**Mendix Documentation**

* + - 1. **Create a detailed microflow in Mendix for retrieving cities based on a selected governorate from an external database**

**Step 1: Install External Database Connector**

* **Open your Mendix project** in Mendix Studio Pro.
* Navigate to the **Marketplace** and search for the "External Database Connector".
* Download and install the connector into your project.

**Step 2: Configure External Database Connector**

* Go to the **Settings** of the External Database Connector module.
* **Set up your database connection** by providing the necessary connection string, username, password, and any other details required.

**Step 3: Create Domain Model Entities**

* Open the **Domain Model** editor from the **Data Model** section of the Project Explorer.
* **Create two entities**: **Governorate** and **City**.
  + For **Governorate**, add attributes like **GovernorateID** and **Name**.
  + For **City**, add attributes such as **CityID**, **Name**, and create an association to **Governorate** via **GovernorateID**.

**Step 4: Retrieve Data from External Database**

* Create two microflows: **MF\_RetrieveGovernorates** and **MF\_RetrieveCities**.

**MF\_RetrieveGovernorates**

* Use the External Database Connector to **fetch a list of governorates** from the external database.
* Map the results to the list of **Governorate** entities.

**MF\_RetrieveCities**

* **Create an input parameter** within the microflow for **GovernorateID**.
* Use the connector to fetch cities related to the selected **GovernorateID**.
* Map the results to the list of **City** entities.

**Step 5: Set Up the Governorate ComboBox**

* Drag a **Data view** onto your page and set it to an object of an entity, say **UserSelection**, which will store the selected governorate.
* Inside the Data view, place a **ComboBox** for governorates.
* Set its **Data source** to **MF\_RetrieveGovernorates**.

**Step 6: Bind the Governorate ComboBox Output**

* Configure the governorate ComboBox to **store the selected GovernorateID** in the **UserSelection** entity when a user makes a selection.
* Set the **Attribute (variable)** property to **UserSelection.GovernorateID**.

**Step 7: Set Up the City ComboBox**

* Add another ComboBox for cities inside the same Data view.
* Set its **Data source** to **MF\_RetrieveCities**.
* Ensure it updates when the governorate ComboBox's value changes.

**Step 8: Set the City ComboBox to Use the Selected Governorate**

* Configure the city ComboBox’s **On change** event to execute a microflow (**MF\_RetrieveCities**).
* Pass **UserSelection.GovernorateID** as an input parameter to this microflow.

**Step 9: Testing and Troubleshooting**

* Run the application locally to test the functionality.
* Select a governorate and verify that the city ComboBox is populated with the correct data.

**Example Microflow Diagram**

The microflow MF\_UpdateSelectedGovernorate could be visually represented as follows:

1. **Start Event**
   * Input: **UserSelection**
2. **Change Object**
   * Target: **UserSelection**
   * Operation: Set **SelectedGovernorate** to the value passed from the ComboBox
3. **Refresh in Client** (Optional)
   * Target: Refresh **UserSelection**
4. **End Event**
   * + 1. **Configure Mendix microflow for saving a "societe" object and then passing this object to the next page**

**Step 1: Create 'societe' Object**

* **Create Object Action**: Start your microflow with a **Create Object** action. Configure this action to initialize a new **demande\_societe** object. Set all necessary attributes that will be included in your insertion query to the external database.

**Step 2: Insert into External Database**

* **Query External Database Action**: Use this action to insert the new 'societe' into the external database. Ensure that your SQL query is correctly set up to insert the data and that this query is designed to either return the number of affected rows or the last inserted ID (if supported by your database and configuration).

**Step 3: Retrieve the Last Inserted ID**

* **Query External Database Action**: If your database setup does not support returning the last inserted ID within the same query (or through a RETURNING clause), use a separate query action immediately after the insert to fetch the last inserted ID. You might use a criteria like fetching the highest ID assuming it is auto-incremented.

**Step 4: Fetch the Complete 'societe' Object**

* **Query External Database Action**: With the last inserted ID obtained, execute another query to retrieve the complete 'societe' object from the external database. This action should be set to return a list containing just the newly inserted 'societe' object.

**Step 5: Extract 'societe' Object from the List**

* **Head List Operation**: Since your previous action returns a list, and you expect this list to contain only one item, use the **Head** list operation to safely extract the first item. This ensures you are working with the correct 'societe' object.

**Step 6: Update Mendix Object**

* **Change Object Action**: Apply a **Change Object** action to update the Mendix internal 'demande\_societe' object with the values retrieved from the external database. Here, you'll map each external attribute back to your Mendix object. This is crucial for ensuring that your app's internal state matches the state of the external database.

**Step 7: Pass Object to Next Page**

* **Show Page Action**: Finally, with the updated 'demande\_societe' object ready, use a **Show Page** action to pass this object to the next page or microflow that requires it. Configure this action to ensure the correct page is loaded with the necessary context.

**Additional Considerations:**

* **Error Handling**: Integrate error handling into each step to manage any potential failures in database operations.
* **Validation**: Prior to executing actions, particularly database inserts or updates, validate the data to maintain integrity.
* **Transactions**: If supported, use transactions to wrap your database operations, ensuring that all steps succeed or fail as a unit, which helps in maintaining data consistency.
  + - 1. **Display the duration in years between two dates in your Mendix application using the calendarYearsBetween function**

**Step 1: Define the Entity**

1. **Go to the Domain Model:**
   * Navigate to the Domain Model in your Mendix app project.
2. **Create a Non-Persistent Entity:**
   * Create a non-persistent entity, name it NominationDuration.
   * Add the following attributes:
     + **DateNomination (Date and time type)**
     + **EndDateNomination (Date and time type)**
     + **Duration (Integer type)**

**Step 2: Create the Microflow to Calculate Duration**

1. **Create a New Microflow:**
   * Name the microflow CalculateDuration.
   * Set the parameter type to NominationDuration.
2. **Retrieve Object:**
   * If the object is not already passed as a parameter, retrieve the NominationDuration object.
3. **Add Decision Node:**
   * Add a decision node to check if both DateNomination and EndDateNomination are not empty.
   * **Expression:**

**$NominationDuration/DateNomination ! = empty and $NominationDuration/EndDateNomination ! = empty**

1. **True Path (Both Dates Available) :**

* Add a Change Object activity:
* Change the Duration attribute.
* Use the expression:

**calendarYearsBetween ($NominationDuration/DateNomination, $NominationDuration/EndDateNomination)**

1. **False Path (One or Both Dates are Null):**

* Add another Change Object activity:
  + Set the Duration attribute to 0.
* End the Microflow:
  + Add an end node to complete the microflow.

**Step 3: User Interface Configuration**

1. **Create a New Page**:
   * Add a data view to the page.
   * Set the data source to the NominationDuration entity.
2. **Add Date Pickers**:
   * Place two date picker widgets for DateNomination and EndDateNomination.
   * Configure each date picker to trigger the CalculateDuration microflow on change.
3. **Display Duration**:
   * Add a text box or label to display the Duration.

To ensure the nanoflow does not navigate to the next page if the validation fails, you need to conditionally navigate to the next page only if the validation is successful. Here’s how to update your nanoflow step-by-step:

**Step-by-Step Guide:**

1. **Add a Decision Node in the Nanoflow:**
   * Add a decision node in your nanoflow after the NF\_CalculateDuration\_2 call to check if the duration and dates are valid.
2. **Check the Validation Result:**
   * Modify the microflow NF\_CalculateDuration\_2 to set a Boolean variable indicating whether the validation passed.
   * Return this variable to the nanoflow.
3. **Conditional Navigation :**
   * Only navigate to the next page if the validation is sucdcessful.

**Detailed Steps:**

**Modify the Microflow NF\_CalculateDuration\_2:**

1. **Add a Boolean Variable :**
   * Add a Boolean variable (e.g., isValid) to store the validation result.
   * Set isValid to true initially.
2. **Update Validation Logic :**
   * In your existing validation logic, if any validation fails, set isValid to false before showing the validation message.
3. **Return the Validation Result :**
   * Add an end event to return the isValid variable.

**Update the Nanoflow:**

1. **Call the Modified Microflow:**
   * Call the modified microflow NF\_CalculateDuration\_2.
   * Capture the returned Boolean variable (e.g., isValid).
2. **Add a Decision Node :**
   * Add a decision node after the microflow call.
   * Expression : $isValid
3. **True Path (Validation Passed) :**
   * On the true path, proceed with the existing flow (e.g., navigate to the next page).
4. **False Path (Validation Failed):**
   * On the false path, do not navigate to the next page.

**Example of the Updated Nanoflow:**

1. **Start :**
   * Call NF\_CalculateDuration.
   * Call NF\_CalculateDuration\_2.
2. **Add Decision Node :**
   * Expression : $isValid
3. **True Path :**
   * Continue with the existing logic (e.g., call MF\_ValidationIDUnique and navigate to the next page).
4. **False Path :**
   * End the nanoflow without navigating to the next page.

**Visual Guide :**

**Nanoflow:**

* **Start :**
  + Call NF\_CalculateDuration
  + Call NF\_CalculateDuration\_2
    - Capture the returned Boolean variable (e.g., isValid).
* **Decision :**
  + Expression : $isValid
  + True Path :
    - Continue with the existing logic (e.g., call MF\_ValidationIDUnique and navigate to the next page).
  + False Path :
    - End the nanoflow without navigating to the next page.

**Update the Microflow NF\_CalculateDuration\_2:**

1. **Add a Boolean Variable :**
   * Add a variable isValid of type Boolean and set its initial value to true.
2. **Update Validation Logic :**
   * If the date\_nomination is not before date\_fin\_nomination, set isValid to false before showing the validation message.
   * If the duration is less than 3 years, set isValid to false before showing the validation message.
3. **Return the Validation Result:**
   * At the end of the microflow, return the isValid variable.