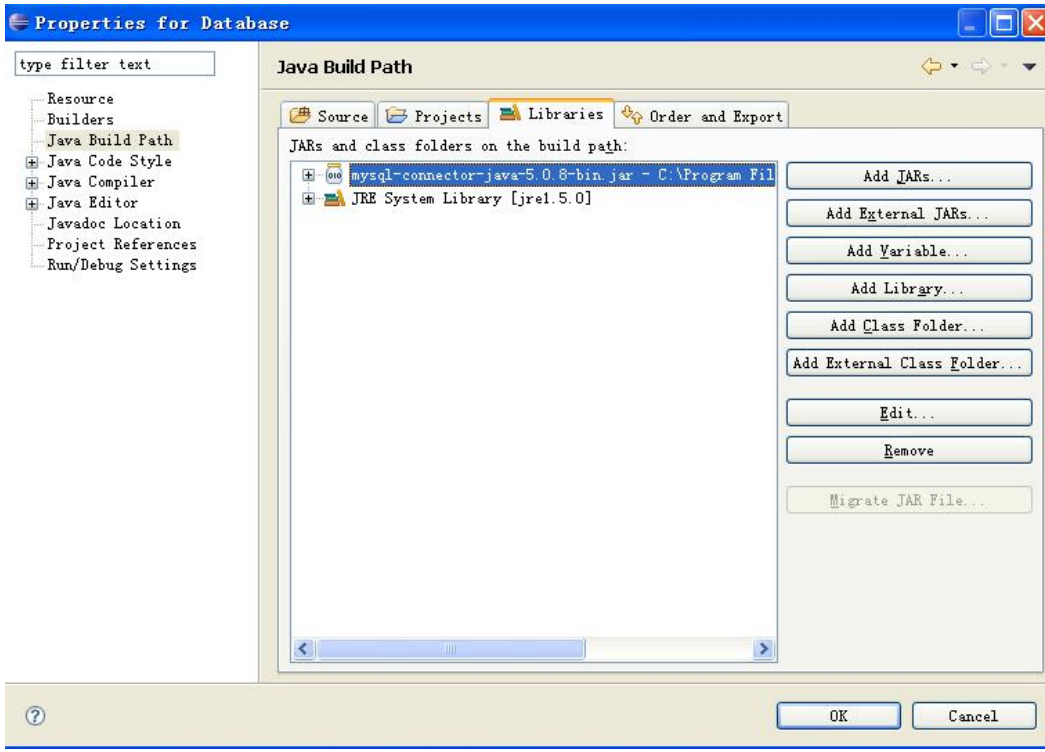
Chapter 24
Manipulating Databases with JDBC

- **Connect to a database**
- **Query the database**
- **Display the results of the query in `JTable`**

24.8.1 Connecting to and Querying a Database

- `DisplayAuthors`
  - Retrieves the entire `authors` table
  - Displays the data in the standard output stream
  - Example illustrates
    - Connect to the database
    - Query the database
    - Process the result



```
import java.sql.SQLException;

public class DisplayAuthors {
   public static void main(String args[]) {
      final String DATABASE_URL = "jdbc:derby:books";
      final String SELECT_QUERY =
         "SELECT authorID, firstName, lastName FROM
authors";
```

通过数据库进行查找

```
// use try-with-resources to connect to and query the database
      try (
         Connection connection = DriverManager.getConnection(
            DATABASE_URL, "deitel", "deitel");
         Statement statement = connection.createStatement();
         ResultSet resultSet = statement.executeQuery(SELECT_QUERY)) {

         // get ResultSet's meta data
         ResultSetMetaData metaData = resultSet.getMetaData();
         int numberOfColumns = metaData.getColumnCount();

         System.out.printf("Authors Table of Books Database:%n%n");

         // display the names of the columns in the ResultSet
         for (int i = 1; i <= numberOfColumns; i++) {
            System.out.printf("%-8s\t", metaData.getColumnName(i));
         }
         System.out.println();
```

存储结果

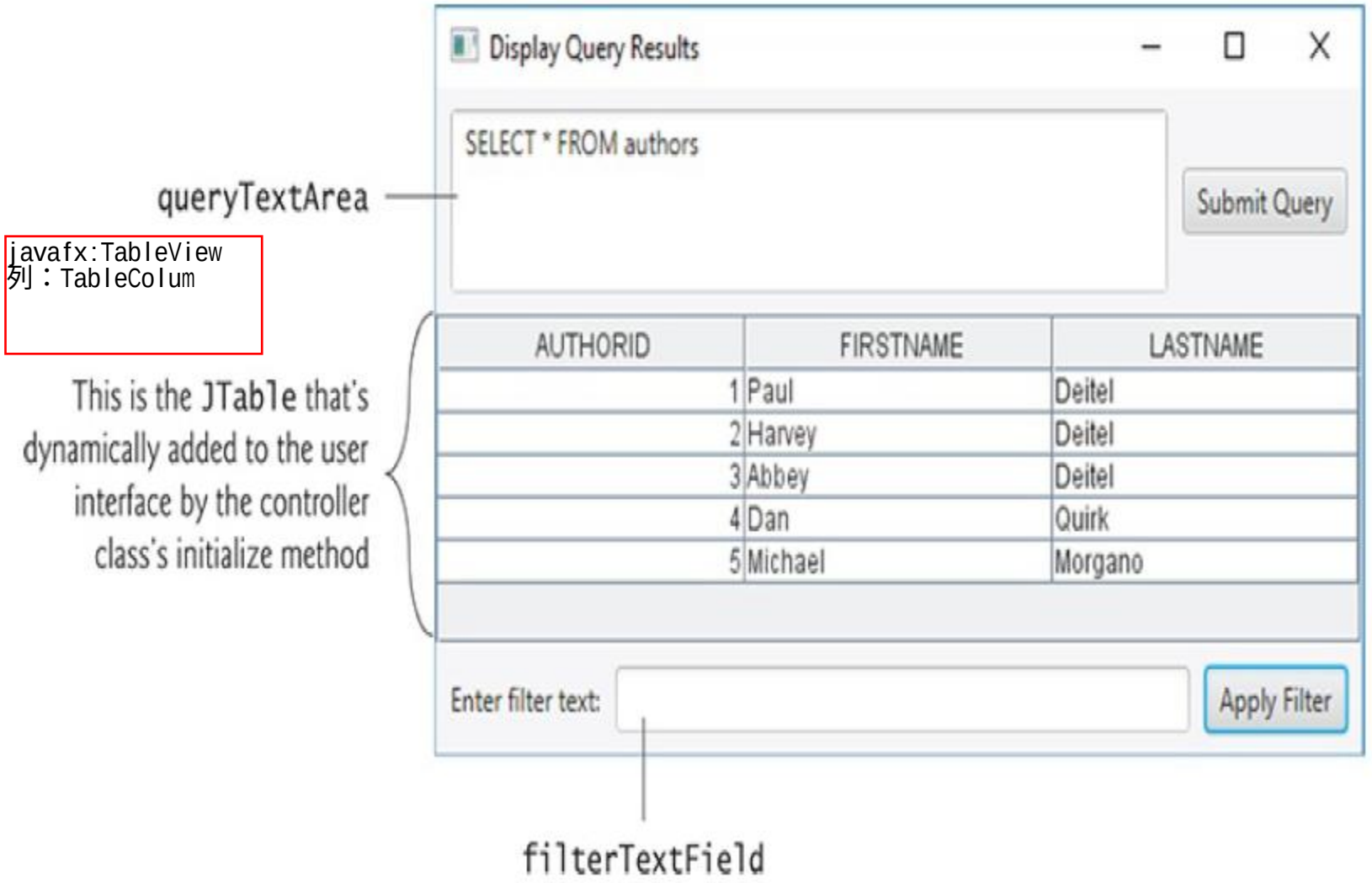
多少行，多少列，每列名字

```
// display query results
         while (resultSet.next()) {
            for (int i = 1; i <= numberOfColumns; i++) {
               System.out.printf("%-8s\t", resultSet.getObject(i));
            }
            System.out.println();
         }
      }
      catch (SQLException sqlException) {
         sqlException.printStackTrace();
      }
   }
}
```

AUTHORID	FIRSTNAME	LASTNAME
1	Paul	Deitel
2	Harvey	Deitel
3	Abbey	Deitel
4	Dan	Quirk
5	Michael	Morgano

RDBMS	Database URL format
MySQL	`jdbc:mysql://`*hostname*`:`*portNumber*`/`*databaseName*
ORACLE	`jdbc:oracle:thin:@`*hostname*`:`*portNumber*`:`*databaseName*
DB2	`jdbc:db2:`*hostname*`:`*portNumber*`/`*databaseName*
PostgreSQL	`jdbc:postgresql://`*hostname*`:`*portNumber*`/`*databaseName*
Java DB/Apache Derby	`jdbc:derby:`*dataBaseName* (embedded; used in this chapter) `jdbc:derby://`*hostname*`:`*portNumber*`/`*databaseName* (network)
Microsoft SQL Server	`jdbc:sqlserver://`*hostname*`:`*portNumber*`;databaseName=`*dataBaseName*
Sybase	`jdbc:sybase:Tds:`*hostname*`:`*portNumber*`/`*databaseName*

Popular JDBC database URL formats.


queryTextArea
javafx:TableView
列：TableColum
This is the JTable that's dynamically added to the user interface by the controller class's initialize method
Display Query Results
SELECT * FROM authors
Submit Query
AUTHORID
FIRSTNAME
LASTNAME
1 Paul Deitel
2 Harvey Deitel
3 Abbey Deitel
4 Dan Quirk
5 Michael Morgano
Enter filter text:
Apply Filter
filterTextField

```
// Fig. 24.29: DisplayQueryResultsController.java
// Controller for the DisplayQueryResults app
import java.sql.SQLException;
import java.util.regex.PatternSyntaxException;

import javafx.embed.swing.SwingNode;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.scene.control.Alert;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.control.TextArea;
import javafx.scene.control.TextField;
import javafx.scene.layout.BorderPane;

import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.RowFilter;
import javax.swing.table.TableModel;
import javax.swing.table.TableRowSorter;
```

```
public class DisplayQueryResultsController {
   @FXML private BorderPane borderPane;
   @FXML private TextArea queryTextArea;
   @FXML private TextField filterTextField;

   // database URL, username and password
   private static final String DATABASE_URL = "jdbc:derby:books";
   private static final String USERNAME = "deitel";
   private static final String PASSWORD = "deitel";

   // default query retrieves all data from Authors table
   private static final String DEFAULT_QUERY = "SELECT * FROM authors";

   // used for configuring JTable to display and sort data
   private ResultSetTableModel tableModel;
   private TableRowSorter<TableModel> sorter;
```

```
public void initialize() {
      queryTextArea.setText(DEFAULT_QUERY);

      // create ResultSetTableModel and display database table
      try {
         // create TableModel for results of DEFAULT_QUERY
         tableModel = new ResultSetTableModel(DATABASE_URL,
            USERNAME, PASSWORD, DEFAULT_QUERY);

         // create JTable based on the tableModel
         JTable resultTable = new JTable(tableModel);

         // set up row sorting for JTable
         sorter = new TableRowSorter<TableModel>(tableModel);
         resultTable.setRowSorter(sorter);

         // configure SwingNode to display JTable, then add to
borderPane
         SwingNode swingNode = new SwingNode();
         swingNode.setContent(new JScrollPane(resultTable));
         borderPane.setCenter(swingNode);
      }
```

```
catch (SQLException sqlException) {
        displayAlert(AlertType.ERROR, "Database Error",
           sqlException.getMessage());
        tableModel.disconnectFromDatabase(); // close connection
        System.exit(1); // terminate application
     }
  }
```

```
// query the database and display results in JTable
   @FXML
   void submitQueryButtonPressed(ActionEvent event) {
      // perform a new query
      try {
         tableModel.setQuery(queryTextArea.getText());
      }
      catch (SQLException sqlException) {
         displayAlert(AlertType.ERROR, "Database Error",
            sqlException.getMessage());

         // try to recover from invalid user query
         // by executing default query
         try {
            tableModel.setQuery(DEFAULT_QUERY);
            queryTextArea.setText(DEFAULT_QUERY);
         }
         catch (SQLException sqlException2) {
            displayAlert(AlertType.ERROR, "Database Error",
               sqlException2.getMessage());
            tableModel.disconnectFromDatabase(); // close connection
            System.exit(1); // terminate application
         }
      }
   }
```

```
// apply specified filter to results
   @FXML
   void applyFilterButtonPressed(ActionEvent event) {
      String text = filterTextField.getText();

      if (text.length() == 0) {
         sorter.setRowFilter(null);
      }
      else {
         try {
            sorter.setRowFilter(RowFilter.regexFilter(text));
         }
         catch (PatternSyntaxException pse) {
            displayAlert(AlertType.ERROR, "Regex Error",
               "Bad regex pattern");
         }
      }
   }
```

```
 // display an Alert dialog
   private void displayAlert(
      AlertType type, String title, String message) {
      Alert alert = new Alert(type);
      alert.setTitle(title);
      alert.setContentText(message);
      alert.showAndWait();
   }
}
```

```
// Fig. 24.25: ResultSetTableModel.java
// A TableModel that supplies ResultSet data to a JTable.
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;

// ResultSet rows and columns are counted from 1 and JTable
// rows and columns are counted from 0. When processing
// ResultSet rows or columns for use in a JTable, it is
// necessary to add 1 to the row or column number to manipulate
// the appropriate ResultSet column (i.e., JTable column 0 is
// ResultSet column 1 and JTable row 0 is ResultSet row 1).
```

```
public class ResultSetTableModel extends AbstractTableModel {
   private final Connection connection;
   private final Statement statement;
   private ResultSet resultSet;
   private ResultSetMetaData metaData;
   private int numberOfRows;

   // keep track of database connection status
   private boolean connectedToDatabase = false;

   // constructor initializes resultSet and obtains its metadata object;
   // determines number of rows
   public ResultSetTableModel(String url, String username,
       String password, String query) throws SQLException {
      // connect to database
      connection = DriverManager.getConnection(url, username, password);

      // create Statement to query database
      statement = connection.createStatement(
         ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);

      // update database connection status
      connectedToDatabase = true;

      // set query and execute it
      setQuery(query);
   }
```

```
// get class that represents column type
  public Class getColumnClass(int column) throws IllegalStateException {
     // ensure database connection is available
     if (!connectedToDatabase) {
        throw new IllegalStateException("Not Connected to Database");
     }

     // determine Java class of column
     try {
        String className = metaData.getColumnClassName(column + 1);

        // return Class object that represents className
        return Class.forName(className);
     }
     catch (Exception exception) {
        exception.printStackTrace();
     }

     return Object.class; // if problems occur above, assume type Object
  }
```

```
// get number of columns in ResultSet
   public int getColumnCount() throws IllegalStateException {
      // ensure database connection is available
      if (!connectedToDatabase) {
         throw new IllegalStateException("Not Connected to
Database");
      }

      // determine number of columns
      try {
         return metaData.getColumnCount();
      }
      catch (SQLException sqlException) {
         sqlException.printStackTrace();
      }

      return 0; // if problems occur above, return 0 for number
of columns
   }
```

```
 // get name of a particular column in ResultSet
   public String getColumnName(int column) throws
IllegalStateException {
      // ensure database connection is available
      if (!connectedToDatabase) {
         throw new IllegalStateException("Not Connected to
Database");
      }

      // determine column name
      try {
         return metaData.getColumnName(column + 1);
      }
      catch (SQLException sqlException) {
         sqlException.printStackTrace();
      }

      return ""; // if problems, return empty string for column
name
   }
```

```
 // return number of rows in ResultSet
   public int getRowCount() throws IllegalStateException {
      // ensure database connection is available
      if (!connectedToDatabase) {
         throw new IllegalStateException("Not Connected to
Database");
      }

      return numberOfRows;
   }
```

```
// obtain value in particular row and column
  public Object getValueAt(int row, int column)
     throws IllegalStateException {

     // ensure database connection is available
     if (!connectedToDatabase) {
        throw new IllegalStateException("Not Connected to Database");
     }

     // obtain a value at specified ResultSet row and column
     try {
        resultSet.absolute(row + 1);
        return resultSet.getObject(column + 1);
     }
     catch (SQLException sqlException) {
        sqlException.printStackTrace();
     }

     return ""; // if problems, return empty string object
  }
```

```
 // set new database query string
   public void setQuery(String query)
      throws SQLException, IllegalStateException {

      // ensure database connection is available
      if (!connectedToDatabase) {
         throw new IllegalStateException("Not Connected to
Database");
      }

      // specify query and execute it
      resultSet = statement.executeQuery(query);

      // obtain metadata for ResultSet
      metaData = resultSet.getMetaData();

      // determine number of rows in ResultSet
      resultSet.last(); // move to last row
      numberOfRows = resultSet.getRow(); // get row number


      fireTableStructureChanged(); // notify JTable that model
has changed
   }
```

```
// close Statement and Connection
  public void disconnectFromDatabase() {
     if (connectedToDatabase) {
        // close Statement and Connection
        try {
           resultSet.close();
           statement.close();
           connection.close();
        }
        catch (SQLException sqlException) {
           sqlException.printStackTrace();
        }
        finally { // update database connection status
           connectedToDatabase = false;
        }
     }
  }
}
```

```
// Fig. 24.29: JdbcRowSetTest.java
// Displaying the contents of the Authors table using JdbcRowSet.
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;

public class JdbcRowSetTest {
   // JDBC driver name and database URL
   private static final String DATABASE_URL = "jdbc:derby:books";
   private static final String USERNAME = "deitel";
   private static final String PASSWORD = "deitel";
```

```
 // display each row
         while (rowSet.next()) {
            for (int i = 1; i <= numberOfColumns; i++) {
               System.out.printf("%-8s\t", rowSet.getObject(i));
            }
            System.out.println();
         }
      }
      catch (SQLException sqlException) {
         sqlException.printStackTrace();
         System.exit(1);
      }
   }
}
```

```
public static void main(String args[]) {
     // connect to database books and query database
     try (JdbcRowSet rowSet =
        RowSetProvider.newFactory().createJdbcRowSet()) {

        // specify JdbcRowSet properties
        rowSet.setUrl(DATABASE_URL);
        rowSet.setUsername(USERNAME);
        rowSet.setPassword(PASSWORD);
        rowSet.setCommand("SELECT * FROM Authors"); // set query
        rowSet.execute(); // execute query

        // process query results
        ResultSetMetaData metaData = rowSet.getMetaData();
        int numberOfColumns = metaData.getColumnCount();
        System.out.printf("Authors Table of Books Database:%n%n");

        // display rowset header
        for (int i = 1; i <= numberOfColumns; i++) {
           System.out.printf("%-8s\t", metaData.getColumnName(i));
        }
        System.out.println();
```