Technical Note 102

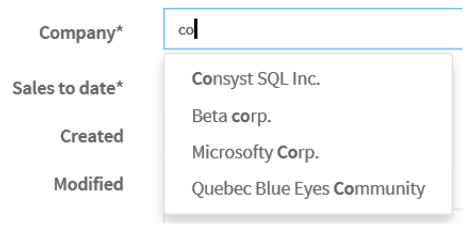
Implementing an auto-complete choice list

Overview

When dealing with choice lists that contain a large number of choice items, is important to find a way to manage them efficiently in order to improve the performance, responsiveness and usability of applications. Auto–complete choice lists make their management easier.

An auto–complete choice list is a list that populates itself with choice items that match a substring: the list adapts as characters are typed in an edit box. The substring may be found at the start, the end or anywhere within the choice item, depending on the SQL command that you provide.

The Rep++ for SPA framework provides an auto-complete feature that you can use on a dynamically built list (i.e. built at execution from an SQL command). It is implemented with the Bootstrap Typeahead plugin, which injects the choice items in the list.



This article describes how to implement the auto-complete feature in your application.

Implementing an auto-complete choice list

An auto–complete choice list is a list that populates itself with choice items that match a substring: the list adapts as characters are typed in an edit box. The substring may be found at the start, the end or anywhere within the choice item, depending on the SQL command that you provide. The auto-complete functionality is included with Rep++ as a specially designed user-defined attribute.

In this article, you will first implement the auto-complete on a field whose choice list data can be built from the same table. For these fields, Rep++ provides two types of auto–complete functions:

- Suggested. In a suggested choice list, the end user can either select an item from the list, or type an entry that does not match any existing choice item.
- Validated. In a validated choice list, the end user can only select an item from the list. An entry that does not match any existing item is cleared when the control loses focus.

In the second implementation, the field's choice list content is extracted from other tables in your database, thereby requiring a join command. The complexity lies in the display and saving of the selected choice list item: the displayed value and the saved value are not the same.

Prerequisites for using the auto-complete functionality for a field

The field on which the auto-complete feature is applied must satisfy the following conditions:

- The type of the field is an Alpha string.
- The control type associated with the field is an edit box.
- The list is built at execution using a SQL command.

Sample system

A sample system designed to demonstrate this capability, Technote102, is included with the Rep++ installation. The information is found in 2 tables, TN102_CLIENT and TN102_COMPANY. The tables contain the following columns:

Company table

Ciecode	Ciename

Client table

Clientcode	ClientFirstname	Clientlastname	Clienttype	**Ciecode**	Clientsalestodate	…

The auto-complete functionality also requires a set of specially designed attributes provided by Rep++, which you will enable.

The functionality will be tested in a single-page application (SPA). You will test the auto-complete on 2 types of fields: a regular field whose content is directly included in the table, and a field whose content is retrieved from another table, thereby requiring a join operation.

In summary, you will:

1. Create the technote's database tables and import the system.
2. Import the user-defined attributes that implement the auto-complete feature.
3. Set up the Clientfirstname field as an auto-complete choice list.
4. Set up the Ciecode field as an auto-complete choice list with a join operation.
5. Create a Rep++ SPA application in Visual Studio® using the Rep++ wizard to test your application.

Prerequisites

Rep++ installed (includes Rep++ studio and SD Tools). Working knowledge of Rep++. SQL Server as database.

Create the Technote102 database tables and import the system

The tables for this system will be added to the repository for the chosen connection.

To create the Technote102 database tables

1. Open SD Tools.
2. Double-click the connection where you want to test the auto-complete feature. A window with your connection contents opens.
3. On the **File** menu, click **Open**.
4. Choose the Demo/Technotes/Technote102_Srv.sql file under the Rep++ installation folder.
5. On the toolbar, click **Execute**. Your tables have been added.

To import the Technote102 system in your connection

1. In your connection window, in the right pane, double-click **Import a system**.
2. Choose the Demo/Technotes/Technote102.sys system under the Rep++ installation folder and click **Open**.

The system has been added to your connection. You can close SD Tools.

Import the user-defined attributes

You may already have imported the FWSPA user-defined attributes in your connection through a previous exercise. If this is the case, skip the import, and include them in the section where you want them, in this case, Technote102.

To import the user-defined attributes

1. Open the Rep++ User-Defined Attributes Editor.
2. On the **Tools** menu, click **User-Defined Attributes Import**.
3. In the *User-Defined Attributes Import* dialog box, select the SPAUDADEF.uda file located under the Rep++ installation folder, in the Rep/base subfolder. Click **OK**.

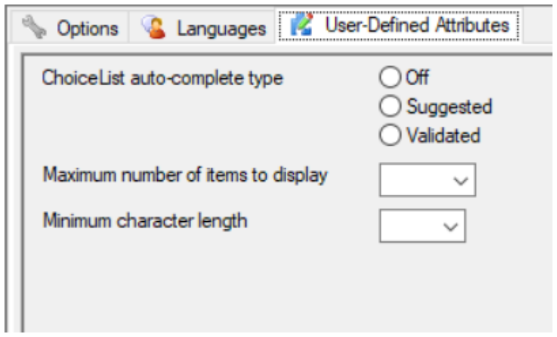
The set of user-defined attributes that implement the auto-complete functionality are included in the FWSPA component, which now appears in the components list. In order to use the auto-complete feature, you must explicitly specify the Rep++ section(s) where the FWSPA user-defined attributes will be available.

To include the user-defined attributes in a section

1. In the *User-Defined Attributes* editor, select the FWSPA component and click the **Sections** tab below the definition section.
2. Click **Add/Remove**.
3. Check the section containing the program where you want to use the auto-complete and click **OK**.
4. Save your modifications.
5. Restart Rep++ studio for the changes to take effect.

The auto-complete attributes

The auto-complete attributes are added in the Rowset editor at the field level. To view the attributes, open the Rowset editor and click the **User-Defined Attributes** tab below the fields list. The attribute values are originally set to null.



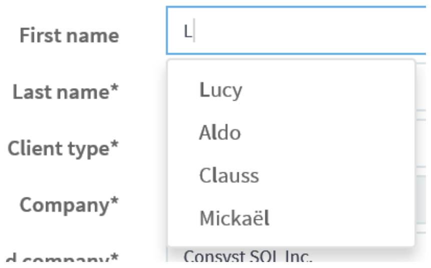
The attributes are described below.

Choice list auto-complete type	Sets the behaviour of the auto-complete feature. **Off**. Disables the auto-complete feature. **Suggested**. Enables the auto-complete feature. The end user is allowed to choose a from the list or type text that does not match any existing choice item. **Validated**. Enables the auto-complete feature. The end user can only save a choice item from the list: Text that does not match any existing item is cleared when the control loses focus.
Maximum number of items to display	Maximum number of items that is proposed to the end user. This limits the number of matching items returned. A value of 0 indicates that all values will be displayed.
Minimum character length	Number of characters that triggers a call to the auto-complete function. Must be 1 or greater: a value of 3 is reasonable with large lists.

Set up the Clientfirstname field as an auto-complete choice list

In this first example, you will implement the auto-complete function on the CLIENTFIRSTNAME field: you will display a list of the existing first names.

You first need to activate the auto-complete function for the CLIENTFIRSTNAME list, define the SQL command to build the list, then modify the settings of the field within the Rowset, and finally test your program.

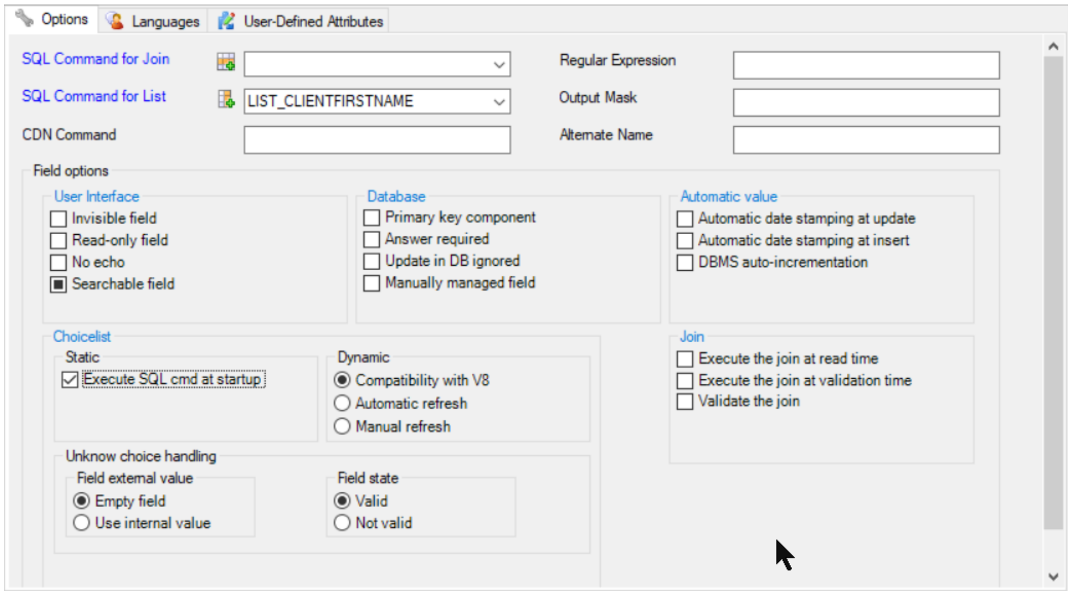


To set up the Clientfirstname choice list

1. In Rep++ studio, open the Technote102 system, then the Technote102 program.
2. Open the CLIENT Rowset.
3. Select the CLIENTFIRSTNAME field from the list.
4. Click the **User-Defined Attributes** tab below the **Fields** list.
5. Set the attributes as follows:
   - Choice list auto-complete type: Suggested or Validated
   - Maximum number of items to display: 15
   - Minimum character length: 1
6. Click the **Options** tab below the **Fields** list.
7. Click the label **SQL Command for List**. The *SQL command editor* window opens.
8. Create a SQL command named *List_Clientfirstname* with the following instructions:

```
SELECT DISTINCT CLIENTFIRSTNAME
FROM CT_CLIENT
WHERE CLIENTFIRSTNAME LIKE
SD_CONCAT(SD_CONCAT('%',:CLIENT.CLIENTFIRSTNAME),'%')
```

9. In the **Options** page, next to **SQL Command for list**, select *List_Clientfirstname*.
10. Under Choicelist group, check **ExecuteSQL cmd at startup**. Make sure the **Compatibility with V8** option is selected. The **Options** page for the CLIENTFIRSTNAME field should look like this.



Options | Languages | User-Defined Attributes

SQL Command for Join:
SQL Command for List: LIST_CLIENTFIRSTNAME
CDN Command:
Regular Expression:
Output Mask:
Alternate Name:

Field options

User Interface
- [ ] Invisible field
- [ ] Read-only field
- [ ] No echo
- [x] Searchable field

Database
- [ ] Primary key component
- [ ] Answer required
- [ ] Update in DB ignored
- [ ] Manually managed field

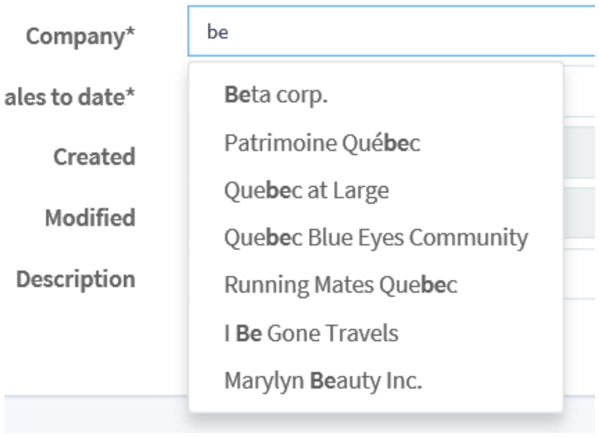
Automatic value
- [ ] Automatic date stamping at update
- [ ] Automatic date stamping at insert
- [ ] DBMS auto-incrementation

Choicelist
- Static: [x] Execute SQL cmd at startup
- Dynamic: [✓ Compatibility with V8] Automatic refresh Manual refresh

Unknow choice handling
- Field external value: [✓ Empty field] Use internal value
- Field state: [✓ Valid] Not valid

Join
- [ ] Execute the join at read time
- [ ] Execute the join at validation time
- [ ] Validate the join

11. Create a Rep++ SPA application using the Rep++ wizard and test your application (see **Create a Rep++ SPA application to test your functionality** at the end of the document).



Company*: be

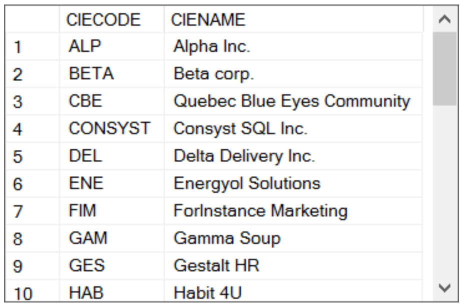
Beta corp.
Patrimoine Québec
Quebec at Large
Quebec Blue Eyes Community
Running Mates Quebec
I Be Gone Travels
Marylyn Beauty Inc.

Notice the difference when you change the type of the auto-complete from **Suggested** to **Validated**.

Set up the Ciecode field as an auto-complete choice list with a join

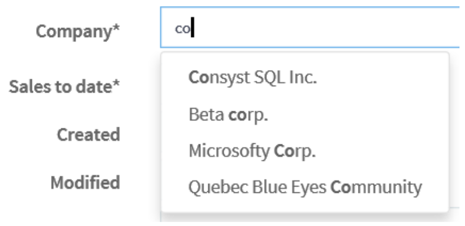
There are situations when the auto-complete field displays choice items located in another table. In this case the join field is a key containing an internal value, typically not user-friendly, to which is associated a more readable value.

In the excerpt below, the CT_COMPANY table contains the CIECODE and CIENAME columns:



	CIECODE	CIENAME
1	ALP	Alpha Inc.
2	BETA	Beta corp.
3	CBE	Quebec Blue Eyes Community
4	CONSYST	Consyst SQL Inc.
5	DEL	Delta Delivery Inc.
6	ENE	Energyol Solutions
7	FIM	ForInstance Marketing
8	GAM	Gamma Soup
9	GES	Gestalt HR
10	HAB	Habit 4U

The company information is included through the CIECODE field in the Rowset. You want however your end users to choose from CIENAME, as in:



Company*	co
Sales to date*	
Created	
Modified	

Consyst SQL Inc.
Beta corp.
Microsofty Corp.
Quebec Blue Eyes Community

For the read operation, you need a join operation that will display the company names instead of the company codes.

Conversely, for the save operation, you also need to use a join operation that will retrieve the company code from the selected company name. The overall procedure consists of:

- Defining a field that will contain the internal value of the selected choice item, namely the value that is saved in the database. This field may be hidden.
- Defining a work field (not linked to the database) that will contain the external (display) value of the selected choice item, whose auto-complete type attribute must be set to **Validated**. This field is only created for display purposes, as its associated hidden field will contain the saved value.
- Defining the SQL command for building the choice list and associating it with the work (display) field.
- Setting up a join operation that will retrieve the internal value associated with the selected display value (for saving).
- Customizing the SELECT SQL command for the Rowset to read the initial values when the application is started.

The following example will demonstrate how to set up the auto-complete for the CIECODE field in the Technote102 program: the end user will be presented with the company name, but the saved field value is the company code.

To set up the Ciecode field as an auto-complete choice list with a join operation

1. In Rep++ studio, open the Technote102 system, then the Technote102 program.
2. Click the **Fields** node.
3. Select the CIECODE field from the list, which will hold the internal value of the choice list. Make sure the following attributes are set:
    - Column: CIECODE
    - Type: Alpha string
    - Control Type: Edit box (required, but not used)

- Max size: 16

4. Select the CIENAME field that will hold the value to be displayed. Make sure the following attributes are set:
   - Column: CIENAME
   - Type: Alpha string
   - Control Type: Edit box
   - Max size: 80
5. Click the **Rowsets** node and select the CLIENT Rowset.
6. Select the CIECODE field, then check all the following options:
   - Invisible field (optional)
   - Read-only field
   - Answer required
7. Click **Add** on the right of the list, then add the CIENAME field from the TN102_COMPANY table.
8. Select the CIENAME field, then check all the following options:
   - Answer required
   - Update in DB ignored
   - Manually managed field
   - Execute a join at validation time
9. Click the **User-Defined Attributes** tab below the **Fields** list and set the attributes as follows:
   - ChoiceList auto-complete type: Validated
   - Maximum number of items to display: 10
   - Minimum character length: 1
10. Click the **Options** tab below the **Fields** list.
11. Click the label **SQL Command for List** to open the SQL command editor.
12. Create the LIST_CIE command. The SQL instructions appear below.

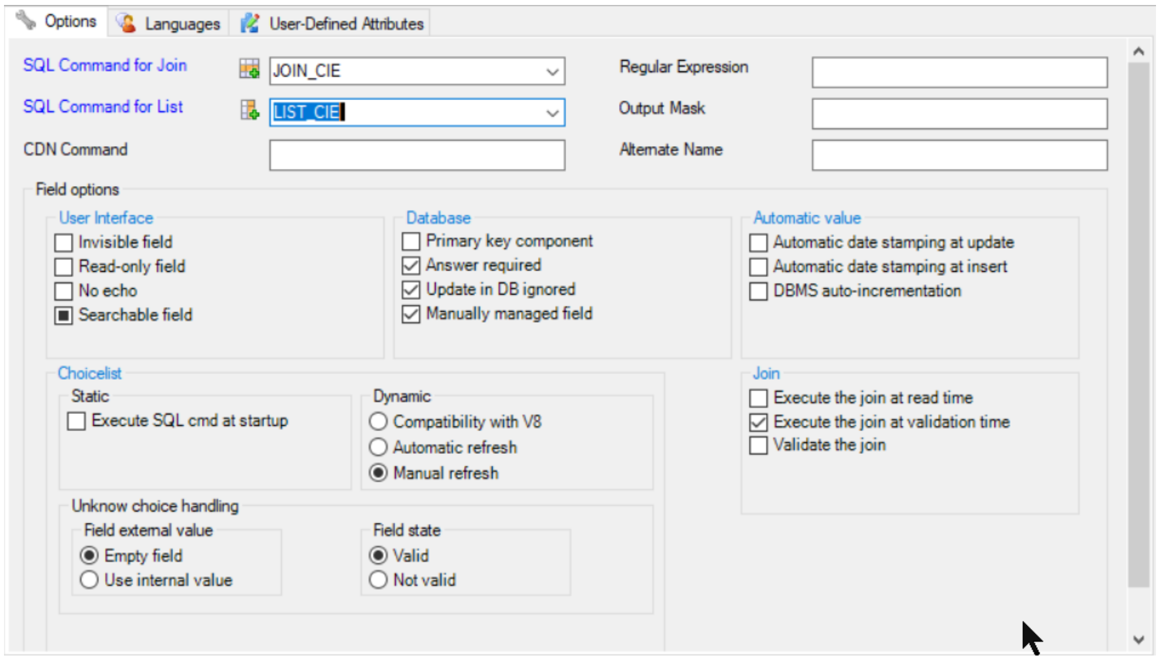
```
SELECT CIECODE, CIECODE, CIENAME
FROM TN102_COMPANY
WHERE CIENAME LIKE SD_CONCAT(SD_CONCAT('%',:CLIENT.CIENAME),'%')
ORDER BY CIENAME
```

13. Create the JOIN_CIE command to retrieve the CIECODE value from the selected CIECODE_DISPLAY value.

```
SELECT CIECODE
INTO :CLIENT.CIECODE
FROM TN102_COMPANY
WHERE TN102_COMPANY.CIENAME = :CLIENT.CIENAME
```

14. Back in the Rowset editor, select JOIN_CIE from the list **SQL Command for join**, and LIST_CIE from the list **SQL Command for List**.

    The **Options** page for the CIECODE_DISPLAY field should look like this.



SQL Command for Join: JOIN_CIE
SQL Command for List: LIST_CIE
CDN Command:
Regular Expression:
Output Mask:
Alternate Name:

Field options

User Interface
- [ ] Invisible field
- [ ] Read-only field
- [ ] No echo
- [■] Searchable field

Database
- [ ] Primary key component
- [x] Answer required
- [x] Update in DB ignored
- [x] Manually managed field

Automatic value
- [ ] Automatic date stamping at update
- [ ] Automatic date stamping at insert
- [ ] DBMS auto-incrementation

Choicelist
Static
- [ ] Execute SQL cmd at startup

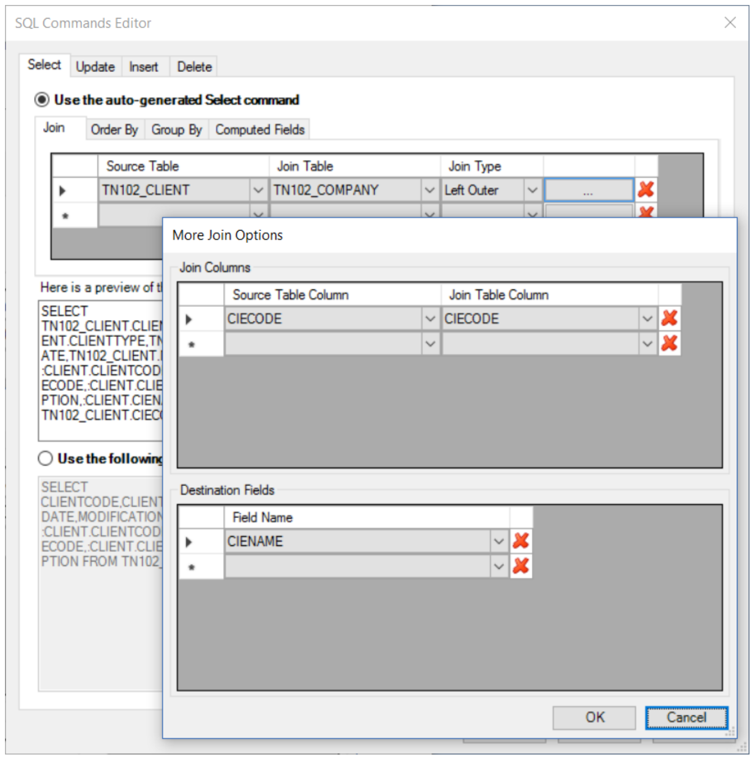
Dynamic
- ○ Compatibility with V8
- ○ Automatic refresh
- ● Manual refresh

Unknow choice handling
Field external value
- ● Empty field
- ○ Use internal value

Field state
- ● Valid
- ○ Not valid

Join
- [ ] Execute the join at read time
- [x] Execute the join at validation time
- [ ] Validate the join

15. **Optional.** Click the **Languages** tab and type ***Company*** as the prompt in the English language.
16. Click the **Info** tab at the top of the Rowset editor.
17. Click **Modify the SQL Commands for this Rowset**.
18. In the *SQL Commands* editor, set up the join operation on the CIECODE field as follows to read the original values when the application starts:
    - Source Table: TN102_CLIENT
    - Join Table: TN102_COMPANY
    - Join Type: Left Outer
    - (Click the ... button)
    - Source Table Column: CIECODE
    - Join Table Column: CIECODE
    - Destination Field Name: CIENAME


SQL Commands Editor

Select | Update | Insert | Delete

(●) Use the auto-generated Select command

Join | Order By | Group By | Computed Fields

	Source Table	Join Table	Join Type
▶	TN102_CLIENT	TN102_COMPANY	Left Outer
*			

Here is a preview of t

```
SELECT
TN102_CLIENT.CLIE
ENT.CLIENTTYPE,TN
ATE,TN102_CLIENT.
:CLIENT.CLIENTCOD
ECODE,:CLIENT.CLIE
PTION,:CLIENT.CIEN
TN102_CLIENT.CIEC
```

( ) Use the following

```
SELECT
CLIENTCODE,CLIENT
DATE,MODIFICATION
:CLIENT.CLIENTCOD
ECODE,:CLIENT.CLIE
PTION FROM TN102_
```

More Join Options

Join Columns

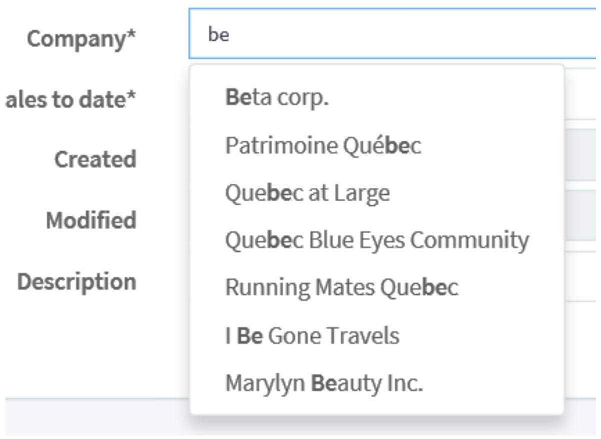
	Source Table Column	Join Table Column
▶	CIECODE	CIECODE
*		

Destination Fields

	Field Name
▶	CIENAME
*	

OK | Cancel

19. Save your Rowset.
20. Create a Rep++ SPA application using the Rep++ wizard and try your application.


Company* be

ales to date*

Created

Modified

Description

Beta corp.
Patrimoine Québec
Quebec at Large
Quebec Blue Eyes Community
Running Mates Quebec
I Be Gone Travels
Marylyn Beauty Inc.

Create a Rep++ SPA application to test your functionality

To test your new functionality, create a Rep++ SPA application using the Rep++ wizards.

To create a Rep++ SPA application using the Rep++ wizard

1. In Visual Studio®, create a standard ASP.NET Web Application (not Rep++).
2. In the *New ASP.NET Project* window, select the **Rep++ MVC V8** template.

3. In the Rep++ wizard's *Connect to the Rep++ repository* page, select the connection, system and program for your technote.
4. In the Select *Rep++ repository components* page, select the *ClientTransaction* and *ClientSelectTransaction* RowsetTrees for the transaction and selection buffer, respectively.
5. In the *Select n-tier options* page, leave the default values.
6. In the *Specify generation information for typed instances* page, leave the default values.
7. In the *Specify generation information for POCO entities* page, clear **Generate POCO entities**.
8. In the *Select template* page, select *MVC SPA using Angle Template*.
9. In the *Select layout options* leave the default values.
10. In the *Select replacement mode* page, click **Finish** to create the application.
11. In the *Search for TypeScript Typings* message box, click **No**.