blueriq
TSS
MAKE YOUR OWN RULES

Collection of Cars

By Maarten P.D. Schadd
Product Consultant at Blueriq B.V.

Contents

1 Introduction

The decision management community [1] is a new initiative that started in 2014 to facilitate the sharing of news and knowledge concerning Decision Management (DM). Next to a product catalog, decision model prototypes and case studies, the decision management community also provides a monthly challenge. Every challenge consists of a problem that should be solved using any business rules and decisions management system or none at all.

As Blueriq is a vendor with an integrated rule engine and decision management capabilities in its BPM suite, we accept this challenge. This article describes how Blueriq solves the September 2015 challenge.

2 Problem definition

This challenge concerns a domain in which there are cars that are part of a collection. Figure 1 shows the example set that is provided in the challenge and used in this article.

Group	Make	Model
G1	Ford	Pickup
G1	Ford	T
G1	Ford	Taurus
G1	Austin	Mini
G1	Hyundai	Santa Fe
G2	Ford	Pickup
G2	Ford	T
G2	Ford	Taurus
G2	Hyundai	Tucson
G2	Hyundai	Santa Fe

Difference	
Make	Model
Hyundai	Tucson
Austin	Mini

Intersection	
Make	Model
Ford	Pickup
Ford	T
Ford	Taurus
Hyundai	Santa Fe

Figure 1: Car test data.

Blueriq should answer the following questions for any number of collections:

1. Find the set of cars that is common to all collections.
2. Identify cars that do not exist in any other collection.
3. For each collection of cars eliminate all but the most expensive instance of each make and model (add another attribute for price).

We define that a car matches the last criterion if it is the most expensive car in at least one of the collections it belongs to. A detailed description can be found at [1].

3 Domain

We create a rather simple domain, as shown in Figure 2. We choose to make a Collection a separate entity, that has a many-to-many collection with Car. Like this, one instance of Car can be part of many collections at the same time. All attributes are self-explanatory. The only attribute which might be surprising is the MaxPrice attribute of Collection. This attribute could be removed as well, but is used as intermediate result for answering the last question, and makes the logic easier to read.

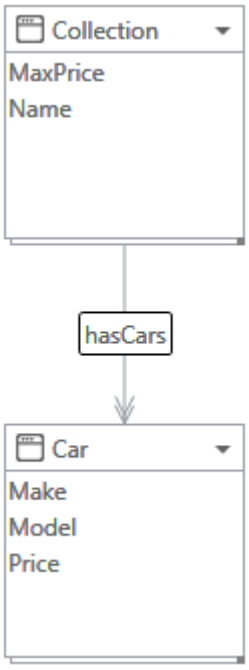

Figure 2: The Entity Relationship Diagram

We model the given dataset as static instances. That means that these instances are created when the application is started, and all information for their attributes and relations is set in the model. We could extend this to read data from a CSV file, or create screens so that the user can fill in the information. As the current challenge is about logic for finding the cars, static instances are appropriate.

4 Answering the questions

This sections answers the three questions in the next subsections. The solutions are general, and not limited to only two collections.

4.1 Question 1: Cars in all collections

In order to calculate an answer, we want to write an expression that returns a set of Car instances. A car should be part of the result set if it is present in each Collection. When

creating a relation in Blueriq, the backwards relation is created as well. This is not optional, so each relation always has a backwards relation. The backwards relation of hasCars from Figure 2 is not visible in the figure. It is called belongsToCollection in our model. We can make use of this backwards relation. The following expression provides an answer to this question:

```
COLLECT Car
FROM ALL CAR
WHERE ( SIZE( Car.belongsToCollection ) = SIZE (ALL Collection) )
```

This is one expression that is split on multiple lines for readability. The first line indicates that we are looking for instance of type Car. The second line states that all cars currently known in the system should be checked. Only those for which the WHERE clause holds true are returned, which is shown in the last line. The trick of the WHERE clause is to use SIZE(Car.belongsToCollection), which counts how many collections a car belongs to. If a car belongs to 2 collections, then the backwards relation is filled with two instance identifiers of Collection instances. The COUNT statement counts instances in a set, and in this case, the relation. We also can count how many collections we have in total in the system with SIZE (ALL Collection). When both these numbers are equal, then the car is part of each collection and should be part of the result set. The results of evaluating this expression at runtime are shown in Figure 3.

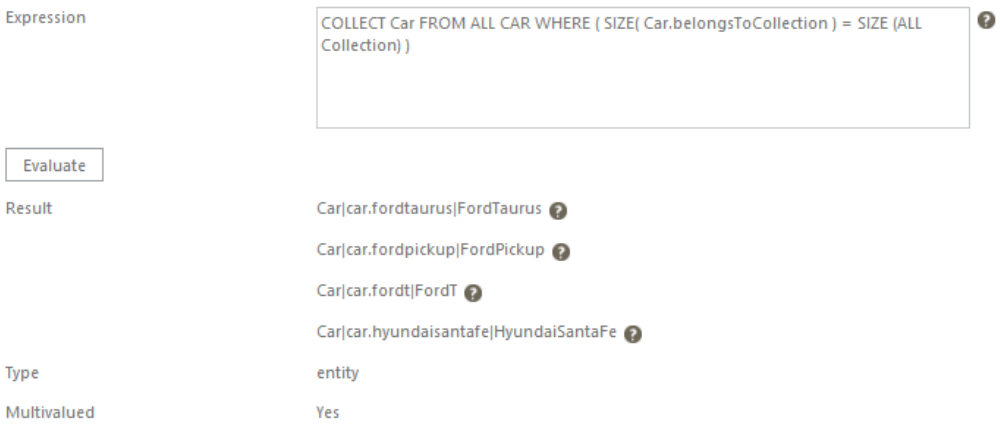

Figure 3: Cars that are part of each collection.

4.2 Question 2: Cars in one collection

The expression used for finding cars that only belong to one collection is rather similar than for find cars that are in each collection. The following expression counts the number of collections a car is part of using the backwards relation, and that number should be one.

```
COLLECT Car
FROM ALL CAR
WHERE ( SIZE( Car.belongsToCollection ) = 1 )
```

The result of this expression are shown in Figure 4.

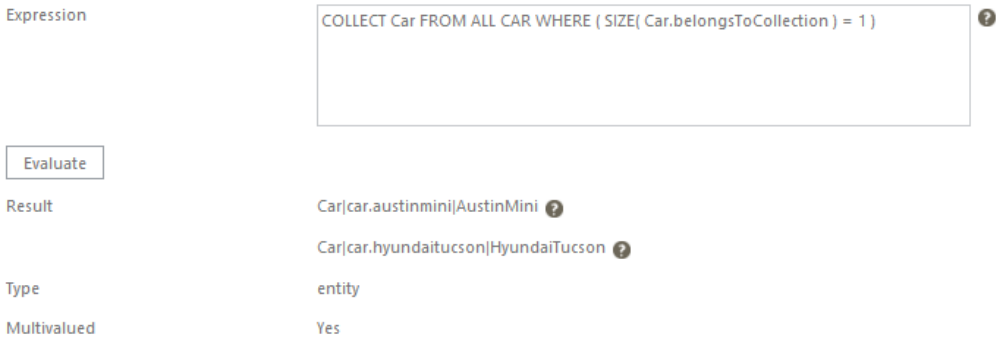

Figure 4: Cars that are part of one collection.

4.3 Question 3: Cars with highest price in a collection

This question is somewhat more difficult to answer. For this, a supporting attribute is created for the Collection entity. This attribute holds the information what the maximum price is for all cars within this collection. The attribute has this default expression:

```
MAX (
COLLECT Car.Price
FROM Collection.hasCars)
```

With the COLLECT statement all prices of cars are retrieved that belong to the corresponding collection, and the MAX function takes the maximum. Now that we know what the maximum price is for each collection, we can find the cars that have the exactly that price with the following expression:

```
COLLECT Car
FROM ALL Car
WHERE ( Car.Price = Car.belongsToCollection.MaxPrice )
```

There are two interesting things happening in this expression. The first is concerning Car.belongsToCollection.MaxPrice. As this backwards relation is multivalued (a car can be part of multiple collections) this expression can result in more than one value. It may return a list of values, the maximum prices of all collections the car is part of. The second interesting thing is the use of the equality sign (=). The runtime now compares the single value Car.Price to the possibly multiple values returned by Car.belongsToCollection.MaxPrice. In this case the runtime checks if the single value is present in the list, and if so returns true. This behavior is exactly what we want in this case.[1]

As for this last question no example data is given, the chosen prices are shown in Table 1 and the results of the complete expression is shown in Figure 5. We chose the prices in a way so that one set has a tie for the maximum price. The Hyundai Tucson is returned as it is the most expensive car of collection 2, and both the Ford Taurus and the Hyundai Santa Fe are returned as they are drawn for most expensive car in Collection one.

[1] There are other constructs available in the expression language to explicitly use set operations.

Car	Price
Austin Mini	10000
Ford Pickup	12000
Ford T	12500
Ford Taurus	13000
Hyundai SantaFe	13000
Hyundai Tucson	15000

Table 1: Example Car Prices.

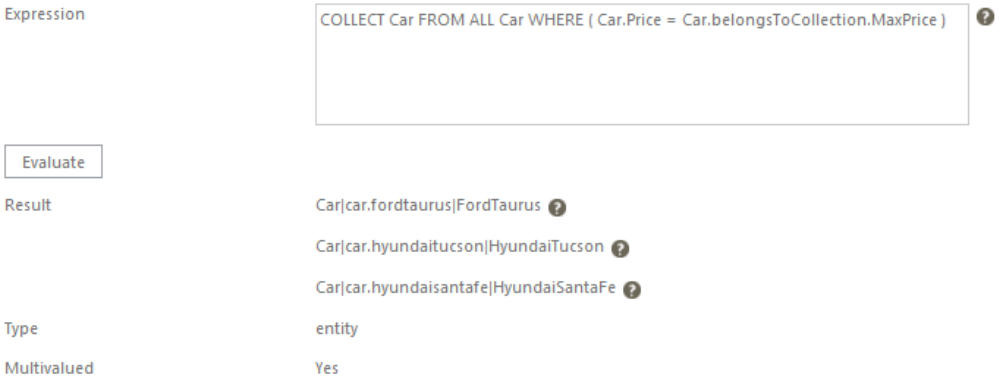

Figure 5: Cars that are most expensive in a collection.

5 Conclusions

Blueriq is able to answer all given questions using the built in expression language. Expressions like these are covered in our Basic Modeling Foundation course, which is the first training new users of our modeling environment receive when they start to learn our product. This expression language has enough power to retrieve all information in a general way without restriction it to two collections, while still being readable and understandable by business users.

Contact Us

If you have any questions about this article or if you would like to start a discussion, do not hesitate to contact us.

Email the author	:	m.schadd@blueriq.com
Email Blueriq	:	info@blueriq.com
Call Blueriq	:	+31 (0)73 6450467
Website Blueriq	:	http://www.blueriq.com

Blueriq BV
Veemarktkade 8
5222 AE s-Hertogenbosch
The Netherlands

About Blueriq

Blueriq is a rule-driven software platform designed to deliver dynamic business solutions for organizations with knowledge-intensive processes. It empowers organizations in fast changing environments to quickly and cost-effectively respond to changing business conditions and regulations. Blueriq provides solutions for Decision Management, Dynamic Case Management and intelligent User Experience Management across multiple channels. Solutions based on Blueriq are modelled, not programmed, giving you the opportunity to respond more quickly to your customers needs and improving your business outcomes. With Blueriq, you make your own rules!

References

[1] Decision Management Community. https://dmcommunity.wordpress.com/home/, 2014.