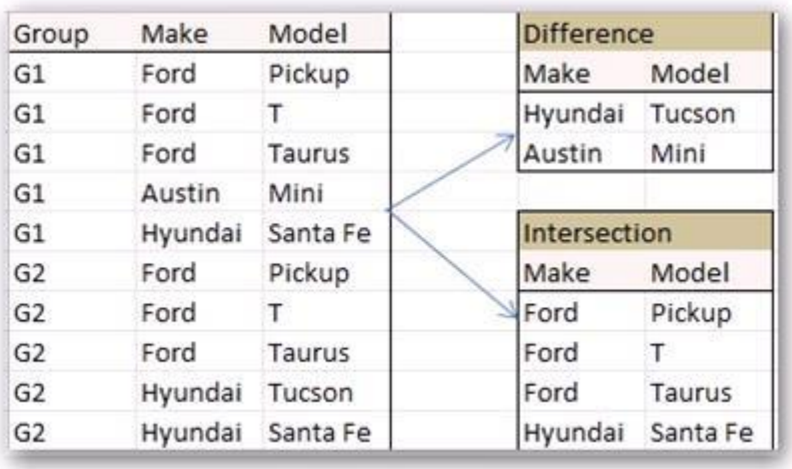
Challenge:



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

Solution Concepts:

There are many ways of tackling this challenge. For instance:

1. Using Decision Table and Counting base on condition (Decision table approach)
2. Using Monadic expression (Advanced expression approach)

All of the mentioned solutions have one simple concept. The idea is we need to get a car one by one from the list (let's name it thisCar) and iterate on the same list (let's call items on this list otherCar). Note thisCar and otherCar both coming from the same list. Now, we count the number of times thisCar matches the condition (Model and Make of thisCar is equal to Model and Make of otherCar). If this number (counting number) is 1, then thisCar is only in one group, if the number is 2, the thisCar appears in two groups and so on…

- For each car in list (name it thisCar)
  - Count numbers of times that thisCar in list *(let's call this number n)*
    - THAT thisCar.Model and thisCar.Make is found (distinctively)

If n is now 1 this thisCar is in only one group and if it is 2, then it is found in 2 groups.

This type of questions and data manipulation are easy by using expressions and relations. So I would not go to implementing this in the Decision Table with iterating on the collections and increasing values related to each group. Although we can use Decision table for answer this question, I found expressions and relations are more intuitive for this type of challenges.

info@pliantframework.com
www.pliantframework.com
+61-399883994
P.O. Box 1024, Mitcham North, VIC 3132, Australia

Intersections:

I don't want to write just 2 lines of expression to answer the question, I want to explain it so it becomes more clear and easier to write the next expressions. FlexRule supports a powerful expression language, which has ability to deal with collections as well as single objects.

The operator that deals with collections in FlexRule has monadic format. What that means is you can pass a result of one operator to next operator. In other word, you can chain them together.

In the explained solution concept, we do have the number "n" equals to two. "2" is not just a magical number, it is because in the sample data we have only 2 groups. In a bit later I show you how to generalize the solution to answer any numbers of groups. So for now, we filter the cars base on the count equals to 2.

```
cars
    |filter (thisCar, COUNTING_WITH_CONDITION == 2)
    |distinct (car, car.Make, car.Model)
```

What the above expression means, is that, we pass a list (cars) to filter operator, and then we pass the result of that to distinct operator. (Simple chaining!)

Now let's see what can be the COUNTING_WITH_CONDITION. Now we need to again filter the cars with some condition and then count the numbers of items in the filtered list. So we will have a 'filter' followed by a 'count' operator.

```
cars
    |filter(otherCar, MATCH_CONDITION_FOR_THIS_AND_OTHER_CAR)
    |count()
```

Alternatively, we can simply re-write the expression like bellow.

```
cars
    |count(otherCar, MATCH_CONDITION_FOR_THIS_AND_OTHER_CAR)
```

(Combining filter-count to count operator)

And now let's see what the matching condition MATCH_CONDITION_FOR_THIS_AND_OTHER_CAR is? We want to compare Model and Make of thisCar to the otherCar, so:

```
(otherCar.Make==thisCar.Make) and (otherCar.Model==thisCar.Model)
```

Now let's put all of them together:

```
Intersections
cars
   |filter (thisCar,
               cars |count(otherCar,
                   (otherCar.Make==thisCar.Make) and (otherCar.Model==thisCar.Model)
               )
          == 2)
   |distinct (car, car.Make, car.Model)
```

Differences:

Almost very identical to the expression we built above, we just need to replace the value 2 to 1, so it counts number of repeating items in each collection but we filter when the count is 1. Which means there is one item in each group.

```
Differences
cars
   |filter (thisCar,
               cars |count(otherCar,
                   (otherCar.Make==thisCar.Make) and (otherCar.Model==thisCar.Model)
               )
          == 1)
   |distinct (car, car.Make, car.Model)
```

Now you know where this expression comes from.

Generic Solution

To support for any numbers of group, as you can guess this value of counts in previous section (1 for differences and 2 for intersection) are the key to answer this part.

That value should be the numbers of groups (distinctively). So we can replace it with bellow expression, instead of fix value 2 in first scenario (intersections).

```
cars
   |distinct(car, car.Group)
   | count()
```

So to write the expression that supports any numbers of groups, we can write the bellow expression:

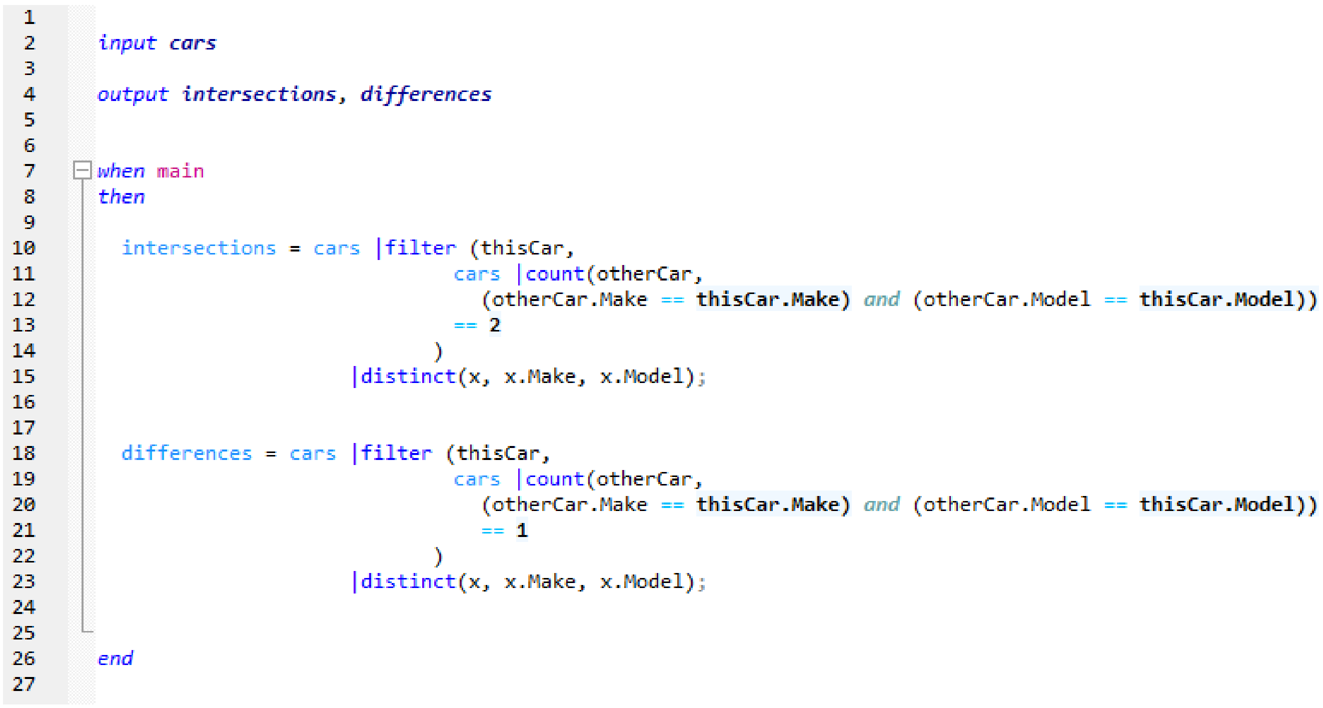
Intersection for Any Numbers of Groups

```
cars
   |filter (thisCar,
               cars |count(otherCar,
                   (otherCar.Make==thisCar.Make) and (otherCar.Model==thisCar.Model)
               )
          == cars
               |distinct(car, car.Group)
               | count()
   )
   |distinct (car, car.Make, car.Model)
```

In Action:

Now we just use FlexRule Designer to simulate and test this rule.

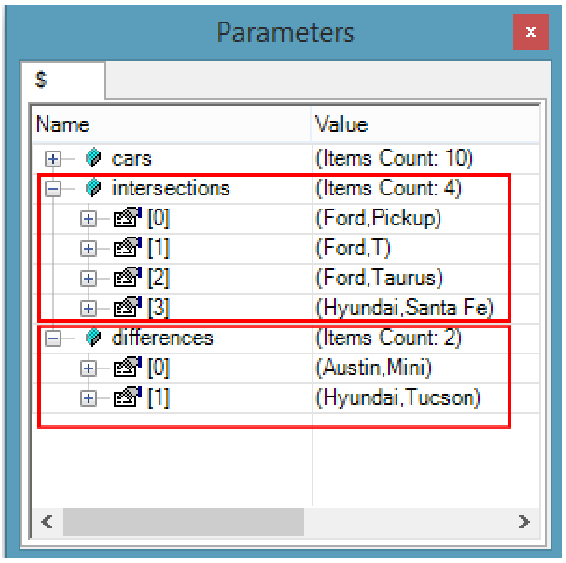
We have already discussed expressions you see bellow:



```

input cars

output intersections, differences


when main
then

  intersections = cars |filter (thisCar,
                             cars |count(otherCar,
                               (otherCar.Make == thisCar.Make) and (otherCar.Model == thisCar.Model))
                             == 2
                           )
                      |distinct(x, x.Make, x.Model);


  differences = cars |filter (thisCar,
                             cars |count(otherCar,
                               (otherCar.Make == thisCar.Make) and (otherCar.Model == thisCar.Model))
                               == 1
                           )
                      |distinct(x, x.Make, x.Model);


end

```

When you use FlexRule Debugger, then you can see the values and the result.

And as the result of execution:


Parameters

$

Name	Value
cars	(Items Count: 10)
intersections	(Items Count: 4)
[0]	(Ford,Pickup)
[1]	(Ford,T)
[2]	(Ford,Taurus)
[3]	(Hyundai,Santa Fe)
differences	(Items Count: 2)
[0]	(Austin,Mini)
[1]	(Hyundai,Tucson)

And if we want to support any number of groups, we can change the first expression to:

```
intersections = cars |filter (thisCar,
                             cars |count(otherCar,
                               (otherCar.Make == thisCar.Make) and (otherCar.Model == thisCar.Model))
                             ==
                               cars|distinct(g, g.Name)|count()
                            )
                    |distinct(x, x.Make, x.Model);
```

To learn more about monadic expressions, please have a look at http://wiki.flexrule.com/index.php?title=Expression/Pipes

For More Information Please Visit

http://www.flexrule.com/

FlexRule

Flexible Software Made Easy

info@pliantframework.com
www.pliantframework.com
+61-399883994
P.O. Box 1024, Mitcham North, VIC 3132, Australia