout.SOUTH);

// listeners
table_db.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (SwingUtilities.isRightMouseButton(e)) {
            int row =
table_db.rowAtPoint(e.getPoint());
            int col =
table_db.columnAtPoint(e.getPoint());

            if (row >= 0 && col >= 0)
                editCellValue(row, col);
        }
    }
});

tables_list.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        columns_list.removeAllItems();
        String table_name = (String)
tables_list.getSelectedItem();

        try {
            DatabaseMetaData meta_data =
connection.getMetaData();
            ResultSet columns =
meta_data.getColumns(null, null, table_name, null);
            while (columns.next()) {

```

```

        String      columnName      =
columns.getString("COLUMN_NAME");
        columns_list.addItem(columnName);
    }
}
catch (SQLException ex) {
    ex.printStackTrace();

JOptionPane.showMessageDialog(DatabaseViewer.this, "Failed to
fetch column names: " + ex.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    }
}
});

columns_list.addActionListener(new ActionListener()
{
    @Override
    public void actionPerformed(ActionEvent e) {
        String      column_name      =      (String)
columns_list.getSelectedItem();
        String      table_name      =      (String)
tables_list.getSelectedItem();

        if (column_name != null && table_name !=
null)

            displayTable(table_name, column_name);

            search_field.setText("");
    }
});

scroll_panel = new JScrollPane(table_db);
add(scroll_panel, BorderLayout.CENTER);

try {
    connectToDatabase();
    loadTables();
}
catch (SQLException e) {
    e.printStackTrace();
    JOptionPane.showMessageDialog(this, "Failed to
connect to a database: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
}

}

private void connectToDatabase() throws SQLException {
    String      url      =
"jdbc:postgresql://localhost:5432/Travel+agency?user=nina";
    connection = DriverManager.getConnection(url);
}

```

```

        private void loadTables() throws SQLException {
            DatabaseMetaData data_db = connection.getMetaData();
            ResultSet tables = data_db.getTables(null, null,
            null, new String[]{"TABLE"});

            while (tables.next()) {
                String table_name =
            tables.getString("TABLE_NAME");
                tables_list.addItem(table_name);
            }
        }

        private void displayTable(String table_name) {
            displayTable(table_name, null);
        }

        private void displayTable(String table_name, String
            order_by_column) {
            try {
                String query = "SELECT * FROM " + table_name;
                if (order_by_column != null) {
                    query += " ORDER BY " + order_by_column;
                    if (!is_ascending) {
                        query += " DESC";
                    }
                }
            }

            Statement statement =
            connection.createStatement();
            ResultSet result =
            statement.executeQuery(query);
            ResultSetMetaData meta_data =
            result.getMetaData();

            int col_counter = meta_data.getColumnCount();
            Vector<String> columns = new Vector<>();
            Vector<Vector<Object>> data = new Vector<>();

            for (int i=1; i<=col_counter; i++)
                columns.add(meta_data.getColumnName(i));

            while (result.next()) {
                Vector<Object> row = new Vector<>();
                for (int i=1; i<=col_counter; i++)
                    row.add(result.getObject(i));
                data.add(row);
            }

            DefaultTableModel model = new
            DefaultTableModel(data, columns);
            table_db.setModel(model);
        }
    }

```



```

        }
        catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Failed to
display a table: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    private void addNewRow() {
        try {
            String table_name = (String)
tables_list.getSelectedItem();
            if (table_name == null) {
                JOptionPane.showMessageDialog(this, "Choose
a table for adding new data", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            Statement statement =
connection.createStatement();

            ResultSet result =
statement.executeQuery("SELECT * FROM " + table_name);
            ResultSetMetaData meta_data =
result.getMetaData();

            int col_counter = meta_data.getColumnCount();
            String[] columns = new String[col_counter];
            input_fields = new JTextField[col_counter];

            JPanel input_panel = new JPanel(new
GridLayout(col_counter, 2));

            for (int i=1; i<=col_counter; i++) {
                columns[i-1] = meta_data洗getColumnName(i);
                input_panel.add(new JLabel(columns[i-1]));

                JTextField text_field = new JTextField();
                input_fields[i-1] = text_field;
                input_panel.add(text_field);
            }

            int insertion =
JOptionPane.showConfirmDialog(null, input_panel, "Enter data",
JOptionPane.OK_CANCEL_OPTION);
            if (insertion == JOptionPane.OK_OPTION) {
                StringBuilder query = new
StringBuilder("INSERT INTO " + table_name + " (");

                for (int i=0; i<col_counter; i++) {
                    query.append(columns[i]);

```

```

        if (i < col_counter - 1)
            query.append(", ");
    }

    query.append(") VALUES (");
    for (int i=0; i<col_counter; i++) {
        query.append(input_fields[i].getText()).append("'");
        if (i < col_counter - 1)
            query.append(", ");
    }
    query.append(")");

    connection.createStatement().executeUpdate(query.toString());
    displayTable(table_name);
    }
    }
    catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this, "Failed to
add a new row: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    }
    }

    private void deleteSelectedRow() {
        try {
            String table_name = (String)
tables_list.getSelectedItem();
            if (table_name == null) {
                JOptionPane.showMessageDialog(this, "Choose
a table for deleting data", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            int selected_row = table_db.getSelectedRow();
            if (selected_row == -1) {
                JOptionPane.showMessageDialog(this, "Choose,
which row to delete", "Error", JOptionPane.ERROR_MESSAGE);
                return;
            }

            TableModel model = table_db.getModel();
            String key_column = model.getColumnNames(0);
            Object id = table_db.getValueAt(selected_row, 0);

            String query;
            if (table_name.equals("clients") ||
table_name.equals("managers"))
                query = "DELETE FROM " + table_name + " WHERE
" + key_column + " = '" + id + "'";

```

```

        else
            query = "DELETE FROM " + table_name + " WHERE
" + key_column + " = " + id;

connection.createStatement().executeUpdate(query);
        displayTable(table_name);
    }
    catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(this, "Failed to
delete a row: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
    }
}

    private void editCellValue(int row, int col) {
        DefaultTableModel model = (DefaultTableModel)
table_db.getModel();
        String table_name = (String)
tables_list.getSelectedItem();
        String column_name = table_db.getColumnName(col);
        String key_column = table_db.getColumnName(0);

        String new_value_str =
JOptionPane.showInputDialog(this, "Enter a new value:", "Edit the
field", JOptionPane.PLAIN_MESSAGE);
        if (new_value_str != null) {
            Object new_value;
            if (new_value_str.startsWith("\"") &&
new_value_str.endsWith("\"")) //string
                new_value = new_value_str.substring(1,
new_value_str.length() - 1);
            else if
(new_value_str.chars().allMatch(Character::isDigit)) //digit
                new_value = Integer.parseInt(new_value_str);
            else
                new_value = new_value_str;

            try {
                Object key_value = model.getValueAt(row, 0);
                String sql = "UPDATE " + table_name + " SET
" + column_name + " = ? WHERE " + key_column + " = ?";
                PreparedStatement preparedStatement =
connection.prepareStatement(sql);

                preparedStatement.setObject(1, new_value);
                preparedStatement.setObject(2, key_value);
                preparedStatement.executeUpdate();

                model.setValueAt(new_value, row, col);
            }

```

```

        catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(this, "Failed
to update a value: " + e.getMessage(), "Error",
JOptionPane.ERROR_MESSAGE);
        }
    }

    private void searchRows() {
        String table_name = (String)
tables_list.getSelectedItemAt();
        String search_query = search_field.getText();
        String search_col = (String)
columns_list.getSelectedItemAt();

        if (!search_query.isEmpty() && search_col != null) {
            try {
                String query;
                if (isInteger(search_query))
                    query = "SELECT * FROM " + table_name +
" WHERE " + search_col + " = " + search_query;
                else {
                    query = "SELECT * FROM " + table_name +
" WHERE " + search_col + " LIKE ?";
                    search_query = "%" + search_query + "%";
                }

                PreparedStatement prep_statement =
connection.prepareStatement(query);
                ResultSet result;
                if (!isInteger(search_query)) {
                    prep_statement.setObject(1,
search_query);

                    result = prep_statement.executeQuery();
                }
                else
                    result = prep_statement.executeQuery();

                table_db.setModel(buildTableModel(result));
            } catch (SQLException e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(this, "Search
failed: " + e.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
            }
        } else {
            JOptionPane.showMessageDialog(this, "Please
enter search criteria.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private boolean isInteger(String str) {

```

```

        try {
            Integer.parseInt(str);
            return true;
        }
        catch (NumberFormatException e) {
            return false;
        }
    }

    public static DefaultTableModel
buildTableModel(ResultSet result) throws SQLException {
        ResultSetMetaData meta_data = result.getMetaData();

        int col_counter = meta_data.getColumnCount();

        DefaultTableModel model = new DefaultTableModel();

        for (int ind=1; ind<=col_counter; ind++)
            model.addColumn(meta_data洗getColumnNane(ind));

        while (result.next()) {
            Vector<Object> row = new Vector<>();

            for (int ind=1; ind<=col_counter; ind++)
                row.add(result.getObject(ind));
            model.addRow(row);
        }

        return model;
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            DatabaseViewer viewer = new DatabaseViewer();
            viewer.setVisible(true);
        });
    }
}

```