# Corticon Rule Modeling Challenge Jan 2018 Order Promotions

Mike Parish

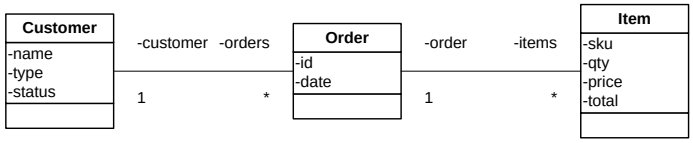
## The Problem

The objective of this challenge is to help merchants to define various promotions for their sales orders and to automatically decide if an order is eligible for a promotion. An order usually consists of order items with known price and quantities. A promotion defines minimal quantities of certain items in the eligible orders. A simple example of promotion: *reduce the total cost of the order by $3.50 if it contains at least 5 items 1108 and at least 4 item 2639.*

## The Approach

Usually the first step in solving any kind of business rule problem is to clearly identify all of the business objects involved, their attributes and their relationships to one another. This is true regardless of which rule engine you may choose to implement the rules. It's also true even if you will be using any other tools.

For this problem we might have a data model something like this




A general approach to a problem like this might be

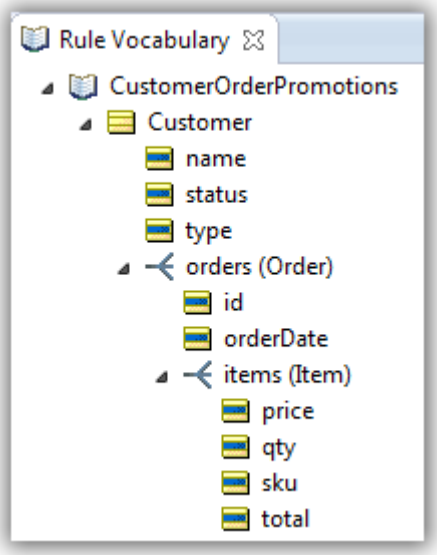
*For each promotion*

1. *Determine if the customer is eligible to participate (based on customer type, status etc)*
2. *Determine which orders are eligible (based on order date etc)*
3. *Determine which items are eligible (based on sku, qty etc)*
4. *Determine discounts and rebates*
5. *Compute the totals*

Each promotion would then be represented as a set of decision tables that go through each of these five steps.

## Corticon Solution

The objects, attributes and relationships would look like this in Corticon.

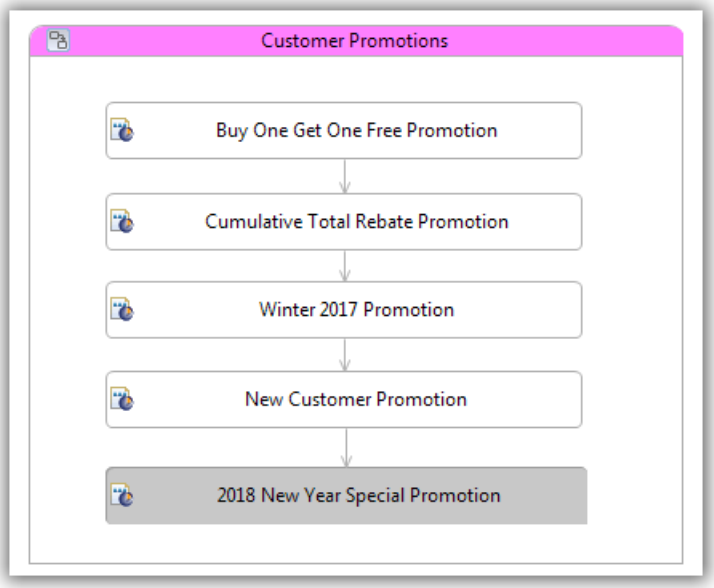



Customers can have many orders; Orders can have many items.

Incidentally, if you already have a UML diagram or database tables then you can create the Corticon vocabulary automatically.

We will probably find, as we develop the rule model, that more attributes need to be added to keep track of intermediate results. For example totals.

A high level flow might look like this where each little box represents a different promotion.




We could keep them all in a single decision like this or we could split them into separate decisions.

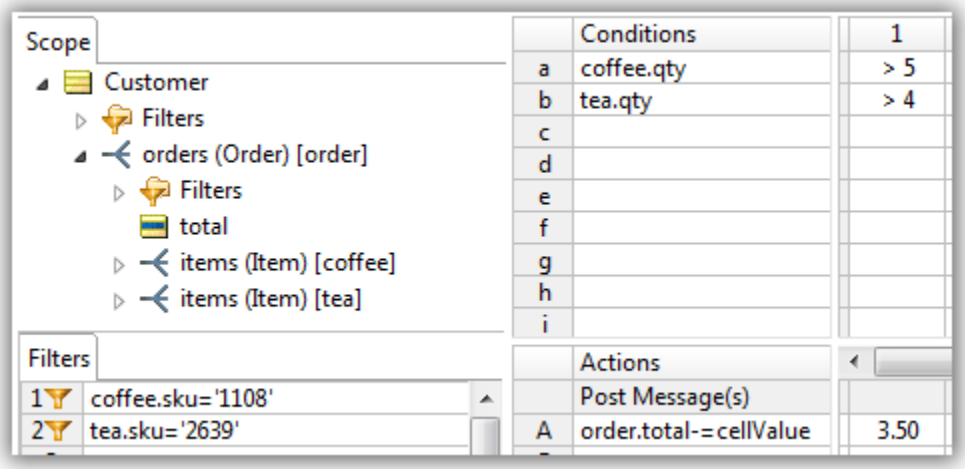
This would depend on whether there were any dependencies between the promotions.

Maybe if you get the New Customer promotion you can't also get the Winter 2017 promotion.

The New Year Promotion is grey because it's not yet active.

Let's take a look at the rule "*reduce the total cost of the order by $3.50 if it contains at least 5 items 1108 and at least 4 item 2639.*"

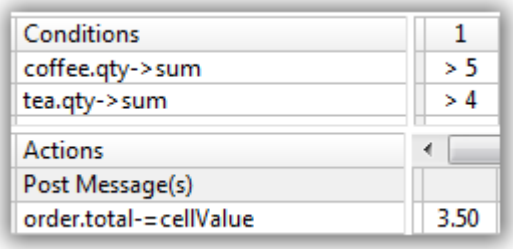
Let's suppose 1108 is coffee and 2639 is tea. Then a simple rule sheet might look like this:




The scope says "for each **order** belonging to each **customer,** consider coffee **items** and tea **items**"

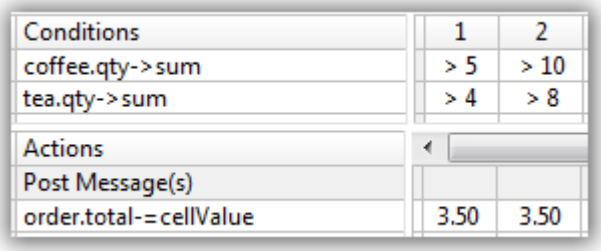
The filters define the two types of item we are interested (by sku) and then we can use more meaningful names (coffee, tea) in the rules.

But will this always work? We are assuming that there is just one item for coffee and one item for tea in the order. What if there were six separate items for coffee each with quantity 1? Then the rules above wouldn't work. Fortunately there is a simple solution using the sum operator:

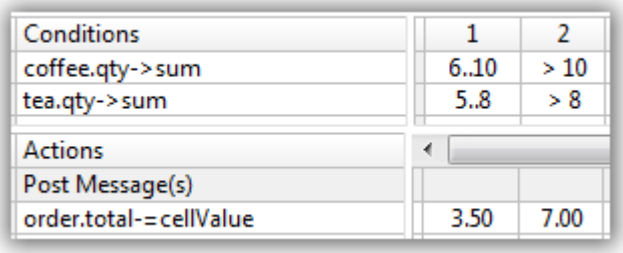



This pattern can easily be extended to other combinations of quantity:

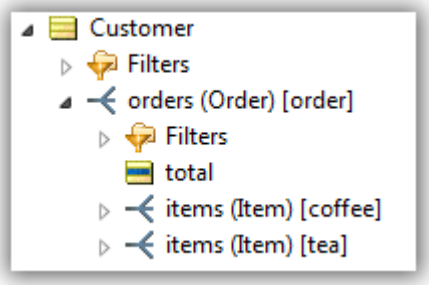
In this case the discounts are cumulative since if you fire rule 2, rule 1 will also fire.




If you don't want cumulative discounts then you could represent it like this by specifying the applicable ranges more precisely.



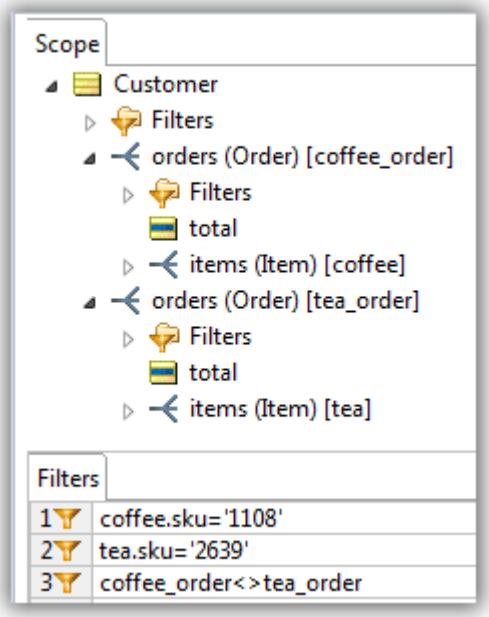
But remember that, since we defined the scope like this, the discount only applies to coffee and tea items WITHIN the SAME order. Not on different orders.




If we needed the quantities to be on different orders then we'd do something along these lines:

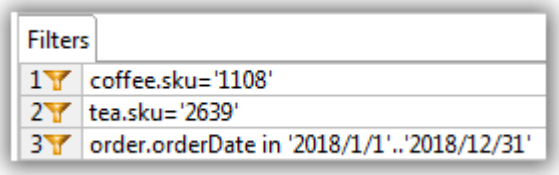
First we modify the scope to create aliases to two different orders

And then we might apportion the $3.50 discount to each order as follows:

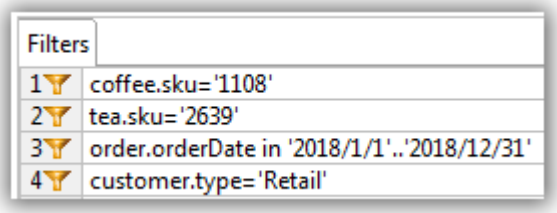



| Conditions | 1 |
|---|---|
| coffee.qty->sum | > 5 |
| tea.qty->sum | > 4 |
| **Actions** | |
| Post Message(s) | |
| tea_order.total-=cellValue | 1.50 |
| coffee_order.total-=cellValue | 2.00 |

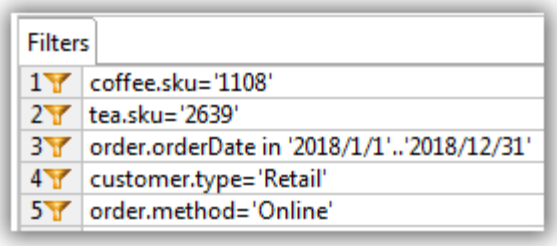
Most promotions have a start date and end date and this is easily handled by adding an appropriate filter to the scope section so that only orders within the range get processed by the rules:




If we further wanted to restrict the promotion to a particular customer type (say Retail) we simply add another filter:




To restrict to Online orders:



For the rest of this discussion let's assume that each order does not have duplicate skus. We can easily arrange for this by using the logic described in an earlier challenge dealing with finding duplicates. (https://dmcommunity.org/challenge/challenge-august-2015/ )

In a traditional rule sheet that does not use the filter mechanism you would need to put all these conditions in the body of the rule sheet; and although a bit cumbersome this will work just fine.

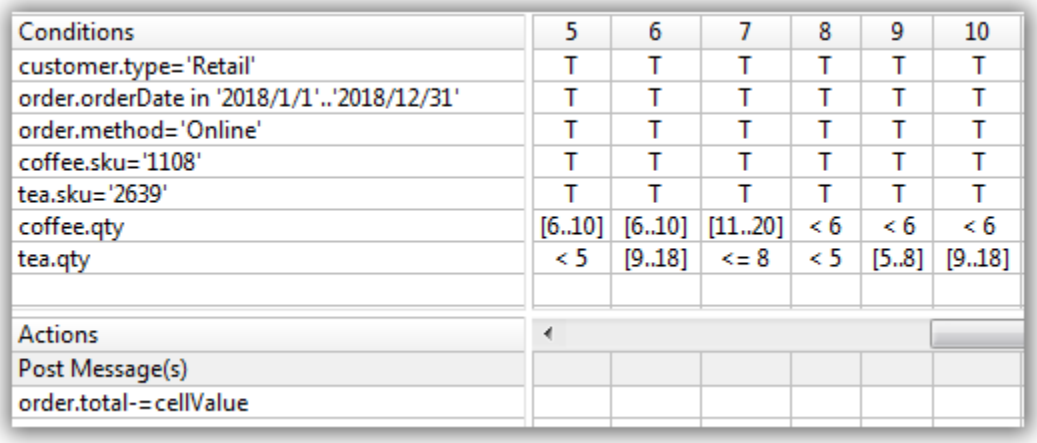
| Conditions | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| customer.type='Retail' | T | T | T | T |
| order.orderDate in '2018/1/1'..'2018/12/31' | T | T | T | T |
| order.method='Online' | T | T | T | T |
| coffee.sku='1108' | T | T | T | T |
| tea.sku='2639' | T | T | T | T |
| coffee.qty | 6..10 | 11..20 | > 20 | - |
| tea.qty | 5..8 | 9..18 | - | > 18 |
| | | | | |
| Actions | | | | |
| Post Message(s) | ✉ | ✉ | ✉ | ✉ |
| order.total-=cellValue | 3.50 | 4.50 | 5.00 | 6.00 |

But as you can see the highlighted conditions have the same values for every rule and so are more simply expressed using the filter mechanism. It's also more efficient to use filters since only the data that gets through the filter is processed by the rules. Without the filter, the same data is evaluated once per rule. The more rules in the decision table, the more efficient it is to use a filter instead.

The filters become even more powerful if the data is actually coming from a database since the rule sheet can use the filters to automatically create a SQL query to retrieve only the records that meet the filter criteria.

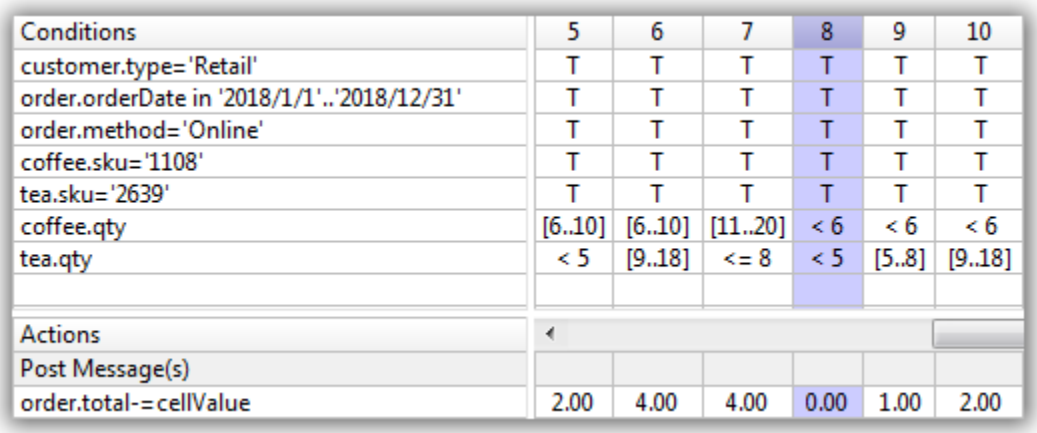
Side note: the decision table above is not complete – can you see where it still has holes?

Corticon can automatically determine that we have not provided discounts in the following cases



| Conditions | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|
| customer.type='Retail' | T | T | T | T | T | T |
| order.orderDate in '2018/1/1'..'2018/12/31' | T | T | T | T | T | T |
| order.method='Online' | T | T | T | T | T | T |
| coffee.sku='1108' | T | T | T | T | T | T |
| tea.sku='2639' | T | T | T | T | T | T |
| coffee.qty | [6..10] | [6..10] | [11..20] | < 6 | < 6 | < 6 |
| tea.qty | < 5 | [9..18] | <= 8 | < 5 | [5..8] | [9..18] |
| | | | | | | |
| Actions | | | | | | |
| Post Message(s) | | | | | | |
| order.total-=cellValue | | | | | | |

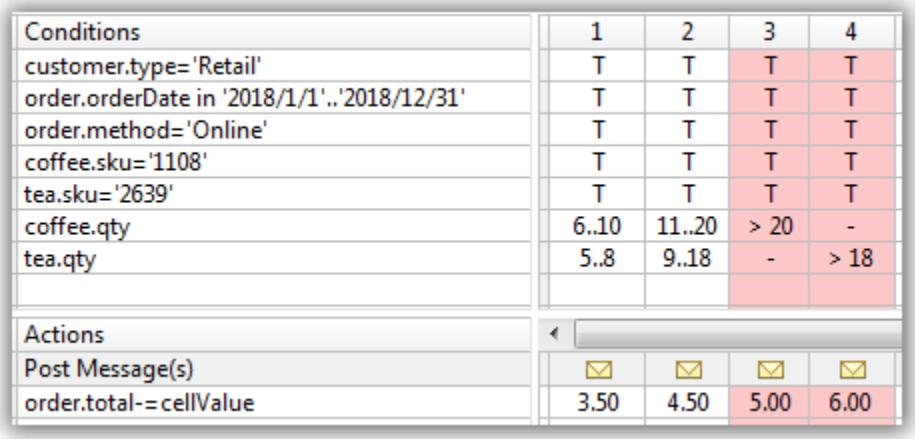
After talking with our business users we might determine that discounts for these situations are as follows:



| Conditions | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|
| customer.type='Retail' | T | T | T | T | T | T |
| order.orderDate in '2018/1/1'..'2018/12/31' | T | T | T | T | T | T |
| order.method='Online' | T | T | T | T | T | T |
| coffee.sku='1108' | T | T | T | T | T | T |
| tea.sku='2639' | T | T | T | T | T | T |
| coffee.qty | [6..10] | [6..10] | [11..20] | < 6 | < 6 | < 6 |
| tea.qty | < 5 | [9..18] | <= 8 | < 5 | [5..8] | [9..18] |
| | | | | | | |
| Actions | | | | | | |
| Post Message(s) | | | | | | |
| order.total-=cellValue | 2.00 | 4.00 | 4.00 | 0.00 | 1.00 | 2.00 |

Note that in rule 8 we can explicitly identify when they receive no discount. This enables us to generate a useful message that tells exactly why they got no discount. In fact as a general best practise, every rule should have an associated message that describes the intent and outcome of the rule in business friendly terms.

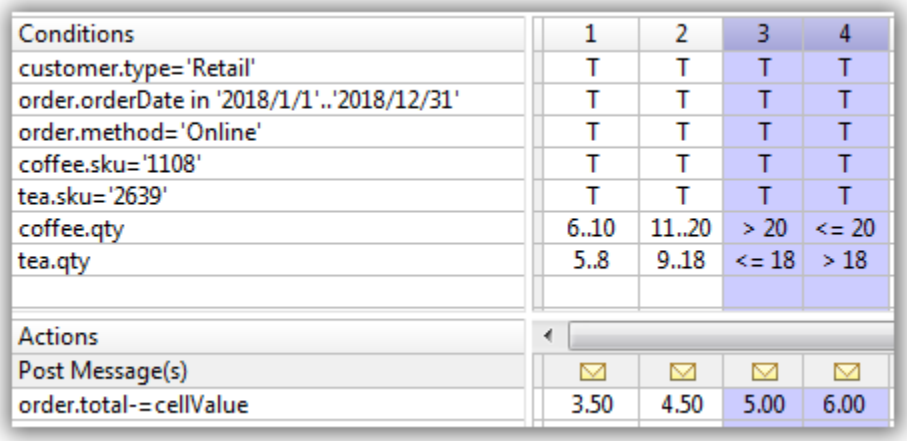
Also note that Corticon is able to find a potential mistake in our logic:



| Conditions | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| customer.type='Retail' | T | T | T | T |
| order.orderDate in '2018/1/1'..'2018/12/31' | T | T | T | T |
| order.method='Online' | T | T | T | T |
| coffee.sku='1108' | T | T | T | T |
| tea.sku='2639' | T | T | T | T |
| coffee.qty | 6..10 | 11..20 | > 20 | - |
| tea.qty | 5..8 | 9..18 | - | > 18 |
| | | | | |
| Actions | | | | |
| Post Message(s) | ✉ | ✉ | ✉ | ✉ |
| order.total-=cellValue | 3.50 | 4.50 | 5.00 | 6.00 |

Rules 3 and 4 are in conflict.

A customer with coffee.qty=21 and tea.qty=19 would get $11.00 discount. That's probably not what we want. This can be resolved as follows by making the rules more precise:



| Conditions | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| customer.type='Retail' | T | T | T | T |
| order.orderDate in '2018/1/1'..'2018/12/31' | T | T | T | T |
| order.method='Online' | T | T | T | T |
| coffee.sku='1108' | T | T | T | T |
| tea.sku='2639' | T | T | T | T |
| coffee.qty | 6..10 | 11..20 | > 20 | <= 20 |
| tea.qty | 5..8 | 9..18 | <= 18 | > 18 |
| | | | | |
| Actions | | | | |
| Post Message(s) | ✉ | ✉ | ✉ | ✉ |
| order.total-=cellValue | 3.50 | 4.50 | 5.00 | 6.00 |

## Summary

Dealing with collections of data and associations between these collections is very common in virtually all non-trivial business rules applications. Corticon manages this by using:

1. The **Scope** to define the context and relationships between objects
2. The **Filters** to define what data is relevant to the rules in a particular rule sheet

By using these features, Corticon is easily able to solve the "Order Promotion" problem. This gives the business user complete control and flexibility in defining eligibility for promotions and what discounts to apply no matter how complex they become.

It's the Scope and Filter sections that distinguish the Corticon decision table from all others and make it possible to solve even very complex problems in a straightforward manner.