1. Querying theDatabasebooks. Write a program which allows the user to enter any query into the program. The program should displays the result of a query in a JTable, using an objectTableModelto provide thedata to the JTable.

Output:



**app.JAVA:**

import javax.swing.\*;  
import javax.swing.table.TableModel;  
import javax.swing.table.TableRowSorter;  
import java.awt.\*;  
import java.awt.event.WindowAdapter;  
import java.awt.event.WindowEvent;  
import java.sql.SQLException;  
import java.util.regex.PatternSyntaxException;  
  
public class app extends JFrame {  
 private static final String *DATABASE\_URL* = "jdbc:mysql://localhost:3306/library";  
 private static final String *USERNAME* = "root";  
 private static final String *PASSWORD* = "";  
 private static final String *DEFAULT\_QUERY* = "SELECT \* FROM Authors";  
 private static ResultSetTableModel *tableModel*;  
  
 public static void main(String[] args) {  
 try {  
 *tableModel* = new ResultSetTableModel(*DATABASE\_URL*, *USERNAME*, *PASSWORD*, *DEFAULT\_QUERY*);  
 final JTextArea queryArea = new JTextArea(*DEFAULT\_QUERY*, 10, 150);  
 queryArea.setWrapStyleWord(true);  
 queryArea.setLineWrap(true);  
  
 JScrollPane scrollPane = new JScrollPane(queryArea, ScrollPaneConstants.*VERTICAL\_SCROLLBAR\_AS\_NEEDED*, ScrollPaneConstants.*HORIZONTAL\_SCROLLBAR\_AS\_NEEDED*);  
 JButton submitButton = new JButton("Submit Query");  
  
 Box boxNorth = Box.*createHorizontalBox*();  
 boxNorth.add(scrollPane);  
 boxNorth.add(submitButton);  
  
 JTable resultTable = new JTable(*tableModel*);  
  
 JLabel filterLabel = new JLabel("Filter : ");  
 final JTextField filterText = new JTextField();  
 JButton filterButton = new JButton("Apply Filter");  
 Box boxSouth = Box.*createHorizontalBox*();  
  
 boxSouth.add(filterLabel);  
 boxSouth.add(filterText);  
 boxSouth.add(filterButton);  
  
 JFrame window = new JFrame("Displaying Query Results");  
  
 window.add(boxNorth, BorderLayout.*NORTH*);  
 window.add(new JScrollPane(resultTable), BorderLayout.*CENTER*);  
 window.add(boxSouth, BorderLayout.*SOUTH*);  
  
 submitButton.addActionListener(event -> {  
 System.*out*.println("inside submit button event");  
 try {  
 *tableModel*.setQuery(queryArea.getText());  
 } catch (SQLException sqlException) {  
 sqlException.printStackTrace();  
 JOptionPane.*showMessageDialog*(null, sqlException.getMessage(), "Database error", JOptionPane.*ERROR\_MESSAGE*);  
  
 try {  
 *tableModel*.setQuery(*DEFAULT\_QUERY*);  
 queryArea.setText(*DEFAULT\_QUERY*);  
 } catch (SQLException sqlException2) {  
 JOptionPane.*showMessageDialog*(null, sqlException2.getMessage(), "Database error", JOptionPane.*ERROR\_MESSAGE*);  
 *tableModel*.disconnectFromDatabase();  
 System.*exit*(1);  
 }  
 }  
 }  
 );  
  
 final TableRowSorter<TableModel> sorter = new TableRowSorter<>(*tableModel*);  
 resultTable.setRowSorter(sorter);  
  
 // pass filter text to listener  
 filterButton.addActionListener(e -> {  
 System.*out*.println("inside filter action");  
 String text = filterText.getText();  
 System.*out*.println(text);  
 if (text.length() == 0)  
 sorter.setRowFilter(null);  
 else {  
 try {  
 sorter.setRowFilter(RowFilter.*regexFilter*(text));  
 } catch (PatternSyntaxException pse) {  
 JOptionPane.*showMessageDialog*(null, "Bad regex pattern", "Bad regex pattern",  
 JOptionPane.*ERROR\_MESSAGE*);  
 }  
  
 }  
 }  
 );  
 window.setDefaultCloseOperation(*DISPOSE\_ON\_CLOSE*);  
 window.setSize(1000, 500);  
 window.setVisible(true);  
  
 window.addWindowListener(new WindowAdapter() {  
 public void windowClosed(WindowEvent event) {  
 *tableModel*.disconnectFromDatabase();  
 System.*exit*(0);  
 }  
 });  
  
 } catch (Exception sqlException) {  
 JOptionPane.*showMessageDialog*(null, sqlException.getMessage(), "Database error", JOptionPane.*ERROR\_MESSAGE*);  
 sqlException.printStackTrace();  
 *tableModel*.disconnectFromDatabase();  
 System.*exit*(1);  
 }  
 }  
}

**ResultSetTableModel.JAVA:**

import java.sql.Connection;  
import java.sql.Statement;  
import java.sql.DriverManager;  
import java.sql.ResultSet;  
import java.sql.ResultSetMetaData;  
import java.sql.SQLException;  
import javax.swing.table.AbstractTableModel;  
  
public class ResultSetTableModel extends AbstractTableModel {  
 private final Connection connection;  
 private final Statement statement;  
 private ResultSet resultSet;  
 private ResultSetMetaData metaData;  
 private int numberOfRows;  
  
 private boolean connectedToDatabase;  
  
 public ResultSetTableModel(String url, String username, String password, String query) throws SQLException, ClassNotFoundException {  
 Class.*forName*("com.mysql.cj.jdbc.Driver");  
 connection = DriverManager.*getConnection*(url, username, password);  
 statement = connection.createStatement(ResultSet.*TYPE\_SCROLL\_INSENSITIVE*, ResultSet.*CONCUR\_READ\_ONLY*);  
 connectedToDatabase = true;  
 setQuery(query);  
 }  
  
 public Class getColumnClass(int column) throws IllegalStateException {  
 if (!connectedToDatabase) throw new IllegalStateException("Not Connected to Database");  
 try {  
 String className = metaData.getColumnClassName(column + 1);  
 return Class.*forName*(className);  
 } catch (Exception exception) {  
 exception.printStackTrace();  
 }  
 return Object.class; // if problems occur above, assume type Object69  
  
 }  
  
 public String getColumnName(int column) throws IllegalStateException {  
 if (!connectedToDatabase)  
 throw new IllegalStateException("Not Connected to Database");// determine column name  
 try {  
 return metaData.getColumnName(column + 1);  
 } catch (SQLException sqlException) {  
 sqlException.printStackTrace();  
 }  
 return "";  
 }  
  
 public void setQuery(String query) throws SQLException, IllegalStateException {  
 System.*out*.println("inside set query");  
 if (!connectedToDatabase) {  
 throw new IllegalStateException("Not Connected to Database"); // specify query and execute it152  
 }  
 resultSet = statement.executeQuery(query);// obtain metadata for ResultSet155  
 metaData = resultSet.getMetaData(); // determine number of rows in ResultSet158  
 resultSet.last(); // move to last row159  
 numberOfRows = resultSet.getRow();  
 System.*out*.println(numberOfRows);  
 fireTableStructureChanged();  
  
 }  
  
 public void disconnectFromDatabase() {  
 if (connectedToDatabase) {  
 // close Statement and Connection  
 try {  
 resultSet.close();  
 statement.close();  
 connection.close();  
 } catch (SQLException sqlException) {  
 sqlException.printStackTrace();  
 } finally {  
 connectedToDatabase = false;  
 }  
 }  
 }  
  
 @Override  
 public int getRowCount() {  
 if (!connectedToDatabase)  
 throw new IllegalStateException("Not Connected to Database");  
 return numberOfRows;  
 }  
  
 @Override  
 public int getColumnCount() {  
 if (!connectedToDatabase)  
 throw new IllegalStateException("Not Connected to Database"); // determine number of columns79  
 try {  
 return metaData.getColumnCount();  
 } catch (SQLException sqlException) {  
 sqlException.printStackTrace();  
 }  
 return 0;  
 }  
  
 @Override  
 public Object getValueAt(int rowIndex, int columnIndex) {  
 if (!connectedToDatabase) throw new IllegalStateException("Not Connected to Database");  
 try {  
 resultSet.absolute(rowIndex + 1);  
 return resultSet.getObject(columnIndex + 1);  
 } catch (SQLException sqlException) {  
 sqlException.printStackTrace();  
 }  
 return "";  
 }  
}