Appian
Academy

Appian Step-by-Step #6

Exercise to Accompany
Design Appian Records Part 2: Record Relationships

The Appian Step-by-Step series consists of 12 exercises that accompany the courses in the Appian Developer learning path. Exercises build upon each other. Complete exercises in order and keep the app and all objects until you are done with the project.

1 Welcome to the Appian Developer Learning Path
2 Create an Application
3 Manage Users and Groups
4 Expressions
5 Records Part 1: Accessing Your Data
6 **Records Part 2: Record Type Relationships**
7 Query Entities
8 Process Modeling 101: Part 1
9 Process Modeling 101: Part 2
10 Reports
11 Sites
12 Task Report

Exercise 6: Records Part 2, Record Type Relationships 3

Create Record Relationships 3

Use Record Relationships in Custom Record Fields 4

Update Summary View Interface 5

Update Category, Status, and Condition Fields 6

Add an Expression-Based User Filter 7

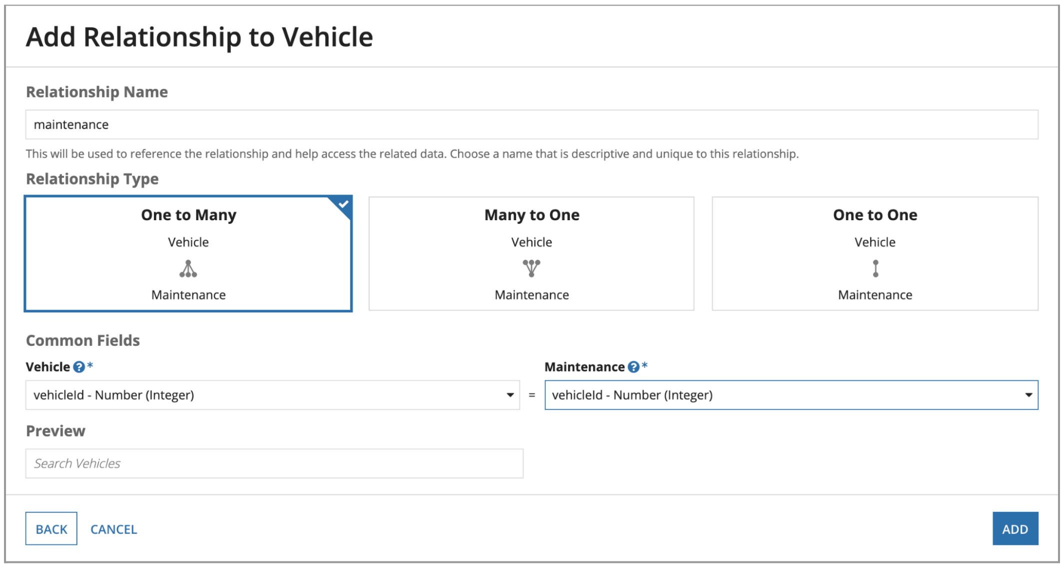
TIP: Use Local Variables to Define and Store Temporary Values 8

Exercise 6: Records Part 2, Record Type Relationships

Create Record Relationships

In this exercise, you will add relationships to the AX Vehicle record. Record relationships allow easy access to data from related records. Once you build relationships into a record, you will be able to access the fields from related records without building queries. For this exercise, you will add four relationships to the AX Vehicle record: Maintenance, Category, Status, and Condition. You will reuse the existing Maintenance, Category, Status, and Condition records from the Acme Automobile Reference Application. Follow the steps below to add these record relationships.

1. In the **AX Vehicle** record, select the **Data Model** tab, and click **Add Relationship**.
2. Type **Maintenance** into the **Related Record Type** field, then click **Next**.
3. Set the **Relationship Name** as **maintenance**, and select **One to Many** as the **Relationship Type**. We use a One to Many type in this instance because one vehicle record can be related to multiple maintenance records.
4. Select **vehicleId** for both the **AX Vehicle** and **Maintenance** fields:



Add Relationship to Vehicle

Relationship Name

maintenance

This will be used to reference the relationship and help access the related data. Choose a name that is descriptive and unique to this relationship.

Relationship Type

[✓ One to Many — Vehicle — Maintenance] Many to One — Vehicle — Maintenance | One to One — Vehicle — Maintenance

Common Fields

Vehicle *: vehicleId - Number (Integer) = Maintenance *: vehicleId - Number (Integer)

Preview

Search Vehicles

5. Click **Add**, and **Save Changes**.
6. Next, you will add a second relationship to the **AX Vehicle** record.
   - Click **Add Relationship**.
   - Type **Category** into the **Related Record Type** field.
   - Set the **Relationship Name** as **category**.

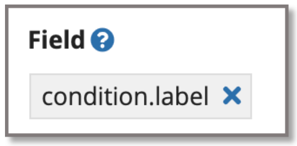
- Select **Many to One** as the **Relationship Type**.
- Select **vehicleCategory** for the **AX Vehicle** field and **id** for the **Category** field.
- Click **Add**, then **Save Changes**.

7. In the same manner, add two more relationships to the **AX Vehicle** record: **Status** and **Condition**:

   - Select **Many to One** for both relationship types.
   - Select **vehicleStatus** for the **Status** relationship and **vehicleCondition** for the **Condition** relationship.

8. Appian automatically creates user filters for Many to One record relationships. Click the **User Filters** tab to view the three newly created user filters. Make the following adjustments to the Condition, Status, and Category user filters:

   - Click the **Expression Editor** icon next to the filter name.
   - Scroll down to the **Field** field.
   - Delete the existing field, and replace it with **label**. For example, for the Condition filter, select **condition.label**:


Field
condition.label

   - Repeat these steps for all three user filters (Condition, Status, and Category).
   - Click **Save Changes**.

9. Click the link next to the AX Vehicle record name to view and test these filters in the business user environment.

Use Record Relationships in Custom Record Fields

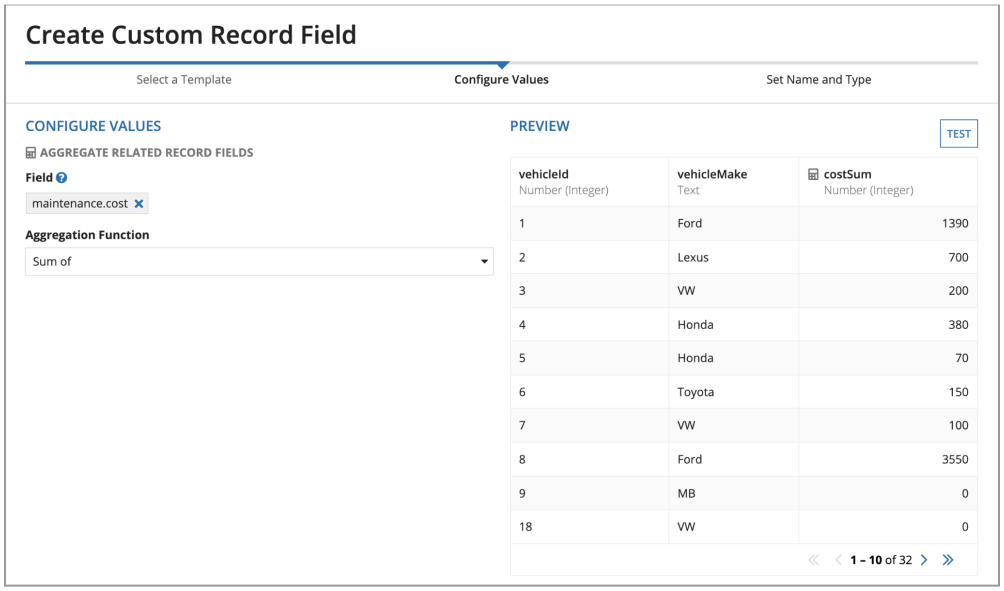
Record relationships can be very useful when you want to aggregate and display data from a related record. In this exercise, you will learn how to aggregate maintenance-related data using custom record fields. You will first create a custom record field for displaying the total cost of all maintenance per vehicle. Then, you will create a custom record field for showing the total count of all maintenance events per vehicle. Follow the steps below to create the two custom record fields.

1. In the **AX Vehicle** record under the **Data Model** tab, click **New Custom Record Field**.

2. Select **Aggregate Related Record Fields**. Click **Next**.

3. Select **maintenance.cost** in the **Field** dropdown menu.

4. Choose **Sum of** in the **Aggregation Function** dropdown menu:



Create Custom Record Field

Select a Template | Configure Values | Set Name and Type

CONFIGURE VALUES

AGGREGATE RELATED RECORD FIELDS

Field: maintenance.cost

Aggregation Function: Sum of

PREVIEW

TEST

vehicleId Number (Integer)	vehicleMake Text	costSum Number (Integer)
1	Ford	1390
2	Lexus	700
3	VW	200
4	Honda	380
5	Honda	70
6	Toyota	150
7	VW	100
8	Ford	3550
9	MB	0
18	VW	0

1 – 10 of 32

5. Click **Test** to preview the new field, and click **Next**.

6. Leave the name as **costSum**, and click **Create**.

7. In the same manner, create a custom record field that displays a count of all maintenance requests for a vehicle.

    - Select the **maintenance.requestId** field.
    - Select the **Count of** aggregation function.
    - Rename the custom record field to **maintenanceRequestCount**.

8. Click **Save Changes**.

Update Summary View Interface

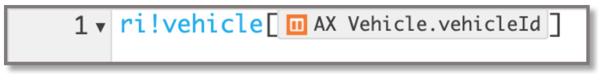
In this exercise, you will add a read-only grid with the vehicle's maintenance data to the AX_VehicleSummary interface. Also, you will update the Category, Status, and Condition fields, so that they display labels instead of numeric IDs. This can be done using the record relationships that you built in the previous exercise. Follow the steps below to complete this exercise.

1. Open **AX_VehicleSummary**, and place a **Read-Only Grid** component under the **Card Layout** with the vehicle details.

2. Select the **Read-Only Grid**, and make the following changes in the **Component Configuration** pane:

- Select **Record Type** as the **Data Source**, and search for the **Maintenance** record.
- Click **Filter Records**. In order to display the maintenance requests for the selected vehicle, you want to filter the record based on **vehicleId**.
- Click **Add Filter**. Select the **vehicleId** Field and the **=** Condition.
- Click the arrow **dropdown** next to **123**, and select **Expression**.
- Click the **null link** to edit the expression. In the **Expression Editor**, enter the following expression:

```
ri!vehicle[recordType!AX Vehicle.fields.vehicleId]
```



```
ri!vehicle[AX Vehicle.vehicleId]
```

- Click **OK**. You will see that the read-only grid now only displays the maintenance requests that are associated with the record's vehicle ID.

3. Scroll down to the bottom of the **Component Configuration** pane, and click **Layout**. Change the **Label Name** to **Maintenance Requests**.

4. Click **Save Changes**.

Update Category, Status, and Condition Fields

Now that you have record relationships, let's update the Category, Status, and Condition fields in the AX_VehicleSummary interface. Currently, these fields display numeric IDs. You will replace them with more user-friendly labels. Complete the steps below.

1. Click the **Category** text box.

2. Click the **Display Value** link, and clear the expression.

3. Type the following expression:

```
ri!vehicle[recordType!AX
Vehicle.relationships.category.fields.label]
```

4. Click **OK**. The field may appear blank, but it will display the correct information once you view the interface in summary view.

5. In the same manner, update the **Status** and **Condition** fields. When selecting the relationship, select the **Status** record for the Status text box and the **Condition** record for the Condition text box.

6. Click **Save Changes**. Return to the AX Vehicle record, and click the link next to the AX Vehicle record name. Click on a **VIN** to view the updated summary view interface.

Add an Expression-Based User Filter

In Exercise 5, you created a simple user filter to use in your AX Vehicle record. In this exercise, you will create an expression-based user filter to filter the AX Vehicle record by vehicle make. Follow the steps below to create an expression-based user filter.

1. In the **AX Vehicle** record, click **User Filters**.
2. Click **New User Filter**, and select the **Expression** radio button.
3. Name the filter **Make**, and type the following expression into the **Expression Editor.** As a reminder, this expression should be typed. If you copy and paste this expression into the editor, you will need to edit the lines with recordType.

```
a!localVariables(
local!vehicleMakes: a!queryRecordType(
  recordType: recordType!AX Vehicle,
  fields: a!aggregationFields(
    groupings: {
      a!grouping(
        Field: recordType!AX Vehicle.vehicleMake,
        alias: "make"
      )
    },
    measures: {
      a!measure(
        field: recordType!AX Vehicle.vehicleMake,
        function: "COUNT",
        alias: "count"
      )
    }
  ),
  pagingInfo: a!pagingInfo(
    startIndex: 1,

          batchSize: 5000
        )
      ),
      a!recordFilterList(
        name: "Make",
        options: a!forEach(
          items: local!vehicleMakes.data,
          expression: a!recordFilterListOption(
            id: fv!index,
            name: fv!item.make,
            filter: a!queryFilter(
              field: recordType!AX Vehicle.vehicleMake,
              operator: "=",
              value: fv!item.make
            ),
            dataCount: fv!item.count
          )
        )
      )
    )
```

This expression first creates a local variable for the record query, so that the record query can be easily referenced in the second half of the expression. In this case, a!queryRecordType() executes a query on the AX Vehicle record type and returns the vehicle makes as the result. Instead of manually creating an option for each unique make, the a!forEach() function writes these options for you based on vehicle makes returned in the record query. The a!forEach() function will take the array of vehicle makes and pass them to an expression one at a time, creating the a!recordFilterListOption function for each item in the array.

4. Click **OK**, and then **Save Changes**. Click the link located next to the record name to access the AX Vehicle record. You will see the Make filter available at the top of your record list. Test that it works as expected.

TIP: Use Local Variables to Define and Store Temporary Values

---

- A local variable is a type of variable that is used to define and store temporary values within an expression. They are useful when you only need values within a particular expression. For example, if you need to temporarily store user-inputted search terms in an interface, this can be easily accomplished by adding a local variable local!search to the interface.
- A local variable will temporarily hold a value, until you save it. For example, in an interface you can store a local variable using a related rule input.
- Use the following syntax to define a local variable: a!localVariables(localVar1, localVarN, expression). LocalVarN means that you can have multiple local variables.
- Local variables are frequently used in interfaces. Although you have to add them in Expression Mode, the advantage is that you can declare as many local variables as you need.
