# Table of Contents

In Datenbankapplikationen ist die Anzeige und Bearbeitung von Daten aus Tabellen eine Grundvoraussetzung. Mit JVx ist genau das ein Kinderspiel und mit minimalem Aufwand verbunden.

Für die Anzeige und Bearbeitung einer Datenbank Tabelle sind folgende Schritte durchzuführen:

- Definieren eines Server Objektes für den Zugriff auf eine Tabelle
- Definieren eines Client Objektes für den Zugriff auf das Server Objekt
- Erstellen einer Maske mit der Tabelle

#### Anwendungsbeispiel

Wir verwenden Die erste JVx Applikation als Grundlage und erstellen eine Maske für die Anzeige und Bearbeitung einer Tabelle aus einer Datenbank.

Das Server Objekt:

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
```

Aus der Tabelle CONTACTS werden die Daten gelesen und Änderungen(Insert, Update, Delete) erfolgen ebenfalls in der Tabelle CONTACTS. Wir empfehlen die Daten aus Views auszulesen und in Tabellen zurückzuschreiben. Dadurch sind Felderweiterungen oder Inhaltsänderungen bequemer durchzuführen.

Es gilt zu beachten, daß wir keine Spaltenliste definieren da wir alle Spalten für die Anzeige im GUI verwenden. Zusätzlich wäre es möglich ein und dasselbe Server Objekt, am Client, für mehrere Tabellen mit unterschiedlichen Spaltendarstellungen zu nutzen!

Die Verwendung in der Maske am Client:

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
```

Die Anbindung der Tabelle erfolgt mit:

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

Wir verwenden ein RemoteDataBook, rdbContacts, und setzen die Verbindung zum Server sowie den Namen des Server Objektes.

Die Erstellung der Benutzeroberfläche kann mehr oder weniger Aufwändig betrieben werden, je nach Anforderung an das Layout.

Die Tabelle wird mit folgendem Code in der Maske angezeigt:

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

group.add(table);
```

## Hints

Die Tabelle ist Standardmäßig änderbar und die Daten werden ungefiltert angezeigt. Die Daten werden on-demand geladen, somit kommt es auch bei Tabellen mit > 100.000 Datensätzen zu keinen Verzögerungen.

Genauere Informationen über die Verwendung entnehmen sie der API Dokumentation oder aus anderen Dokumentationen wie z.B.: Filterung, Master/Detail.