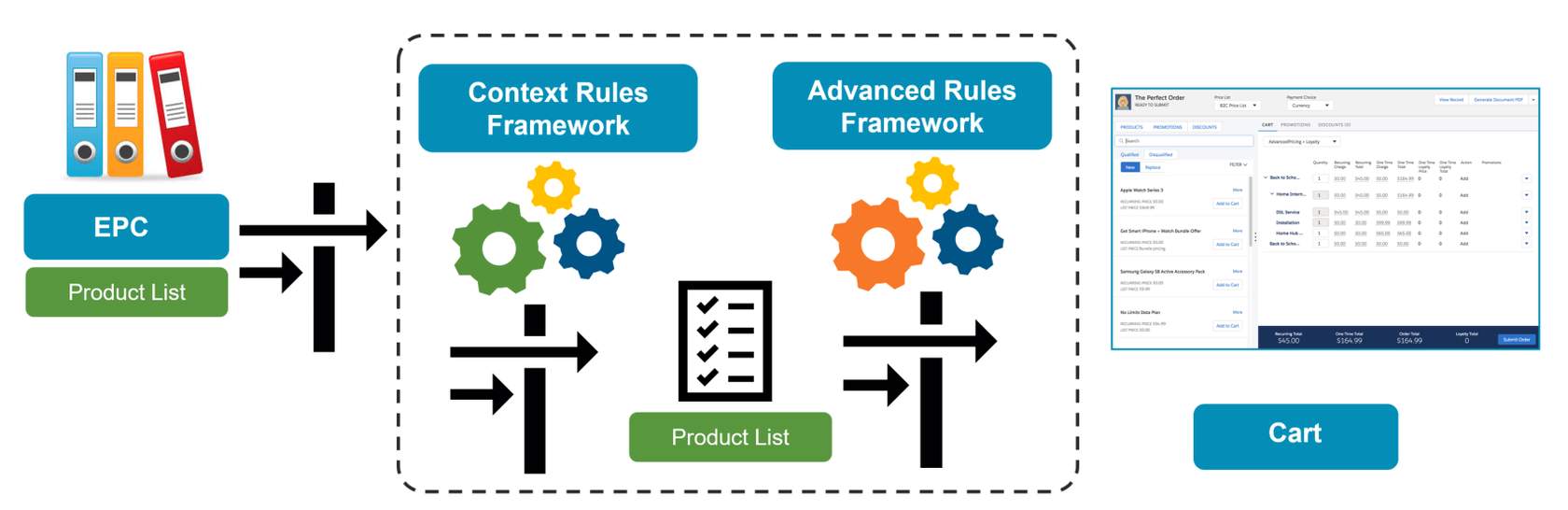
**Dual Frameworks, Working Together (Recap)**

The Context Rules and Advanced Rules frameworks work in tandem to run business rules in your Industries CPQ environment. These business rules tailor the Shared Catalog (EPC) data into product and service offers that are relevant and available to customers.

**Context Rules** qualify products, promotions, price lists, price list entries and pricing adjustments in the Cart. This framework can also be used for Salesforce Industries API Caching, which is used for digital commerce.

**Advanced Rules** is Salesforce Industries' original rules framework, and it is used primarily to create rules for product compatibility or configuration. It's important to understand each framework's strengths in order to decide which type of rule to use. You can learn more about when to use each type of rule at the end of this course.

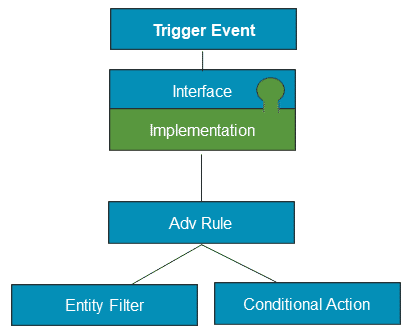
Both rules’ frameworks operate in tandem to provide comprehensive rules to govern all Industries CPQ interactions.



The diagram shows how the two rules’ frameworks work together in the Cart. The process begins by gathering all active products in EPC, moves through the Context Rules Framework to filter the product list, then to the Advanced Rules Framework to further refine the product list, and finally presents available and eligible products and promotions in the Cart.

**Advanced Rules Framework and Rule Types**

**The Advanced Rules Framework**



The Advanced Rules Framework includes the following elements:

* A trigger event such as an event change in the Cart or selecting a price list
* An interface and implementation
* An advanced rule based on an entity filter and a conditional action

**Summary of Advanced Rule Types**

You can create the following types of advanced rules:

* **Availability:**Filter the Products list to show only the products available for a customer.
* **Eligibility:**Filter the Products list to show only the items for which the customer is eligible.
* **Compatibility (AKA Configuration/Validation):**Define the related products to add when a product is added to the Cart. You can also automatically add, remove, or recommend products based on other products in the Cart.
* **Pricing:**Change standard pricing, and operate on order line items in the Cart. The PricingRulesImplementation allows you to create advanced rules to implement pricing rules that call calculation procedures as actions.

**Availability Rules**

Availability rules filter the Products list to show only the items available to the customer.

Examples:

* The gold iPhone is not available in New York.
* Office Internet Solution is not available in Alaska or Hawaii.
* Office Internet Solution is available only in California.
* Products are not available based on the postal code returned by the Canadian post portal.
* Products are not available if they have already been ordered.

**Eligibility Rules**

Eligibility rules filter the Products list to show only the items for which the customer is eligible.

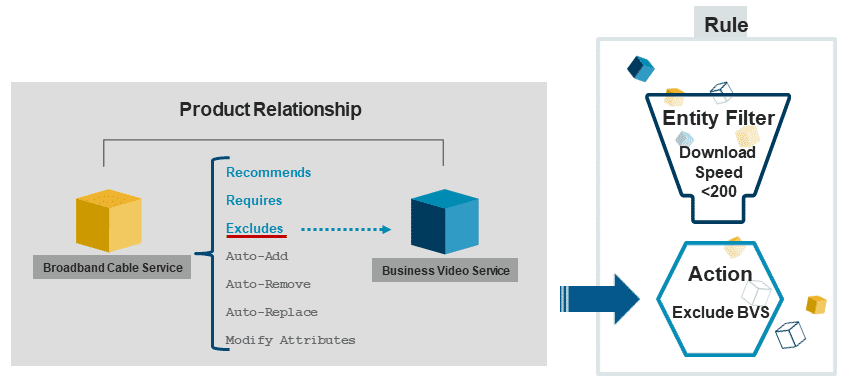
Examples:

* Free installation offer is available to customers with accounts newer than 12 months.
* Free installation offer is shown only to customers who have not used it in the last year.
* Do not ship orders when the number of contract months is less than or equal to 10.
* Billing Accounts are not eligible to purchase Office Internet Solution.
* Bronze and silver Accounts are not eligible to purchase Office Internet Solution.
* New customers are not eligible for specific products.

**Compatibility Rules**

Compatibility rules (also known as configuration/validation rules) do the following:

* Define the relationship between products to ensure that a valid combination of products is added to the Cart. Using product relationships, you can define that when a product is added to the Cart, related products are added with it. You can also automatically add, remove, or recommend products based on other products in the Cart.
* Ensure products ordered in the Cart are compatible based on conditions in order line items and related objects.
* Run on Order Line Item objects—Opportunity Line Item, Quote Line Item, or Order Line Item. The rule actions involve product relationships.
* A compatibility (configuration) advanced rule is required when the rule depends on:
  + **Context:** For example, if you want to recommend a particular product only to consumer accounts when another product is ordered.
  + **Multiple Criteria:**For example, if you want recommend a particular product only when two other products are present in the Cart.
  + **Product Attributes:** For example, if the Business Video Service should not be available in low bandwidth situations, i.e., when Broadband Cable Service has a download speed attribute less than 200 Mbps. (See diagram)



**Pricing Rules**

Pricing rules change standard pricing, and operate on order line items in the Cart.  
Pricing advanced rules require the PricingRulesImplementation, which uses price books rather than price lists. These rules are used in older deployments. Newer deployments use attribute-based pricing using Pricing Plans and context rules for price list entries to provide the same functionality.

# Interface Implementations for Advanced Rules

The core of the Salesforce Industries open architecture is the interface and implementation paradigm. An interface manages the application and its call out to business logic; an implementation is that business logic. Salesforce Industries offers multiple implementations for each interface. You can choose the implementation that best fits your needs, or you can create your own implementation. When used judiciously, you can define interfaces once and deploy them many times to consistently maintain and apply business logic.

Salesforce Industries provides a variety of interface implementations for Advanced Rules with default implementations. The default implementations allow the Cart to run, but they don’t invoke any rules. You must activate different implementations to apply rules.

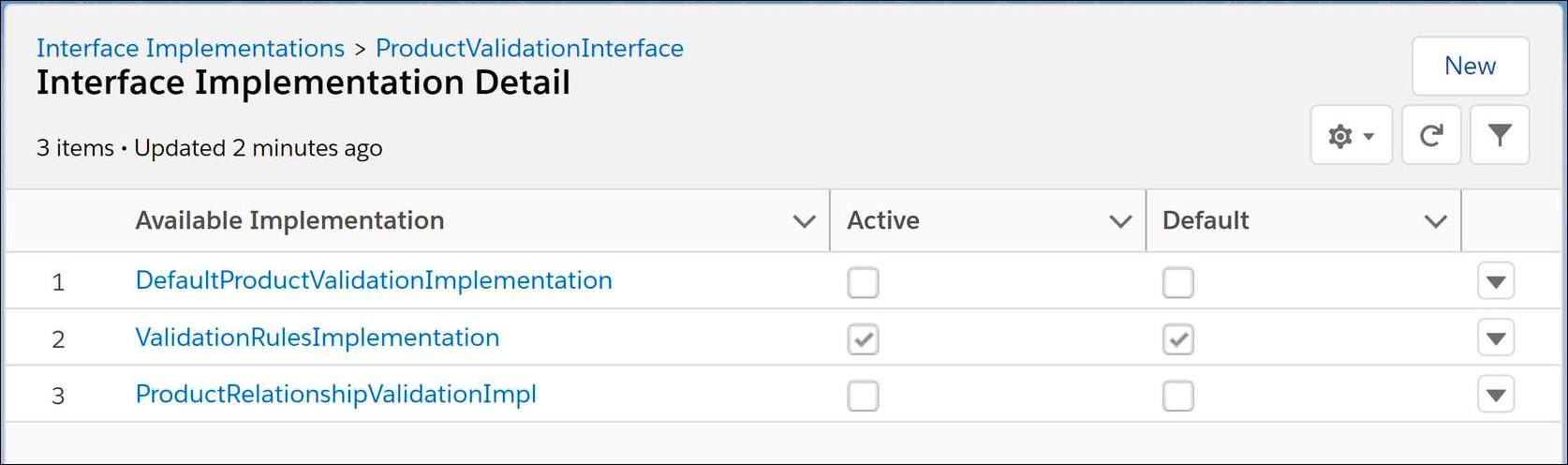
The implementations available to you out of the box will vary depending on whether you have a managed package or an unmanaged package.

**Product Validation / Compatibility Rules**

The **ProductValidationInterface** implements compatibility rules, also known as configuration rules or validation rules. Compatibility rules specify the relationships between products. For example, one product could require, recommend, or exclude another product. The ProductValidationInterface validates the compatibility of the products in an opportunity, order, or quote.

The ProductValidationInterface is typically triggered when a product is added to, modified in, or deleted from an opportunity, order, or quote.

The ProductValidationInterface includes the implementations outlined below, or you can choose to create your own custom/hybrid implementation.



* **ProductRelationshipValidationImpl (deprecated)**
  + A legacy implementation that is no longer supported.
* **ValidationRulesImplementation**
  + Runs all product relationships and active compatibility rules for the opportunity, order, or quote.
* **DefaultProductValidationImplementation**
  + This implementation returns whatever input it receives as output, without any changes; i.e. it does not implement your validation/compatibility rules.

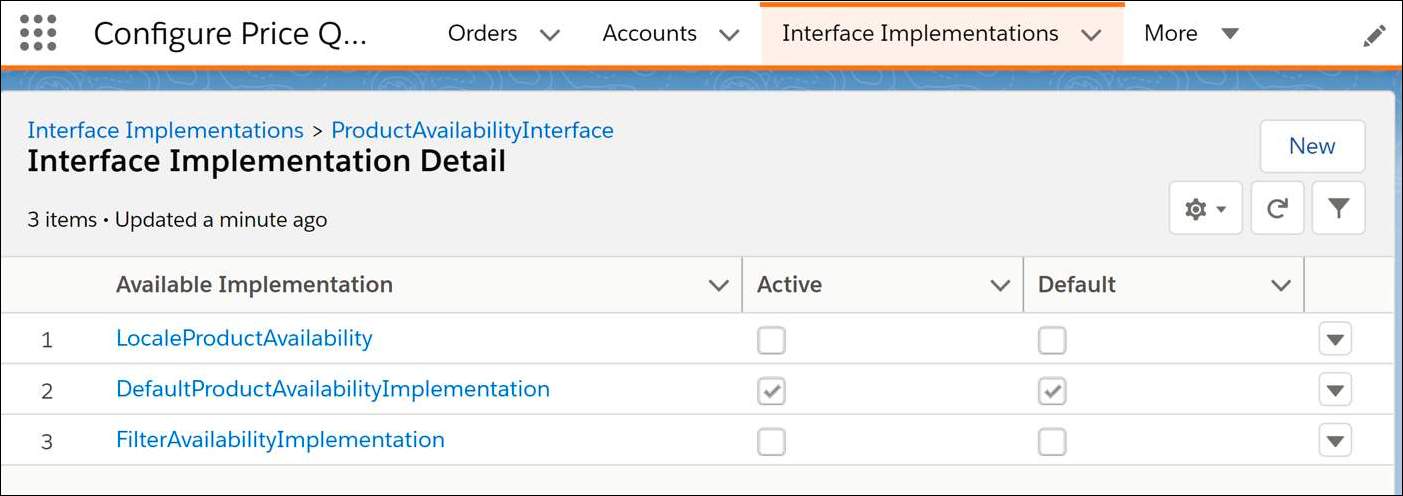
**Availability Rules**

Use availability implementations to specify which products are available for a specific shipping location, account, or account attributes.

Availability rules use the **ProductAvailabilityInterface** and associated implementations.

The ProductAvailabilityInterface is called when a request is made to provide a list of products that are available to the customer to select.

You can create your own custom or hybrid implementation, or you can use the implementations supplied out of the box.



**Eligibility Rules**

Use eligibility implementations to specify products that may be available in certain locations but the customer isn't eligible for other reasons.

Eligibility rules use the **ProductEligibilityInterface** and associated implementations.

The ProductEligibilityInterface is called immediately after the ProductAvailability interface and implementation.

You can create your own custom or hybrid implementation, or you can use the implementations supplied out of the box. Click on the cards below to find out more about some of the key eligibility implementations.

* AccountTypeProductEligibilityImplementation
  + Runs standard eligibility rules.
* DefaultProductEligibilityImplementation
  + Returns the input without any modifications in relation to eligibility.
* FilterEligibilityImplementation
  + Runs advanced eligibility rules when the sequence of those rules is unimportant.

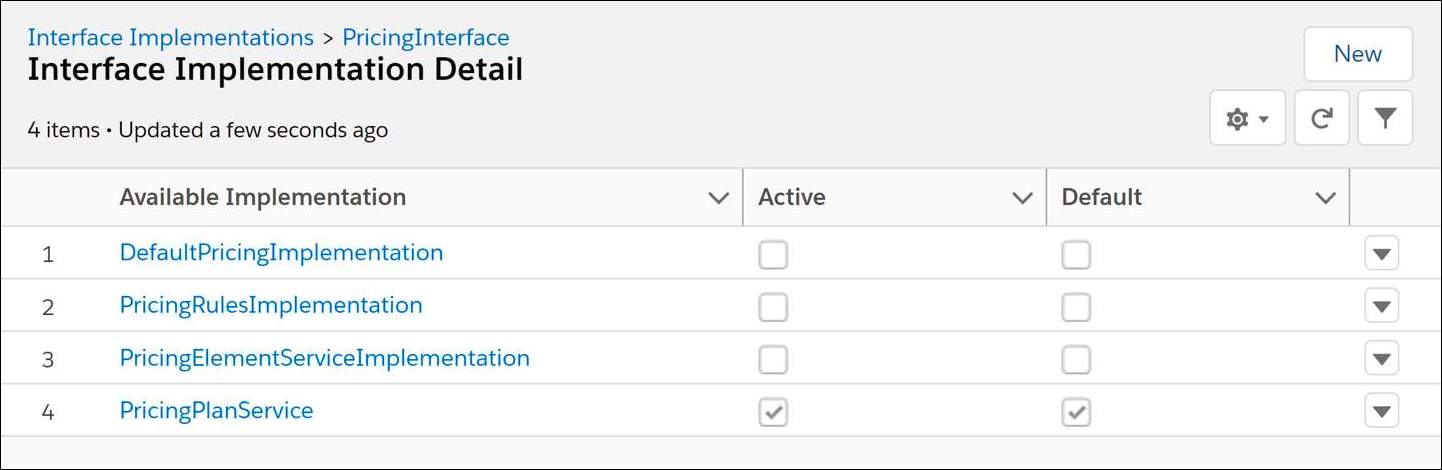
**Pricing Rules & PricingInterface**

Pricing rules use the **PricingInterface** and associated implementations to calculate correct pricing. The PricingInterface and implementation are triggered when a product is added to, deleted from, or modified in an opportunity, order, or quote.

It allows you to create advanced rules that call calculation procedures as actions. The implementation uses price books, rather than price lists, to price products.

Pricing rule implementations include:

* **DefaultPricingImplementation**: use with standard pricing rules that are not dependent on any external factors.
* **PricingRulesImplementation:**allows you to create advanced rules to implement pricing rules that call calculation procedures as actions.
* **PricingElementServiceImplementation:**uses pricing element pricing capabilities to retrieve the charges and adjustments for products from price lists. It also calculates price adjustments based on promotions also defined in Vlocity Product Console.
* **PricingPlanService** is used when you require pricing in an implementation which crosses multiple application domains.



# Product Relationships

Product relationships are used to determine if a product combination is valid. Using product relationships, you can define that, when a product is added to the Cart, related products are added with it. You can also automatically add, automatically remove, or recommend products based on other products in the Cart. Product relationships are triggered using advanced Configuration/Validation rules. Advanced Configuration / Validation rules run on order line item objects—Opportunity Line Item, Quote Line Item, or Order Line Item, and use product relationships as rule actions.

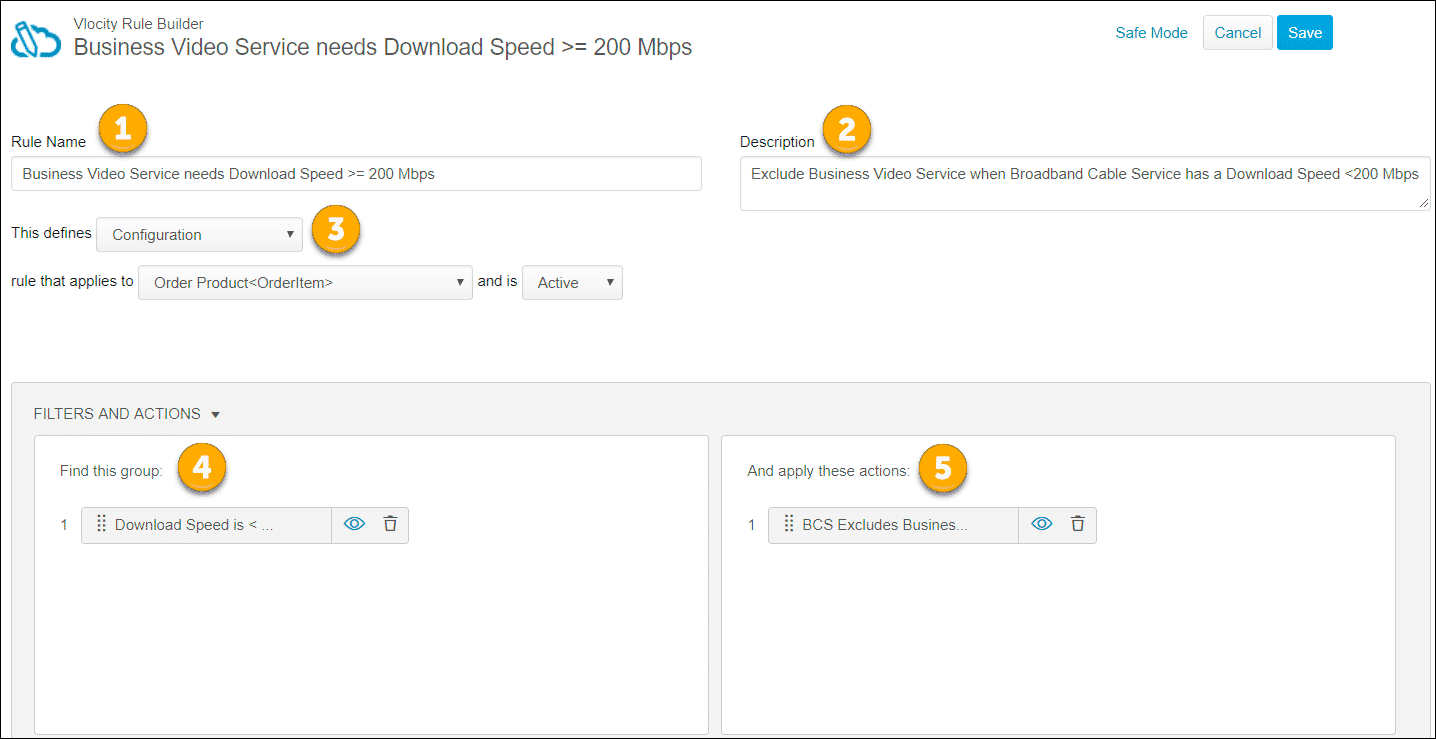
**Rule Builder**

**Rule Builder**

The **Vlocity Rule Builder** is a drag ‘n drop design tool available on the Vlocity Rules tab.

* The natural language and visual Rule Builder allows you to design availability, eligibility, configuration, and pricing rules to ensure the right products are presented to the right customers at the right prices.
* Advanced rules determine under what circumstances they apply using entity filters.
* You can apply one or more rule actions to each item qualified by the entity filter. You can choose from existing filters and actions, or create new ones.

This is an example of a compatibility rule that excludes Business Video Service when the Download Speed attribute is less than 200 Mbps.

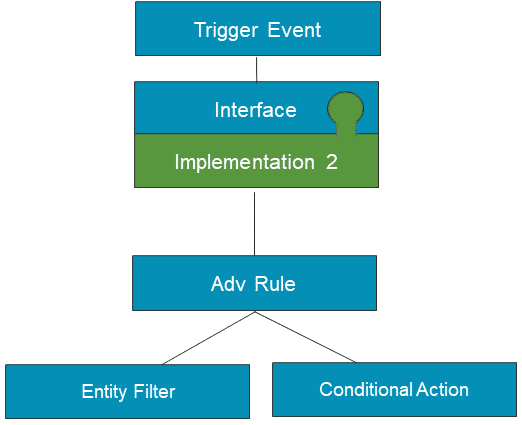


1. Write the**Rule Name** in natural language, and make it as short yet descriptive as possible.
2. Make the **Description** longer to explain to other users the purpose of the rule.
3. Specify the rule type and object. A compatibility rule runs against the following line-item objects: Opportunity Line Item, Quote Line Item, Order Item. You can activate and deactivate rules while testing.
4. **Filters:** In the left pane, select the entity filter to trigger when the rule executes. We'll learn about entity filters in the next module.
5. **Actions:** In the right pane, select the rule action to execute. Rule actions are simple, clear indications of the actions performed. Depending on the type of rule you're building:
   1. Compatibility rules use product relationships as their rule action (as seen above).
   2. Pricing rules use calculation procedures to define pricing logic.
   3. Offering procedures are custom objects that allow you to exclude or include products in the product list. Inclusions include all products qualified by the entity filter in the rule action. Exclusions exclude all products qualified by the entity filter in the rule action.

**Advanced Compatibility/Validation Rules**

Here, we can graphically see the sequence of implementation tasks. Review the diagram from the bottom up:

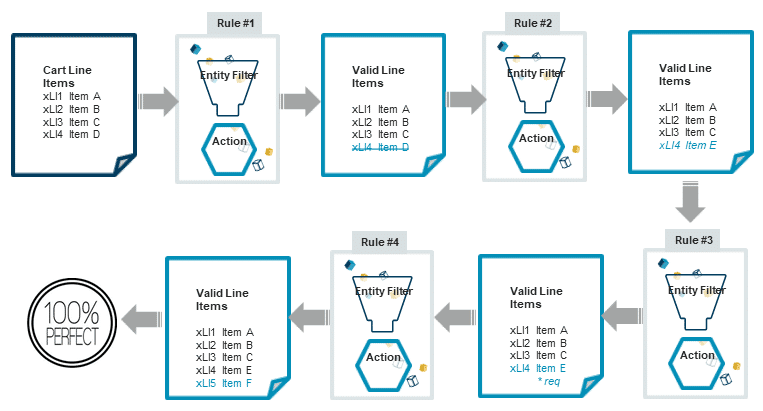
1. Create an entity filter and/or a conditional action.
2. Create the advanced compatibility/validation rule.
3. Enable the implementation to the interface/object.
4. The trigger event causes the rule to run.



**Multiple Advanced Rules: Working Together**

You can create multiple compatibility advanced rules to work together to create the “perfect order.” In the scenario below, 4 different compatibility rules do the following:

1. Rule#1 excludes Item D.
2. Rule#2 auto-adds Item E.
3. Rule#3 determines that Item F is now required because Item E was added.
4. Rule#4 determines all line items are valid now that Item F is in the Cart, and you can now submit the order for fulfillment.



**Entity Filters**

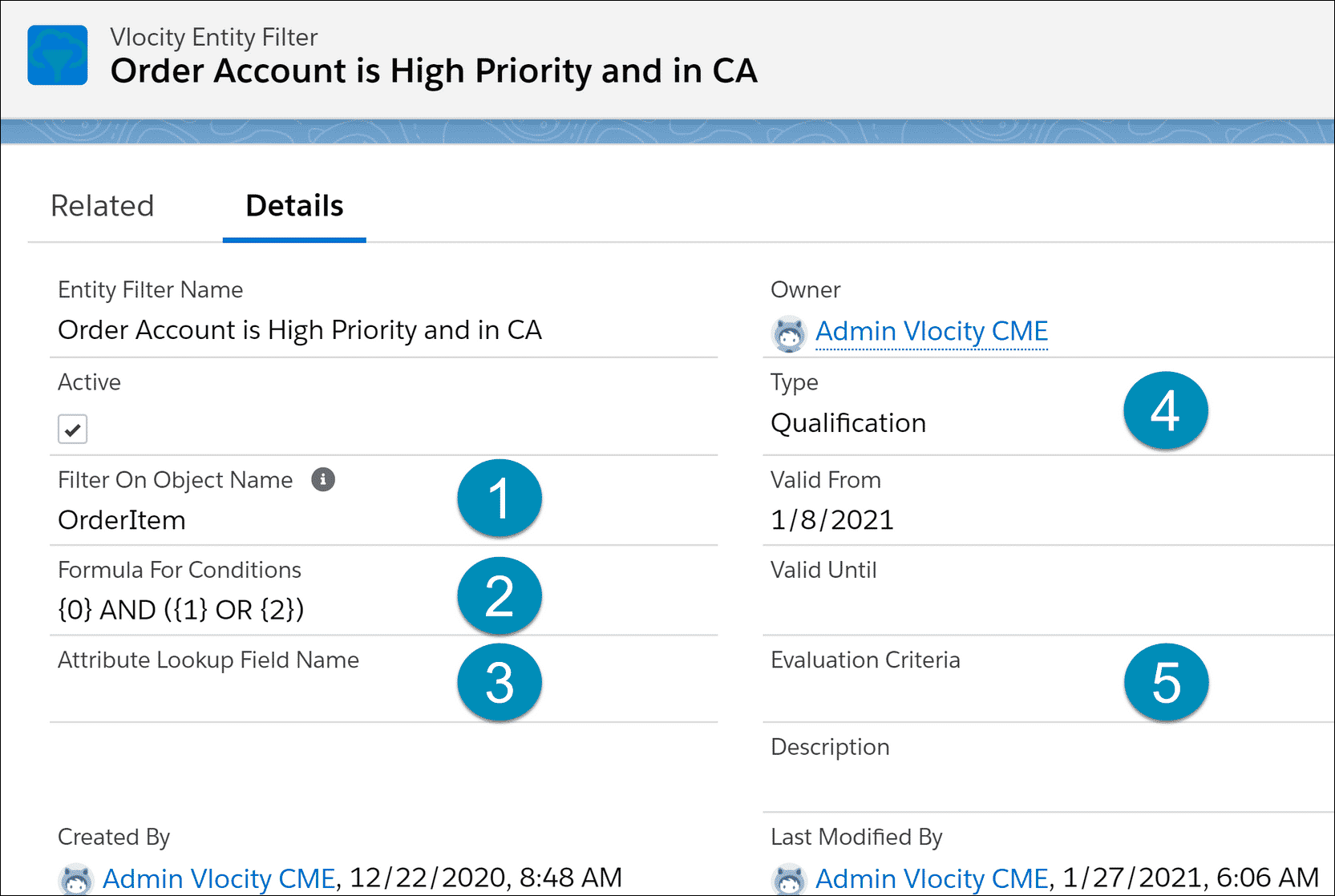
In the first exercise, we created product relationships and added them as actions to an advanced rule, which fired every time the products appeared in the Cart. However, sometimes you will want to create rules that fire only when certain conditions arise. To do that you will add entity filters to your advanced rules. Advanced Rules use entity filters to determine which products and under what circumstances they apply.

**What are Entity Filters?**

Entity filters create context for advanced rules by filtering order line items and searching for defined conditions. They consist of one or more conditions that evaluate fields or attributes on an object, and if true, apply a rule action, which can be a product relationship, a matrix, a calculation, or a procedure.

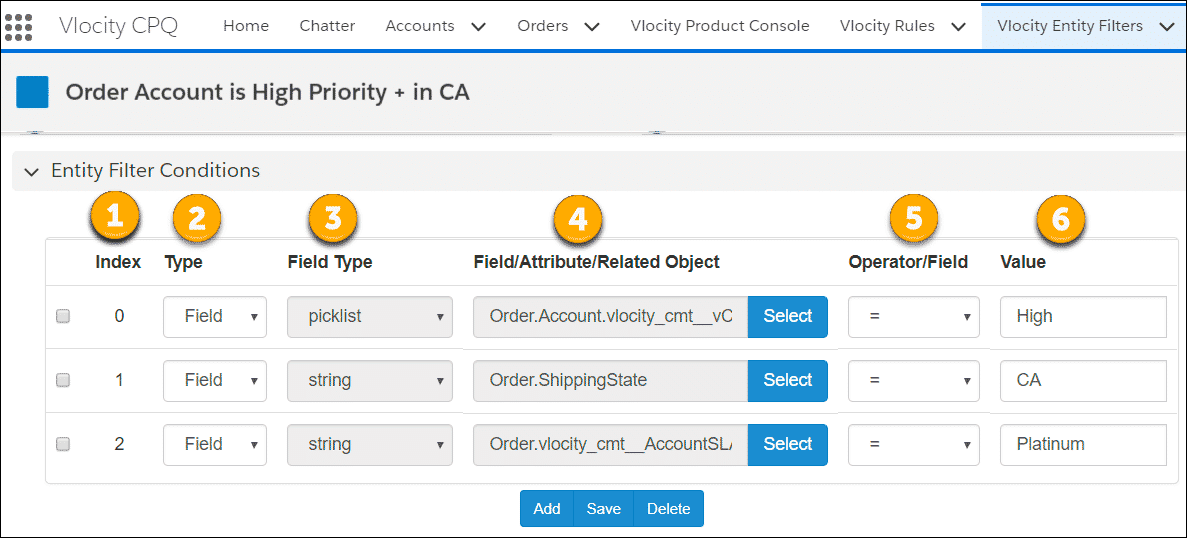
* Rules can contain one or more filters, depending on the business logic that the rule implements.
* Filters are reusable and additive. Once you create a filter, it's available for use in any rule that needs it.

**Entity Filter Details**



1. **Filter on Object Name:** The object against which the filter conditions are evaluated. In this case, OrderItem or order line item.
2. **Formula for Conditions:** Defines the formula used to evaluate the filter conditions (AND if not specified).
3. **Attribute Lookup Field Name:**Defines the field name containing a JSON attribute string.
4. **Type:**Qualification or Evaluation.
   1. Qualification determines if a set of records qualify to be acted on, returning qualified line items.
   2. Evaluation determines if a set of object records matches specific conditions, returning true or false.
5. **Evaluation Criteria:** Required if the filter Type is Evaluation. It evaluates if All, Any, or None of a set of objects satisfy a set of conditions.

**Entity Filter Conditions**



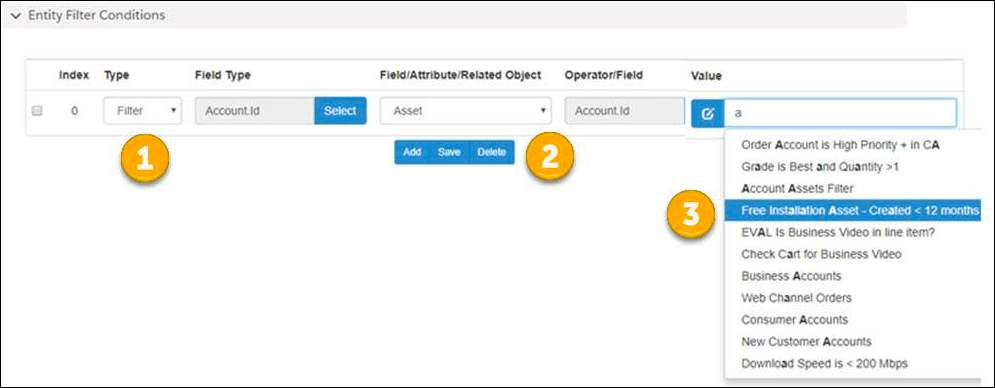
*A single entity filter can have one or more conditions. See the descriptions of the columns below.*

1. **Index:** lists the number to use for identifying the condition when creating a formula for conditions.
2. **Type:**specifies what will be checked. The available types of filter condition are:
   1. **Attribute:** select products based on attributes.
   2. **Field:** select products based on fields in an object.
   3. **Filter:** the system looks up another filter and does something else with it. For this type, you provide a comma-delimited path to the list of objects to pass to the internal filter.
   4. **Function:** Instead of creating a filter and defining a comma-delimited path to the list of objects, you can use a function that offers predefined data sets to pass to the evaluation filter, so that you don't have to specify them.
3. **Field Type:** Picklist or string.
4. **Field/Attribute Name/Related Object:** Defines the Field or Attribute Name to be evaluated. When the condition Type is Filter, this value holds the reference of the internal evaluation filter.
5. **Operator:** defines the condition logical operator.
6. **Value:**specifies the values for comparison and can include:
   1. Literal String values
   2. Date/DateTime functions
   3. Internal filters
   4. Variables, using syntax {!$Variable.variableName}, where variableName is the name of the Rule Variable.
      1. Forms a nested entity filter

**Filter Condition Types**

You can create entity filters to look across objects, tracing data from the order to the account to the assets. You can evaluate the results of another entity filter using an evaluation or compound entity filter, which pulls the account ID from the current order, goes to the asset object, and gathers the records with a matching account. Then it applies those records against the entity filter.

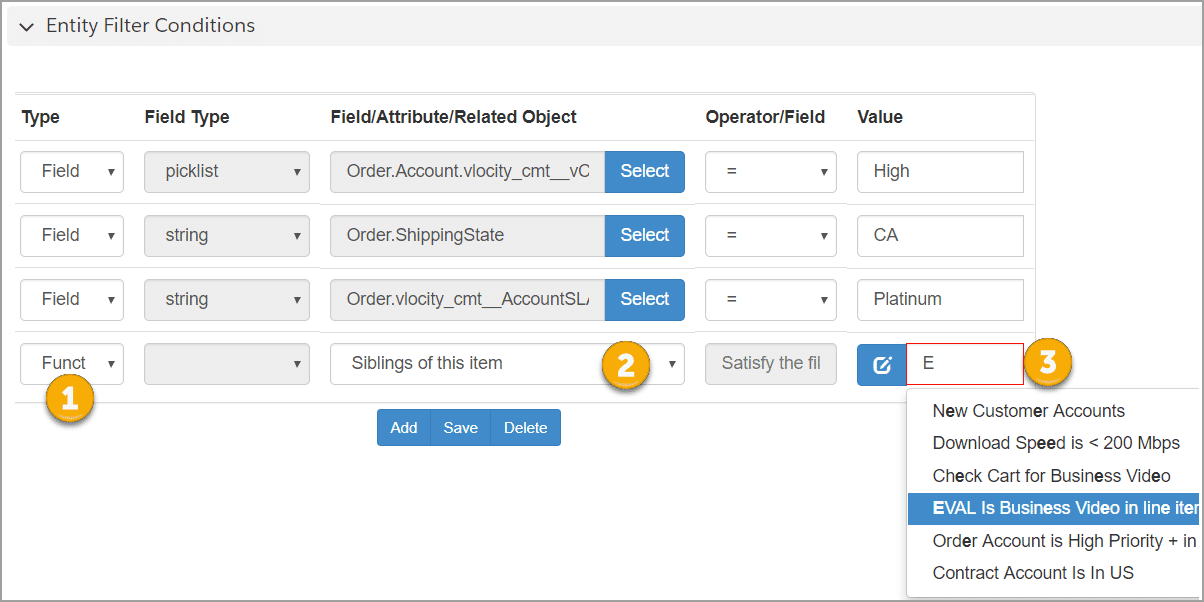
When using entity filters with conditions of type **Filter**, you define a path to the list of objects to pass to the internal filter. The internal filter must be an **Evaluation**filter and the evaluation criteria must be set.



In this example, the entity filter of Type **Filter (1)** finds the **account assets** **(2)**, and then calls an **evaluation**filter named "Sample Free Installation Asset created in last 12 months" **(3)**. This evaluation filter looks at each asset record to determine if it is the "Free Installation" product and dated in the last 12 months.

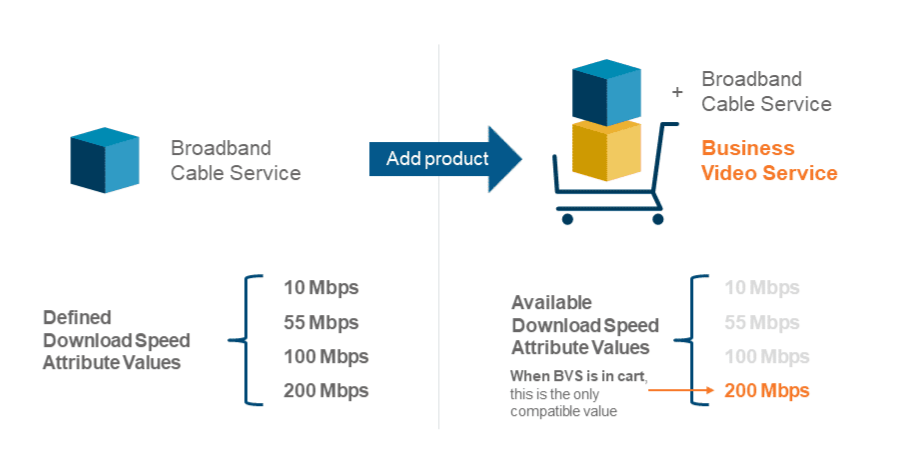
When you chain a qualification filter with an evaluation filter, it's called a compound filter.

**Function Condition Types**



1. **Type = Function:**Instead of creating a filter, you can use a function that offers predefined data sets to pass to the evaluation filter, so that you don't have to specify them.
2. **Predefined Data Sets:**Replaces the Compound Filter Object Path with common options such as Parent of this item, Siblings of this item, Root of this item, Items in cart, and Items in this bundle.
3. **Value:** Stores the name of the evaluation filter to use.

# Using JSON Notation in Advanced Rules

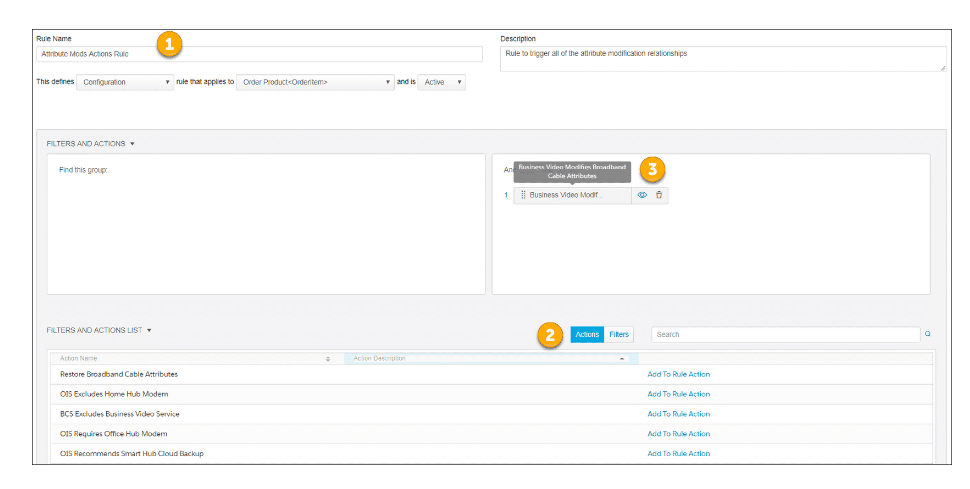


This diagram shows an advanced compatibility rule that modifies attributes. In this case, a Business Video Service product requires a high-bandwidth broadband connection.

A compatibility rule can use product relationships to evaluate the Download Speed Attributes for the Broadband Cable Service product.

**Creating a Rule to Trigger the Attribute Modification Relationships**

Go to the **Vlocity Rules** tab and click **New**to access the rule builder.

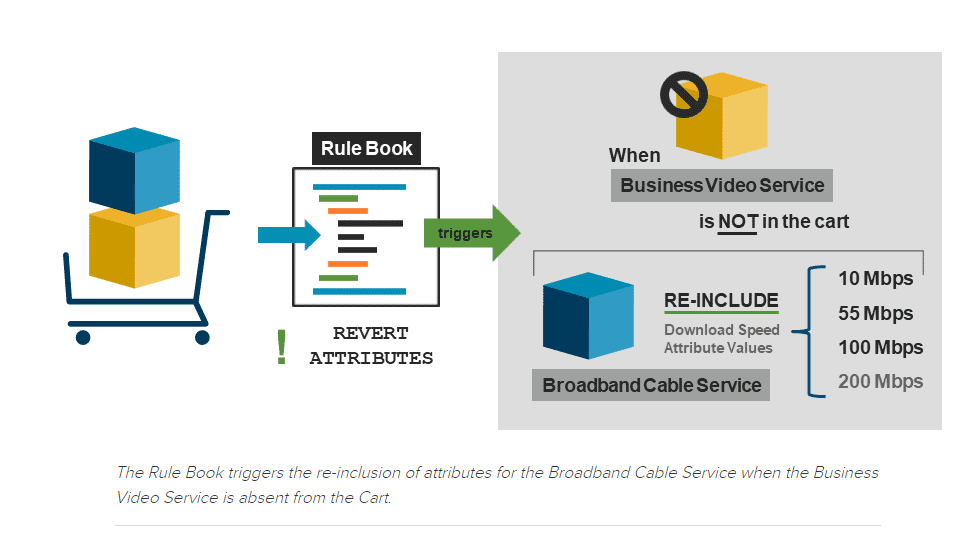


1. Enter the rule name, description, and definition in the rule header.
2. Click **Actions** to display a list of available actions, then click **Add to Rule Action** to apply the action to this rule.
3. The selected action displays in the right pane.

# Creating a Product Relationship to Restore Attributes to Their Original State

**Restoring Attributes to Their Original State**

When users remove the product that triggers the attribute modification, what can you do to ensure the modified attributes return to their original state? In the example below, we need to restore the download speed attribute values to the Broadband Cable Service product now that Business Video Service is no longer in the Cart.



# Creating an Evaluation Entity Filter to Check for Absence of Product

We need the product relationship we described in the previous lesson to fire ONLY when Business Video Service does not exist in any line item of the Cart.

To do this, we use an **evaluation entity filter**to check for the product name.

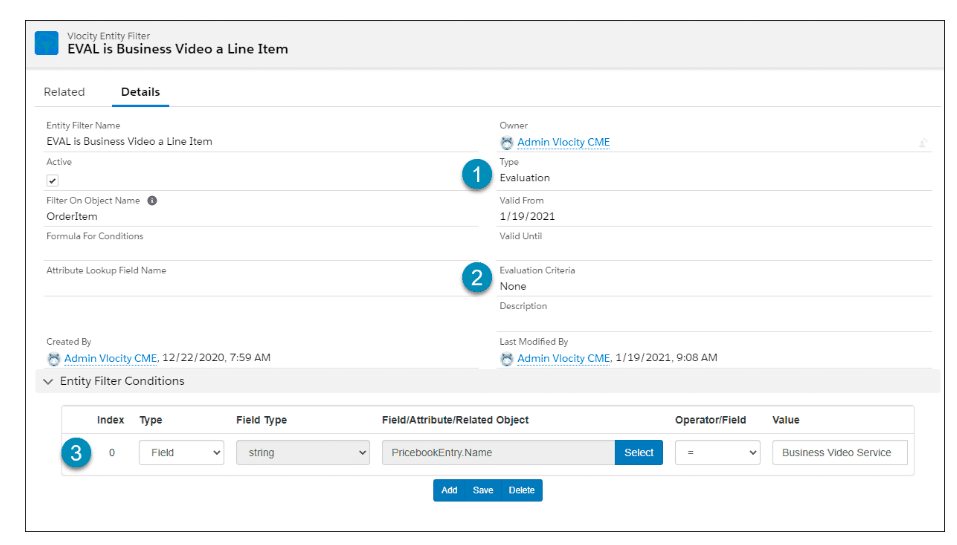
Next, we create a **compound entity filter**that chains the evaluation entity filter to a **qualification entity filter** that prepares the list of input items for evaluation. When true, the entity filter passes the list of qualified items to the Advanced Rules Engine.

**Creating an Evaluation Entity Filter to Check for Absence of Product**

Evaluation entity filters are not used directly in rule building and instead are “internal filters” to qualification entity filters. They evaluate if all, any, or none of a set of records satisfy certain conditions.

Evaluation entity filters accept a list of items as input and return true or false as the output. For example, an evaluation filter evaluates to true if any of the product children has a maximum quantity of less than 10 and a minimum quantity larger than 2.

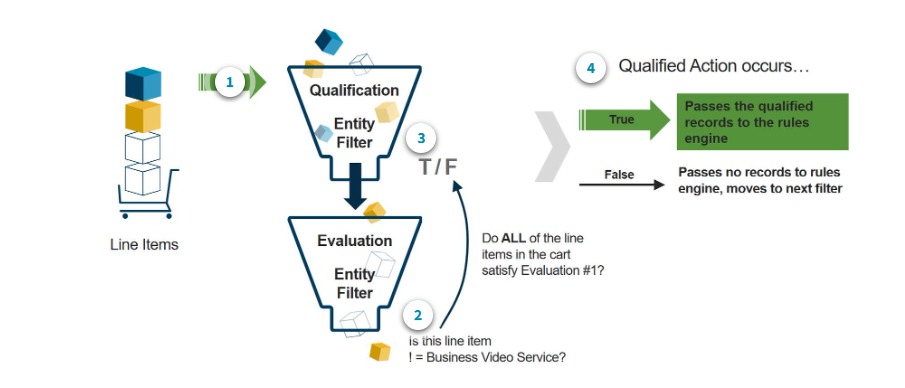
You create evaluation entity filters within the Entity Filters tab.  Here's an example of an evaluation entity filter that checks if any of the line items are Business Video Service.



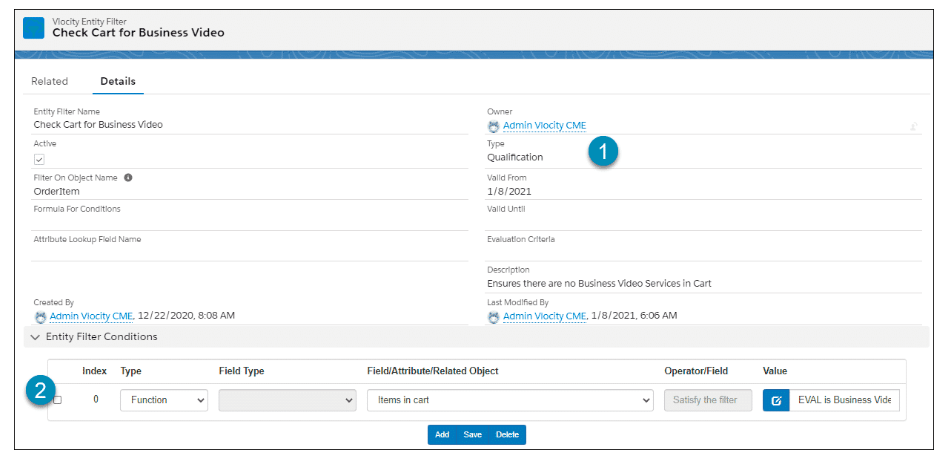
1. **Type**is Evaluation.
2. **Evaluation Criteria** specifies the criteria in which the filter should return a True result. In this case, it says that if none of the line items are Business Video Service, all the line items have satisfied the condition, and thus can be passed either directly to the rules engine or to other conditions in the calling qualification filter.
3. **Entity Filter Condition** is evaluating the PricebookEntry.Product Name field. This is the correct field since the object we're evaluating is an order line item, and the primary key to this object from a data model perspective is the pricebook entry object.

In a real world implementation, it's unlikely you would create a rule that evaluates a product name, since that might change, and if it did, it would “break” this filter and the rule that uses it. Instead, you would create the filter to evaluate the product code, which is less likely to change.

**How Does a Compound Entity Filter Work?**



**Creating a Compound Qualification Entity Filter**



1. **Type** is Qualification.
2. **Entity Filter Condition**is a function that has a defined scope; in this case, **Items in cart.**  
     
   Defined scopes in function conditions reduce the number of records to optimize processing.  
     
   The reduced set of records is passed to the linked evaluation entity filter for processing.

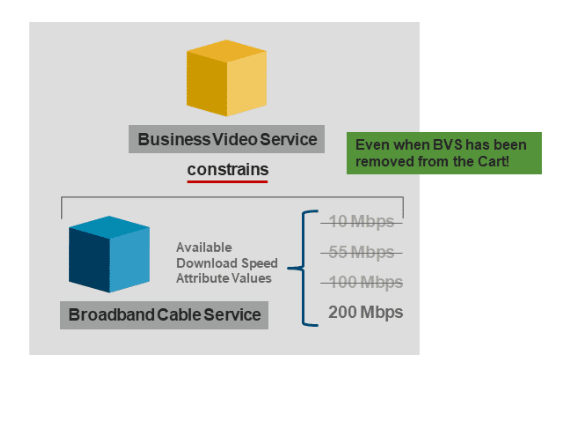
# Creating an Advanced Rule to Trigger Reverse Attribute Modifications

# 

1. Enter the rule name, description, and definition in the rule header.
2. Click **Actions** to display the list of available actions, then click **Add to Rule Action** to apply the action to this rule. In this case, the**Restore Broadband Cable Attributes** product relationship.
3. Click **Filters** to display the list of available filters. Drag the **Check Cart for Business Video** filter on top of the **Restore Broadband Cable Attributes** action.

Using the filter as a modifier to the action increases reusability of this rule. We can add multiple actions to this one rule, and each action will only fire for records qualified by the attached filter.

**Modifying Attributes Unless Explicitly Reversed**



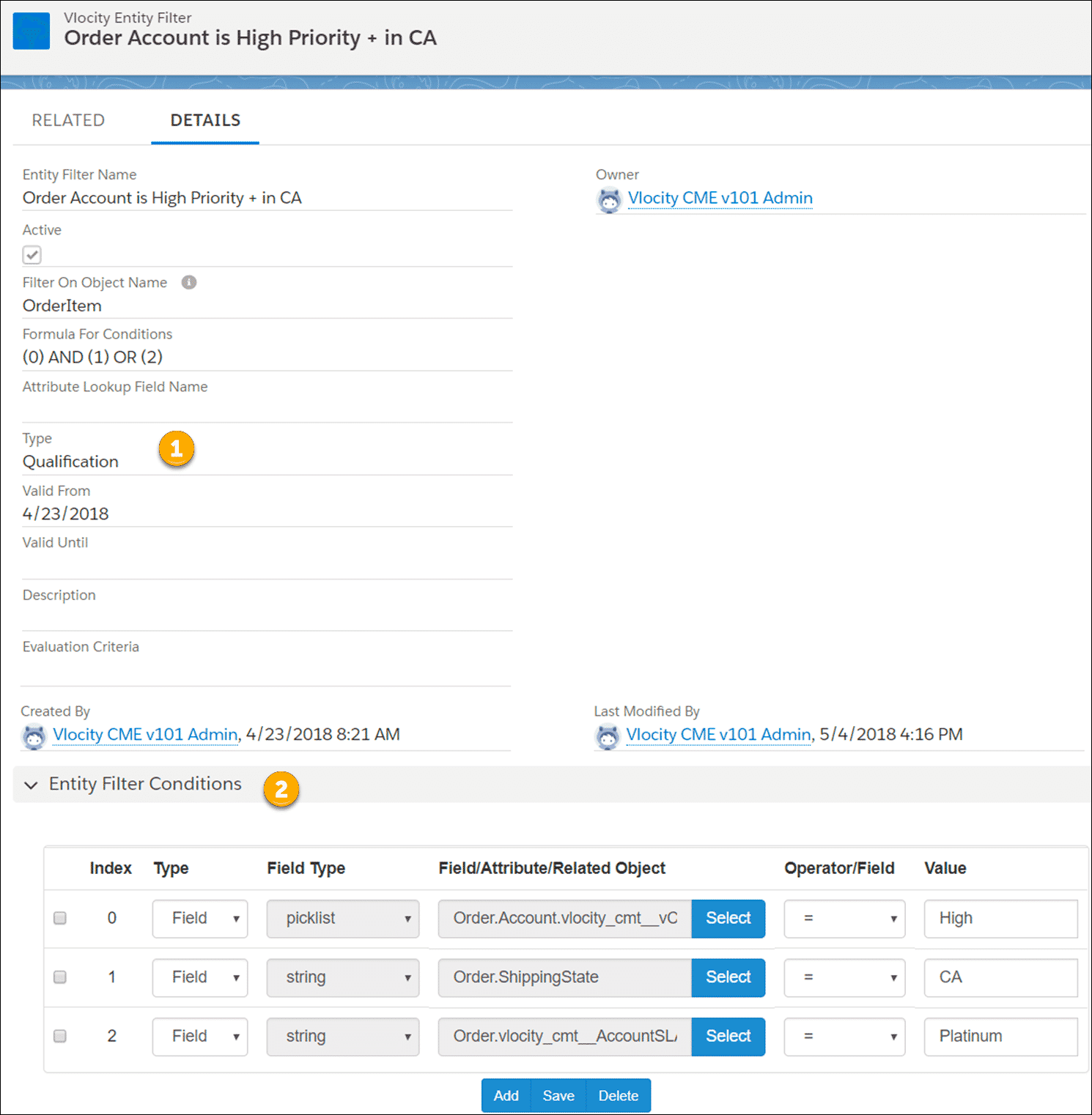
This diagram shows a rule that applies constraints to Business Video Service so that the Broadband Cable Service is only available at 200Mbps. The rule stays active even with the Business Video Service is no longer in the Cart.

To do this, create a product relationship where you select the product to be modified and leave the related product field blank. As we covered above, a blank related product alerts the rules engine to apply the given action parameters to the product on its own.

# Creating an Entity Filter for High Priority Accounts

What if we want our free phone case to be added only to customers we have identified as high priority, who live in CA, and have a platinum level service agreement?

We need to create an **Entity Filter** for the rule, to specify where it applies. Begin by going to the **Entity Filter tab**, then click on the **New**button in the top right to create a new entity filter. Follow the instructions below to complete the entity filter details.

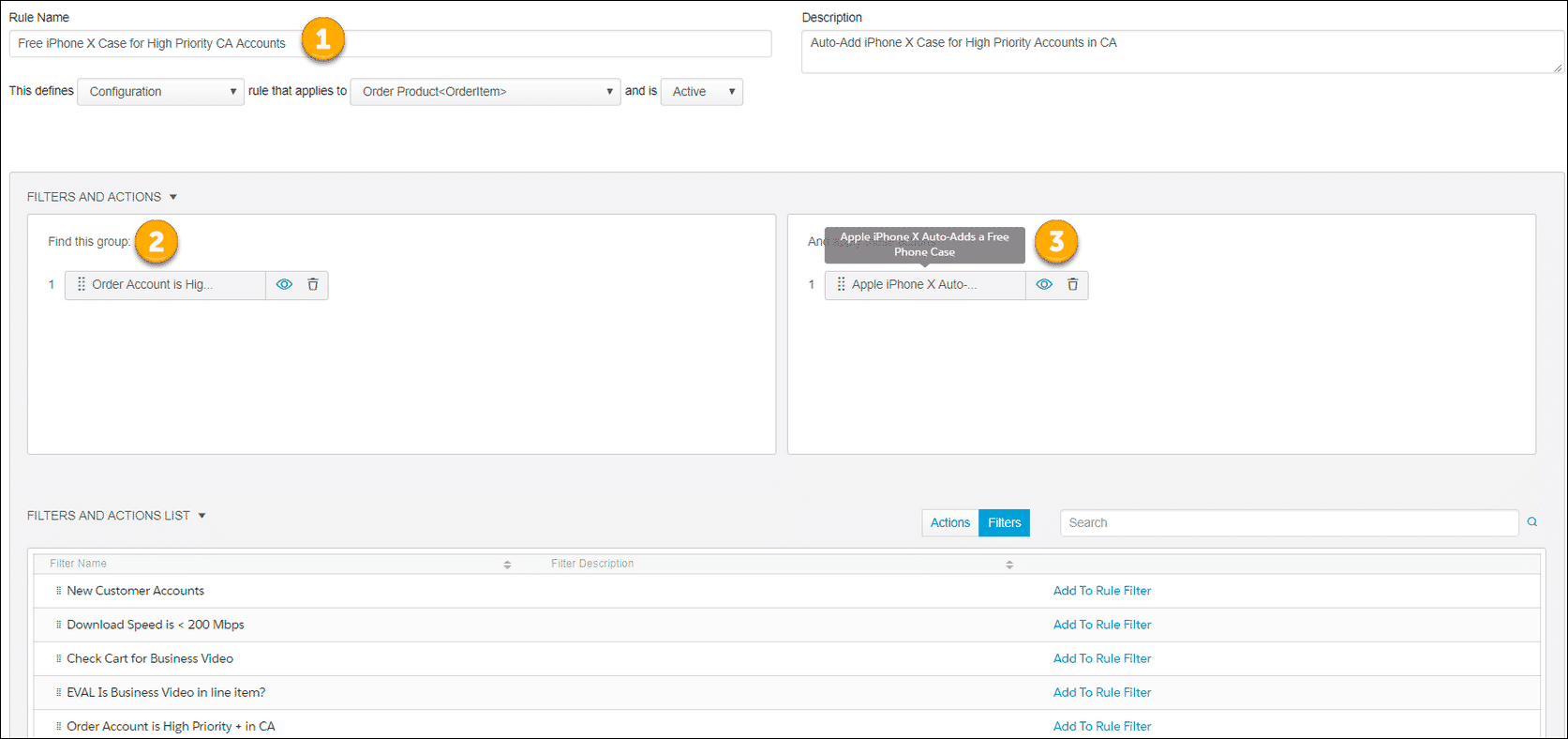


1. **Type** is Qualification.
2. **Create 3 Entity Filter Conditions**to do the following:
   1. Filter high-priority accounts.
   2. Filter for accounts in California.
   3. Filter for accounts that have a Platinum service level agreement (SLA).

# Combining the Product Relationship and Entity Filter

Now it's time to combine the entity filter (which chooses which customers get the free phone case) and the product relationship (which determines the products and the action to be taken) into a rule.

Go to the **Vlocity Rules** tab and click on the **New**button to create a new rule. Follow the instructions below to complete the rule details.



1. Enter the rule name, description, and definition in the rule header.
2. Click **Filters** to display the list of available filters. Select **Order Account is High Priority + in CA**.
3. Click **Actions** to display the list of available actions, then click **Add to Rule Action** to apply the action to this rule. In this case, the**Apple iPhone X Auto-Adds a Free iPhone Case** product relationship.

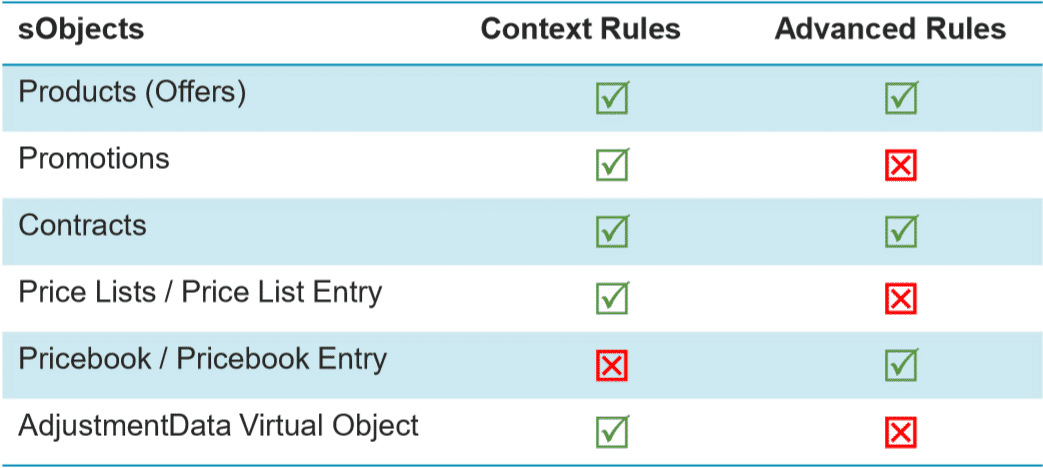
**Rules: How to Decide?**

**Why do you need this rule?**

Do you need to check product compatibility? Do you need to check whether a customer is eligible for a product? Or whether a product is available in a geographic region? Do you need to apply a penalty? Do you need to automatically add a product to the Cart based on other conditions in the Cart?

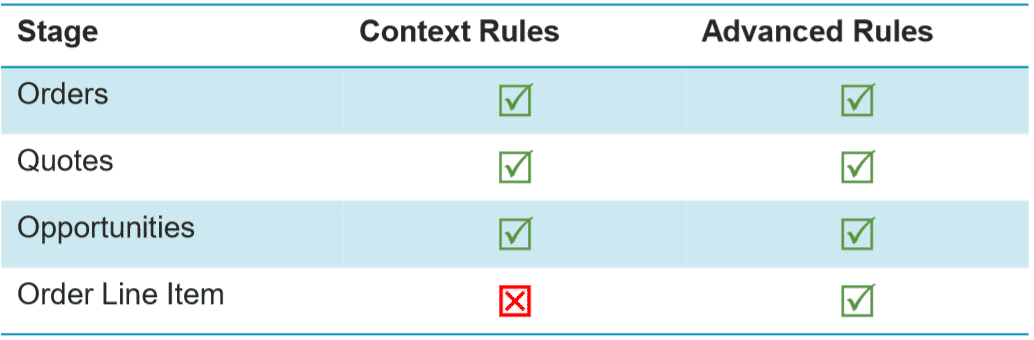
**What object will the rule apply to?**

Will it be used to determine eligibility for a product or promotion? Will it apply pricing? Will it apply a penalty when canceling a promotion or a contract?



**When will this rule apply?**

At what point in the order capture process will this rule apply? Will the rule evaluate items before they are placed in the Cart in an order, quote, or opportunity? Will it be used to evaluate the line items after they are in the Cart?



**How will you administer or manage the rule?**

Will this rule apply to a single product or to thousands? Do you need maximum performance?

