



# Red Hat Process Automation Manager 7.8

ガイド付きデシジョンテーブルを使用したデシ  
ジョンサービスの作成



# Red Hat Process Automation Manager 7.8 ガイド付きデシジョンテーブル を使用したデシジョンサービスの作成

---

Red Hat Customer Content Services  
brms-docs@redhat.com

## 法律上の通知

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

Java<sup>®</sup> is a registered trademark of Oracle and/or its affiliates.

XFS<sup>®</sup> is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL<sup>®</sup> is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js<sup>®</sup> is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack<sup>®</sup> Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

本書は、Red Hat Process Automation Manager 7.8 で、ガイド付きデシジョンテーブルを使用してデシジョンサービスを作成する方法を説明します。

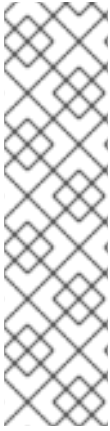
## 目次

前書き .....	3
第1章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット .....	4
第2章 ガイド付きデシジョンテーブル .....	8
第3章 データオブジェクト .....	9
3.1. データオブジェクトの作成 .....	9
第4章 ガイド付きデシジョンテーブルの作成 .....	11
第5章 ガイド付きデシジョンテーブルのヒットポリシー .....	13
5.1. ヒットポリシーの例: 映画観賞券の割引価格を定めるデシジョンテーブル .....	14
5.1.1. ガイド付きデシジョンテーブルの種類 .....	16
第6章 ガイド付きデシジョンテーブルへの列の追加 .....	18
第7章 ガイド付きデシジョンテーブルの列の種類 .....	20
7.1. "ADD A CONDITION" .....	20
7.1.1. 条件列セルへの any other 値の追加 .....	22
7.2. "ADD A CONDITION BRL FRAGMENT" .....	22
7.3. "ADD A METADATA COLUMN" .....	25
7.4. "ADD AN ACTION BRL FRAGMENT" .....	25
7.5. "ADD AN ATTRIBUTE COLUMN" .....	28
7.6. "DELETE AN EXISTING FACT" .....	29
7.7. "EXECUTE A WORK ITEM" .....	29
7.8. "SET THE VALUE OF A FIELD" .....	30
7.9. "SET THE VALUE OF A FIELD WITH A WORK ITEM RESULT" .....	30
第8章 ガイド付きデシジョンテーブルのルール名列の表示 .....	32
第9章 ガイド付きデシジョンテーブルで列の編集または削除 .....	33
第10章 ガイド付きデシジョンテーブルで行の追加およびルールの定義 .....	34
第11章 ルールアセットのドロップダウンリストの列挙定義 .....	36
11.1. ルールアセットの詳細列挙オプション .....	37
第12章 ガイド付きデシジョンテーブルのリアルタイム検証および妥当性確認 .....	40
12.1. ガイド付きデシジョンテーブルの問題の種類 .....	40
12.2. 通知の種類 .....	41
12.3. ガイド付きデシジョンテーブルの検証および妥当性確認の無効化 .....	41
第13章 スプレッドシート形式のデシジョンテーブルへの、ガイド付きデシジョンテーブルの変換 .....	43
第14章 ルールの実行 .....	44
第15章 次のステップ .....	50
付録A バージョン情報 .....	51



## 前書き

ビジネス分析者またはビジネスルールの開発者は、ガイド付きデシジョンテーブルを使用して、ウィザード主導のテーブル形式でビジネスルールを定義することができます。このルールは Drools Rule Language (DRL) に組み込まれ、プロジェクトのデシジョンサービスの中心となります。



### 注記

ルールベースやテーブルベースのアセットではなく、Decision Model and Notation (DMN) モデルを使用してデシジョンサービスを設計することもできます。Red Hat Process Automation Manager 7.8 の DMN サポートに関する詳細は、以下の資料を参照してください。

- 『[デシジョンサービスのスタートガイド](#)』 (DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- 『[DMN モデルを使用したデシジョンサービスの作成](#)』 (Red Hat Process Automation Manager の DMN サポートおよび機能の概要)

### 前提条件

- ガイド付きデシジョンテーブルのチームおよびプロジェクトが Business Central に作成されており、各アセットが、スペースに割り当てられたプロジェクトに関連付けられています。詳細は『[デシジョンサービスのスタートガイド](#)』を参照してください。

## 第1章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、ゴールおよびニーズに合わせて、アセットを1つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表1.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメンテーション
DMN (Decision Model and Notation) モデル	<ul style="list-style-type: none"> <li>Object Management Group (OMG) が定義する標準記法に基づくデシジョンモデルである</li> <li>1つまたは複数の意思決定要件グラフ (DRG: decision requirements graph) を含むグラフィカルな意思決定要件ダイアグラム (DRD: decision requirements diagram) を使用してビジネスの意思決定フローを追跡する</li> <li>DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する</li> <li>DMN デシジョンテーブルおよび他の DMN ボックス式 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする</li> <li>Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる</li> <li>包括性、具体性および安定性のあるデシジョンフローの作成に最適である</li> </ul>	Business Central または DMN 準拠のエディター	『 <a href="#">DMN モデルを使用したデシジョンサービスの作成</a> 』



アセット	主な特徴	オーサリングツール	ドキュメンテーション
ガイド付きデシジョンテーブル	<ul style="list-style-type: none"> <li>● Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブル</li> <li>● デシジョンテーブルにスプレッドシートで対応する代わりにウィザードで対応する</li> <li>● 条件を満たした入力に、フィールドとオプションを指定する</li> <li>● ルールテンプレートを作成するためのテンプレートキーと値をサポートする</li> <li>● その他のアセットではサポートされていない、ヒットポリシー、リアルタイム検証などの追加機能をサポートする</li> <li>● コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適</li> </ul>	Business Central	『ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成』
スプレッドシートのデシジョンテーブル	<ul style="list-style-type: none"> <li>● Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである</li> <li>● ルールテンプレートを作成するためのテンプレートキーと値をサポートする</li> <li>● Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適</li> <li>● アップロード時に適切にルールをコンパイルするために厳密な構文要件がある</li> </ul>	スプレッドシートエディター	『スプレッドシート形式のデシジョンテーブルを使用したデシジョンサービスの作成』
ガイド付きルール	<ul style="list-style-type: none"> <li>● Business Central の UI ベースのルールデザイナーで作成する個別ルール</li> <li>● 条件を満たした入力に、フィールドとオプションを指定する</li> <li>● コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適</li> </ul>	Business Central	『ガイド付きルールを使用したデシジョンサービスの作成』

アセット	主な特徴	オーサリングツール	ドキュメンテーション
ガイド付きルールテンプレート	<ul style="list-style-type: none"> <li>● Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造</li> <li>● 条件を満たした入力に、フィールドとオプションを指定する</li> <li>● (このアセットの基本となる) ルールテンプレートを作成するためのテンプレートキーと値をサポートする</li> <li>● ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適</li> </ul>	Business Central	『 <a href="#">ガイド付きルールテンプレートを使用したデジジョンサービスの作成</a> 』
DRL ルール	<ul style="list-style-type: none"> <li>● <code>.drl</code> テキストファイルに直接定義する個別ルール</li> <li>● 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる</li> <li>● スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能</li> <li>● 詳細にわたる DRL オプションが必要なルールを作成するのに最適</li> <li>● ルールを適切にコンパイルするために厳密な構文要件がある</li> </ul>	Business Central または統合開発環境 (IDE)	『 <a href="#">DRL ルールを使用したデジジョンサービスの作成</a> 』

アセット	主な特徴	オーサリングツール	ドキュメンテーション
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	<ul style="list-style-type: none"><li>● Data Mining Group (DMG) が定義する標準記法に基づく予測データ分析モデルである</li><li>● PMML モデルを PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する</li><li>● 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする</li><li>● スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる</li><li>● Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である</li></ul>	PMML または XML エディター	『 <a href="#">PMML モデルでのデシジョンサービスの作成</a> 』

## 第2章 ガイド付きデシジョンテーブル

ガイド付きデシジョンテーブルは、デシジョンテーブルのスプレッドシートに代わる方法で、ウィザードを用いて表形式でビジネスルールを定義します。ガイド付きデシジョンテーブルでは、プロジェクトで指定したデータオブジェクトをもとに、Business Central のUI ベースのウィザードに従ってルール属性、メタデータ、条件、およびアクションを定義します。ガイド付きデシジョンテーブルを作成すると、定義したルールは、その他のすべてのルールアセットとともに Drools Rule Language (DRL) ルールにコンパイルされます。

ガイド付きデシジョンテーブルに関連するすべてのデータオブジェクトは、ガイド付きデシジョンテーブルと同じプロジェクトパッケージに存在する必要があります。同じパッケージのアセットはデフォルトでインポートされます。必要なデータオブジェクトとガイド付きデシジョンテーブルの作成後、ガイド付きデシジョンテーブルデザイナーの **Data Objects** タブを使用して、必要なデータオブジェクトがすべてリストされていることを検証したり、**新規アイテム** を追加してその他の既存データオブジェクトをインポートしたりできます。

## 第3章 データオブジェクト

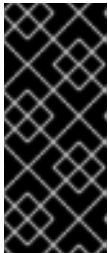
データオブジェクトは、作成するルールアセットの構成要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータ型です。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

### 3.1. データオブジェクトの作成

以下の手順は、データオブジェクトを作成する際の一般的な概要で、特定のビジネスアセットに固有のものではありません。

#### 手順

1. Business Central で、**Menu** → **Design** → **Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset** → **Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

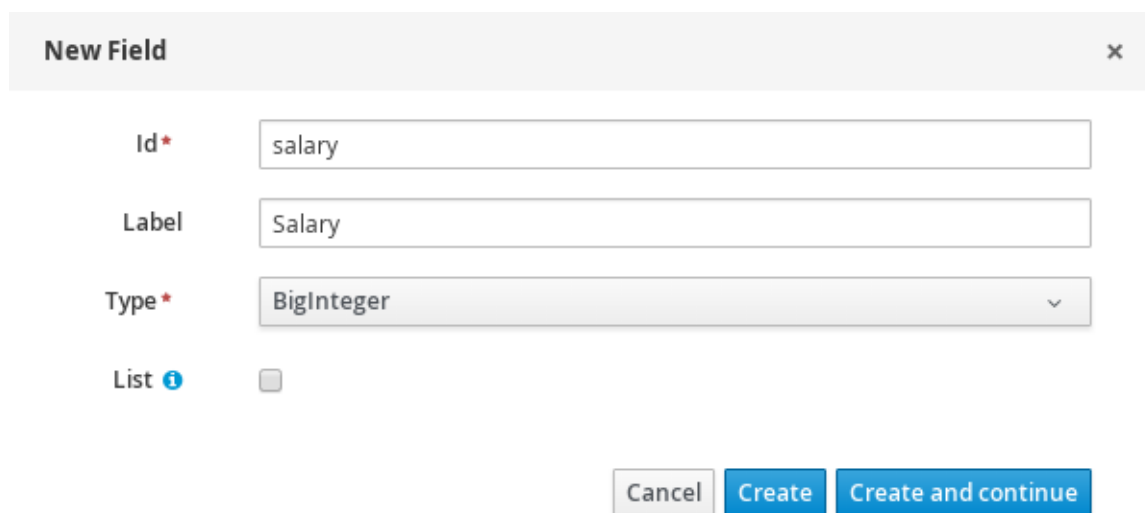


#### 別のパッケージからのデータオブジェクトのインポート

別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトに関連するルールアセットを選択し、アセットデザイナーで **Data Objects** → **New item** に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、**Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (\*) マークが付いています。
  - **Id**: フィールドの一意の ID を入力します。
  - **Label**: (任意) フィールドのラベルを入力します。
  - **Type**: フィールドのデータタイプ\を入力します。
  - **List**: (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図3.1 データオブジェクトへのデータフィールドの追加



**New Field** x

**Id\*** salary

**Label** Salary

**Type\*** BigInteger

**List**

Cancel Create Create and continue

7. **Create** をクリックして新規フィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き追加できます。



#### 注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

## 第4章 ガイド付きデシジョンテーブルの作成

ガイド付きデシジョンテーブルを使用して、ルール属性、メタデータ、条件、およびアクションを表形式で定義し、ビジネスルールプロジェクトに追加できます。

### 手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Guided Decision Table** をクリックします。
3. **ガイド付きデシジョンテーブル** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトが割り当てられている、またはこれから割り当てるパッケージにする必要があります。
4. **ウィザードを使用** を選択して、ウィザードでテーブルの設定を終わらせるか、このオプションは選択せずにテーブル作成を終わりにし、ガイド付きデシジョンテーブルデザイナーで残りの設定を行います。
5. テーブルのルール行が準拠するヒットポリシーを選択します。詳細は「[5章ガイド付きデシジョンテーブルのヒットポリシー](#)」を参照してください。
6. テーブルの形式を、**拡張エントリ** と **制限エントリ** から選択します。詳細は「[ガイド付きデシジョンテーブルの種類](#)」を参照してください。
7. **OK** をクリックして設定を完了します。**ウィザードを使用** をクリックすると、ガイド付きデシジョンテーブルが表示されます。**ウィザードを使用** オプションを選択しないとこのプロンプトは表示されず、テーブルデザイナーが表示されます。  
たとえば、以下のウィザード設定は、ローン申請のデシジョンサービスで使用するガイド付きデシジョンテーブルです。

図4.1 ガイド付きデシジョンテーブルの作成

**Create new Guided Decision Table**

**Guided Decision Table \***

Pricing loans

**Package**

mortgages.mortgages

Use Wizard

**Hit Policy:** None

None

This is the normal hit mode. Old decision tables will use this by default, but since 7.0 uses PHREAK the row order now matters. There is no migration tooling needed for the old tables. Multiple rows can fire. Verification warns about rows that conflict.

**Table Format:**

Extended entry, values defined in table body

Limited entry, values defined in columns

+ Ok Cancel

8. ウィザードを使用する場合は、利用可能なインポート、ファクトパターン、制約、およびアクションを追加して、テーブルの列を拡張するかどうかを選択します。**Finish** をクリックしてウィザードを閉じ、テーブルデザイナーを表示します。

図4.2 ガイド付きデシジョンテーブルのウィザード

**Guided Decision Table Wizard**

Summary of fields for the decision table.

Summary

Imports

Add Fact Patterns

Add Constraints

Add Actions to update Facts

Add Actions to insert Facts

Columns to expand

**Name:** Pricing loans

**Path:** default//master@myrepo/mortgages/src/main/resources/mortgages/mortgages

**Table Format:** Extended entry, values defined in table body

**Hit Policy:** None

This is the normal hit mode. Old decision tables will use this by default, but since 7.0 uses PHREAK the row order now matters. There is no migration tooling needed for the old tables. Multiple rows can fire. Verification warns about rows that conflict.

< Previous Next > Cancel Finish

ガイド付きデシジョンテーブルデザイナーで、列および行を追加または編集し、最終調整を行います。



## 第5章 ガイド付きデシジョンテーブルのヒットポリシー

ヒットポリシーは、ガイド付きデシジョンテーブルのルール (行) を適用する順番 (上から下、優先順位を指定した順など) を指定します。

次のヒットポリシーが利用できます。

- **None:** (デフォルトのヒットポリシー) 複数の行を実行できます。競合している行については検証により警告されます。(ガイド付きデシジョンテーブル以外のスプレッドシートを使用して) アップロードされているデシジョンテーブルには、このヒットポリシーが適用されます。
- **Resolved Hit:** 指定されている優先順位に従って同時に実行できる行は1つだけです。リストされている順序は無視されます (たとえば、行10の優先順位を行5よりも高くできます)。したがって、視覚的な分かりやすさを基準に行の順番を決め、それとは別に優先順位を指定することができます。
- **Unique Hit:** 同時に実行できる行は1つだけです。一致する条件は重複しないようにする必要があります。複数の行を実行すると、開発時の検証により警告が生成されます。
- **First Hit:** 同時に実行できる行は1つだけです。テーブルの順番に従って、上から下に実行します。
- **Rule Order:** 複数の行を実行できます。複数の行の実行は想定されているため、検証では複数の行の競合は報告されません。

図5.1 利用可能なヒットポリシー

## 5.1. ヒットポリシーの例: 映画観賞券の割引価格を定めるデシジョンテーブル

以下の表は、顧客の年齢、学生かどうか、軍隊経験の有無、または3つの条件すべてをもとにして映画観賞券の割引価格を求める時に使用するデシジョンテーブル例の一部です。

表5.1 映画観賞券で利用可能な割引価格を定めたデシジョンテーブル例

行番号	割引の種類	割引価格
1	高齢者 (60 歳以上)	10%
2	学生	10%
3	軍隊経験	10%

以下の例では、最終的に適用される割引合計は、テーブルに指定したヒットポリシーによって変わります。

- **None/Rule Order:** ヒットポリシー **None** および **Rule Order** では、適用可能なルールがすべて組み込まれ、それぞれの顧客に適用される割引が積み重ねられます。  
例: 現在学生で、軍隊経験がある高齢者には、3つの割引がすべて適用され、合計30%の割引になります。

重要な相違点: **None** では、複数の行が適用されると警告が作成されます。**Rule Order** では、この警告が作成されません。

- **First Hit/Resolved Hit:** ヒットポリシー **First Hit** および **Resolved Hit** ポリシーでは、割引の中から1つだけが適用されます。  
**First Hit** の場合は、リストの中で最初に満たされた割引が適用され、その他の割引は適用されません。

例: 現在学生で、軍隊経験がある高齢者には、テーブルに最初にリストされている高齢者割引10%だけが適用されます。

**Resolved Hit** の場合は、テーブルの修正が必要になります。テーブルで優先順位の例外を割り当てた割引が、リストされた順番にかかわらず最初に適用されます。この例外を割り当てるには、割引(行)の優先順位を指定する新しい列を追加します。

例: 軍隊経験者向けの割引の優先順位が、高齢者向けまたは学生向けの割引よりも高く設定されている場合、現在学生で、軍隊経験がある高齢者には、軍隊経験者の割引10%が適用されます。ここでは、高齢者もしくは学生であるかどうかは関係ありません。

**Resolved Hit** ポリシーを適用して修正したデシジョンテーブルをご覧ください。

表5.2 **Resolved Hit** ポリシーを適用して修正したデシジョンテーブル

行番号	割引の種類	優先する行	割引価格
1	高齢者 (60 歳以上)		10%
2	学生		10%
3	軍隊経験	1	10%

この修正したテーブルでは、軍人経験者向けの割引が事実上の行1になるため、高齢者向けと学生向けの割引よりも優先され、その後でその他の割引が追加されます。優先順位は、行1に対する優先だけを指定する必要があり、「行1および行2」の両方に指定する必要はありません。この変更により、行のヒット順は、3→1→2となり、さらに行が増えた場合はこの後に続きます。



### 注記

ここで変更した行の順番は、軍隊経験者向けの割引を行1に移動して **First Hit** ポリシーをテーブルに適用した場合と同じになります。ただし、ルールを特定の順番で並べ(アルファベット順)、ルールの適用順を変更する場合は、**Resolved Hit** ポリシーが便利です。

重要な相違点: **First Hit** を使用すると、ルールの適用順はリストした順番に固定されます。**Resolved Hit** を使用すると、指定された優先順位の例外を除いて、リストされた順にルールが適用されます。

- **Unique Hit:** テーブルの修正が必要になります。**Unique Hit** ポリシーを使用する場合は、一度

に複数の行を適用できないように行を作成する必要があります。ただし、ルールを1つまたは複数適用するかどうかは、行ごとに指定できます。この方法では、**Unique Hit** ポリシーを使用した場合に重複の警告が表示されないように、デシジョンテーブルの粒度をより細かくできます。

**Unique Hit** ポリシーを適用して修正したデシジョンテーブルをご覧ください。

表5.3 Unique Hit ポリシーを適用して修正デシジョンテーブル

行番号	高齢者 (65歳以上)	学生	軍隊経験	割引価格
1	はい	いいえ	いいえ	10%
2	いいえ	はい	いいえ	10%
3	いいえ	いいえ	はい	10%
4	はい	はい	いいえ	20%
5	はい	いいえ	はい	20%
6	いいえ	はい	はい	20%
7	はい	はい	はい	30%

この修正したテーブルでは、各行が一意で、重複はできず、1つまたは複数の割引が適用されます。

### 5.1.1. ガイド付きデシジョンテーブルの種類

Red Hat Process Automation Manager では、拡張エントリーテーブルと制限エントリーテーブルの2種類のデシジョンテーブルに対応します。

- **拡張エントリー:** 拡張エントリーのデシジョンテーブルの列定義には、パターン、フィールド、演算子を指定します。値は指定しません。値または状態 (state) は、デシジョンテーブルの本体に保持されます。

Pricing loans		application : LoanApplication				ome : IncomeSou	application		
#	Description	amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

- **制限エントリー:** 制限エントリーのデシジョンテーブルの列定義には、パターン、フィールド、演算子に加えて、値を指定します。デシジョンテーブルの本体には、デシジョンテーブルの状態 (state) をブール値で指定します。正の値 (チェックボックスを選択した場合) はその列が適用され、負の値 (チェックボックスを選択しない場合) はその列が適用されないことを示しています。

Credit rating						
#	Description	CR = AA	CR = OK	CR = Sub prime	Approve	Decline
1		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>

## 第6章 ガイド付きデシジョンテーブルへの列の追加

ガイド付きデシジョンテーブルを作成したら、ガイド付きデシジョンテーブルデザイナーで、さまざまなタイプの列を定義して追加できます。

### 前提条件

- ファクトやフィールドなど、列パラメーターに使用されるデータオブジェクトは、ガイド付きデシジョンテーブルと同じパッケージに作成されています。もしくは、ガイド付きデシジョンテーブルデザイナーの **Data Objects** → **New item** で、別のパッケージからインポートされています。

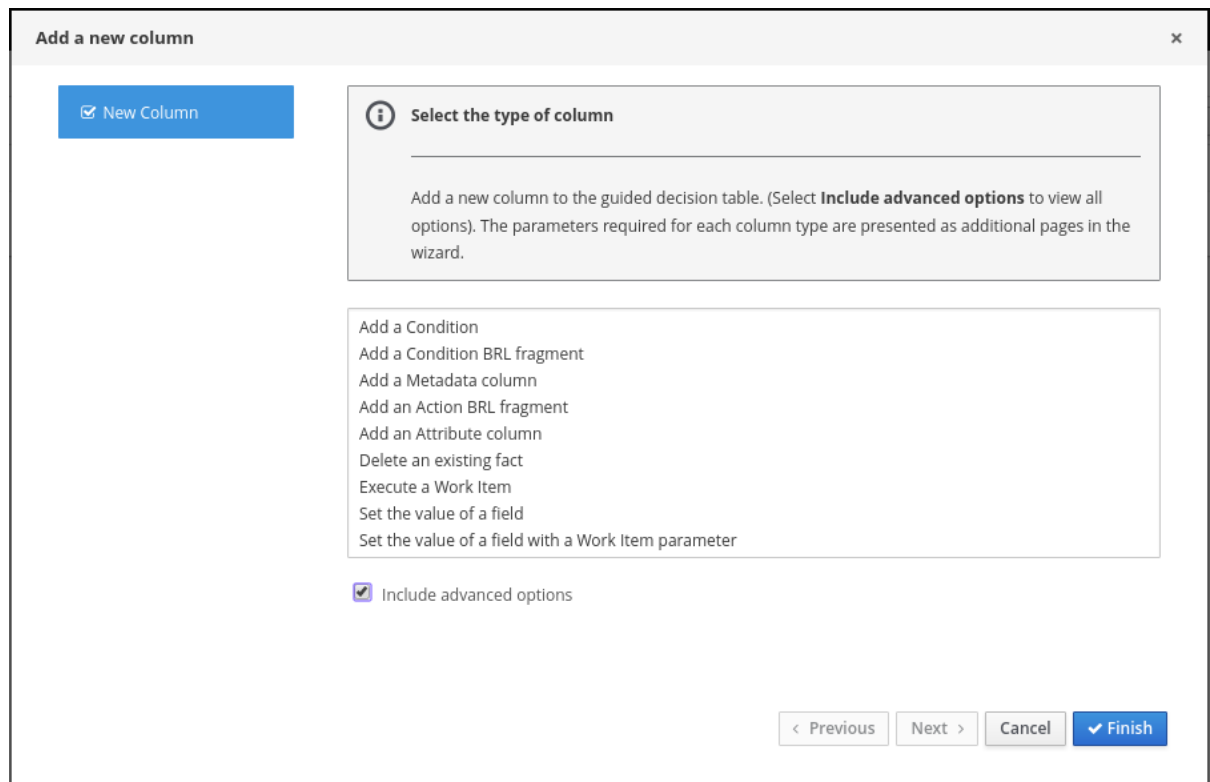
この列パラメーターの説明は「[7章 ガイド付きデシジョンテーブルの列の種類](#)」の「必須の列パラメーター」を参照してください。

データオブジェクトの作成は「[データオブジェクトの作成](#)」を参照してください。

### 手順

- ガイド付きデシジョンテーブルで、**Columns** → **Insert Column** をクリックします。
- Include advanced options** をクリックして、列の全オプションを表示します。

図6.1 列の追加



- 追加する列の種類を選択して **Next** をクリックし、ウィザードの手順に従って、列を追加するのに必要なデータを指定します。  
列の各種類と、設定に必要なパラメーターは「[7章 ガイド付きデシジョンテーブルの列の種類](#)」を参照してください。
- Finish** をクリックして、設定した列を追加します。

列を追加したら、関連するルール行を列に追加し、デシジョンテーブルを完了します。詳細は「[10章 ガイド付きデシジョンテーブルで行の追加およびルールの定義](#)」を参照します。

以下は、ローン申請のデシジョンサービスのデシジョンテーブル例です。

図6.2 完成したガイド付きデシジョンテーブルの例

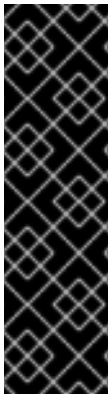
Pricing loans		application : LoanApplication				ome : IncomeSou	application		
#	Description	amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

## 第7章 ガイド付きデシジョンテーブルの列の種類

ガイド付きデシジョンテーブルの **Add a new column** ウィザードは、次の列オプションを提供します (**Include advanced options** を選択して、すべてのオプションを表示します)。

- [Add a Condition \(条件の追加\)](#)
- [Add a Condition BRL fragment \(条件 BRL フラグメントの追加\)](#)
- [Add a Metadata column \(メタデータ列の追加\)](#)
- [Add an Action BRL fragment \(アクション BRL フラグメントの追加\)](#)
- [Add an Attribute column \(属性列の追加\)](#)
- [Delete an existing fact \(既存ファクトの削除\)](#)
- [Execute a Work Item \(作業アイテムの実行\)](#)
- [Set the value of a field \(フィールド値の設定\)](#)
- [Set the value of a field with a Work Item result \(作業アイテムの結果でフィールド値の設定\)](#)

**Add a new column** ウィザードに必要な列タイプとパラメーターは、以下のセクションで説明します。



### 重要: 列パラメーターに必要なデータオブジェクト

ファクトパターン、フィールドなど、このセクションで説明するいくつかの列パラメーターは、ガイド付きデシジョンテーブルと同じパッケージにすでに定義されているデータオブジェクトだけで構成されるドロップダウンオプションを提供します。パッケージで利用可能なデータオブジェクトは、Project Explorer の **Data Objects** パネル、およびガイド付きデシジョンテーブルデザイナーの **Data Objects** タブに一覧表示されます。必要に応じてパッケージにデータオブジェクトを追加したり、ガイド付きデシジョンテーブルデザイナーの **Data Objects** → **New item** で、別のパッケージからインポートしたりできます。データオブジェクトの作成の詳細は「[データオブジェクトの作成](#)」を参照してください。

### 7.1. "ADD A CONDITION"

条件はファクトパターンを表し、ルールの左側 (「WHEN」) に定義されます。この列オプションで、特定のフィールド値を使用して、データオブジェクトが存在するかどうかを確認し、ルールのアクション (「THEN」) 部分に影響を及ぼす条件列を1つ以上定義します。条件テーブルでファクトにバインディングを定義したり、以前定義したものを選択できます。パターンを無効にすることもできます。

#### ルール条件の例

```
when
  $i : IncomeSource( type == "Asset" ) // Binds the IncomeSource object to the $i variable
then
  ...
end
```

```
when
  not IncomeSource( type == "Asset" ) // Negates matching pattern
then
```



...  
end

バインディングを指定したら、フィールド制約を定義できます。同じファクトパターンのバインディングを使用して列を2つ以上定義すると、フィールド制約は、同じパターンに定義された複合フィールド制約になります。1つのモデルクラスに複数のバインディングを定義すると、それぞれのバインディングは、そのルールの条件(「WHEN」)で異なるモデルクラスになります。

## 必須の列パラメーター

この列タイプを設定するには、**Add a new column**ウィザードに以下のパラメーターが必要です。

- **Pattern:** テーブルの条件に使用しているファクトパターンのリストから選択するか、新しいファクトパターンを作成します。ファクトパターンは、パッケージで利用可能なデータオブジェクト(詳細は「**必要なデータオブジェクト**」の注記を参照)と、指定するモデルクラスバインディングの組み合わせとなります(例: **LoanApplication [application]** または **IncomeSource [income]**。括弧内は、指定したファクトタイプへのバインディング)。
- **Entry point:** 可能な場合は、ファクトパターンのエントリーポイントを定義します。エントリーポイントは、指定するとファクトがデシジョンエンジンに組み込まれるゲートまたはストリームです(例: **Application Stream**、**Credit Check Stream**)。
- **Calculation type:** 以下の計算タイプの中から1つ選択します。
  - **Literal value:** 演算子を使用して、セルの値とフィールドを比較します。
  - **Formula:** セルの表現を評価して、フィールドと比較します。
  - **Predicate:** フィールドは必要ありません。表現を **true** または **false** で評価します。
- **Field:** 以前指定したファクトパターンからフィールドを選択します。フィールドオプションは、プロジェクトの **Data Objects** パネルのファクトファイルに定義されます(例: **LoanApplication** ファクトタイプの **amount** フィールドまたは **lengthYears** フィールド)
- **Binding (任意):** 必要に応じて、以前選択したフィールドにバインディングを定義します(例: **LoanApplication [application]** パターン、**amount** フィールド、および **equal to** 演算子に、バインディング **\$amount** を設定すると、終了条件は **application : LoanAppplication(\$amount : amount == [value])** になります)。
- **Operator:** 事前に指定したファクトパターンおよびフィールドに適用する演算子を選択します。
- **Value list (任意):** コンマおよび空白文字で区切った値オプションのリストを入力して、ルールの条件(「WHEN」)部分のテーブル入力データを制限します。この値リストを定義すると、値は、その列のテーブルセルにドロップダウンリストとして提供され、ユーザーは、そこからオプションを1つだけ選択できます(リスト例: **Monday**、**Wednesday**、**Friday** の3つだけを指定可能)。
- **Default value (任意):** 事前定義した値オプションのいずれかを、新しい列のセルに自動的に表示するデフォルト値として選択します。デフォルト値が指定されていないと、テーブルのセルはデフォルトでは空欄となります。また、Project Explorer の **Enumeration Definitions** パネルにリストした、プロジェクトに事前に設定したデータの列挙からデフォルト値を選択できます(列挙は、**Menu** → **Design** → **Projects** → **[select project]** → **Add Asset** → **Enumeration** から作成できます)。
- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

### 7.1.1. 条件列セルへの **any other** 値の追加

ガイド付きデシジョンテーブルにおける単純な条件列では、以下のパラメーターを設定した場合に、列のテーブルセルに **any other** 値を適用できます。

- 条件列の **Calculation type** を **Literal value** に設定している。
- **Operator** を、等価演算子 **==** または非等価演算子 **!=** に設定している。

**any other** の値を使用すると、テーブルに既存のルールに明示的に定義されていない他のフィールド値にルールを定義できます。DRL ソースでは、**any other** は **not in** と表記されます。

#### **any other** に使用される **not in** が設定されたルール条件の例

```
when
  IncomeSource( type not in ("Asset", "Job") )
  ...
then
  ...
end
```

#### 手順

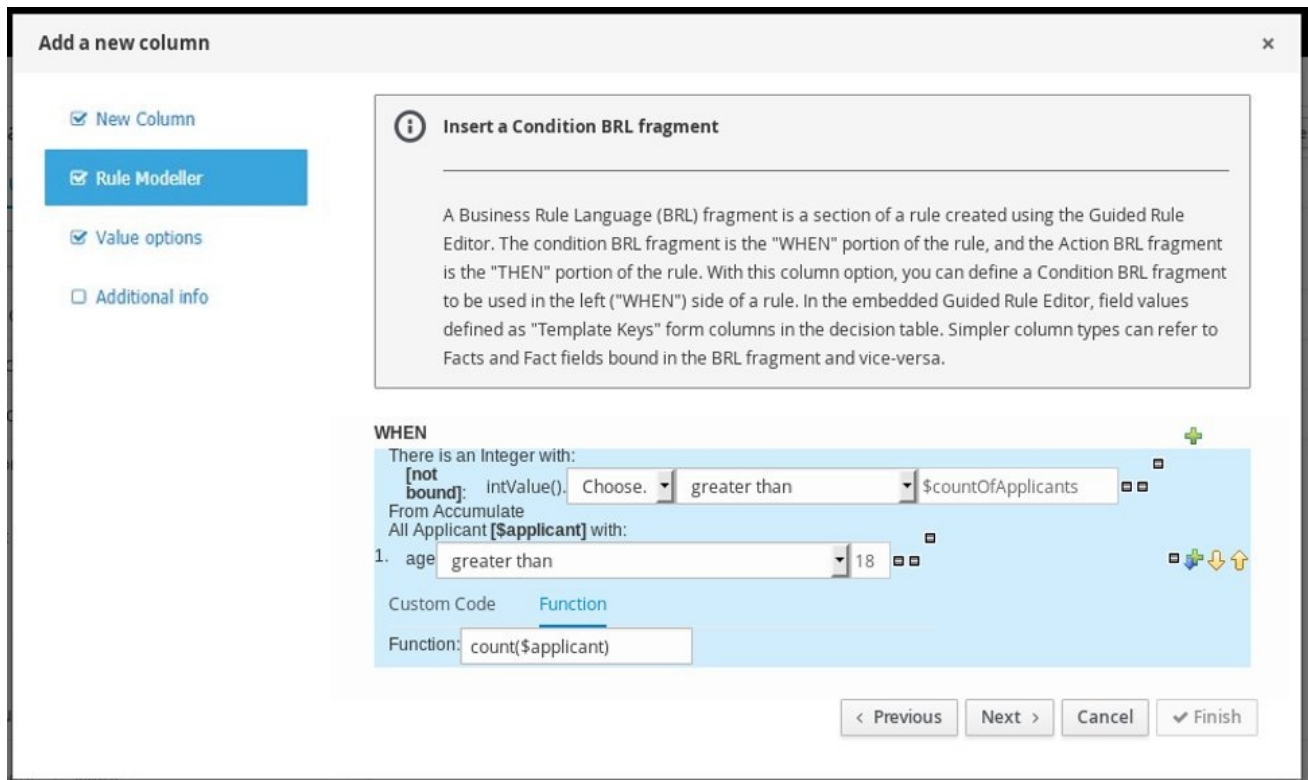
1. **==** 演算子または **!=** 演算子を使用する条件列のセルを選択します。
2. テーブルデザイナーの右上のツールバーで、**Edit** → **Insert "any other" value** をクリックします。

## 7.2. "ADD A CONDITION BRL FRAGMENT"

BRL (Business Rule Language) フラグメントは、ガイド付きルールデザイナーを使用して作成したセクションです。条件 BRL フラグメントはルールの「WHEN」部分で、[action BRL fragment \(アクション BRL フラグメント\)](#) はルールの「THEN」の部分です。この列オプションを使用して、ルールの左側 (WHEN) 部分で使用する条件 BRL フラグメントを定義できます。単純な列タイプは、BRL フラグメントでバインドされているファクトおよびファクトフィールドを参照でき、これらのファクトおよびファクトのフィールドから列タイプへの参照も可能です。

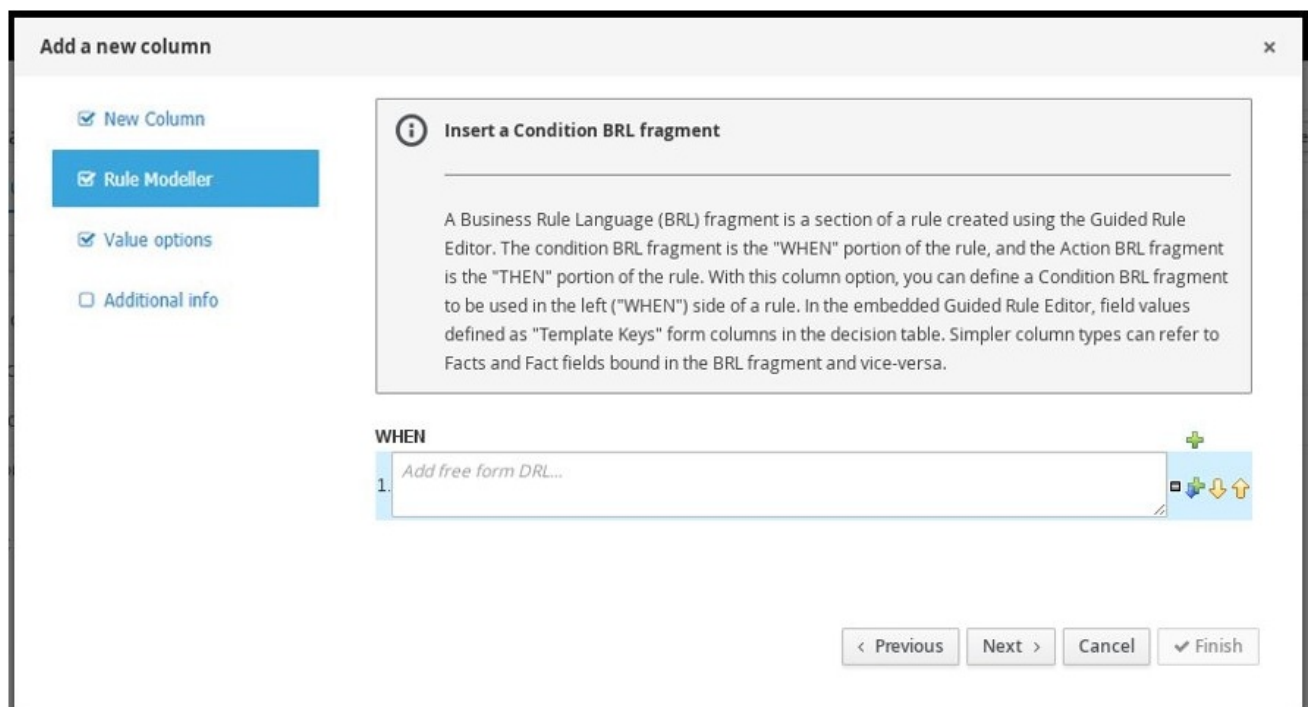
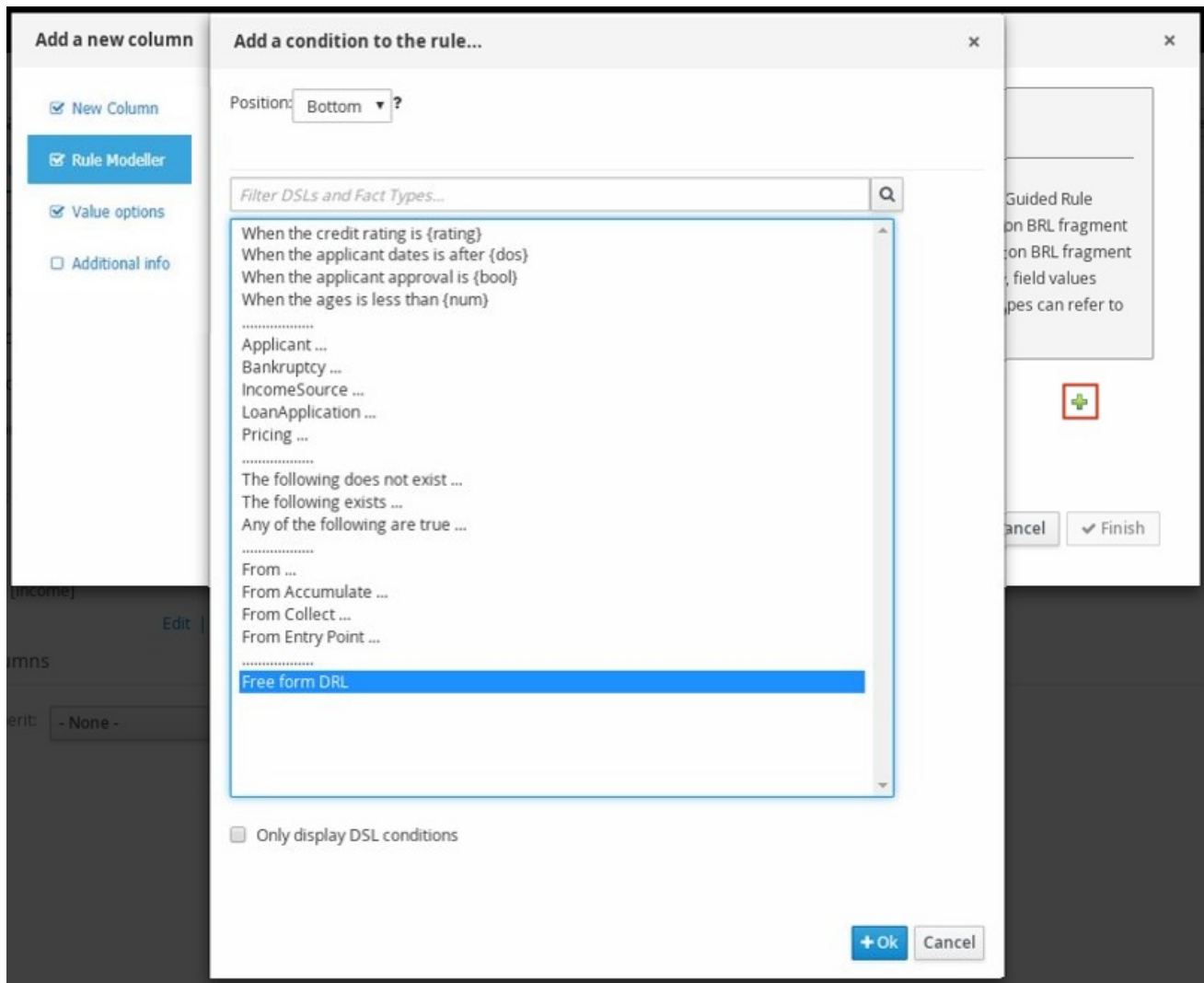
以下の例は、ローン申請の条件 BRL フラグメントです。

図7.1 組み込みガイド付きルールデザイナーを使用する条件 BRL フラグメントの追加



条件オプションのリストから **Free form DRL** を選択して、組み込みガイド付きルールデザイナーを使用せずに条件 BRL フラグメントを定義します。

図7.2 フリーフォーム DRL を使用する条件 BRL フラグメントの追加





## テンプレートキー

条件 BRL フラグメントにフィールドを追加すると、値オプションの1つが (**Literal** または **Formula** ではなく) **テンプレートキー** になります。テンプレートキーは、ガイド付きデシジョンテーブルを生成時に指定した値と切り替え可能なプレースホルダー変数で、指定したテンプレートキーごとに、テーブル内で別の列を形成します。**Value options** ページでテンプレートキーのデフォルト値を指定できます。デシジョンテーブルでは Literal および Formula の値は静的ですが、テンプレートキーの値は必要に応じて修正できます。

組み込みのガイド付きルールデザイナーでは、**Template key** フィールドオプションを選択し、エディターに **\$key** 書式で値を入力し、テンプレートのキー値をフィールドに追加できます。たとえば、**\$age** は、デシジョンテーブルに **\$age** 列を作成します。

フリーフォーム DRL では、**@{key}** の書式でテンプレートのキー値をファクトに追加できます。たとえば、**Person( age > @{age} )** にすると、デシジョンテーブルに **\$age** 列が作成されます。

テンプレートキーを使用して追加した新しい列のデータ型は String です。

## 必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Rule Modeller:** ルールの条件 BRL フラグメント ("WHEN" 部分) を定義します。
- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

## 7.3. "ADD A METADATA COLUMN"

この列オプションを使用して、デシジョンテーブルでメタデータを列として定義できます。各列には、DRL ルールの通常のメタデータアノテーションが表示されます。デフォルトでは、メタデータ列は非表示になっています。列を表示するには、ガイド付きデシジョンテーブルデザイナーで **Edit Columns** をクリックし、**Hide column** チェックボックスの選択を解除します。

## 必須の列パラメーター

この列タイプを設定する **Add a new column** ウィザードには、以下のパラメーターが必要です。

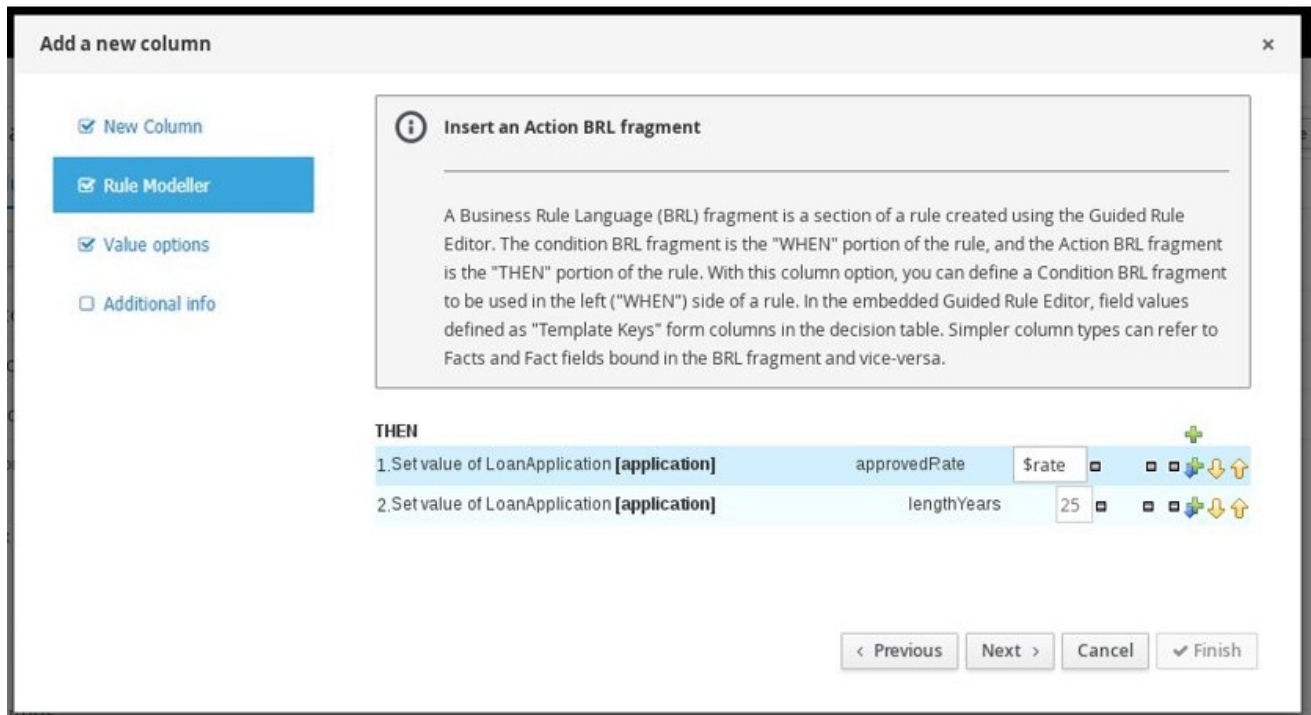
- **Metadata:** Java 変数書式のメタデータ項目の名前を入力します (つまり、空白文字または特殊文字を使用することはできません)。

## 7.4. "ADD AN ACTION BRL FRAGMENT"

BRL (Business Rule Language) フラグメントは、ガイド付きルールデザイナーを使用して作成したルールのセクションです。**条件 BRL フラグメント** はルールの「WHEN」部分で、アクション BRL フラグメントはルールの「THEN」部分です。この列オプションを使用すると、ルールの右側 (THEN) で使用するアクション BRL フラグメントを定義できます。BRL フラグメントでバインドされているファクトおよびファクトのフィールドは簡単な列タイプで参照でき、これらのファクトおよびファクトのフィールドから列タイプの参照も可能です。

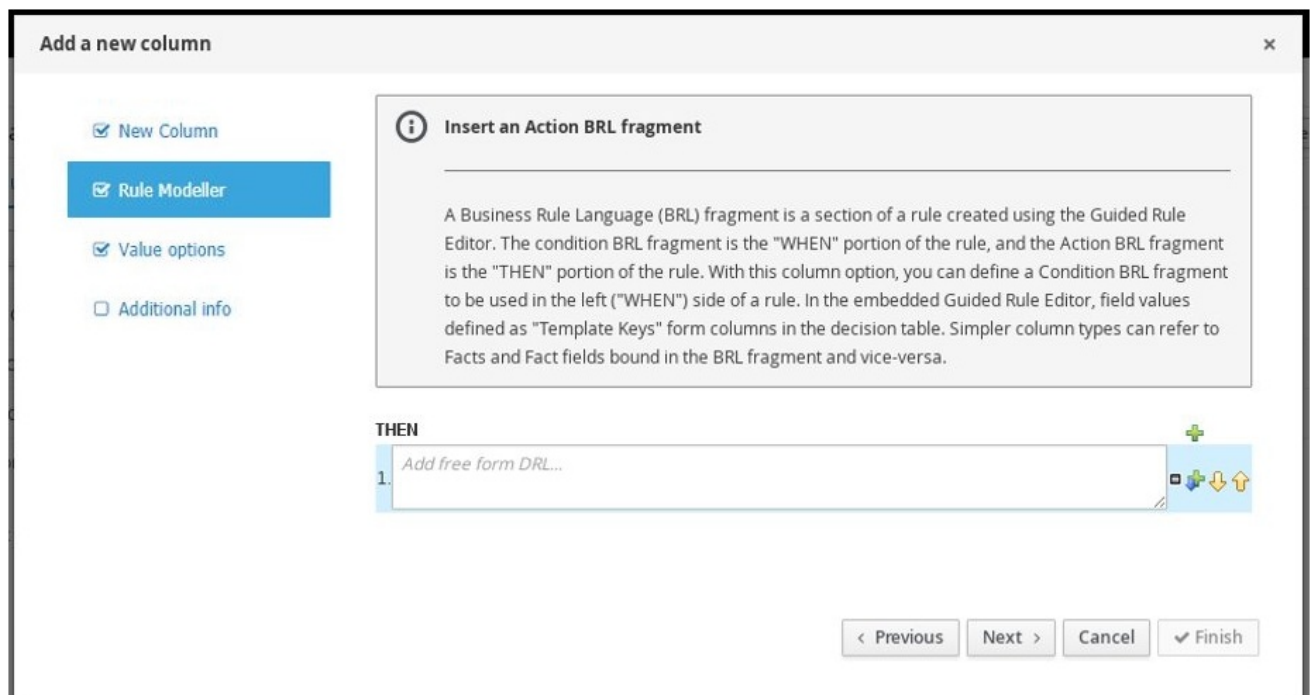
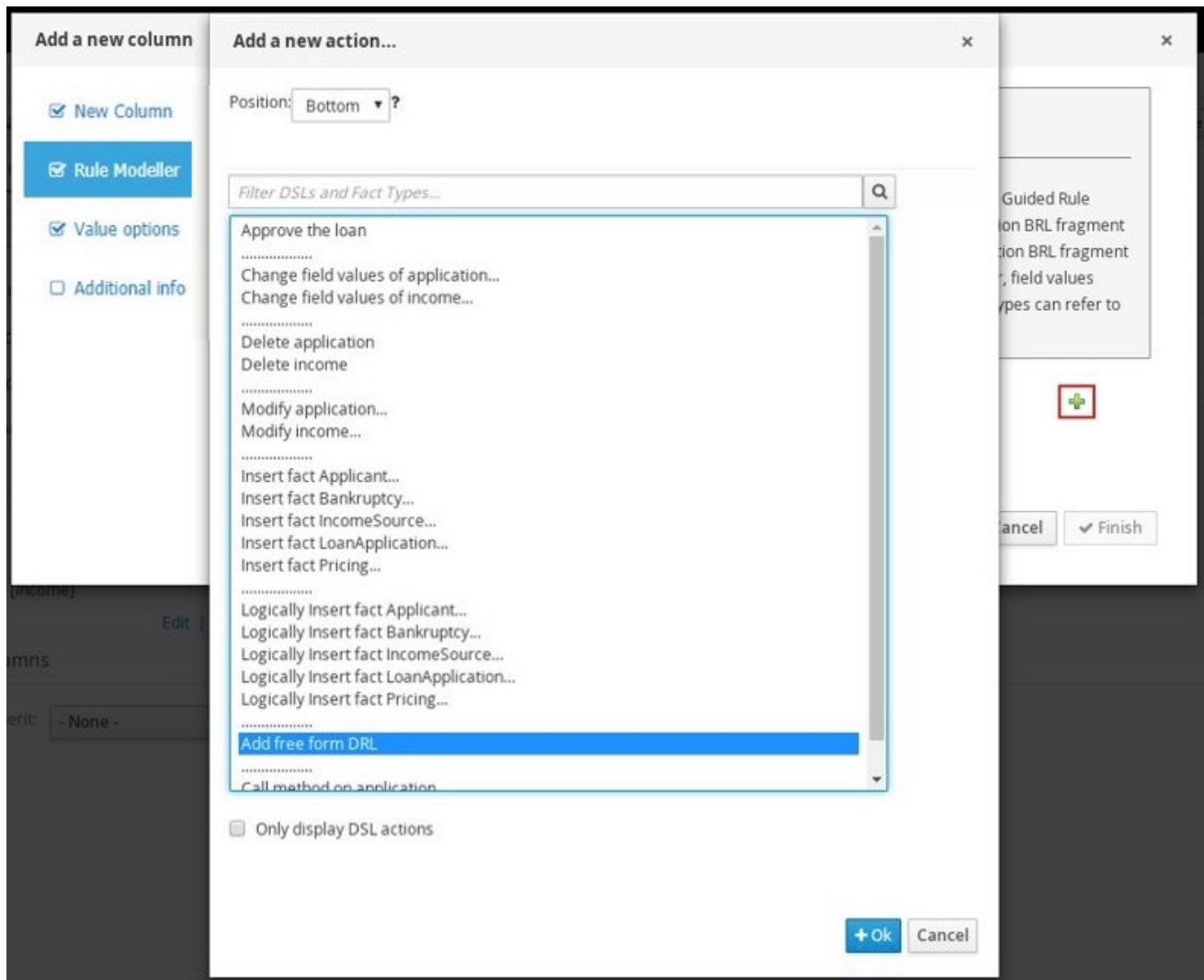
以下の例は、ローン申請のアクション BRL フラグメントです。

図7.3 組み込みガイド付きルールデザイナーを使用するアクション BRL フラグメントの追加



アクションオプションのリストから **Add free form DRL** を選択して、組み込みガイド付きルールデザイナーを使用しないアクション BRL フラグメントを定義します。

図7.4 フリーフォーム DRL を使用するアクション BRL フラグメントの追加





## テンプレートキー

アクション BRL フラグメントにフィールドを追加すると、値オプションの1つが (Literal または Formula ではなく) テンプレートキー になります。テンプレートキーは、ガイド付きデシジョンテーブルを生成時に指定した値と切り替え可能なプレースホルダー変数で、指定したテンプレートキーごとに、テーブル内で別の列を形成します。Value options ページでテンプレートキーのデフォルト値を指定できます。デシジョンテーブルでは Literal および Formula の値は静的ですが、テンプレートキーの値は必要に応じて修正できます。

組み込みのガイド付きルールデザイナーでは、Template key フィールドオプションを選択し、エディターに \$key 書式で値を入力し、テンプレートのキー値をフィールドに追加できます。たとえば、\$age は、デシジョンテーブルに \$age 列を作成します。

フリーフォーム DRL では、@{key} の書式でテンプレートのキー値をファクトに追加できます。たとえば、Person( age > @{age} ) にすると、デシジョンテーブルに \$age 列が作成されます。

テンプレートキーを使用して追加した新しい列のデータ型は String です。

## 必須の列パラメーター

この列タイプを設定するには、Add a new column ウィザードに以下のパラメーターが必要です。

- Rule Modeller: ルールのアクション BRL フラグメントの定義) (「THEN」部分)
- Header (説明): 列にヘッダーテキストを追加します。
- Hide column: 選択すると列が非表示になり、選択を解除すると列が表示されます。

## 7.5. "ADD AN ATTRIBUTE COLUMN"

この列オプションを使用して、Salience、Enabled、Date-Effective などの DRL ルール属性を表現する属性列を1つ以上追加できます。

たとえば、以下のデシジョンテーブルでは salience 属性を使用してルールの優先度を、enabled 属性を使用して評価のルールを有効化または無効化します。salience の値が大きいルールが最初に評価され、enabled 属性が指定されたルールは、チェックボックスが選択されている場合にのみ、評価されます。

図7.5 評価の動きを定義する salience 属性および enabled 属性が含まれるルールの例

Pricing loans		application : LoanApplication						ome
#	Description	salience	enabled	amount min	amount max	period	deposit max	
1		100	<input checked="" type="checkbox"/>	131000	200000	30	20000	
2			<input checked="" type="checkbox"/>	10000	100000	20	2000	
3			<input type="checkbox"/>	100001	130000	20	3000	

## ルール属性を含むルールソースの例

```
rule "Row 1 Pricing loans"
  salience 100
  enabled true
  when
```



```

...
then
...
end
...
rule "Row 3 Pricing loans"
  enabled false
  when
    ...
  then
    ...
  end
end

```

各属性の説明について、ウィザードのリストから属性を選択します。



### ヒットポリシーおよび属性

デシジョンテーブルに定義したヒットポリシーに応じて、ヒットポリシーが内部的に使用されているため、一部の属性を無効にできます。たとえば、**Resolved Hit** ポリシーをこのテーブルに割り当てて、テーブルに指定した優先順位に従って行 (ルール) を適用すると、Salience 属性が使用されなくなります。Salience 属性は、定義した salience (優先順位) 値に従って、ルールの優先順位をエスカレーションし、値は、テーブルの **Resolved Hit** ポリシーによって上書きされるためです。

### 必須の列パラメーター

この列タイプを設定する **Add a new column** ウィザードには、以下のパラメーターが必要です。

- **属性:** 列に適用する属性を選択します。

## 7.6. "DELETE AN EXISTING FACT"

この列のオプションを使用して、これまでにファクトパターンとしてテーブルに追加したファクトを削除するアクションを実装できます。この列を作成すると、ファクトタイプは、その列のテーブルセルにドロップダウンリストとして提供され、ユーザーは、そこからオプションを1つだけ選択できます。

### 必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

## 7.7. "EXECUTE A WORK ITEM"

この列オプションを使用して、Business Central に以前作成した作業アイテム定義に基づいて、作業アイテムハンドラーを実行できます (作業アイテムは、**Menu → Design → Projects → [select project] → Add Asset → Work Item definition** で作成できます)。

### 必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Work Item:** 事前設定した作業アイテムのリストから選択します。

- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。

## 7.8. "SET THE VALUE OF A FIELD"

この列オプションを使用して、ルールの「THEN」部分に事前にバインドしたファクトにフィールドの値を設定するアクションを実装できます。その他のルールを再度アクティブにするように修正した値をデシジョンエンジンに通知するオプションがあります。

### 必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Pattern:** テーブルの条件または条件 BRL フラグメントに使用しているファクトパターンのリストから選択するか、新しいファクトパターンを作成します。ファクトパターンは、パッケージで利用可能なデータオブジェクト (詳細は「[必要なデータオブジェクト](#)」の注記を参照) と、指定するモデルクラスバインディングの組み合わせとなります (例: **LoanApplication [application]** または **IncomeSource [income]**。括弧内は、指定したファクトタイプへのバインディング)。
- **Field:** 以前指定したファクトパターンからフィールドを選択します。フィールドオプションは、プロジェクトの **Data Objects** パネルのファクトファイルに定義されます (例: **LoanApplication** ファクトタイプの **amount** フィールドまたは **lengthYears** フィールド)
- **Value list (任意):** 値オプションをコンマおよび空白文字で区切ったリストを入力して、ルールのアクション (「THEN」) 部分に対するテーブルの入力データを制限します。この値リストを定義すると、値は、その列のテーブルセルにドロップダウンリストとして提供され、ユーザーは、そこからオプションを1つだけ選択できます (リスト例: **Accepted, Declined, Pending**)。
- **Default value (任意):** 事前定義した値オプションのいずれかを、新しい列のセルに自動的に表示するデフォルト値として選択します。デフォルト値が指定されていないと、テーブルのセルはデフォルトでは空欄となります。また、Project Explorer の **Enumeration Definitions** パネルにリストした、プロジェクトに事前に設定したデータの列挙からデフォルト値を選択できます (列挙は、**Menu** → **Design** → **Projects** → **[select project]** → **Add Asset** → **Enumeration** から作成できます)。
- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。
- **Logically insert (論理的な挿入):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの別の列に現在使用されていない場合に表示されます (次のフィールドの説明を参照)。ファクトパターンをデシジョンエンジンに論理的に挿入する場合はこれを選択し、定期的に挿入する場合は選択を解除します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。定期的な挿入、または指定した挿入の後に、ファクトを明示的に取り消す必要があります。論理挿入の後に、ファクトをアサートした条件が TRUE ではなくると、ファクトは自動的に取り消されます。
- **Update engine with changes (変更でエンジンをアップデート):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの別の列ですでに使用されている場合に表示されます。フィールドの値を変更してデシジョンエンジンを更新する場合には、これを選択し、デシジョンエンジンを更新しない場合には、この選択を解除します。

## 7.9. "SET THE VALUE OF A FIELD WITH A WORK ITEM RESULT"

この列オプションを使用して、ルールの「THEN」部分の作業アイテムハンドラーの結果値に、事前定義したファクトフィールドの値を設定するアクションを実行できます。作業アイテムは、return パラメーターにフィールドを設定するために、バインドしたファクトのフィールドと同じデータ型の結果パラメーターを設定する必要があります (作業アイテムは **Menu → Design → Projects → [select project] → Add Asset → Work Item definition** に作成できます)。

**Execute a Work Item (作業アイテムの実行)** 列は、この列オプションを作成するために、テーブルですでに作られている必要があります。

### 必須の列パラメーター

この列タイプを設定するには、**Add a new column** ウィザードに以下のパラメーターが必要です。

- **Pattern:** テーブルに使用しているファクトパターンのリストから選択するか、新しいファクトパターンを作成します。ファクトパターンは、パッケージで利用可能なデータオブジェクト (詳細は「**必要なデータオブジェクト**」の注記を参照) と、指定するモデルクラスバインディングの組み合わせとなります (例: **LoanApplication [application]** または **IncomeSource [income]**。括弧内は、指定したファクトタイプへのバインディング)。
- **Field:** 以前指定したファクトパターンからフィールドを選択します。フィールドオプションは、プロジェクトの **Data Objects** パネルのファクトファイルに定義されます (例: **LoanApplication** ファクトタイプの **amount** フィールドまたは **lengthYears** フィールド)
- **Work Item:** 事前定義した作業アイテムのリストから選択します (作業アイテムは、return パラメーターにフィールドを設定するために、バインドしたファクトにフィールドと同じデータ型の result パラメーターを設定する必要があります)。
- **Header (説明):** 列にヘッダーテキストを追加します。
- **Hide column:** 選択すると列が非表示になり、選択を解除すると列が表示されます。
- **Logically insert (論理的な挿入):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの別の列に現在使用されていない場合に表示されます (次のフィールドの説明を参照)。ファクトパターンをデシジョンエンジンに論理的に挿入する場合はこれを選択し、定期的に挿入する場合は選択を解除します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。定期的な挿入、または指定した挿入の後に、ファクトを明示的に取り消す必要があります。論理挿入の後に、ファクトをアサートした条件が TRUE でなくなると、ファクトは自動的に取り消されます。
- **Update engine with changes (変更でエンジンをアップデート):** このオプションは、選択したファクトパターンが、ガイド付きデシジョンテーブルの別の列ですでに使用されている場合に表示されます。フィールドの値を変更してデシジョンエンジンを更新する場合には、これを選択し、デシジョンエンジンを更新しない場合には、この選択を解除します。

## 第8章 ガイド付きデシジョンテーブルのルール名列の表示

必要に応じて、ガイド付きデシジョンテーブルで **Rule Name** 列を表示できます。

### 手順

1. ガイド付きデシジョンテーブルで、**Columns** をクリックします。
2. **Show rule name column**のチェックボックスを選択します。
3. **Finish** をクリックして保存します。

デフォルトのルール名形式は **Row (row\_number)(table\_name)** です。**Source** には、ルール名を指定しない場合にはデフォルト値が含まれます。ガイド付きデシジョンテーブルでは、**Rule Name** の列にルール名を追加して、デフォルト値を上書きできます。

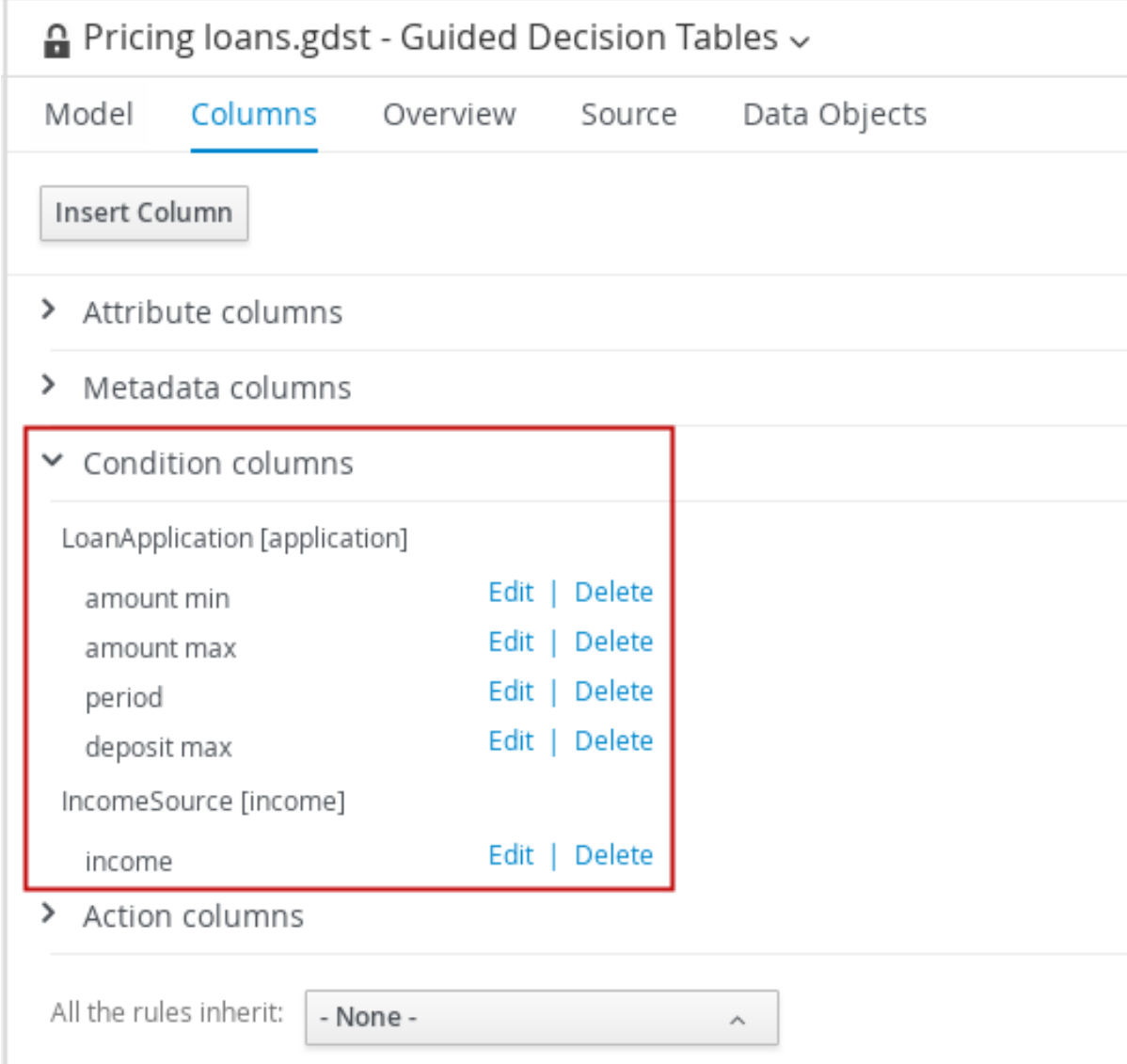
## 第9章 ガイド付きデシジョンテーブルで列の編集または削除

ガイド付きデシジョンテーブルデザイナーに作成した列は、いつでも編集または削除できます。

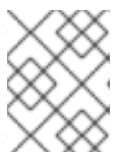
### 手順

1. ガイド付きデシジョンテーブルで、**Columns** をクリックします。
2. 適切なセクションを展開し、列名の隣にある **Edit** または **Delete** をクリックします。

図9.1 列の編集または削除



The screenshot shows the 'Pricing loans.gdst - Guided Decision Tables' interface. The 'Columns' tab is selected. Below the 'Insert Column' button, there are sections for 'Attribute columns', 'Metadata columns', 'Condition columns', and 'Action columns'. The 'Condition columns' section is expanded and highlighted with a red box. It contains two groups of columns: 'LoanApplication [application]' and 'IncomeSource [income]'. Each group lists several columns with 'Edit' and 'Delete' buttons next to them. At the bottom, there is a dropdown menu for 'All the rules inherit:' set to '- None -'.



### 注記

既存のアクション列が、条件列と同じパターン一致パラメーターを使用している場合は、条件列を削除できません。

3. 列を変更したら、ウィザードの **Finish** をクリックして保存します。

## 第10章 ガイド付きデシジョンテーブルで行の追加およびルールの定義

ガイド付きデシジョンテーブルで列を作成したら、ガイド付きデシジョンテーブルデザイナーに行を追加してルールを定義します。

### 前提条件

- 「6章ガイド付きデシジョンテーブルへの列の追加」の手順に従って、ガイド付きデシジョンテーブルの列が追加されている。

### 手順

- ガイド付きデシジョンテーブルデザイナーで、**Insert** → **Append row**、またはいずれかの **Insert row** オプションをクリックします (もしくは、**Insert column** をクリックして列ウィザードを開いて、新しい列を定義することもできます)。

図10.1 行の追加

Pricing loans		application : LoanApplication				ome : IncomeSou	application		
#	Description	amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4

- 各セルをダブルクリックしてデータを入力します。入力する値が決まっている場合は、セルのドロップダウンオプションから選択します。

図10.2 各セルに入力データの入力

Pricing loans		application : LoanApplication				ome : IncomeSou	application		
#	Description	amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

- ガイド付きデシジョンテーブルですべてのデータ行を定義したら、ガイド付きデシジョンテーブルデザイナーの右上のツールバーで **Validate** をクリックして、テーブルの妥当性を確認します。テーブルの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、テーブルの全コンポーネントを見直し、エラーが表示されなくなるまでテーブルの妥当性確認を行います。



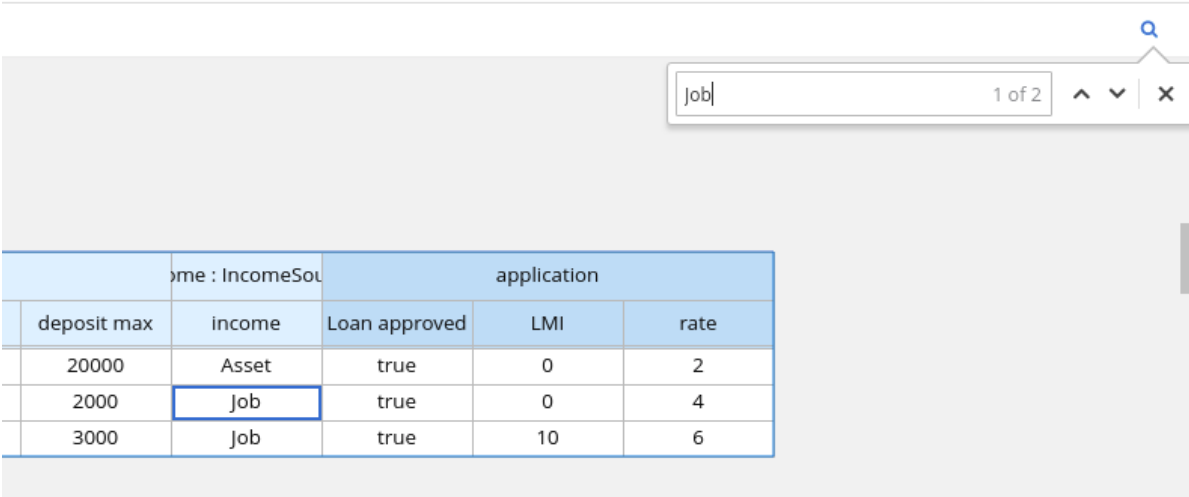
### 注記

ガイド付きデシジョンテーブルには、リアルタイム検証および妥当性確認がありますが、最善の結果を確実に得るために、完了したデシジョンテーブルの妥当性確認を手動で行うことができます。

- テーブルデザイナーで **Save** をクリックして、変更を保存します。  
ガイド付きのデシジョンテーブルのコンテンツを定義した後に、ガイド付きデシジョンテーブルに表示されているテキストを検索するには、ガイド付きデシジョンテーブルデザイナーの右上隅の検索バーを使用します。検索機能は、特に多数の値が指定された、複雑なガイド付きデ

シジョンテーブルで有用です。

図10.3 ガイド付きデシジョンテーブルのコンテンツ検索



The screenshot shows a search interface for a decision table. A search bar at the top right contains the text 'Job' and '1 of 2' results. Below the search bar is a table with the following data:

	ome : IncomeSou	application		
deposit max	income	Loan approved	LMI	rate
20000	Asset	true	0	2
2000	Job	true	0	4
3000	Job	true	10	6

## 第11章 ルールアセットのドロップダウンリストの列挙定義

Business Central での列挙定義では、ガイド付きルール、ガイド付きルールテンプレート、ガイド付きデシジョンテーブルの条件やアクションのフィールドで使用可能な値を指定します。列挙定義には、ルールアセットで該当するフィールドのドロップダウンリストとして表示される対応値一覧に対する **fact.field** マッピングが含まれています。列挙定義と同じファクトとフィールドをベースにしたフィールドを選択すると、定義した値のドロップダウンリストが表示されます。

列挙は、Business Central または Red Hat Process Automation Manager プロジェクトの DRL ソースで定義できます。

### 手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Enumeration** をクリックします。
3. 分かりやすい **Enumeration** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なデータオブジェクトと適切なルールアセットが割り当てられているか、これから割り当てるパッケージと同じでなければなりません。
4. **Ok** をクリックして列挙を作成します。  
**Project Explorer** の **Enumeration Definitions** パネルに、新しい列挙が追加されました。
5. 列挙デザイナーの **Model** タブで、**Add enum** をクリックし、以下の列挙値を定義します。
  - **Fact:** この列挙を関連付けるプロジェクトの同じパッケージ内に、既存のデータオブジェクトを指定します。**Project Explorer** で **Data Objects** パネルを開き、利用可能なデータオブジェクトを表示するか、必要に応じて新規アセットとして適切なデータオブジェクトを作成します。
  - **Field:** **Fact** 用に選択したデータオブジェクトの一部として定義した既存のフィールド ID を指定します。**Project Explorer** で **Data Objects** パネルを開き、適切なデータオブジェクトを選択して、利用可能な **Identifier** オプションの一覧を表示します。必要に応じて、データオブジェクトに関連する ID を作成してください。
  - **Context:** **Fact** と **Field** の定義にマッピングする **['string1','string2','string3']** または **[integer1,integer2,integer3]** 形式の値一覧を定義します。これらの値は、ルールアセットの適切なフィールドに、ドロップダウンリストとして表示されます。

たとえば、以下の列挙は、ローン申請デシジョンサービスの申請者でクレジットスコアに使用するドロップダウンの値を定義します。

図11.1 Business Central での申請者のクレジットスコアの列挙例

Model Overview Source			
Add enum			
Fact	Field	Context	
Applicant	creditRating	['AA', 'OK', 'Sub prime']	<span>Remove</span>

### DRL ソースの申請者のクレジットスコアの列挙例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```



以下の例では、プロジェクトと同じパッケージ内にあり、**Applicant** データオブジェクトと **creditRating** フィールドを使用するガイド付きルール、ガイド付きルールテンプレートまたはガイド付きデシジョンテーブルであれば、設定値がドロップダウンオプションとして利用できます。

図11.2 ガイド付きルールまたはガイド付きルールテンプレートでの列挙ドロップダウンオプション例

WHEN

1. There is a LoanApplication [app]
  - Any of the following are true:
    - There is an Applicant with:
      - creditRating equal to
    - There is an Applicant with:
      - creditRating equal to

THEN

1. Set value of LoanApplication [app]
  - approved
2. Set value of LoanApplication [app]
  - explanation
2. delete LoanApplication [app]

図11.3 ガイド付きデシジョンテーブルでの列挙ドロップダウンオプション例

Model Columns Overview Source Data Objects

Pricing loans		application : LoanApplication				income : IncomeSource	applicant : Applicant	application	
#	Description	amount min	amount max	period	deposit max	income	creditRating	Loan approved	LMI
1		131000	200000	30	20000	Asset	AA	true	0
2		10000	100000	20	2000	Job	AA	true	0
3		100001	130000	20	3000	Job	OK Sub prime	true	10

## 11.1. ルールアセットの詳細列挙オプション

Red Hat Process Automation Manager プロジェクトの列挙定義を使用した詳細ユースケースについては、列挙を定義する時に、以下の拡張オプションの使用を検討してください。

### Business Central の値との DRL の値におけるマッピング

DRL ソースと Business Central インターフェースで、異なる列挙値またはより複雑な列挙値を表示するには、列挙の定義値に **'fact.field' : ['sourceValue1=UIValue1','sourceValue2=UIValue2', ... ]** 形式のマッピングを使用します。

たとえば、ローンの状態に関する以下の列挙定義では、**A** または **D** のオプションを DRL ファイルで使用しますが、Business Central では **Approved** または **Declined** のオプションが表示されます。

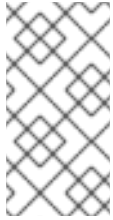
```
'Loan.status' : ['A=Approved','D=Declined']
```

### 列挙値の依存関係

選択した値を1つのドロップダウンリストにまとめて、後続のドロップダウンリストで利用可能なオプションを判断する場合は、列挙定義で **'fact.fieldB[fieldA=value1]' : ['value2', 'value3', ... ]** の形式を使用します。

たとえば、保険契約に関する以下の列挙定義では、**policyType** フィールドに **Home** または **Car** の値を使用できます。ユーザーが選択する保険契約タイプにより、利用できる契約 **coverage** のフィールドオプションが決まります。

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```



## 注記

列挙依存関係は、ルールの条件およびアクションをまたいで適用されません。たとえば、保険契約のユースケースでは、ルール条件で選択した契約をもとに、ルールアクションで利用可能な補償オプションが決定されるわけではありません (該当する場合)。

## 列挙の外部データソース

列挙定義で直接、値を定義するのではなく、外部のデータソースから列挙値の一覧を取得する場合には、プロジェクトのクラスパスで、文字列の `java.util.List` の一覧を返すヘルパークラスを追加します。列挙定義で、値を指定する代わりに、外部の値を取得するように設定したヘルパークラスを特定します。

たとえば、ローン申請者の地域に関する以下の列挙定義では、`'Applicant.region' : ['country1', 'country2', ... ]` の形式で明示的に申請者の地域を定義するのではなく、外部で定義した値の一覧を返すヘルパークラスを使用します。

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

この例では、`DataHelper` クラスに、文字列の一覧を返す `getListOfRegions()` メソッドが含まれます。列挙は、ルールアセットの関連フィールドのドロップダウンリストに、読み込まれます。

また、通常通り従属フィールドを特定して、引用符でヘルパーへの呼び出しを括弧することで、ヘルパークラスから動的に、従属の列挙定義を読み込むこともできます。

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}')
```

リレーショナルデータベースなど、外部データソースから列挙データすべてを読み込む場合には、`Map<String, List<String>>` マッピングを返す、Java クラスを実装できます。マップのキーは、`fact.field` マッピングで、値は、値の `java.util.List<String>` 一覧です。

たとえば、以下の Java クラスでは、関連する列挙のローン申請者の地域を定義します。

```
public class SampleDataSource {

    public Map<String, List<String>> loadData() {
        Map data = new HashMap();

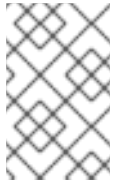
        List d = new ArrayList();
        d.add("AU");
        d.add("DE");
        d.add("ES");
        d.add("UK");
        d.add("US");
        ...
        data.put("Applicant.region", d);

        return data;
    }
}
```

以下の列挙定義は、この Java クラスの例に相关します。参照は Java クラスで定義されているので、列挙にはファクトまたはフィールド名への参照は含まれません。

```
=(new SampleDataSource()).loadData()
```

= 演算子を使用して、Business Central がヘルパークラスから全列挙データを読み込めるようにします。エディターで使用するよう列挙定義を要求すると、ヘルパーメソッドが静的に評価されます。



### 注記

現在、Business Central では、ファクトおよびフィールド定義なしで列挙を定義することはできません。この方法で関連の Java クラスの列挙を定義するには、Red Hat Process Automation Manager プロジェクトで DRL ソースを使用します。

## 第12章 ガイド付きデシジョンテーブルのリアルタイム検証および妥当性確認

Business Central は、ガイド付きデシジョンテーブルにリアルタイム検証および妥当性確認を提供し、テーブルのエラーがなくなったことを確認します。ガイド付きデシジョンテーブルは、各セルが変更するたびに妥当性が確認されます。ロジックに問題が検出されたら、エラー通知が表示され、問題を確認できます。

### 12.1. ガイド付きデシジョンテーブルの問題の種類

検証および妥当性確認の機能は、以下のタイプの問題を検出します。

#### 冗長性 (Redundancy)

冗長性は、デシジョンテーブルの2つの行で、同じファクトセットの同じ結果を実行する際に生じます。たとえば、顧客の誕生日をチェックして誕生日割引を提供する行が2つあると、割引は2倍になる可能性があります。

#### 包含 (Subsumption)

包含は冗長と似ていますが、2つルールが同じ結果を実行し、1つのルールがもう1つのルールのファクトのサブセットに対して実行する場合に生じます。たとえば、以下の2つのルールを見ましょう。

- when Person age > 10 then Increase Counter
- when Person age > 20 then Increase Counter

この場合、対象者の年齢が15歳の場合はルールが1つだけ実行されますが、対象者の年齢が20歳を超えている場合は2つのルールが実行されます。これにより、実行時に冗長性の場合と同様の問題が生じます。

#### 競合 (Conflict)

デシジョンテーブルの2つの行(ルール)または2つのセルの条件が類似し、異なる結果が設定されている場合は、競合が発生する場合があります。

以下の例は、デシジョンテーブルの2つの行で競合が発生しているのを示しています。

- when Deposit > 20000 then Approve Loan
- when Deposit > 20000 then Refuse Loan

この場合、ローンが承認されるかどうかについて確認することはできません。

以下の例は、デシジョンテーブルの2つのセル間の競合を示します。

- when Age > 25
- when Age < 25

競合セルを持つ行は実行しません。

#### Unique Hit ポリシーの違反 (Broken Unique Hit Policy)

Unique Hit ポリシーをデシジョンテーブルに適用する際は、同時に1行しか実行できず、各列は一意であり、満たした条件が重複しないようにする必要があります。複数の行が実行した場合は、検証レポートが、違反したヒットポリシーを特定します。たとえば、価格の割引資格を決定するテーブルで、以下の条件を見てください。

- when Is Student = true
- when Is Military = true

顧客が学生であり、軍隊に所属している場合は、両方の条件が適用され、**Unique Hit** ポリシーに違反します。したがって、この種のテーブルの行は、複数の行が同時に発生しないように作成する必要があります。ヒットポリシーについては「5章 [ガイド付きデシジョンテーブルのヒットポリシー](#)」を参照してください。

### 欠陥 (Deficiency)

欠陥は競合と似ていますが、デシジョンテーブルのルールのロジックが未完成の場合に生じます。たとえば、欠陥がある以下の2つのルールを見てみましょう。

- when Age > 20 then Approve Loan
- when Deposit < 20000 then Refuse Loan

この2つのルールは、20歳を超え、預金が20000より少ない場合に混乱が発生する場合があります。さらに制約を追加すると、競合を避けることができます。

### 列の欠落 (Missing Column)




列を削除したため、不完全または不正確なロジックが発生した場合は、ルールが正しく発生しません。これを検出し、不足している列に対処したり、ロジックを調整して、意図的に削除した条件またはアクションに依存しないようにすることができます。

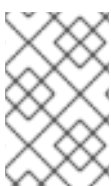
### 不完全な範囲 (Incomplete Ranges)

可能なフィールド値に対する制約がテーブルに追加されているにもかかわらず、可能な値がすべて定義されていない場合は、フィールド値の範囲が不完全です。検証レポートは、提供された不完全な範囲を特定します。たとえば、アプリケーションが承認されたかどうかをテーブルが確認した場合、検証レポートにより、アプリケーションが承認されていない状況も処理できるようになります。

## 12.2. 通知の種類

検証および妥当性確認の機能では、3種類の通知を使用します。

-  Error: ガイド付きデシジョンテーブルが、実行時に設計された通りに機能しなくなるような重大な問題があることを意味します。たとえば、競合はエラーとしてレポートされます。
-  Warning: おそらく、ガイド付きデシジョンテーブルが動作しなくなるような重要な問題ではありませんが、注意が必要な重要な問題があることを示します。たとえば、包含は警告として報告されます。
-  Information: ガイド付きデシジョンテーブルの動作が止まることはないかもしれませんが、注意が必要です。たとえば、列が不明な場合は、情報として報告されます。



### 注記

Business Central の確認および検証機能は、誤った変更の保存を防ぐものではありません。この機能は、編集時に問題のみをレポートするので、これらの問題を無視して変更を保存することができます。

## 12.3. ガイド付きデシジョンテーブルの検証および妥当性確認の無効化

Business Central のデシジョンテーブルの検証および妥当性確認機能は、デフォルトで有効になっています。この機能を使用すると、ガイド付きデシジョンテーブルの検証が容易になりますが、複雑なガイド付きデシジョンテーブルの場合は、この機能が原因でデシジョンエンジンのパフォーマンスが低下してしまう可能性があります。お使いの Red Hat Process Automation Manager ディストリビューションで **org.kie.verification.disable-dtable-realtime-verification** のシステムプロパティを **true** に設定して、この機能を無効にできます。

## 手順

~/standalone-full.xml に移動して、以下のシステムプロパティを追加します。

```
<property name="org.kie.verification.disable-dtable-realtime-verification" value="true"/>
```

たとえば、Red Hat JBoss EAP では、このシステムプロパティを **\$EAP\_HOME/standalone/configuration/standalone-full.xml** に追加します。

## 第13章 スプレッドシート形式のデシジョンテーブルへの、ガイド付きデシジョンテーブルの変換

Business Central でガイド付きデシジョンテーブルを定義した後に、オフライン参照やファイル共有向けに、ガイド付きデシジョンテーブルを XLS スプレッドシート形式のデシジョンテーブルファイルに変換できます。ガイド付きデシジョンテーブルを変換するには、拡張エントリーのガイド付きデシジョンテーブルを使用する必要があります。変換ツールは、制限エントリーのガイド付きデシジョンテーブルをサポートしていません。

デシジョンテーブルの詳細は、『[アップロードしたデシジョンテーブルを使用したデシジョンサービスの作成](#)』を参照してください。



### 警告

ガイド付きデシジョンテーブルとスプレッドシートのデシジョンテーブルは、テーブル形式が異なり、サポート対象機能も異なります。別のデシジョンテーブル形式(例: ヒットポリシー)に変換する場合には、サポート対象機能が両方で異なる分は変更されるか、失われることになります。

### 手順

Business Central で、変換するガイド付きデシジョンテーブルに移動して、デシジョンテーブルデザイナーの右上隅で **Convert to XLS** をクリックします。

#### 図13.1 アップロードしたデシジョンテーブルの変換

#	Description	application : LoanApplication				IncomeSol	application		
		amount min	amount max	period	deposit max	Income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

変換後に、変換したデシジョンテーブルはプロジェクト内のスプレッドシート形式のデシジョンテーブルアセットとして利用でき、ダウンロードしてオフラインで参照できるようになります。

## 第14章 ルールの実行

ルールの例を特定するか、Business Central でルールを作成したら、関連のプロジェクトをビルドしてデプロイし、ローカルまたは KIE Server でルールを実行してテストできます。

### 前提条件

- Business Central および KIE Server がインストールされ、実行されている。インストールオプションは、『[Red Hat Process Automation Manager インストールの計画](#)』を参照してください。

### 手順

1. Business Central で、**Menu** → **Design** → **Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Assets** ページの右上にある **Deploy** をクリックして、プロジェクトをビルドして KIE Server にデプロイします。ビルドに失敗したら、画面下部の **Alerts** パネルに記載されている問題に対処します。  
プロジェクトデプロイメントのオプションに関する詳細は、『[Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ](#)』を参照してください。

### 注記

デフォルトでプロジェクト内のルールアセットが実行可能なルールモデルからビルドされていない場合には、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれているか確認して、プロジェクトを再構築してください。

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるようにするために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルに関する詳細は、『[Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ](#)』を参照してください。

3. ローカルでのルール実行に使用するか、KIE Server でルールを実行するクライアントアプリケーションとして使用できるように、まだ作成されていない場合には、Business Central 外に Maven または Java プロジェクトを作成します。プロジェクトには、**pom.xml** ファイルと、プロジェクトリソースの実行に必要なその他のコンポーネントを含める必要があります。  
テストプロジェクトの例については、『[その他の DRL ルールの作成および実行方法](#)』を参照してください。
4. テストプロジェクトまたはクライアントアプリケーションの **pom.xml** ファイルを開き、以下の依存関係が追加されていない場合は追加します。
  - **kie-ci**: クライアントアプリケーションで、**ReleasesId** を使用して、Business Central プロ



ジェクトデータをローカルにロードします。

- **kie-server-client**: クライアントアプリケーションで、KIE Server のアセットを使用してリモートに接続します。
- **slf4j**: (オプション) クライアントアプリケーションで、KIE Server に接続した後に、SLF4J (Simple Logging Facade for Java) を使用して、デバッグのログ情報を返します。

クライアントアプリケーションの **pom.xml** ファイルにおける、Red Hat Process Automation Manager 7.8 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.39.0.Final-redhat-00005</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.39.0.Final-redhat-00005</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

このアーティファクトで利用可能なバージョンについては、オンラインの [Nexus Repository Manager](#) でグループ ID とアーティファクト ID を検索してください。



## 注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Process Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.8.0.redhat-00005</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、「[What is the mapping between Red Hat Process Automation Manager and the Maven library version?](#)」を参照してください。

5. モジュールクラスを含むアーティファクトの依存関係が、クライアントアプリケーションの **pom.xml** ファイルに定義されていて、デプロイしたプロジェクトの **pom.xml** ファイルに記載されているのと同じであることを確認します。モデルクラスの依存関係が、クライアントアプリケーションとプロジェクトで異なると、実行エラーが発生します。

Business Central でプロジェクトの **pom.xml** ファイルを利用するには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View** → **pom.xml** を選択します。

たとえば、以下の **Person** クラスの依存関係は、クライアントと、デプロイしたプロジェクトの **pom.xml** ファイル両方に表示されます。

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. デバッグ向けロギングを行うために、**slf4j** 依存関係を、クライアントアプリケーションの **pom.xml** ファイルに追加した場合は、関連するクラスパス (Maven の **src/main/resources/META-INF** 内など) に **simplelogger.properties** ファイルを作成し、以下の内容を記載します。

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

7. クライアントアプリケーションに、必要なインポートを含む **.java** メインクラスと、KIE ベースをロードする **main()** メソッドを作成し、ファクトを挿入し、ルールを実行します。たとえば、プロジェクトの **Person** オブジェクトには、名、姓、時給、賃金を設定および取得する getter メソッドおよび setter メソッドが含まれます。プロジェクトにある以下の **Wage** ルールでは、賃金と時給を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
  end
```

(必要に応じて) KIE Server の外でローカルにこのルールをテストするには、**.java** クラスで、KIE サービス、KIE コンテナ、および KIE セッションをインポートするように設定し、その後、**main()** メソッドを使用して、定義したファクトモデルに対してすべてのルールを実行するようにします。

### ローカルでのルールの実行

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseIdImpl("com.myspace", "MyProject", "1.0.0");

      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.newKieContainer(rid);
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }
  }
}
```

```

    catch (Throwable t) {
        t.printStackTrace();
    }
}
}
}

```

KIE Server でこのルールをテストするには、ローカルの例と同じように、インポートとルール実行情報で **.java** クラスを設定し、KIE サービス設定および KIE サービスクライアントの詳細を指定します。

## KIE Server でのルールの実行

```

package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {

    private static final String containerName = "testProject";
    private static final String sessionName = "myStatelessSession";

    public static final void main(String[] args) {
        try {
            // Define KIE services configuration and client:
            Set<Class<?>> allClasses = new HashSet<Class<?>>();
            allClasses.add(Person.class);
            String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
            String username = "$USERNAME";
            String password = "$PASSWORD";
            KieServicesConfiguration config =
                KieServicesFactory.newRestConfiguration(serverUrl,
                    username,
                    password);
            config.setMarshallingFormat(MarshallingFormat.JAXB);
            config.addExtraClasses(allClasses);
            KieServicesClient kieServicesClient =
                KieServicesFactory.newKieServicesClient(config);

```

```

// Set up the fact model:
Person p = new Person();
p.setWage(12);
p.setFirstName("Tom");
p.setLastName("Summers");
p.setHourlyRate(10);

// Insert Person into the session:
KieCommands kieCommands = KieServices.Factory.get().getCommands();
List<Command> commandList = new ArrayList<Command>();
commandList.add(kieCommands.newInsert(p, "personReturnId"));

// Fire all rules:
commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
BatchExecutionCommand batch = kieCommands.newBatchExecution(commandList,
sessionName);

// Use rule services client to send request:
RuleServicesClient ruleClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
ServiceResponse<ExecutionResults> executeResponse =
ruleClient.executeCommandsWithResults(containerName, batch);
System.out.println("number of fired rules:" +
executeResponse.getResult().getValue("numberOfFiredRules"));
}

catch (Throwable t) {
t.printStackTrace();
}
}
}
}

```

8. 設定した **java** クラスをプロジェクトディレクトリーから実行します。(Red Hat CodeReady Studio などの) 開発プラットフォーム、またはコマンドラインでファイルを実行できます。(プロジェクトディレクトリー内の) Maven の実行例:

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

(プロジェクトディレクトリー内の) Java の実行例

```
javac -classpath ".*$DEPENDENCIES/*:." RulesTest.java
java -classpath ".*$DEPENDENCIES/*:." RulesTest
```

9. コマンドラインおよびサーバーログで、ルール実行のステータスを確認します。ルールが期待通りに実行しない場合は、プロジェクトに設定したルールと、メインのクラス設定を確認して、提供されるデータの妥当性を確認します。

## 第15章 次のステップ

- 『[テストシナリオを使用したデシジョンサービスのテスト](#)』
- 『[Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ](#)』

## 付録A バージョン情報

本書の最終更新日: 2020 年 9 月 8 日 (木)