OpenRules Decision Model

for DMCommunity April-2024 Challenge “Lookup Tables”

Problem

The problem was formulated as follows:

Your decision model needs to validate each claim against the standard lists of compatible and incompatible pairs of procedures and diagnoses represented as ICD-10 codes. While real claim processing systems deal with tens of standard lists, we will limit our challenge only to two lists. Here are the validation rules for claim procedures of types X, Y, and Z:

1. If a Procedure has Type X or Y, then check if this procedure has a corresponding diagnosis such that the pair [procedure, diagnosis] can be found in the standard list “CompatibleCodes.csv” that consists of 260K compatible pairs [Diagnosis Code, Procedure Code]. If no corresponding pair is found, report all procedures without compatible diagnoses.
2. If a Procedure has Type Z, then check that this procedure against the standard list “IncompatibleCodes.csv” which consists of only 83 records of incompatible ranges [Procedure Code Min, Procedure Code Max, Diagnosis Code Min, Diagnosis Code Max]. If a pair [procedure, diagnosis] is found, report the incompatible pairs (Procedure Code – Diagnosis Code).

The example of a claim is provided in the JSON format “Claim.json“.

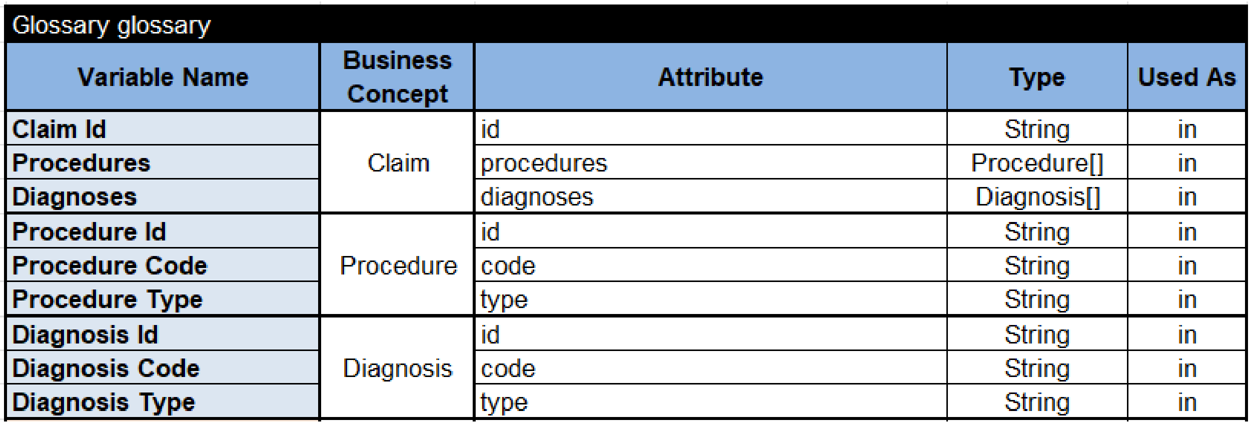
OpenRules-based Solution

Input Data

First, we created the OpenRules project “LookupTables” – you can download it from OpenRules samples. We placed all provided files “Claim.json”, CompatibleCodes.csv”, and “IncompatibleCodes.csv” in the subfolder “data”.

Glossary

Then we created the glossary with all input decision variables corresponding to the provided JSON file.



Glossary glossary

Variable Name	Business Concept	Attribute	Type	Used As
Claim Id	Claim	id	String	in
Procedures		procedures	Procedure[]	in
Diagnoses		diagnoses	Diagnosis[]	in
Procedure Id	Procedure	id	String	in
Procedure Code		code	String	in
Procedure Type		type	String	in
Diagnosis Id	Diagnosis	id	String	in
Diagnosis Code		code	String	in
Diagnosis Type		type	String	in

To specify the expected output of our decision model, we added the business concept "Results" to the Glossary:

Procedures Without Compatible Diagnoses	Results	proceduresWithoutCompatibleDiagnoses	String[]	out
Incompatible Procedures and Diagnoses		incompatibleProceduresAndDiagnoses	String[]	out
Errors		errors	String[]	out
Validation Result		result	String	out

Rules

The following main tables specify how we want to calculate the results:

Decision ValidateClaim
ActionExecute
Rules
ValidateProcedures
SetValidationResult

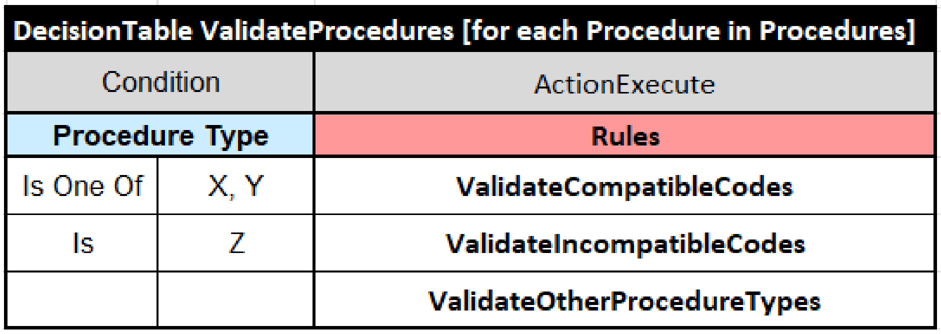
The first step "ValidateProcedure" should specify the validation rules for all procedures and related diagnoses within the claim. They will define 3 output arrays:

- Procedures Without Compatible Diagnoses
- Incompatible Procedures and Diagnoses
- Errors.

The overall Validation Result can be defined as SUCCESS or FAILURE by the second step:

DecisionTable SetValidationResult						
Condition		Condition		Condition		Action
Errors		**Procedures Without Compatible Diagnoses**		**Incompatible Procedures and Diagnoses**		**Validation Result**
Is Empty	TRUE	Is Empty	TRUE	Is Empty	TRUE	**SUCCESS**
						FAILURE

The high-level rules "ValidateProcedures" are defined in the following table:

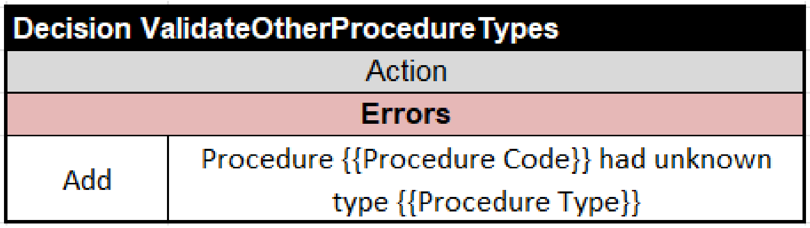


DecisionTable ValidateProcedures [for each Procedure in Procedures]		
Condition		ActionExecute
Procedure Type		Rules
Is One Of	X, Y	**ValidateCompatibleCodes**
Is	Z	**ValidateIncompatibleCodes**
		ValidateOtherProcedureTypes

This table iterates over all claim Procedures executing the following rules:

- ValidateCompatibleCodes for the procedures of the type X or Y
- ValidateIncompatibleCodes for the procedures of the type X
- ValidateOtherProcedureTypes for all other types not specified in the Challenge.

We implemented the last step as a stub that simply adds an error to the array "Errors":

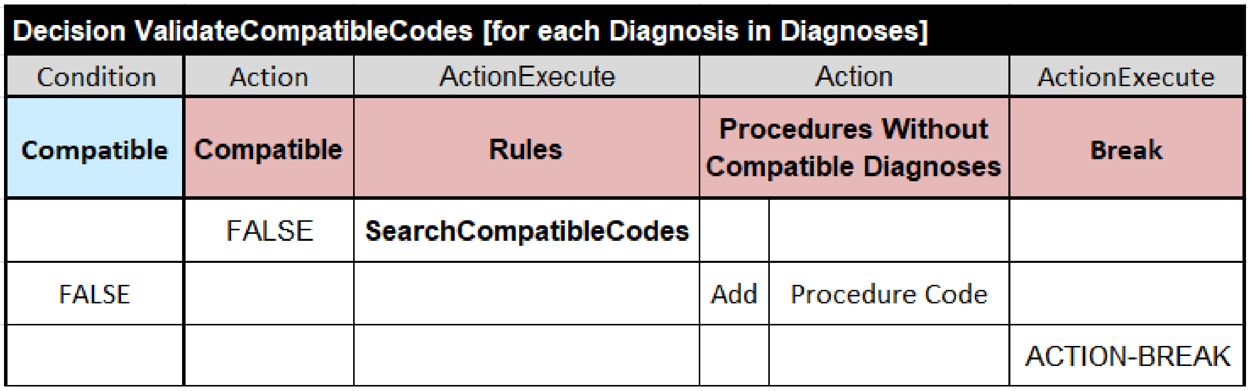


Decision ValidateOtherProcedureTypes	
Action	
Errors	
Add	Procedure {{Procedure Code}} had unknown type {{Procedure Type}}

Let's look at how two major validations against files "CompatibleCodes.csv" and "IncompatibleCodes.csv" can be implemented.

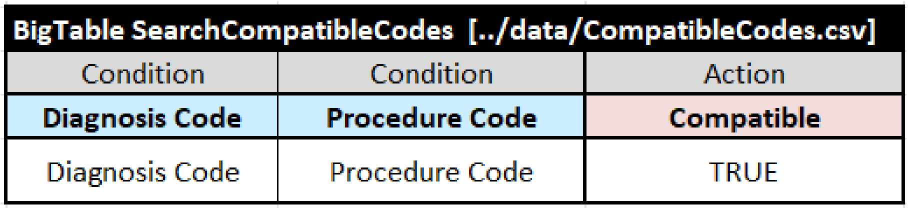
Validating Compatible Codes

The following table first executes the search inside the file "CompatibleCodes.csv" to set the intermediate Boolean variable "Compatible":



Decision ValidateCompatibleCodes [for each Diagnosis in Diagnoses]					
Condition	Action	ActionExecute	Action		ActionExecute
Compatible	Compatible	Rules	Procedures Without Compatible Diagnoses		Break
	FALSE	**SearchCompatibleCodes**			
FALSE			Add	Procedure Code	
					ACTION-BREAK

If the search is unsuccessful, this variable will remain "FALSE" as defined in the first rule before the search and the second rule will add the current Procedure to the array "Procedures Without Compatible Diagnoses". The third rule will break the iteration over the claim Diagnoses for the current Procedure. Here is how the search in the file "CompatibleCodes.csv" is implemented:

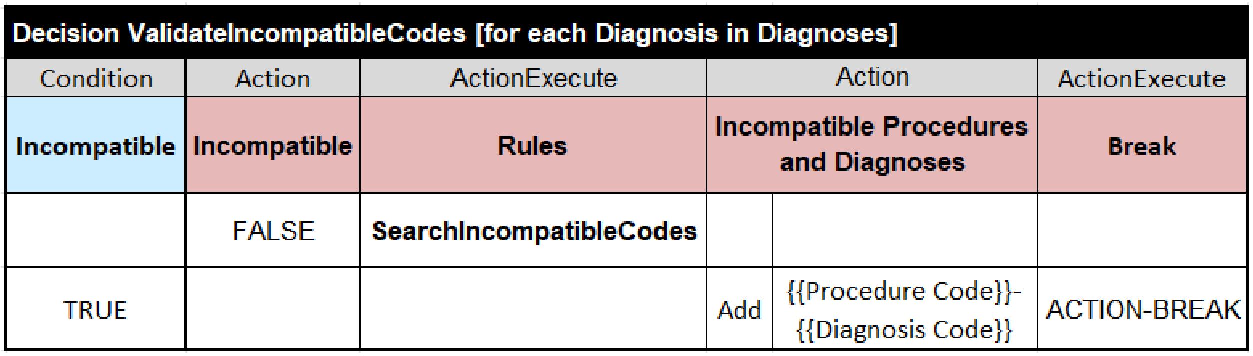


BigTable SearchCompatibleCodes [../data/CompatibleCodes.csv]		
Condition	Condition	Action
Diagnosis Code	**Procedure Code**	**Compatible**
Diagnosis Code	Procedure Code	TRUE

It utilizes OpenRules **BigTable** which does search in large tables highly efficiently (~100 times faster compared with regular decision tables).

Validating Incompatible Codes

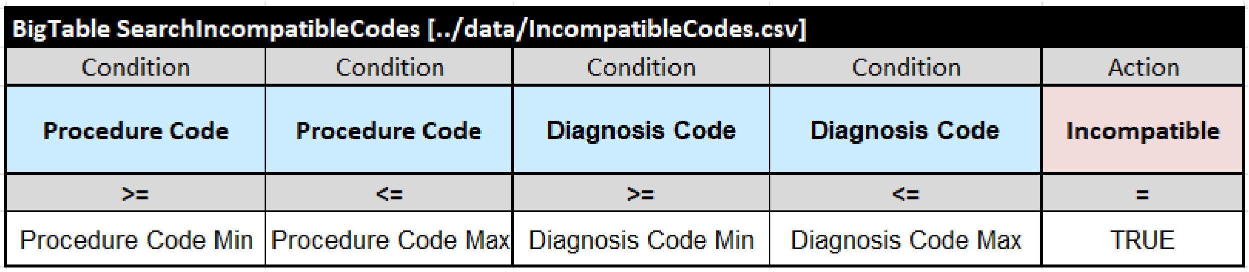
The following table first executes the search inside the file "IncompatibleCodes.csv":



Decision ValidateIncompatibleCodes [for each Diagnosis in Diagnoses]					
Condition	Action	ActionExecute	Action		ActionExecute
Incompatible	**Incompatible**	**Rules**	**Incompatible Procedures and Diagnoses**		**Break**
	FALSE	**SearchIncompatibleCodes**			
TRUE			Add	{{Procedure Code}}-{{Diagnosis Code}}	ACTION-BREAK

If the search is unsuccessful, the second rule will add the current pair [Procedure Code-Diagnosis Code] to the array "Incompatible Procedures and Diagnoses". It will also break the iteration over the claim Diagnoses for the current Procedure. Otherwise, the search will continue for the remaining diagnoses.

Here is how the search in the file "IncompatibleCodes.csv" is implemented:

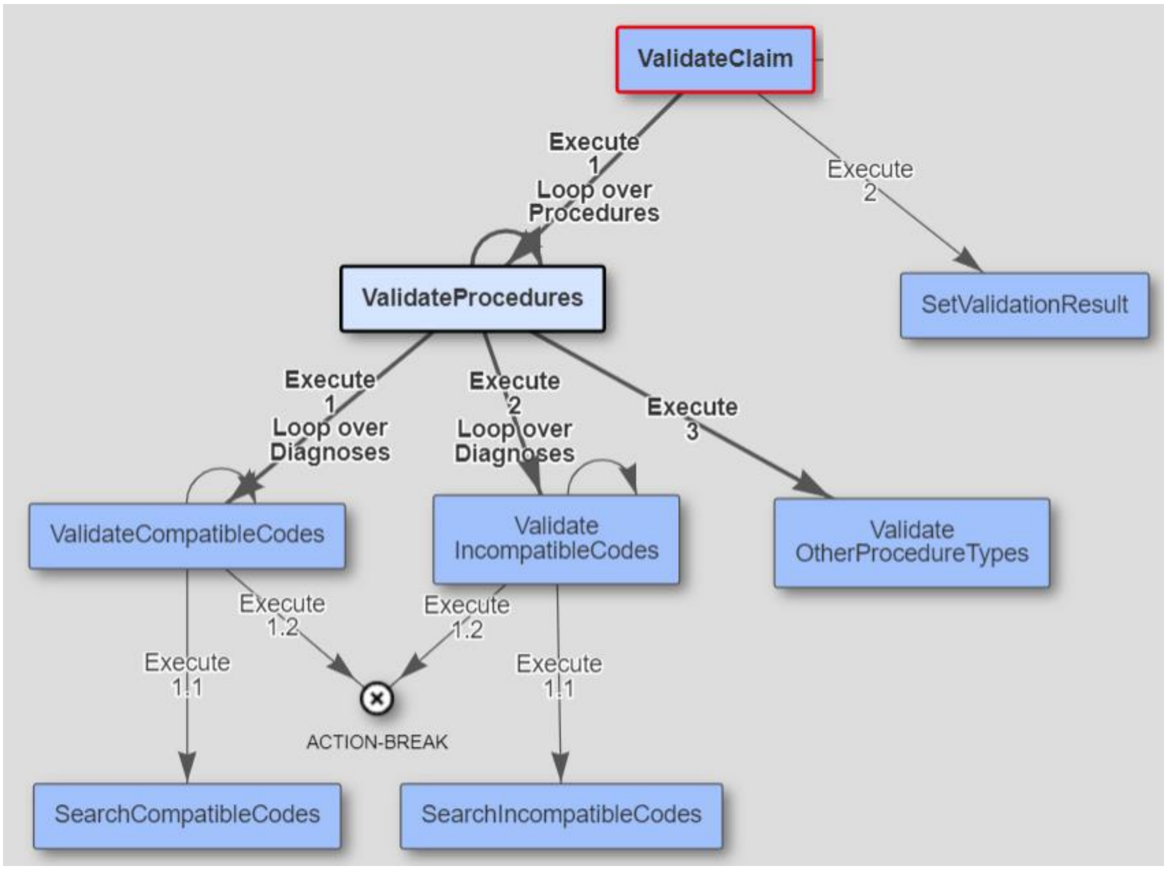


BigTable SearchIncompatibleCodes [../data/IncompatibleCodes.csv]				
Condition	Condition	Condition	Condition	Action
Procedure Code	**Procedure Code**	**Diagnosis Code**	**Diagnosis Code**	**Incompatible**
>=	**<=**	**>=**	**<=**	**=**
Procedure Code Min	Procedure Code Max	Diagnosis Code Min	Diagnosis Code Max	TRUE

This search again utilizes the highly efficient **BigTable**.

Decision Diagram

The decision diagram created by OpenRules Explorer is presented in the following picture:



Testing Decision Model

To test our decision model, we created the following test table in Excel that uses the provided JSON file "Claim.json" and specifies the expected results:

DecisionTest testCases					
#	ActionUseJson	ActionExpect	ActionExpect	ActionExpect	ActionExpect
Test ID	JSON File	Validation Result	Procedures Without Compatible Diagnoses	Incompatible Procedures and Diagnoses	Errors
1	../data/Claim.json	FAILURE	12001, 28003, 28005, 11000, 27675, 11011, 49657, 62319	28415-I10, 30420-I10	Procedure 49180 had unknown type N

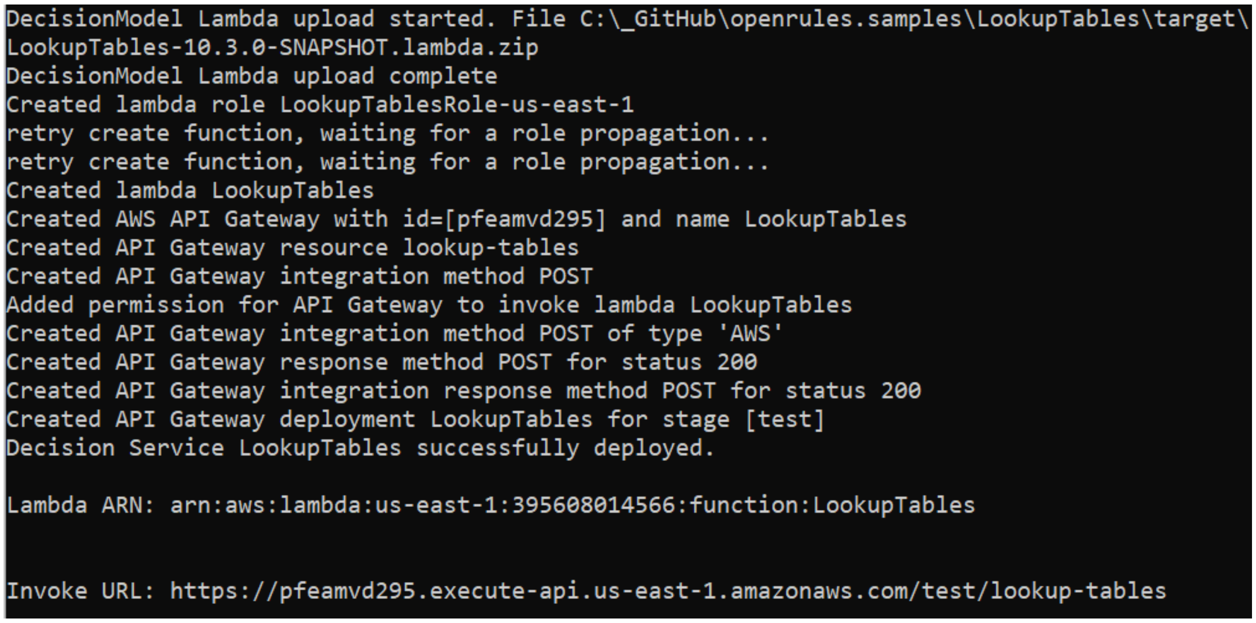
When we ran this test locally using the standard batch file "test.bat" we received the following results:

Results (
proceduresWithoutCompatibleDiagnoses=[12001, 28003, 28005, 11000, 27675, 11011, 49657, 62319],
incompatibleProceduresAndDiagnoses=[28415-I10, 30420-I10],
errors=[Procedure 49180 had unknown type N],result=FAILURE)
Test 'testCases-1' completed OK. Elapsed time **669.49** milliseconds.

The relatively large execution time mainly was spent on Java warm-up. So, we decided to deploy this decision model as a web service to test the real performance.

Deployment

We selected AWS Lambda as our deployment target. To deploy our decision model we just ran the standard batch file “deployLambda.bat”. Here is the deployment protocol:

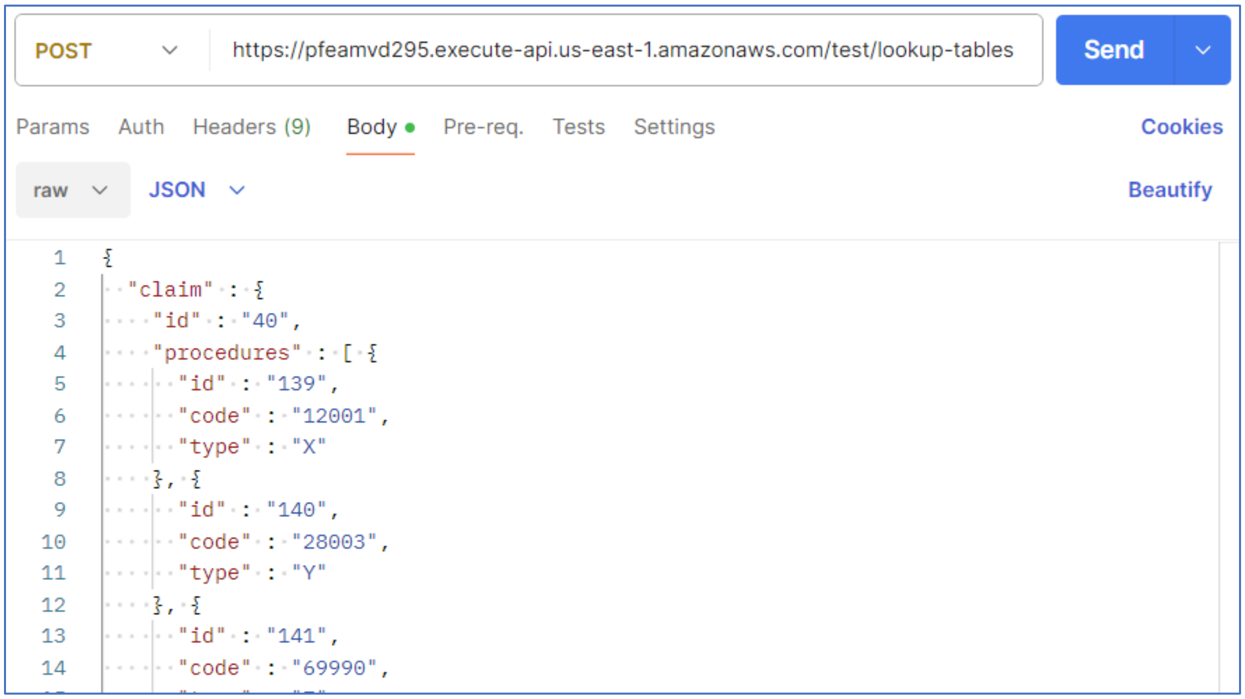


```
DecisionModel Lambda upload started. File C:\_GitHub\openrules.samples\LookupTables\target\
LookupTables-10.3.0-SNAPSHOT.lambda.zip
DecisionModel Lambda upload complete
Created lambda role LookupTablesRole-us-east-1
retry create function, waiting for a role propagation...
retry create function, waiting for a role propagation...
Created lambda LookupTables
Created AWS API Gateway with id=[pfeamvd295] and name LookupTables
Created API Gateway resource lookup-tables
Created API Gateway integration method POST
Added permission for API Gateway to invoke lambda LookupTables
Created API Gateway integration method POST of type 'AWS'
Created API Gateway response method POST for status 200
Created API Gateway integration response method POST for status 200
Created API Gateway deployment LookupTables for stage [test]
Decision Service LookupTables successfully deployed.

Lambda ARN: arn:aws:lambda:us-east-1:395608014566:function:LookupTables


Invoke URL: https://pfeamvd295.execute-api.us-east-1.amazonaws.com/test/lookup-tables
```

Thus, our decision service became available with the following endpoint URL:

https://pfeamvd295.execute-api.us-east-1.amazonaws.com/test/lookup-tables

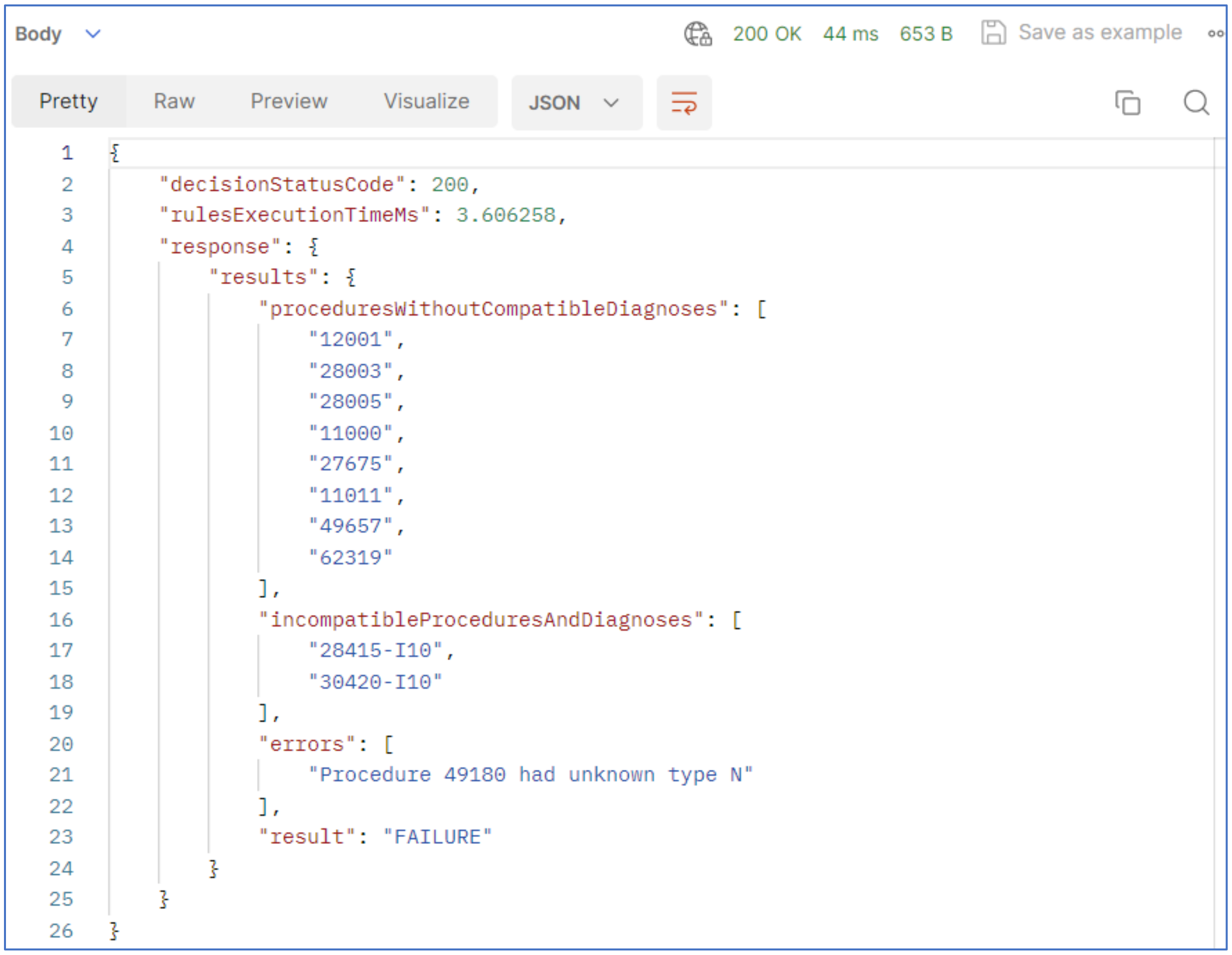
To test this service as it will be used in production, we ran POSTMAN using this URL and Claim.json as a request body:



https://pfeamvd295.execute-api.us-east-1.amazonaws.com/test/lookup-tables

```
{
  "claim" : {
    "id" : "40",
    "procedures" : [ {
      "id" : "139",
      "code" : "12001",
      "type" : "X"
    }, {
      "id" : "140",
      "code" : "28003",
      "type" : "Y"
    }, {
      "id" : "141",
      "code" : "69990",
```

After clicking on “Send”, POSTMAN executed our decision service (a lambda function) producing the following response:



```
{
    "decisionStatusCode": 200,
    "rulesExecutionTimeMs": 3.606258,
    "response": {
        "results": {
            "proceduresWithoutCompatibleDiagnoses": [
                "12001",
                "28003",
                "28005",
                "11000",
                "27675",
                "11011",
                "49657",
                "62319"
            ],
            "incompatibleProceduresAndDiagnoses": [
                "28415-I10",
                "30420-I10"
            ],
            "errors": [
                "Procedure 49180 had unknown type N"
            ],
            "result": "FAILURE"
        }
    }
}
```

Performance Considerations

As you can see in the JSON Response, the pure rules execution time for this large claim was **only 3.6 milliseconds.** Together with sending the JSON request and receiving back JSON response, the total service execution time was **44 milliseconds**. It means that one instance of our decision service will easily handle **1,000 claims in less than 1 minute** and **60,000 claims in less than 1 hour**.

We relied on the standard deployment options. However, we have several ways to essentially improve this already high performance:

1) We may process claims in batches and the performance will be improved from 10 to 100 times.
2) We may allocate more memory for the Lambda function which usually makes it run much faster.

3) We may increase the AWS Concurrency limit, and multiple instances of the same decision service working in parallel will add more 10-20 times performance improvement.

Thus, our deployed decision service **will handle 1,000,000 claims within an hour**.

Conclusion

We used OpenRules to develop a working decision model that utilizes large external CSV files. Adding, removing, or modifying the records inside these files does not require any modifications in the decision model. In such cases, the decision model will only need to be redeployed. The deployed decision service shows great performance and will easily handle the requested millions of claims per day.