BPMN: Database Operations with OData

BPMN's advocates – myself included – like to proclaim that the language allows non-programmers to define executable processes themselves. But that's only one-third true. Yes, in BPMN 2.0 the executable process steps through the shapes of the diagram as drawn by the modeler. That in itself was a monumental achievement a decade ago. "What you model is what you execute," we liked to say. But it left out two key ingredients that still required programming. First, the process data, mappings, and expression language were assumed to be based on Java or some similar programming language. So while the BPMN diagram describes the paths the process may follow, the logic controlling which path is followed normally requires a programmer. And, of course, the actions provided by the process tasks themselves must be created by programmers. A decade ago, in the heyday of SOA, vendors liked to pretend that all the actions you might want to use in your process have already been nicely programmed and wrapped in a service API, just waiting to be orchestrated by BPMN. Really?

Now you may have noticed Trisotech as begun marketing "business automation as a service," executable process apps combining BPMN, DMN, and CMMN that can be created entirely by non-programmers, and has been quietly building out the platform that turns that promise into reality. To solve the problem of data modeling, mapping, and expressions they borrowed FEEL and boxed expressions from DMN. Brilliant, in my opinion! But that still leaves the issue of the tasks. How can non-programmers create those? For that, Trisotech currently offers the following options:

- A decision task can invoke a DMN decision service created by a non-programmer.
- A service task can invoke an existing REST service via its OpenAPI specification. In today's microservices era, this is becoming more widely available.
- Microsoft Office and a few other apps can be invoked via a Trisotech Connector, although the Connector Library is still beta and not as well developed as it will be.

What's missing in all this is a solution for basic CRUD operations: database record Create, Read, Update, and Delete. Actually, that's available as well, though a standard called OData. I just never knew how to use it. This post will show you how.

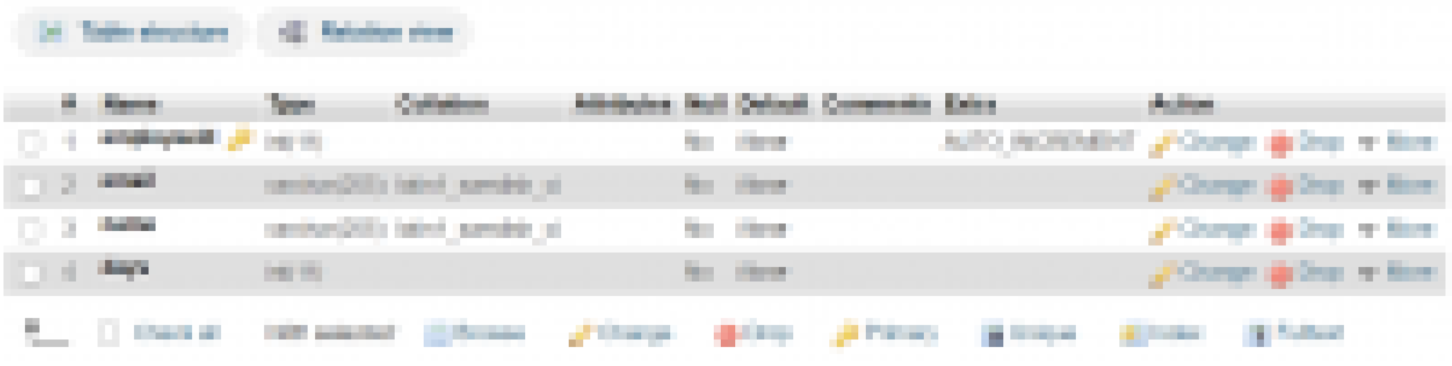
Databases are, as a rule, not cloud-friendly. Each DBMS provider has its own proprietary protocols and APIs. OData is a standard that provides a common abstraction layer for databases, exposing CRUD operations as cloud-friendly REST services. Trisotech has written a white paper on this, which I recommend and will summarize briefly here.

The solution assumes you have some DBMS on a website: SQL Server, Oracle, mySQL, whatever. Between Trisotech and that DBMS is a *data gateway*, a service that "virtualizes" the DBMS using OData. After you point the gateway to your database, it exposes a *Metadata XML* file that you download into Trisotech Workflow Modeler or Decision Modeler. This file defines the data structures used in your database, which Trisotech uses to generate the required FEEL datatypes and service interface. This makes it possible for non-programmers to define CRUD operations through basic boxed expressions. At runtime, deployed Trisotech services call the data gateway using the OData protocol to execute the operations.

Sounds easy, right? Well, first you have to provide the data gateway. You can use free open source code from Teiid (http://teiid.io/), or Red Hat JBoss Data Virtualization, a commercial product, or Skyvia, a commercial cloud service. The white paper provides a configuration example for Teiid to get started. The rest of this post assumes you've got that in place.

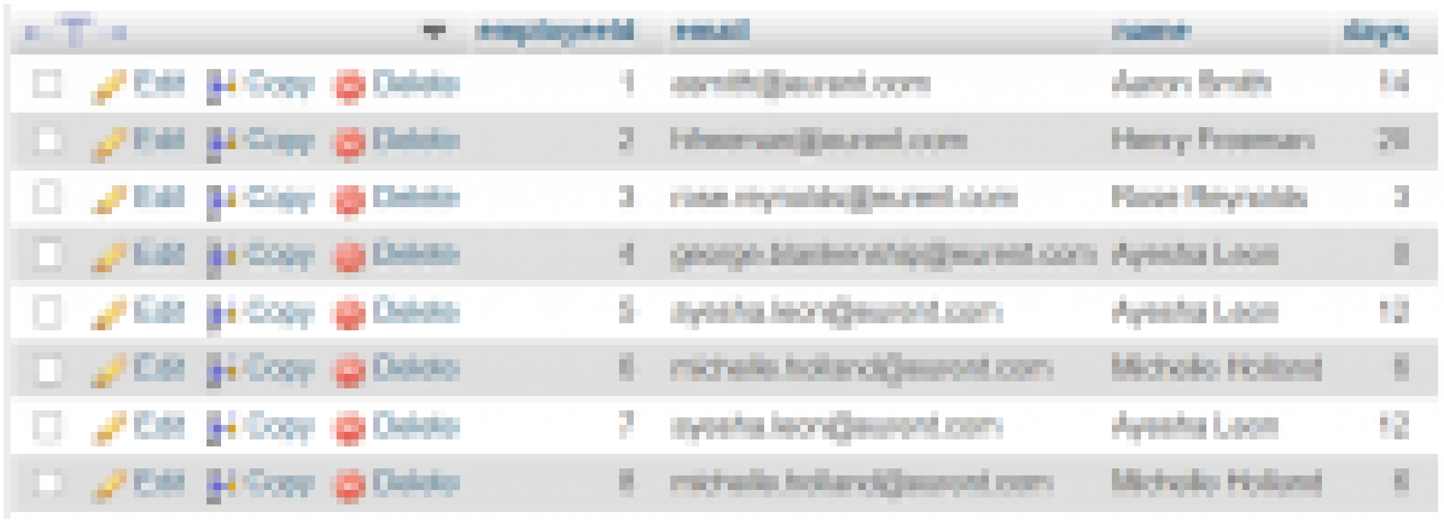
In your Trisotech workspace you have a repository of example models called EU-Rent. The example I'm going to show you is based on the Vacation Request process there. The scenario is you have a database table of employees and their accrued vacation days. The employee submits a vacation request, specifying the vacation start and return dates. Some decision logic controls whether to automatically approve, refuse, or manually adjudicate the request. And if the request is approved, it updates the remaining available vacation days in the database. That's pretty straightforward. So let's see how a non-programmer like me can implement this.

We start with the database. I've got a simple mySQL table *vacation* on a website. The structure is created using phpMyAdmin:



#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	employeeId	int(11)			No	None		AUTO_INCREMENT	Change Drop More
2	email	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
3	name	varchar(255)	latin1_swedish_ci		No	None			Change Drop More
4	days	int(11)			No	None			Change Drop More

For OData to work, you need a primary key field, which we have here with *employeeId*. It's common to have the primary key field defined to autoincrement as records are added. phpMyAdmin lets you populate the tables from a CSV file, so let's add a few records:



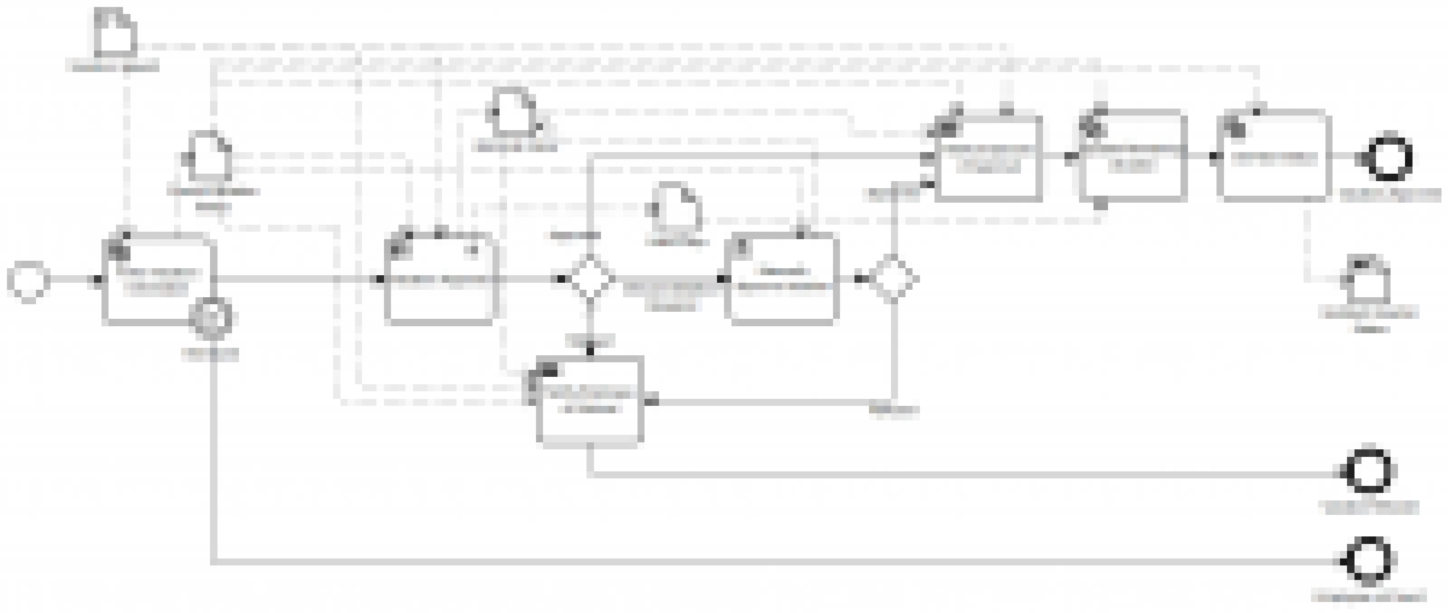
	employeeId	email	name	days
Edit Copy Delete	1	asmith@aurent.com	Aaron Smith	14
Edit Copy Delete	2	hfreeman@aurent.com	Henry Freeman	28
Edit Copy Delete	3	rose.reynolds@aurent.com	Rose Reynolds	3
Edit Copy Delete	4	george.blankenship@aurent.com	Ayesha Leon	8
Edit Copy Delete	5	ayesha.leon@aurent.com	Ayesha Leon	12
Edit Copy Delete	6	michelle.holland@aurent.com	Michelle Holland	6
Edit Copy Delete	7	ayesha.leon@aurent.com	Ayesha Leon	12
Edit Copy Delete	8	michelle.holland@aurent.com	Michelle Holland	6

Now the data gateway provides a downloadable Metadata XML file that looks like this:

[illegible]

Save that on your PC. You'll need it to configure the service task that accesses the database.

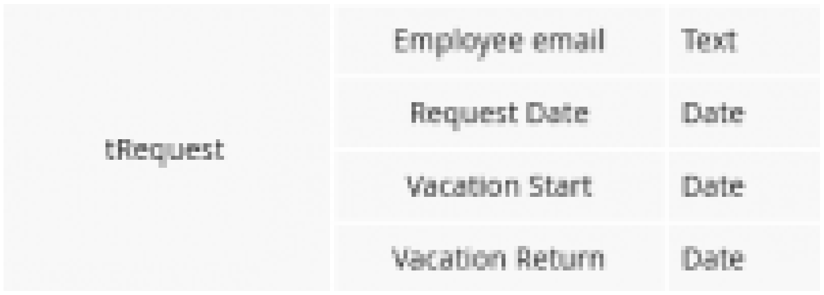
Here is the process we want to make executable. The main difference from normal non-executable BPMN diagrams is all the data objects, data inputs, and data outputs – the dog-eared page shapes representing process variables – and the dotted arrow data associations mapping them to the task inputs and outputs.



Our *vacation* table isn't represented in the diagram, although you could add a datastore as a kind of annotation to represent it. We have a data input *Vacation request*, specifying the employee email, vacation start and return dates. And we have a data output, *Updated Vacation Status*, that returns the new employee record after updating the vacation days. When you publish this process as a service, only the data outputs and process end states are returned on execution. The regular data objects are not.

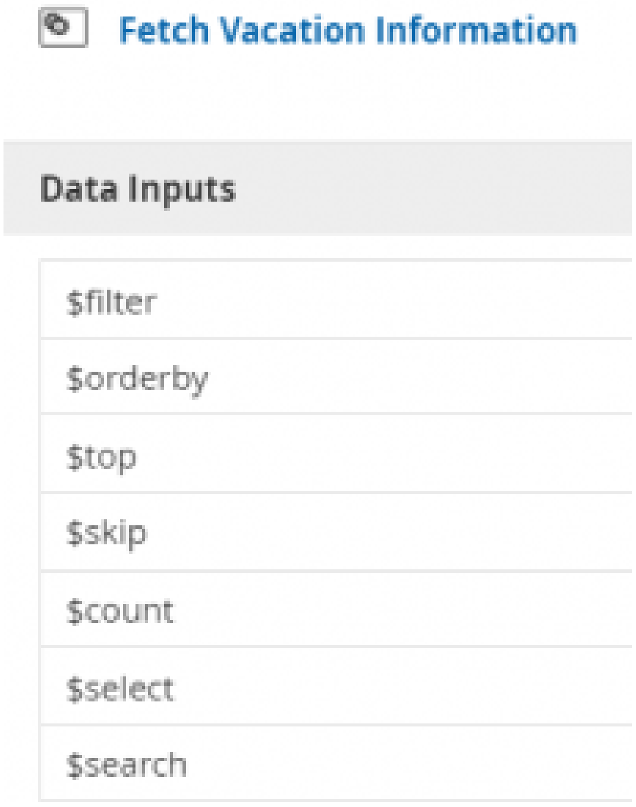
First we need to import the OData service for vacation into our model. In the BPMN ribbon's Operation Library, click on Import OData and upload the *Metadata XML* file you saved. Select the table or tables you want imported, and magically Trisotech exposes a REST interface containing several standard operations: Find, Get, Create, Update, Delete, etc. It also creates FEEL datatypes for the table as a whole, a table record, and a query input structure.

The data input Vacation request looks like this:



tRequest	Employee email	Text
	Request Date	Date
	Vacation Start	Date
	Vacation Return	Date

The first service task, *Fetch Vacation Information*, queries our *vacation* table using the *employee email* from *Vacation request* to obtain the employee's accrued vacation days. Right-click Attributes/Service task and select the interface for the table and the operation for this task, in this case *Find*, meaning a query of this table. Right-click Attributes/Data inputs to see the parameters of the Find operation:



Fetch Vacation Information

Data Inputs

- $filter
- $orderby
- $top
- $skip
- $count
- $select
- $search

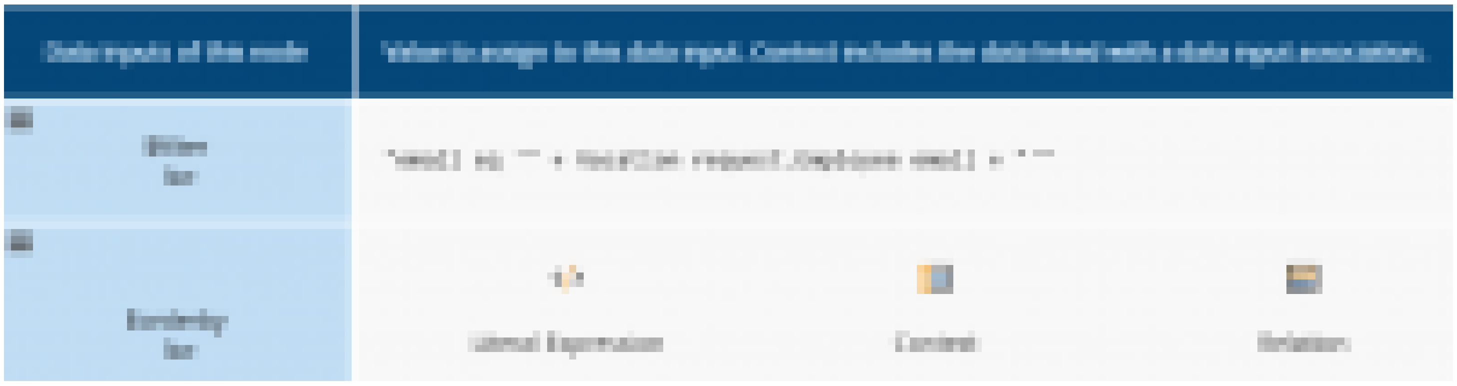
It wasn't obvious to me what to make of this. You need to consult the OData documentation, which explains how these parameters are used to construct the REST query URI. We want the $filter parameter, which is used in a URI like this:

```
<myDB>/vacation?filter=<match condition string>
```

The Data Input Mapping screen provides a boxed expression for all of the parameters. What you need to do is, for the parameters you use, create a string that goes to the right of the = in the URI. What was confusing to me at first is that this string is a FEEL expression that returns a boolean condition in OData (not FEEL) syntax. For example, if Aaron Smith is issuing the vacation request, we want the Find operation to use the URI

```
<myDB>/vacation?filter="email eq 'asmith@eurent.com'"
```

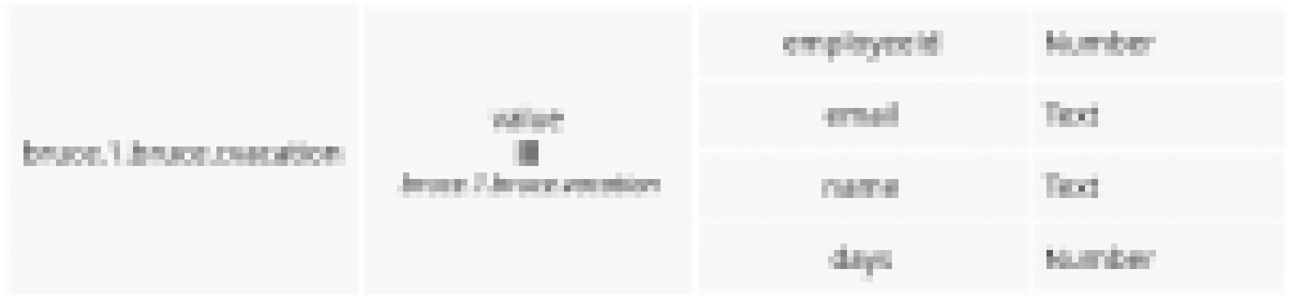
In that case, the Data Input Mapping for *Fetch Vacation Information* looks like this:



Data inputs of this node	Value to assign to this data input. Context includes the data linked with a data input association.		
$filter Text	"email eq '" + Vacation request.Employee email + "'"		
$orderby Text	Literal Expression	Context	Relation

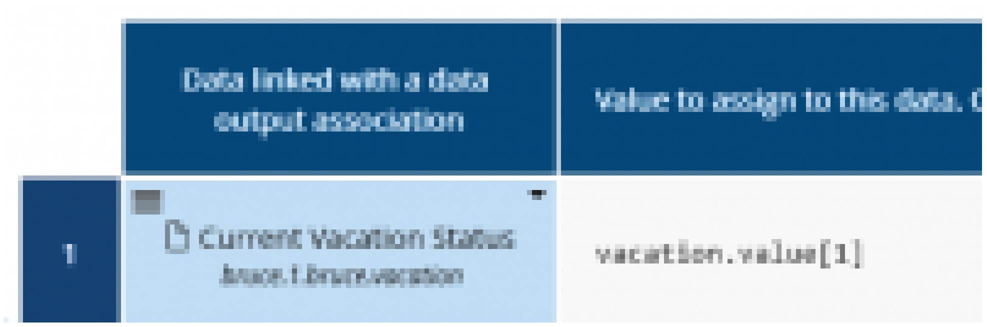
Note that the FEEL expression has to include the single quotes to wrap the value returned by *Vacation request.Employee email.* We don't use parameters besides $filter, so their mapping expressions are blank.

If we right-click Data Outputs, we see the OData output is called *vacation*, the name of our mySQL table, converted to the FEEL type shown below:



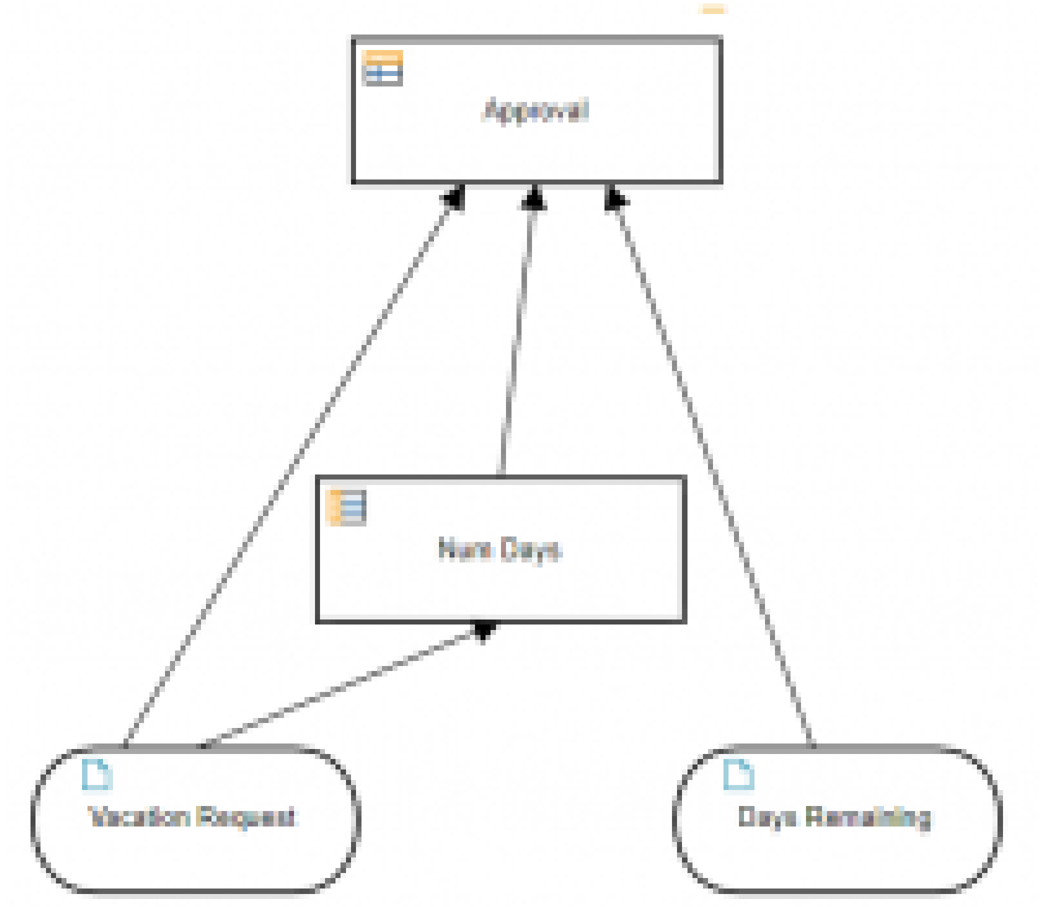
bruce.1.bruce.cvacation	value bruce.1.bruce.vacation	employeeid	Number
		email	Text
		name	Text
		days	Number

The output is not the whole table, just records matching our filter query, which should be just one. We save this record in the data object *Current Vacation Status* using the Data Output Mapping shown below. Because *vacation.value* is a list, we need [1] to extract the item.



	Data linked with a data output association	Value to assign to this data. C
1	Current Vacation Status bruce.1.bruce.vacation	vacation.value[1]

The decision task *Vacation Approval* invokes a DMN decision service. You need to create the decision service first in Decision Modeler and then link to it from the BPMN decision task. The DRD is shown below:

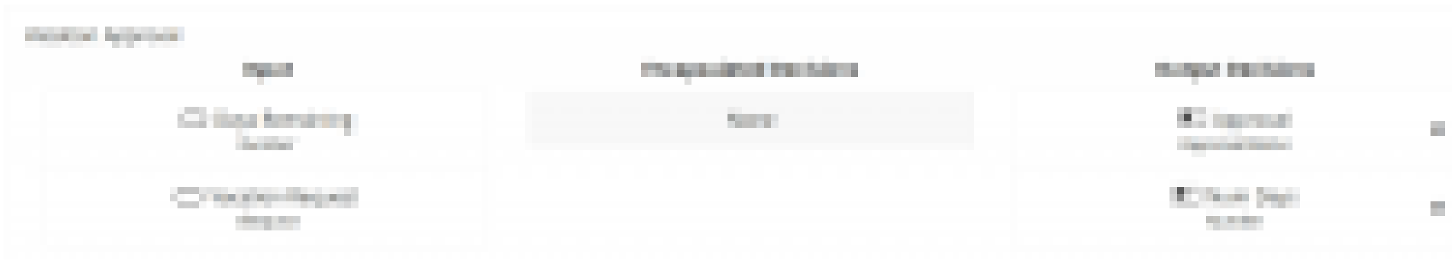


It takes the *Vacation Request* in combination with the *Days Remaining* from *Current Vacation Status* to calculate the number of vacation days requested and then either automatically approving, rejecting, or referring the request for a human decision. *Num Days* is a context that determines the number of vacation days requested based on the start and return dates. Most of the logic is figuring out the count of weekend days in that span.

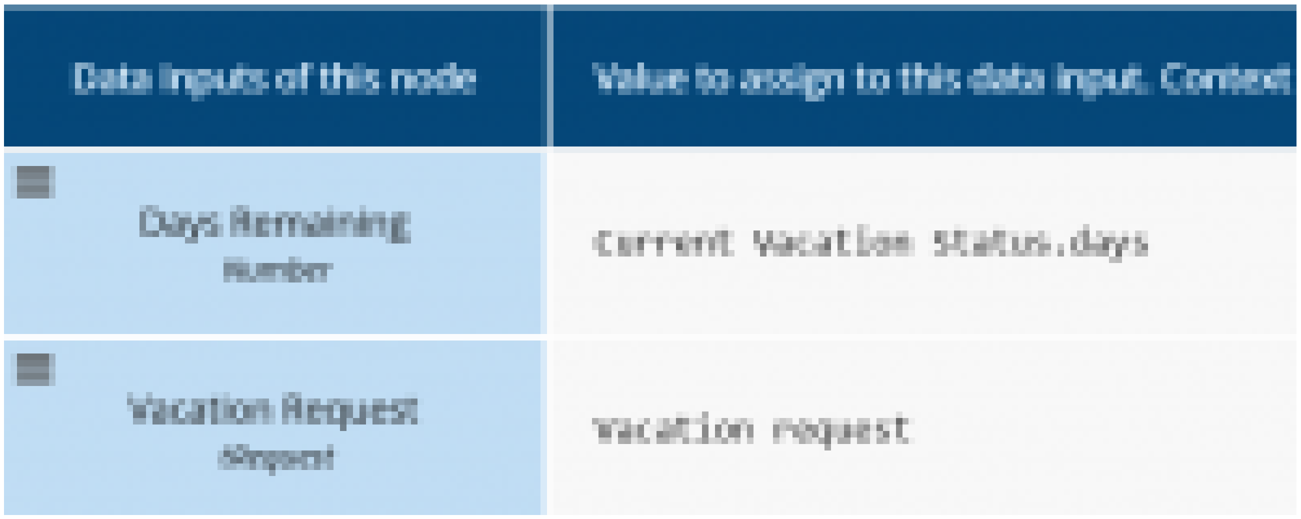
Approval returns a structure with components *Status* and *Reason*, the latter used in an email to the employee in case the request is refused. The decision table also returns “Refused” in the case of invalid values, such as a return date not later than start date, or either start or return date on a weekend. Readers interested in the calendar arithmetic used in these tables are referred to my DMN Method and Style book or training.

[illegible]

By default, Trisotech generates a decision service for the whole DMN model, but this returns only top-level decisions. We want both *Approval* and *Num Days*returned, so we define a decision service *Vacation Approval* that includes both as output decisions:


[illegible]

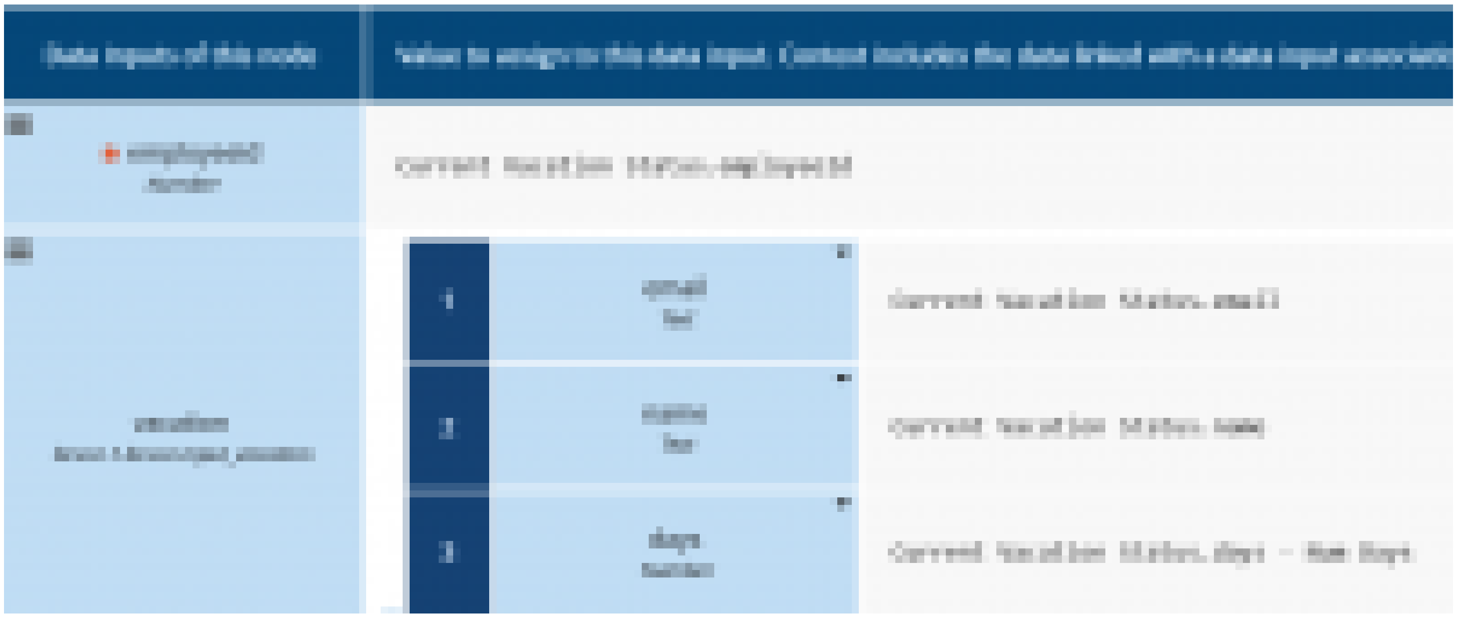
Back in BPMN, if we click on the task type icon for our decision task, we navigate our repository and select a decision service to invoke. When we select *Vacation Approval,* note the decision task reuses it by reference, denoted by the closed lock icon. If we modify the decision model, the decision task automatically uses the updated decision logic. This decision service has two inputs, Days Remaining and Vacation Request. The process data available to the decision task are indicated by the incoming data associations from data objects *Current Vacation Status* and *Vacation Request*. The Data Input Mapping looks like this:



Data Inputs of this node	Value to assign to this data input. Context
Days Remaining Number	Current Vacation Status.days
Vacation Request tRequest	Vacation Request

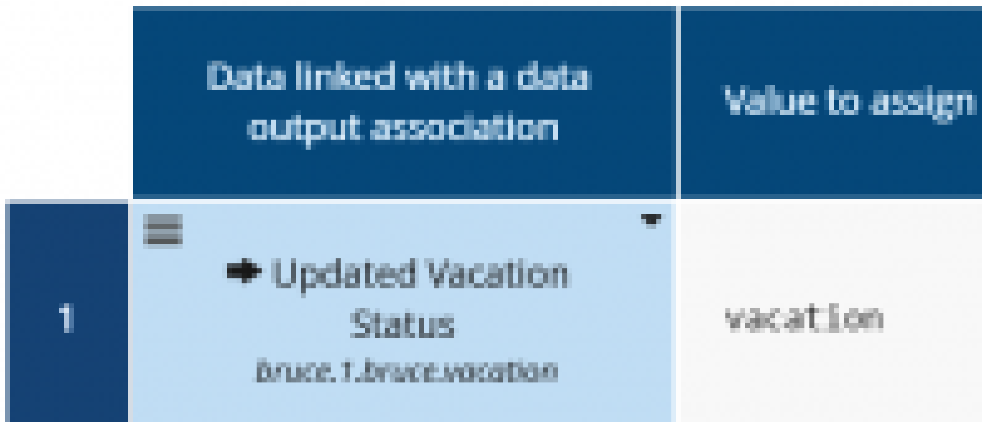
Because our decision service contains two output decisions, the decision task has two data outputs. We map *Approval* to the data object *Approval Status* and *Num Days* to the data object *Num Days*. The gateway following the decision task tests the value of *Approval.Status*, entered as the Condition attribute on each gate. The Send tasks in the model use a Trisotech system service to send an email, which incorporates the *Reason* value of *Approval Status*.

In the case where the vacation request is approved, we next need to update the *vacation* table. The service task *Update Remaining Vacation* uses the same OData interface we used before, this time with the *Update vacation* operation. The inputs to an update are the primary key of the database, which we call *employeeId*, and other columns of the record. The only change is we need to subtract *Num Days* from the original days remaining. The Data Input Mapping is shown below:



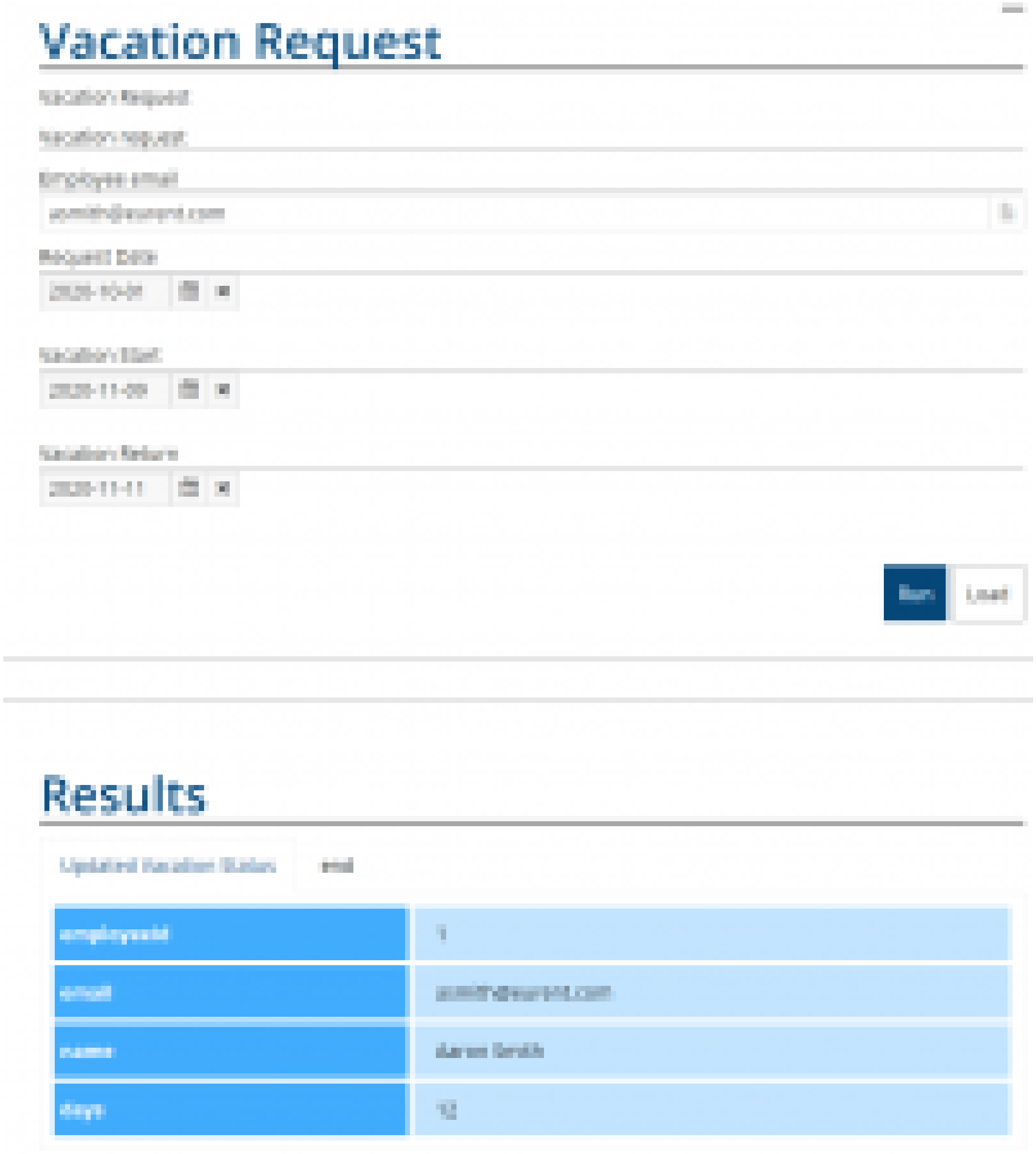
Data inputs of this node	Value to assign to this data input. Context includes the data linked with a data input associatio		
employeeId number	Current Vacation Status.employeeId		
vacation [illegible]	1	email Text	Current Vacation Status.email
	2	name Text	Current Vacation Status.name
	3	days Number	Current Vacation Status.days - Num Days

The only output of the update is the id of the updated record. We already know that, but just to check that the operation worked, we can use the *Get vacation*operation to return the updated record. The output of the Get operation is referenced by the table name, *vacation*, and we will save it in the process data output *Updated Vacation Status*.



	Data linked with a data output association	Value to assign
1	Updated Vacation Status *bruce.1.bruce.vacation*	vacation

We also need to model the User task Manually Approve Vacation, but we'll leave that for another time. Once that is done, we can publish the BPMN model as an executable service simply clicking Cloud Publish in the Execution ribbon. In the Service Library we can try it out.



Vacation Request

Vacation Request

Vacation request

Employee email

asmith@acme.com

Request Date

2020-10-09

Vacation Start

2020-11-09

Vacation Return

2020-11-11

Results

employeeid	1
email	asmith@acme.com
name	Aaron Smith
days	12

Here Aaron Smith, with 14 days remaining, requested 2 vacation days. This is automatically approved, so the *Updated Vacation Status* data output now shows him with 12 days remaining.

So let's recap what we've just seen. Our employee vacation status is in a mySQL database *vacation* that is exposed via OData. When an employee submits a *Vacation request*, this triggers a process that first uses the OData *Find vacation* operation with a *$filter* parameter to return the employee record based on his email address. The remaining *days* value from that record in combination with values of the *Vacation request* are used in a decision service that either approves, refuses, or refers the request, and sends an email to the employee with the result. If the request is approved, a second service task uses the OData *Update vacation* operation to deduct the requested vacation days from the original count, and then uses the OData *Get vacation* operation to output the updated record for that employee. We published this in one click as an executable service, and verified that it gives the expected result. And none of it required programming!

If you are playing around with Trisotech BPMN and didn't think you could make your processes executable, maybe you can. A good place to start is by OData-enabling your databases.