Table of Contents

Viewing and editing database tables are basic requirements for database applications. With JVX, is this a piece of cake and takes minimal effort.

The following steps need to be taken to display and edit a database table:

- Define a server object for access to a table
- Define a client object for access to the server object
- Create a form using the table

Example

We want to create a form for viewing and editing of a database table based on the first JVx Application.

The server object:

DBEdit.java

```
package apps.firstapp.frames;

import com.sibvisions.rad.persist.jdbc.DBStorage;

import apps.firstapp.Session;

/**
 * The LCO for the DBEdit WorkScreen.
 * <p/>
 * @author René Jahn
 */
public class DBEdit extends Session
{
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  // User-defined methods
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  /**
   * Returns the contacts storage.
   *
   * @return the contacts storage
   * @throws Exception if the initialization throws an error
   */
  public DBStorage getContacts() throws Exception
  {
    DBStorage dbsContacts = (DBStorage)get("contacts");

    if (dbsContacts == null)
    {
      dbsContacts = new DBStorage();
      dbsContacts.setDBAccess(getDBAccess());
      dbsContacts.setFromClause("CONTACTS");
      dbsContacts.setWritebackTable("CONTACTS");
```

```
      dbsContacts.open();

      put("contacts", dbsContacts);
    }

    return dbsContacts;
  }

}  // DBEdit
```

Data is read from the table CONTACTS; changes (insert, update, delete) are also made in the table CONTACTS. We recommend reading the data from views and writing it back in tables. This allows for easier field expansion and content changes.

Please note that we do not define a column list, since all columns are used for display in the GUI. In addition, it would be possible to use a single server object at the client for multiple tables with different column displays!

The screen code:

DBEditFrame.java

```
/**
 * A simple database table editor.
 * <p/>
 * @author René Jahn
 */
public class DBEditFrame extends UIInternalFrame
{
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  // Class members
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  /** the application. */
  private Application application;

  /** the communication connection to the server. */
  private AbstractConnection connection;

  /** the DataSource for fetching table data. */
  private RemoteDataSource dataSource = new RemoteDataSource();

  /** the contacts tabl. */
  private RemoteDataBook rdbContacts = new RemoteDataBook();

  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  // Initialization
  //~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  /**
```

```
   * Creates a new instance of DBEditFrame for a specific application.
   * <p/>
   * @param pApp the application
   * @throws Throwable if the remote access fails
   */
  public DBEditFrame(Application pApp) throws Throwable
  {
    super(pApp.getDesktopPane());

    application = pApp;

    initializeModel();
    initializeUI();
  }

  /**
   * Initializes the model.
   * <p/>
   * @throws Throwable if the initialization throws an error
   */
  private void initializeModel() throws Throwable
  {
    //we use a new "session" for the screen
    connection = ((MasterConnection)application.getConnection()).
                 createSubConnection("apps.firstapp.frames.DBEdit");
    connection.open();

    //data connection
    dataSource.setConnection(connection);
    dataSource.open();

    //table
    rdbContacts.setDataSource(dataSource);
    rdbContacts.setName("contacts");
    rdbContacts.open();
  }

  /**
   * Initializes the UI.
   * <p/>
   * @throws Exception if the initialization throws an error
   */
  private void initializeUI() throws Exception
  {
    UIGroupPanel group = new UIGroupPanel();
    group.setText("Available Contacts");

    UITable table = new UITable();
    table.setDataBook(rdbContacts);

    group.setLayout(new UIBorderLayout());
```

```
    group.add(table);

    //same behaviour as centered component in BorderLayout
    setLayout(new UIBorderLayout());
    add(group);

    setTitle("Contacts");
    setSize(new UIDimension(400, 500));
  }

}  // DBEditFrame
```

The binding to the table:

```
rdbContacts.setDataSource(dataSource);
rdbContacts.setName("contacts");
rdbContacts.open();
```

We use a RemoteDataBook, rdbContacts, to set the connection to the server as well as the name of the server object.

The user interface can be more or less elaborate depending on the requirements for the layout.

The table is displayed in the form using the following code:

```
UITable table = new UITable();
table.setDataBook(rdbContacts);

group.add(table);
```

Hints

Tables can be changed by default, and the data is displayed unfiltered. Data is loaded on demand so that no delays are caused, even with large datasets (> 100.000).

Additional information can be found in the API documentation or in other documents, such as Filtering, Master/Detail.