Table of Contents

Version: 1.1 / 2024-11-18

Introduction

VisionX let's you effortlessly develop a data model by choosing a visual representation of the created data relation. This tutorial aims to explain the coherency between the model and the visual components. You can download the example from the VisionX Solution Store.

Starting Point

Let's assume we have already created an application with VisionX and PostgreSQL and want to develop a specific screen. We have an idea how the screen should look like, but we have not analyzed the required data model yet. This is the perfect starting point for our low-code platform VisionX! The example is based on the well-known scott/tiger schema from oracle.

Create an Input Screen With Dropdown Lists

We want to create an input screen for employees and their personal data, such as name, contact information, hire date, the current job, and the employee's department. Most input fields are text fields. We want a date-picker for the hire date, but department and job shall be selected from a dropdown list of possible values. How can we achieve that?

Define Data Model

Defining the data model in VisionX is straight forward: create a new table and specify the required fields and their respective data-types. Now select the field “Department” (1) and press “Make Combobox” (2).

To understand the implication on the created data model, press “More” (3). As simple as that, we have just created an additional table DEPARTMENTS and defined a foreign-key constraint from EMPLOYEES to DEPARTMENTS.

You can now drag and drop the input controls to your screen.

Create Foreign-Key Relation in an Existing Data Model

“Nice, but not always applicable”, you may think. You do not always develop an application from scratch but may want to reuse an existing data model. That's even more simple: if your existing data model already uses a foreign-key relation, VisionX will automatically create a dropdown list for all related subtables. Otherwise, you can create that relation with just a few mouse clicks.

Let's assume we have a JOBS table and we want that all jobs in that table are available as a dropdown list in our Employees screen. Our data model currently looks like this:

To create the relation and change the text-field “job” to a dropdown list, we edit the data-object “Employees”. In the wizard, press “More...”. Also make sure that the button “Database changes” (1) is toggled so that VisionX can create the foreign-key relation in the database. We create a new column “Jobs Id” and press “Make Combobox” (2).

> **Note**: You have to add a new column and choose “Make Combobox” in one step to create a foreign-key relation!

All we have to do is to specify the table that we want to use as lookup table; in our case, it is the JOBS table (1).

If we want additional data from the related table to be available in our Employees screen, we can check the corresponding columns here. Let's say we want to see the minimum salary for the employee's job. Just select the corresponding column (1) and the field is joined to the dropdown list (2). Press “Finish” (3) to apply your changes.

Voilá! We have just created a foreign-key relation between the tables EMPLOYEES and JOBS. Next to that, we have a new dropdown input control “Title” that provides a lookup to the JOBS table.

This is our updated data model:

If you want, you can insert some sample data into the tables using the script hr_popul_1.sql.

Your Employees screen, created with VisionX, could look like this now (for desktop and web):

Create Dropdown Without Foreign-Key Relation

You may still hesitate: what if you cannot, or do not want to, alter the data model? Don't worry, you can also create a dropdown list without an underlying foreign-key relation.

Let's try based on an example: I created a new Departments screen for managing departments and their managers using the existing database table DEPARTMENTS. Let's add a column “Manager Id” to our DEPARTMENTS table by editing the data-object “Departments” in VisionX. Make sure that “Database changes” (1) is toggled and press the (+) button (2) to define the column (3). Press “Finish” (4) to finally add the column to the table.

You could also have added the column directly in the database. Let's verify our current data model. Note that there is no relation whatsoever between DEPARTMENTS.MANAGER_ID and the table EMPLOYEES:

What we want now is a dropdown list to select the department's manager. Use the data wizard of the data-object “Departments” again, press “More…”, select the column “Manager Id”, and press “Define Combobox”. In the “CombobBox Details”, select the table “Employees” (1) and check its columns “First Name” and “Last Name” (2). Apply your changes by pressing “Finish” (3).

> **Note**: You have to choose “Define Combobox” on an already existing column to avoid the creation of a foreign-key relation!

That's it. We have created a dropdown list without an underlying foreign-key relation. For desktop, the result will look like this:

Just in case you want to know how this works: JVx offers a a simple solution for linking tables called “Automatic Link Reference”. For every database table (or view) you access from within your screen, a so-called “Storage” is created in the screen's server-class.

With our EPlug Plugin, you can jump to the storage's source code in Eclipse by opening the Configure Server-Side Data Storages Wizard (1) and pressing “Show Source code” (2).

The code of that storage definition is short and almost self-explaining:

```
    /**
     * Departments.java (Generated server-class for
DepartmentsWorkScreen.java)
     *
     * Gets the departments database storage.
     *
     * @throws Exception if the DBStorage couldn't initialized.
```

```
     * @return the departments DBStorage.
     */
    public DBStorage getDepartments() throws Exception
    {
        // get storage configuration from cache
        DBStorage dbsDepartments = (DBStorage)get("departments");

        if (dbsDepartments == null)
        {
            // create Database storage
            dbsDepartments = new DBStorage();

            // specify table or view to access via that storage
            dbsDepartments.setWritebackTable("departments");

            // specify the database access object to use (default)
            dbsDepartments.setDBAccess(getDBAccess());

            // link the columns "ID", "FIRST_NAME", "LAST_NAME" from the
table "employees"
            // and reference them in this storage as "MANAGER_ID",
"MANAGER_FIRST_NAME", "MANAGER_LAST_NAME".
            dbsDepartments.createAutomaticLinkReference(
                    new String[] { "MANAGER_ID", "MANAGER_FIRST_NAME",
"MANAGER_LAST_NAME" },
                    "employees",
                    new String[] { "ID", "FIRST_NAME", "LAST_NAME" });

            // open the storage
            dbsDepartments.open();

            // cache the storage configuration internally
            put("departments", dbsDepartments);
        }
        return dbsDepartments;
    }
```

Restrict Content of Dropdown List

Of course, JVx offers many more features to enhance our application without manipulating the data model. For example, we assume that the manager of each department must be assigned to the department he or she manages. Therfore, we may want to restrict the employees in the manager listbox to employees which are assigned to the corresponding department.

With our EPlug Plugin, you can jump to the editor's client source code in Eclipse by selecting the editor and pressing the eclipse icon (1).

In JVx, the dropdown list is an ILinkedCellEditor. We restrict the data in the dropdown list by setting an

additional condition to the cell editor:

```
    /**
     * DepartmentsWorkScreen.java (Generated client-class)
     *
     * Initializes the UI.
     *
     * @throws Throwable if the initialization throws an error
     */
    private void initializeUI() throws Throwable
    {
        // code removed

        // definition of the cell editor
        editManagerLastName.setDataRow(rdbDepartments);
        editManagerLastName.setColumnName("MANAGER_LAST_NAME");

        // set additional condition to the cell editor:
        // Only show employees whose "DEPA_ID" equals the "ID" of the
selected department.
        ((ILinkedCellEditor)editManagerLastName.getCurrentCellEditor())
            .setAdditionalCondition(new Equals(rdbDepartments, "ID",
"DEPA_ID"));

        // code removed
    }
```

That's it! This script updates the department's managers: hr_popul_2.sql.

Create an Input Screen With Table and Subtable

Let's dive into another feature of VisionX: we can easily relate data entities, regardless of their cardinality. Let me explain what I mean with the following example: each of the employees may have fix assets assigned, such as a notebook, a mobile phone, or a coffee maker. How can we list the employee's assets in the Employees screen?

Create a One-to-Many Relation

From a technical point of view, the relation between an employee and its assets is a one-to-many relation. Each employee can have multiple assets assigned, but each asset is assigned to one employee (at a time) only. In VisionX, this concept is called “Subtable”. Let's create ASSETS as a subtable of EMPLOYEES.

Open the designer of your Employee screen and edit the data-object “Employees” (1). In the wizard,

toggle “Database changes” (2) and add a new column by pressing the (+) button (3). Name the new column “Assets” (4) and press “Make Subtable” (5).

> **Note**: You have to add a new column and choose “Make Subtable” in one step to create a one-to-many-relation!

We can now specify the details of the new table. Let's rename the column “Assets” to “Asset” (1) and add additional columns “Issue Date” and “Type” (2). VisionX automatically creates a dropdown list for the employee that is currently holding the asset. Press “Finish” to apply your changes.

The model pane (lower panel) of the VisionX designer now contains a new data-object “Assets”:

On data level, VisionX has created a new table ASSETS and a foreign-key relation between the tables ASSETS and EMPLOYEES.

But a clean one-to-many relation in the data model does not fully do the trick. We want each record of EMPLOYEES to be linked to its records in ASSETS, so that whenever we select an employee (only) the assets assigned to that employee are shown. In JVx, this kind of link between table and subtable is called “Master Reference”: the subtable only shows records linked to the currently selected record in the master table. When we created the subtable via VisionX, the “Master Reference” was set automatically. In the data-object wizard of the subtable ASSETS, the table EMPLOYEES is specified as master table:

If you are interested in the underlying Java code, use the EPlug Plugin to jump to the client source code in Eclipse by pressing the eclipse icon for the data-object.

```
    /**
     * EmployeesWorkScreen.java (Generated client-class)
     *
     * Initializes the model.
     *
     * @throws Throwable if the initialization throws an error
     */
    private void initializeModel() throws Throwable
    {
        // code removed

        // link databook to the server storage "assets"
        rdbAssets.setName("assets");
        // specify the data source to use(default)
        rdbAssets.setDataSource(getDataSource());
        // ASSETS is a subtable of EMPLOYEES: available ASSETS rows depend
on the currently selected EMPLOYEES row
```

```
        rdbAssets.setMasterReference(new ReferenceDefinition(new String[] {
"EMPL_ID" }, rdbEmployees, new String[] { "ID" }));
        // open databook
        rdbAssets.open();

        // code removed
    }
```

> **See**: More information on the ReferenceDefinition of a Master Reference is available in the documentation of the JVx API.

After populating the ASSETS table using the script hr_popul_4.sql, your extended web application may now look like this now:

Create a Many-to-Many Relation

What about many-to-many relations? There are plenty of real-life scenarios for that, e.g., we may want to select the asset's type from a predefined list of types. Assets from each type may be assigned to multiple employees, and each employee can have assets of multiple types assigned.

As VisionX is an easy-to-use low-code-platform, it does not bother you with the cardinality nor the required relation. Instead of analyzing the data model, you visually design what you need.

From the employee screen as starting point, we simply want a dropdown list for asset's type instead of a textfield editor. Not a big deal with VisionX.

Let's define the table ASSET_TYPES by opening the wizard for the data-object “Assets”. In order to create a foreign-key relation between the tables ASSETS and ASSET_TYPES, we have to recreate the column “Type” (remember that foreign-key relations are only created for new columns, whereas existing columns are linked via an Automatic Link Reference only, as explained in Create foreign-key relation in an existing data model).

In the wizard, toggle “Database changes” (1), press the (+) Button to create a new column “Type” (2), and, finally, press “Make Combobox” (3).

In the section “ComboBox Details” of the wizard, we name the table “Asset Types” (1), and the columns ID and TYPE (2). Press “Finish” (3) to apply the changes and close the wizard.

That's it! As you can see in the screenshot of the desktop application, the asset's type is now selectable via a dropdown list. You may use the script hr_popul_5.sql to update the test data.

The many-to-many relation between the tables EMPLOYEES and ASSET_TYPES is graphically represented as two linked tables (EMPLOYEES and ASSETS) with a dropdown list for the ASSET_TYPES.

Of course, VisionX updated the data model for us with the new table and the appropriate foreign-key relation:

Specify a Table-Subtable Relation in an Existing Data Model

What about defining a table-subtable relation for an existing data model? Let's update our data model with a new table LOCATIONS and link it to DEPARTMENTS with a foreign-key relation using a PostgreSQL-Script.

The data model looks like this now:

Let's extend the departments screen by adding a data-object “Locations” for the table LOCATIONS and dragging some of its editors. On the left, I have the list of departments. On the right, the department's details and its location.

Looks nice, but when selecting through the departments, the location's details do not change. Why is that? Well, as we created the data model outside VisionX, we have to specify the master table manually with just a few mouse clicks.

Set the Master Reference

To link the DEPARTMENTS and the LOCATIONS records so that the current department's location is automatically selected, we have to specify that DEPARTMENTS is the master table of LOCATIONS (or, in other words, that LOCATIONS is a subtable of DEPARTMENTS).

Let's open the data-object wizard of “Locations”, press “More..” (1), select “Departments” as the “Master Table” (2), and confirm with “Finish” (3).

We are done!

For the sake of completeness: creating a data model that gives a correct reflection of the relationships among the entities is the preferable way to ensure consistent and coherent data. However, under some circumstances you cannot (or may not want to) create a foreign-key relation between table and subtable. In this case, simply create an “Automatic Link Reference” as described in Create drop-down without foreign-key relation and VisionX will offer the referenced table as the master table.

To give an example, I added a column LOCA_ID to the assets table (you may use my script) and created an automatic link reference between ASSETS and LOCATIONS in the assets screen - check it out!

Views and Storages

Until now, to keep it simple, we have been working with database tables. Of course, we are in no way limited to tables as data sources! We can use database views and queries of any complexity. We will investigate some of that more advanced features with the help of a concrete example:

Let's create a screen “Assets” that shows all available assets by their locations. In my mental concept, the location information of my assets is redundant: the asset is assigned to a location (after acquisition) and optionally to an employee (when it becomes issued). But what if the employee changes their department or the employee's department changes its location? Let's assume that the employee (and the department) take their stuff along. That means the employee's location overrides the asset's location. I need a data source that provides me the current location of all assets according to that rule.

Manipulate the Server-Side Storage of Your Data Source

Adapting the data provided by a table is straightforward with VisionX. Remember that for every data object, a server-side storage is created. We can manipulate that storage to contain the required information.

Open the editor for the server-side storage that VisionX created for the data object by pressing the “Edit storages” icon on the right side of the lower pane:

On the left, we have a list of available storages for the screen. We select the storage “assets” (1) and specify the query that shall be used for the storage:

The <query columns> (2) are:

```
asse.id,
asse.asset,
asse.empl_id,
asse.issue_date,
asse.asty_id,
asty.type asty_type,
empl.last_name empl_last_name,
COALESCE(emlo.id, aslo.id) empl_loca_id,
COALESCE(emlo.name, aslo.name) empl_loca_name
```

The <from clause> (3) is:

```
assets asse
INNER JOIN asset_types asty ON asse.asty_id = asty.id
LEFT JOIN employees empl ON asse.empl_id = empl.id
LEFT JOIN departments depa ON empl.depa_id = depa.id
LEFT JOIN locations emlo ON depa.loca_id = emlo.id
LEFT JOIN locations aslo ON asse.loca_id = aslo.id
```

We can verify the generated SQL query by pressing the “Show SQL Query” button (4).

As a foreign-key-relation exists between the table ASSETS and EMPLOYEES, the automatic link reference between the corresponding data objects is created automatically. In the case of ASSETS and LOCATIONS, we have to create the automatic link reference manually (as described in Create Dropdown Without Foreign-Key Relation). The generated code for this server-side looks like this:

```
    /**
     * Gets the assets database storage.
     *
     * @throws Exception if the DBStorage couldn't initialized.
     * @return the assets DBStorage.
     */
    public DBStorage getAssets() throws Exception
    {
        DBStorage dbsAssets = (DBStorage)get("assets");
        if (dbsAssets == null)
        {
            dbsAssets = new DBStorage();
            dbsAssets.setWritebackTable("assets");
            dbsAssets.setDBAccess(getDBAccess());
            dbsAssets.setQueryColumns(new String[] { "asse.id",
                    "asse.asset",
                    "asse.empl_id",
                    "asse.issue_date",
                    "asse.asty_id",
                    "asty.type asty_type",
                    "empl.last_name empl_last_name",
                    "COALESCE(emlo.id, aslo.id) empl_loca_id",
                    "COALESCE(emlo.name, aslo.name) empl_loca_name" });
            dbsAssets.setFromClause(
                    "assets asse "
                    + "INNER JOIN asset_types asty ON asse.asty_id = asty.id
"
                    + "LEFT JOIN employees empl ON asse.empl_id = empl.id "
                    + "LEFT JOIN departments depa ON empl.depa_id = depa.id
"
                    + "LEFT JOIN locations emlo ON depa.loca_id = emlo.id "
                    + "LEFT JOIN locations aslo ON asse.loca_id = aslo.id");
            dbsAssets.createAutomaticLinkReference(new String[] { "EMPL_ID",
"EMPL_LAST_NAME" }, "employees", new String[] { "ID", "LAST_NAME" });
            dbsAssets.createAutomaticLinkReference(new String[] {
"EMPL_LOCA_ID", "EMPL_LOCA_NAME" }, "locations", new String[] { "ID", "NAME"
});
            dbsAssets.open();

            put("assets", dbsAssets);
        }
        return dbsAssets;
```

```
    }
```

That's it. The data object “assets” will now reference the department's location instead of the asset's location.

Use a View as Data Source

If necessary, we could have alternatively created a database view like this:

```
CREATE OR REPLACE VIEW V_ASSETS_LOCALIZED AS
SELECT asse.id,
       asse.asset,
       asse.empl_id,
       asse.issue_date,
       asse.asty_id,
       asty.type asty_type,
       empl.last_name empl_last_name,
       COALESCE(emlo.id, aslo.id) empl_loca_id,
       COALESCE(emlo.name, aslo.name) empl_loca_name
  FROM assets asse
 INNER JOIN asset_types asty
    ON asse.asty_id = asty.id
  LEFT JOIN employees empl
    ON asse.empl_id = empl.id
  LEFT JOIN departments depa
    ON empl.depa_id = depa.id
  LEFT JOIN locations emlo
    ON depa.loca_id = emlo.id
  LEFT JOIN locations aslo
    ON asse.loca_id = aslo.id
;
```

We can use a view in the VisionX data-object wizard just like any table.

In the VisionX lower pane, select the “NEW table” tab and press the (+) button to open the wizard. In the wizard, select “Use existing data from database tables” and confirm with “Next >”, keep the selection “Use Application Database User” and, again, confirm with “Next >”. You can now select the view as data source. Note that views have a different icon:

Let's verify the server-side storage that VisionX created for the data object by pressing the “Edit storages” icon on the right side of the lower pane:

On the left, we have the list of available storages for the screen. Remember that for every data object, a server-side storage is created. We select the storage “vAssetsLocalized”, which is the storage of our currently created view, to review its settings.

Per default, the view is set as “Writeback Table”. This means, that the application will try to save changes on the data object's data in the view itself (which works fine for updateable views or views using INSTEAD OF triggers).

Define the Writeback Table

Of course, we can name another table or view as the target relation for an INSERT, UPDATE, or DELETE. Let's choose the table “assets”. Change the “Writeback Table” to “assets” (1), the “from” clause to “v_assets_localized” (2), and confirm your changes with “Finish”.

Whenever we execute an insert or delete on the data object now, the row will actually be inserted into or deleted from the table ASSETS. Given that the columns of the view and the writeback table are named equally, any UPDATES on that columns will also target the ASSETS table.

This is the generated code for the views server-side storage (after creating an additional Automatic Link Reference between V_ASSETS_LOCALIZED and LOCATIONS):

```
    /**
     * Gets the v_assets_localized database storage.
     *
     * @throws Exception if the DBStorage can't be initialized.
     * @return the vAssetsLocalized DBStorage.
     */
    public DBStorage getVAssetsLocalized() throws Exception
    {
        DBStorage dbsVAssetsLocalized = (DBStorage)get("vAssetsLocalized");
        if (dbsVAssetsLocalized == null)
        {
            dbsVAssetsLocalized = new DBStorage();
            dbsVAssetsLocalized.setWritebackTable("assets");
            dbsVAssetsLocalized.setDBAccess(getDBAccess());
            dbsVAssetsLocalized.setFromClause("v_assets_localized");
            dbsVAssetsLocalized.createAutomaticLinkReference(new String[] {
"EMPL_ID", "EMPL_LAST_NAME" }, "employees", new String[] { "ID", "LAST_NAME"
});
            dbsVAssetsLocalized.createAutomaticLinkReference(new String[] {
"EMPL_LOCA_ID", "EMPL_LOCA_NAME" }, "locations", new String[] { "ID", "NAME"
});
            dbsVAssetsLocalized.open();

            put("vAssetsLocalized", dbsVAssetsLocalized);
        }
        return dbsVAssetsLocalized;
    }
```

We can use any of the upper data storages (“Assets” or “VAssetsLocalized”) for our data object “Assets”. I designed this screen to contain all relevant information about the asset, the employee currently holding the asset, and that employee's department. Therefore, I created data objects for the

tables LOCATIONS and EMPLOYEES and linked them to the “Assets” data object by setting Master References. Moreover, I created a data object for the table DEPARTMENTS and made “EMPLOYEES” the master table for it. The resulting assets screen looks like this:

However, I do not want all that information to be editable in this screen. Let's set some of the data as read-only.

Set Data as Read-Only

Select the assets screen in designer mode and press the “Customize” button (1) at the top left corner. In the lower part of the popup, VisionX will suggest some of the events provided by the currently selected control. We want that the data is set to read-only in this screen immediately after the screen is created. let's. therefore. click on “Create On Load” (2).

The “Edit Action” window opens. The text in the tab page “Description” will be copied to the documentation of the source code. Let's select the tab page “Action” (1) to specify what shall be done whenever the screen is loaded. Press the dropdown list button (2) to see all predefined commands. To avoid scrolling through the list, I filter the list by entering “disable” in the editor (3) and press the drop down list button again. Select “Disable Edit in table” (3).

> **See**: For detailed information on all available VisionX Actions and how to use them, check out the documentation.

Next, we have to specify the data object that shall not be editable; we choose “[Table: Departments]”. To add another action for the same event, drag and drop a command from the right pane to the action table. Let's select “Disable Edit in table” again, this time for “[Table: Employees]”.

I still want to be able to change the employee that is the current holder of the asset. Therefore, I bind the editor “Last Name” to the column “Last Name” of the “Localized Assets” storage (which is editable). In designer mode, select the editor and press the “Customize” icon (1). Open the Binding dropdown list (2) and choose “[Localized Assets.Last Name]” (3) as binding for that editor.

Done! My asset screen is ready to use without writing even one line of Java code:

Filters

To be honest, showing the location table in the departments screen is dispensable. As my use-case is to show all available assets by their locations, I could have done so more easily by creating a filter

editor for the asset's location. Let's give that a try:

Open the “Assets with Filter” screen in designer view and drag and drop the “Search” editor (1) to the screen (2).

Select the editor, press the “Customize” icon (1), and specify the search mode in the lower part of the popup. Let's select “Like” as search mode (2) and “[Localized Assets.Location]” as column (3) to search in.

VisionX automatically creates a dropdown list for the location filter editor due to the automatic link reference between the storage [Localized Assets.Location] and the table LOCATIONS. But the “Like” filter does more than simple text matching. You can use the wildcard character ? as placeholder for any single character and * or % as placeholders for any number of characters.

For example, you may want to filter for assets at locations starting with “South” (such as **South**lake, **South** San Francisco or **South** Brunswick) by entering “South*” (or “South%”) into the location filter editor. Or you enter “*en*” (or “%en%”) to get all assets at locations containing “en” (such as “UK C**en**tral” or “V**en**ice”).

I added another “Like” filter editor for the employee, and this is the result:

Filters Use IConditions

How does this work? Each filter editor creates an ICondition and is connected to other conditions by the logical operator AND. In other words, if multiple filters are set, the result contains only those rows that match all of the filters.

Of course, VisionX supports more than one search mode to use in filters and conditions. These are:

Search mode	Description
Like	Compares literals ignoring the character casing. Supports ?, * and % as wildcards.
Equals	Compares literals case sensitively. No wildcards allowed.
Less	Matches all values that are less the entered value. The comparison algorithm depends on the column's data type.
Less equals	Matches all values that are less or equal the entered value. The comparison algorithm depends on the column's data type.
Greater	Matches all values that are greater the entered value. The comparison algorithm depends on the column's data type.
Greater equals	Matches all values that are greater or equal the entered value. The comparison algorithm depends on the column's data type.
Contains	Matches all literals that contain the entered value ignoring the character casing. Supports ?, * and % as wildcards.

Search mode	Description
Starts with	Matches all literals that start with the entered value ignoring the character casing. Supports ?, * and % as wildcards.

All of the above filters operate on a specific column.

Full Text Search

What about the default search editor we've used in all of our screens so far? If you just drag and drop the “Search” editor to a screen, a filter with the following specification is created:

Search mode	Description
Full text search	Matches all literals that contain the entered value in any column. Supports ?, * and % as wildcards.

The “Full text search filter” operates on all columns of the data object.

Specify a BETWEEN Filter

Let's implement the possibility to filter for assets issued in a specific time span, in other words, between two dates. Simply drag and drop two search editors and bind them to “[Localized Assets.Issue Date]” (1). Use “Greater equals” (2) for the first filter and “Less equals” for the second filter.

That's it. The web application looks nice too:

Download

You can download the entire example application from the VisionX Solution Store. The data model we created during this tutorial is available as data-model.sql.