

# Advanced N-way Constraints

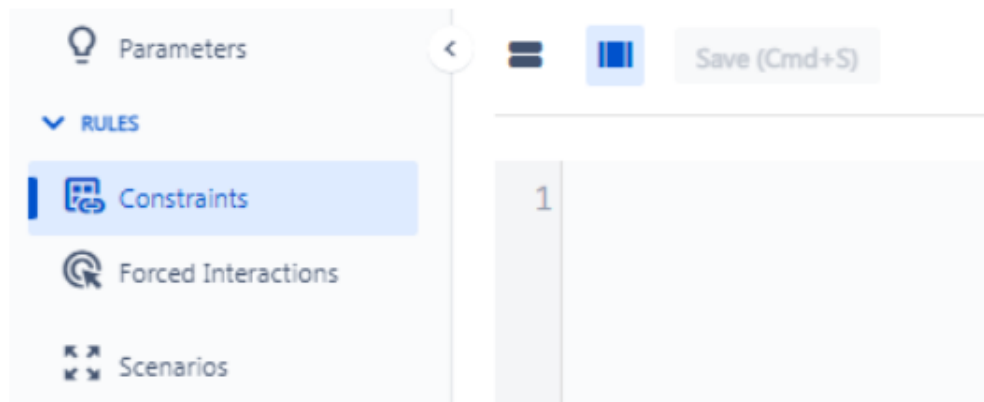
- [Overview](#)
- [Basics - New Functions](#)
  - [All\(\)](#)
  - [Some\(\)](#)
- [Advanced Rule Composition](#)
  - [Example 1](#)
  - [Example 2](#)

## Overview

One of our features is Constraints which can be used to exclude invalid or irrelevant combinations of parameter values from the Scenarios table. Advanced N-way constraints (where N can be 2 or higher) make it possible to create dependencies across any number of parameters which should be a significant time saver when defining complex rules and should expand your ability to apply our tool to highly conditional workflows.

The summary is below:

- The advanced N-way constraints can only be created in the bulk editor.



- One "Net New" grouping function - all().
- One "Efficiency Improvement" grouping functions - some().
- You can use these grouping functions on the left side of any constraint and on the right side of all but the >> skip constraint. The order of parameters within the function does not matter.
- You cannot use more than one grouping function per side of the constraint, but you can use two functions per rule (see "Advanced Rule Composition" section below).
- The number of involved parameters is not limited.
- Rules with these functions are not visually reflected in the Coverage Matrix or the Standard constraints view. I.e. the only artifacts for review are the bulk editor and the Scenarios table.

As a separate part of this update, we have added the ability to leave comments in the Constraints bulk editor using the # syntax. It is valid when used:

- as the first symbol on a line - to indicate that the whole line is a comment,



- after the valid constraint syntax - to indicate that the rest of the line is a comment.



## Basics - New Functions

*Note: we are using the same set of parameters throughout the article, but that does not mean rules would co-exist in the same model, at the same time. Treat each example as an isolated constraint.*

### All()

This is a “Net New” function because implementing this rule type is pretty much impossible via other methods. All() provides a **logical “AND” operation** on values from more than one parameter.

Let’s take a look at the example from an insurance quoting application:

all(VIP[No] Student[No] Tier[Bronze, Silver]) <-> Discount[No]

This rule says “WHEN Vip is No AND Student is No AND Tier is Bronze or Silver, THEN, and ONLY THEN (*note the mutually-bound constraint type*), Discount should be No.”

I.e., there will be only 2 scenarios in the table where the Discount is No:

VIP	Student	Tier	Discount
No	No	Bronze	No
No	No	Silver	No

When more than 1 value of a given parameter is included in the all() function (like with Tier above), **all combinations** of values will be treated as valid triggers for the rule (see more in the “Advanced” section below).

### Some()

This is an “Efficiency Improvement” function because even though implementing this rule type is possible via multiple lines without functions, it takes longer.

Some() is semantically equivalent to a **logical “OR” operation** on values from more than one parameter, for example:

some(VIP[Yes] Student[Yes] Tier[Gold, Platinum]) <-> Discount[Yes]

This rule says “WHEN Vip is Yes OR Student is Yes OR Tier is Gold or Platinum, THEN, and ONLY THEN (*note the mutually-bound constraint type*), Discount should be Yes.”

I.e., some of the possible scenarios (depending on the coverage strength) where the Discount is Yes are:

VIP	Student	Tier	Discount
Yes	No	Bronze	Yes
No	Yes	Silver	Yes
No	No	Gold	Yes
Yes	No	Platinum	Yes
Yes	Yes	Gold	Yes

## Advanced Rule Composition

As we have mentioned, two grouping functions can be used in the same rule, one on each side (except for the skip constraint, where the right side behaves the same way it did before this release).

In this section, we will cover some of the trickier function combinations. Overall, mutually bound constraints with functions on both sides are the most difficult to get right, so we recommend starting with invalid and bound relationships to get some practice.

### Example 1

**all**(VIP[No] Student[No, N/A] Tier[Bronze, Silver]) **!= some**(Discount[Yes] Reserved Seat[Yes])

This rule says “WHEN Vip is No AND Student is No or N/A AND Tier is Bronze or Silver, THEN Discount cannot be Yes OR Reserved Seat cannot be Yes”

The following scenarios will be **excluded** from the generated table as invalid:

VIP	Student	Tier	Discount	Reserved Seat
No	No	Bronze	Yes	Yes
No	No	Bronze	Yes	No
No	No	Bronze	No	Yes
No	N/A	Bronze	Yes	Yes
No	N/A	Bronze	Yes	No
No	N/A	Bronze	No	Yes
No	No	Silver	Yes	Yes
No	No	Silver	Yes	No
No	No	Silver	No	Yes
No	N/A	Silver	Yes	Yes
No	N/A	Silver	Yes	No
No	N/A	Silver	No	Yes

Only the Discount = No AND Reserved Seat = No is the valid combination for the trigger in all(). Note that some(Discount[Yes] Reserved Seat[Yes]) still includes the Yes/Yes combination.

### Example 2

**some**(Coverage B[125k, 175k] Coverage C[100k]) -> **all**(Coverage A[200k] Coverage D[No])

This rule says “WHEN Coverage B is 125k or 175k OR Coverage C is 100k, THEN Coverage A should be 200k AND Coverage D should be No”

The following scenarios will be **excluded** from the generated table as invalid (non-exhaustive list):

Coverage B	Coverage C	Coverage A	Coverage D
------------	------------	------------	------------

125k	100k	250k (i.e. NOT[200k])	No
175k	50k	200k	Yes
225k	100k	250k (i.e. NOT[200k])	Yes

*Note: we changed the order of columns for easier review*