[Stambia]
Designer チュートリアル

株式会社クライム

Climb Inc.
Growing to Meet Your Needs

作成日: 2017/06/05(月)
更新日: 2017/06/05(月)
バージョン: 1.0

目次

1 はじめに

- 本ドキュメントに記載されたイラスト、写真、文章の一部またはすべてを無断で複製、転載することを禁止します。
- 本ドキュメントは製品を購入されたお客様、評価版をご使用のお客様向けに株式会社クライムが提供しております。

1.1 範囲

本ドキュメントは、Stambia のインストール手順、操作方法について記載しております。

1.2 対象バージョン

本ドキュメントは、以下の製品のバージョンに対応しております。

- Stambia Designer S18.3
- Stambia Runtime S17.4
- Stambia Analytics 2.2

2 チュートリアルの目標

当チュートリアルは、**Stambia** の基本的な機能を使いこなせるようになることを目標としています。下記の項目を取り扱います。

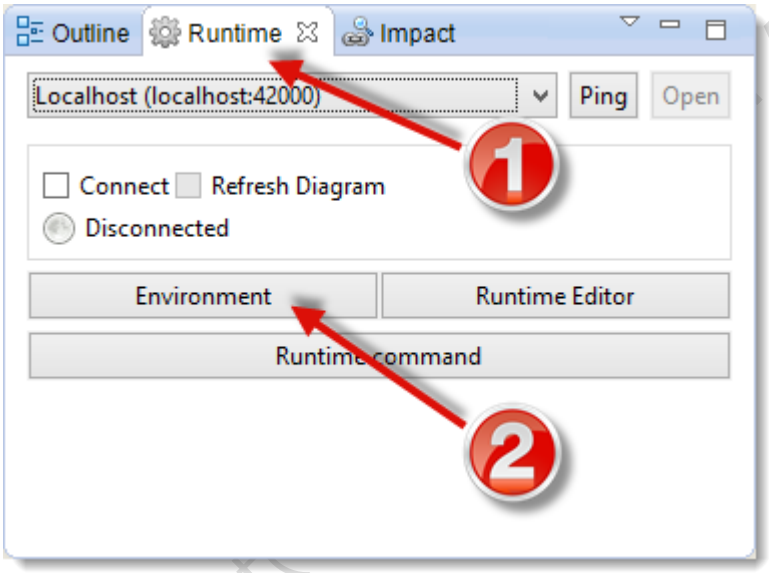
- プロジェクトの作成
- メタデータの作成とリバースエンジニアリング
- マッピングの作成
- プロセスの作成と実行

使用される技術は下記の通りです。

- データベース
- フラットファイル

3 チュートリアル環境を起動

Runtime ビューの **Environment** ダイアログを開きます。



View(ビュー)とは、**Stambia Designer** の画面における移動可能なウィンドウを指し、当チュートリアルでしばしば引用されます。ビューは **Stambia Designer** 画面上、各種ファイルが開かれる中央エリア（編集エリア）に表示されます。ビューの内容は、中央エリアあるいは **Project Explorer** において選択されているオブジェクトに応じて同期化される場合があります。

Start local Runtime ボタンと **Start demo Databases** ボタンをクリックします。

Start local Runtime ボタンは、ローカルの **Runtime** を起動します。この Runtime が開発環境における各種処理の実行を管理します。Runtime は手動でも起動することができ、そのパラメータをすべて制御することも可能です。また、Windows Service として開始することも可能です。詳しくは、他のチュートリアルで紹介します。

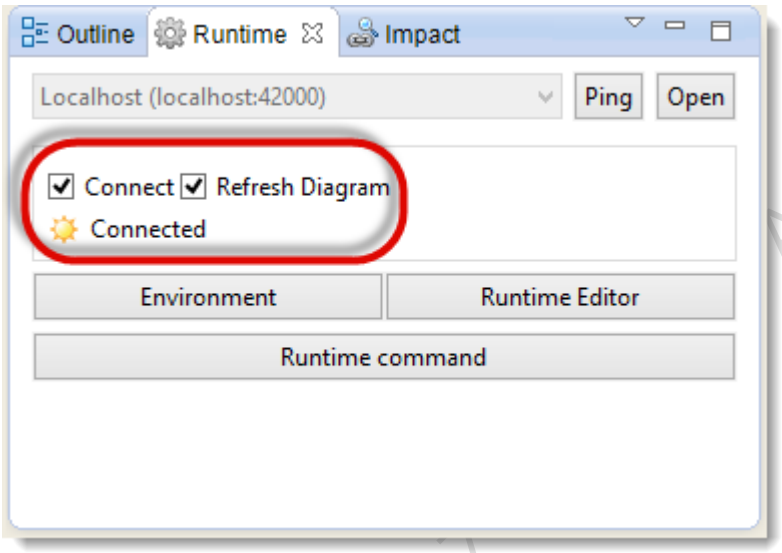
Start demo Databases ボタンは、当チュートリアルで使用するデモ用の簡易データベースをローカルのマシン上に起動します。

デモ環境あるいはマシンをシャットダウンし、その後、チュートリアルを途中から再開する場合、デモ環境を再起動するのを忘れないでください。

最後に、**Connect** のチェックボックスをクリックします。

Connect チェックボックスは、**Designer** の Runtime への接続を指定します。**Refresh Diagram** のチェックボックスは、実行結果（統計、変数、色分け表示、ステータスなど）のリアルタイム表示を指定します。

Designer で開発したプロセスを **Designer** 上で実行するには、Runtime への接続が必須となります。



4 チュートリアルプロジェクトを作成

4.1 プロジェクトとは

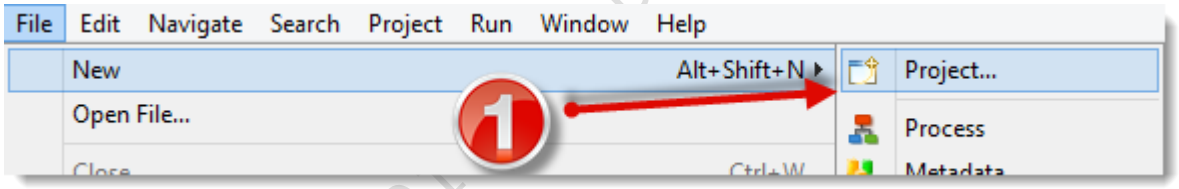
Project（プロジェクト）は、開発に必要なファイルや設定をまとめたグループであり、ワークスペースの中に実際に存在する物理的ディレクトリです。

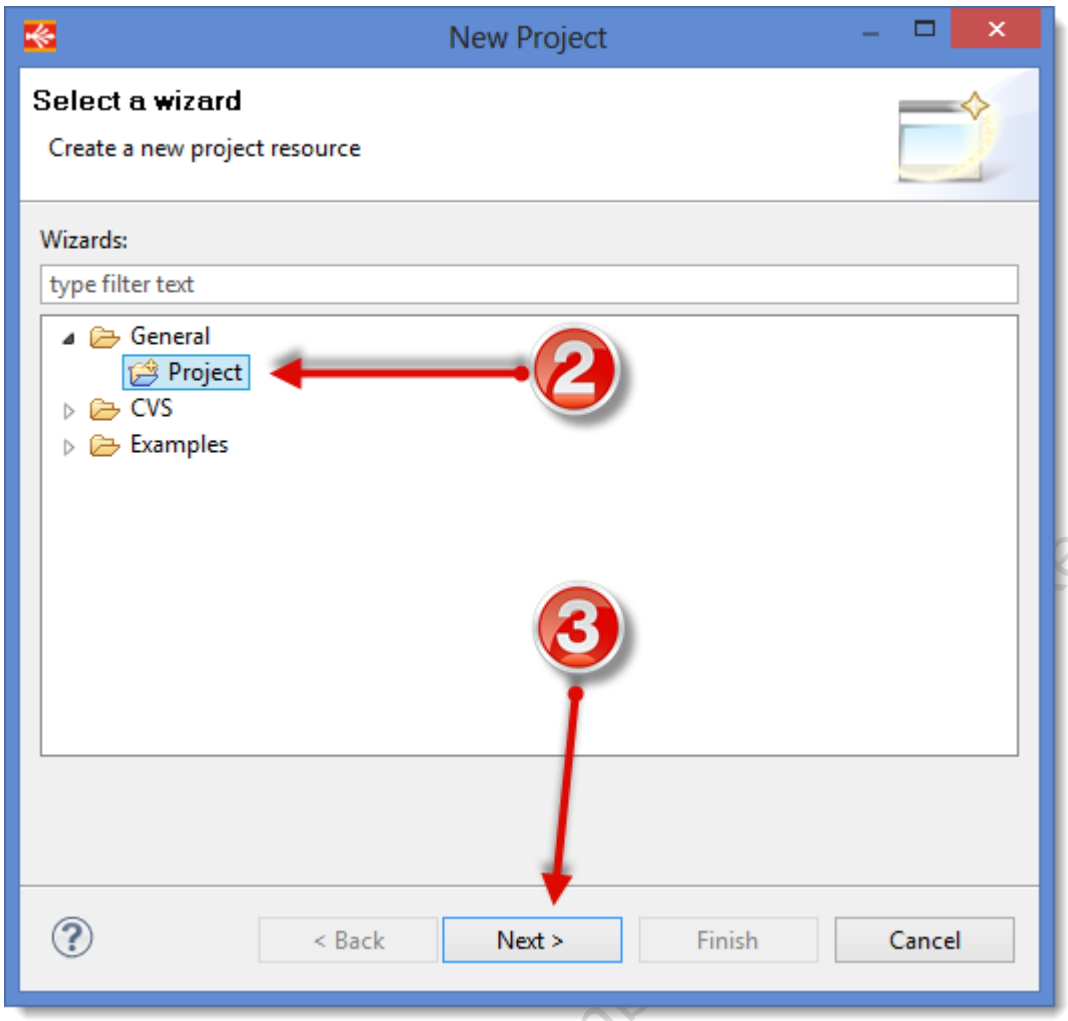
プロジェクトに含まれるすべてのファイルが、そのディレクトリのツリー構造の中に保存されます。つまり、それらのファイルが、通常のファイルシステムやフォルダのように取り扱うことができ、アーカイブされます。

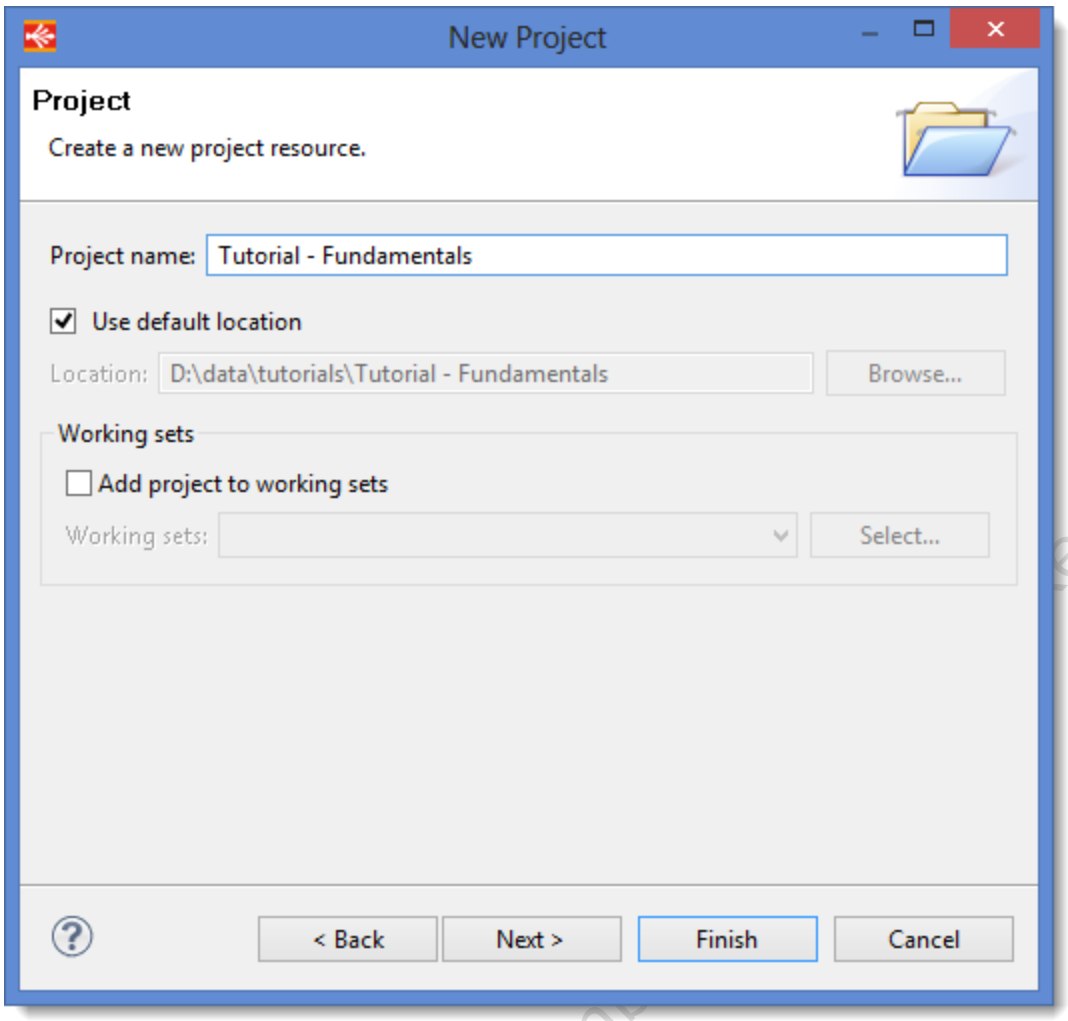
4.2 プロジェクトの作成

チュートリアルプロジェクトを作成してみましょう。

1. **Project Explorer** ビューを開きます。
2. **File** メニューをクリックし、次に **New**、**Project** をクリックします。
3. **Select a Wizard** ウィンドウの **General** ノードを展開して、**Project** を選択します。
4. **Next** ボタンをクリックします。
5. プロジェクトに名前を付けます。**Project name** 欄に Tutorial – Fundamentals と入力します。
6. **Finish** をクリックします。







5 メタデータを作成する

5.1 メタデータとは

Stambia は**Metadata**（メタデータ）にもとづいています。メタデータがデータ統合のプロセスの設計、生成、そして実行を可能にします。例えば、データ統合の工程に使用されるフラットファイルや XML ファイル、テーブルの構造はメタデータにより定義されます。

Stambia で管理される **Metadata** ファイルが通常、データベースサーバーやテーブルあるいはファイルで構成されるフォルダのように、データモデルを表現します。

データベースサーバーやファイルシステムなどに接続し、テーブルやファイルの構造を取得し反映するリバースエンジニアリング（リバース）によって、メタデータファイルが作成できます。

さらに、**Stambia** では、このメタデータファイルの基本要素を **Datastore**（データストア）と呼びます。個々のテーブルやファイルなどが単一のデータストアに対応します。また、メタデータファイルはマッピングのソースおよびターゲットとしても使用されます。

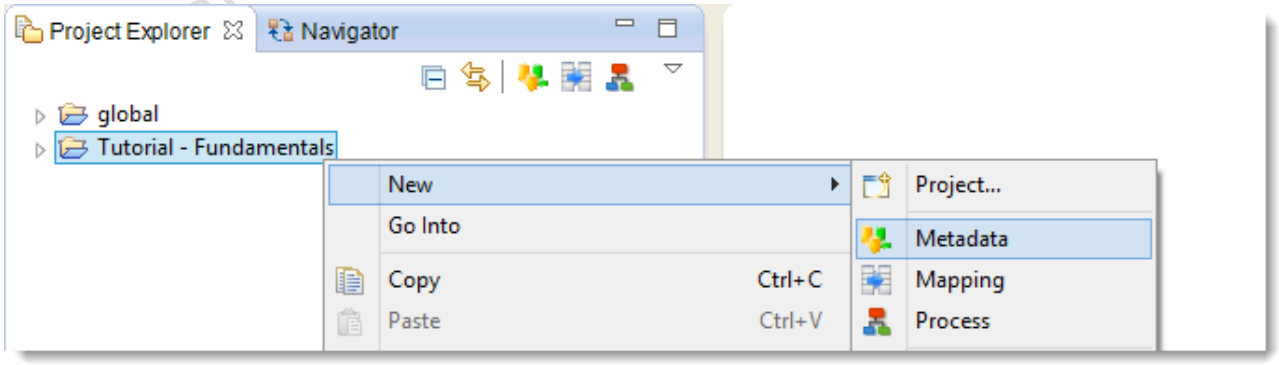
本章では、**Metadata** ファイルの作成方法と、各データストア定義を取得するためのリバースエンジニアリングの方法を実施します。

5.2 ターゲットデータベースのリバース

5.2.1 メタデータファイルの作成

ターゲットとなるモデルは、デモ環境ベースのうちの 1 つである Datamart に対応するデータベースです。

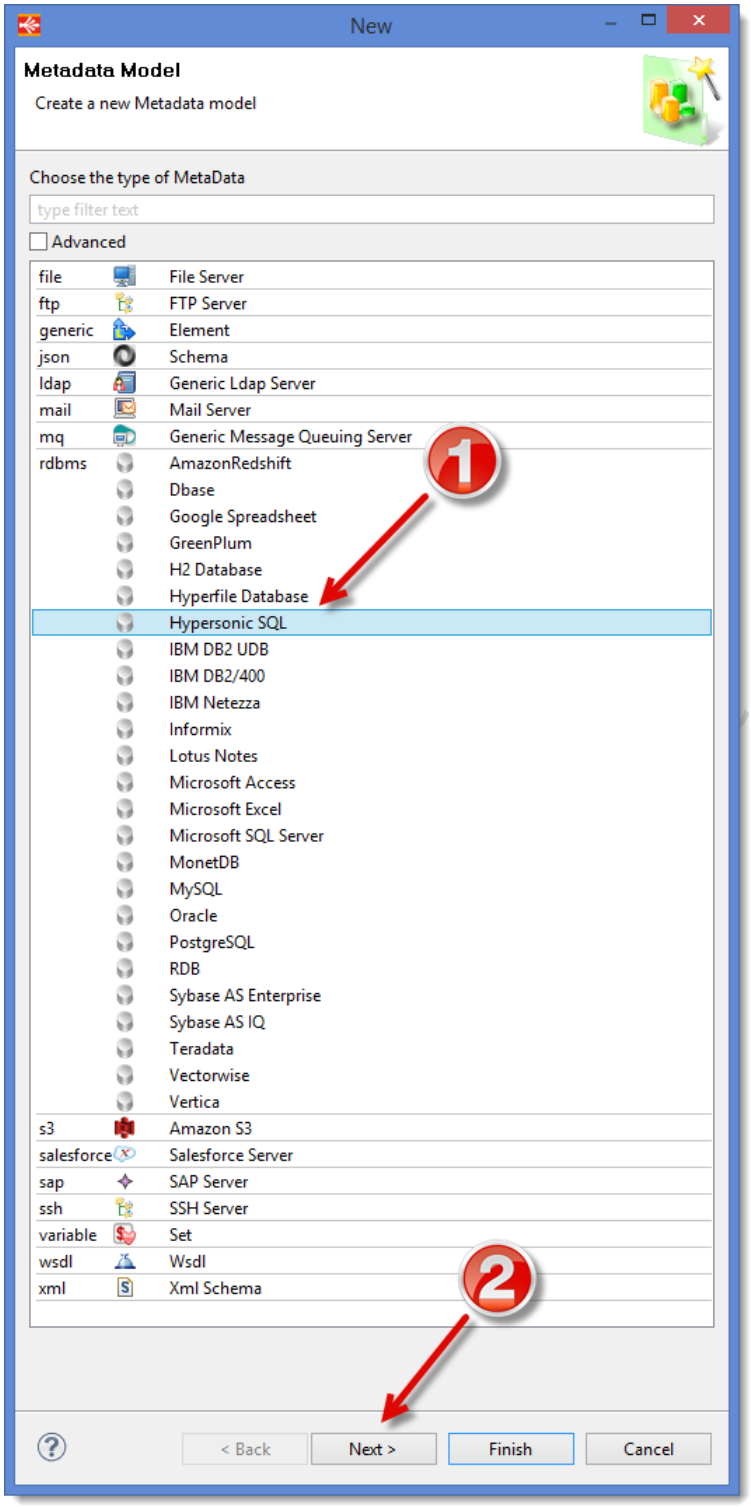
1. **Project Explorer** ビューにおいて、Tutorial – Fundamentals プロジェクトをクリックします。
2. **New** を選択し、続いて**Metadata** を選択します。



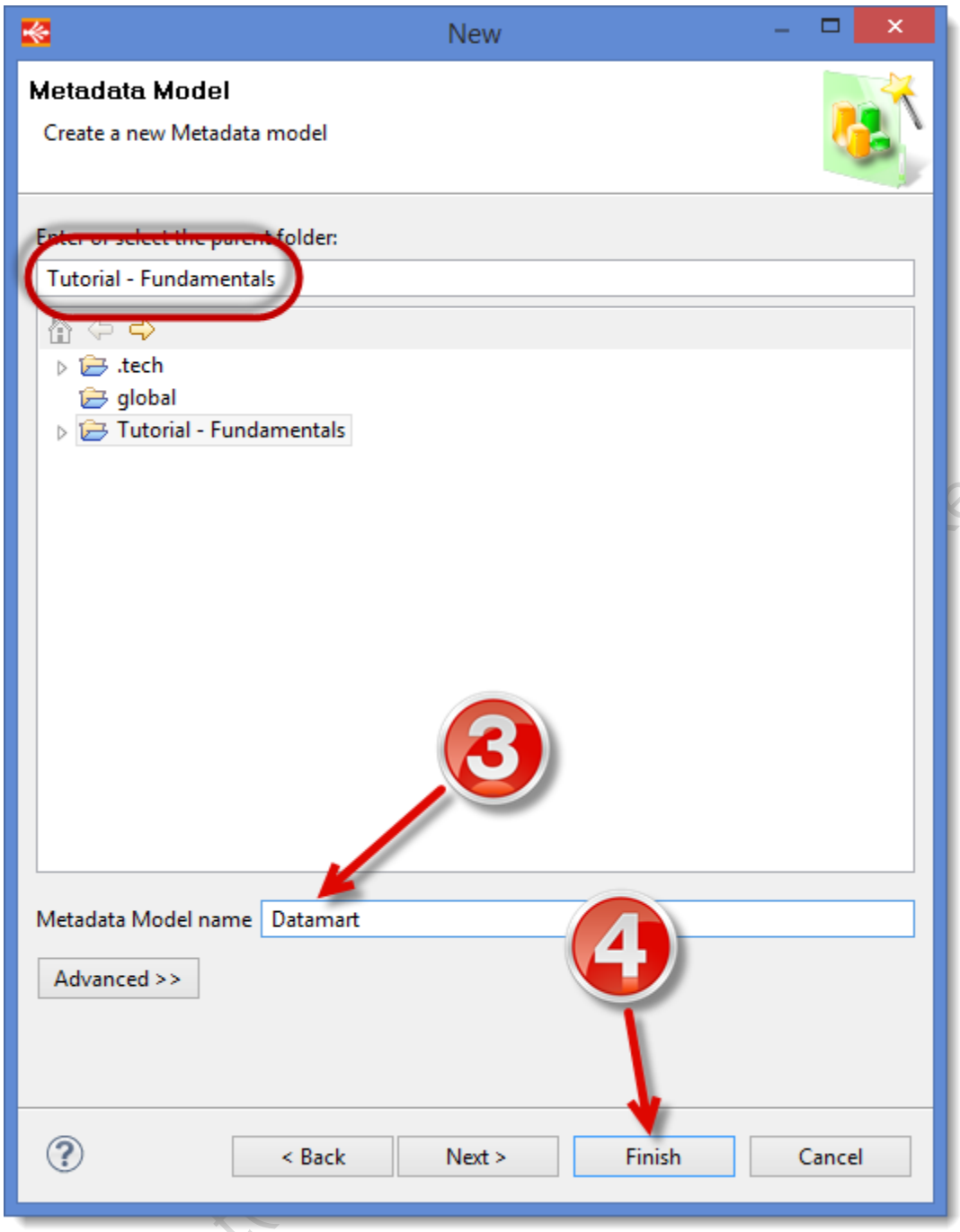
Stambia は基盤となるテクノロジーによって作業工程を最適化しますが、同時に、各テクノロジーに合ったメタデータを提案することもできます。そのため、作成するメタデータのタイプを選択する必要があります。今回のケースで使用する

Datamart は Hypersonic SQL データベースです。（デモ環境に使用される 2 つのデータベースのうちの 1 つです）。

表示されたリストから **Hypersonic SQL** を選択し、**Next** をクリックしてください。



Metadata Model name 欄に Datamart と記入し、**Finish** をクリックします。



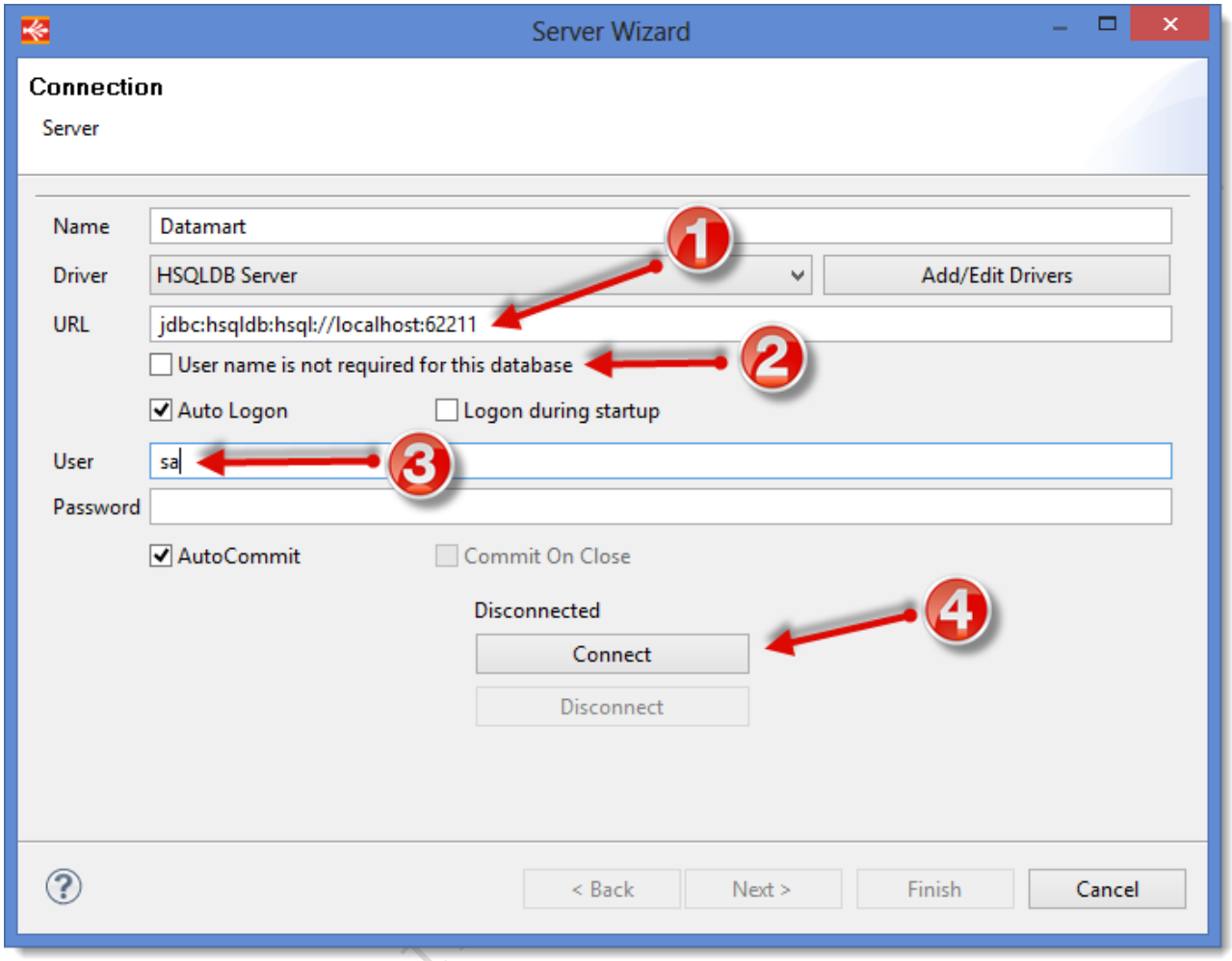
上記の手順では、プロジェクト名と一致する親フォルダが自動的に使われます。しかし、プロジェクトを右クリックしてメタデータファイルを作成していない場合、親フォルダを手動で選択してください。

以上により、メタデータファイルが作成され、**Stambia** は自動的に **Dataserver Assistant**（データサーバーアシスタント）を開いて、それを定義します。そして、Dataserver Assistant がリバースエンジニアリングによってデータベーステーブルの定義を取得します。

5.2.2 データベースサーバーの定義

1. データベースサーバーの JDBC **URL** : jdbc:hsqldb:hsql://localhost:62211 を入力します。

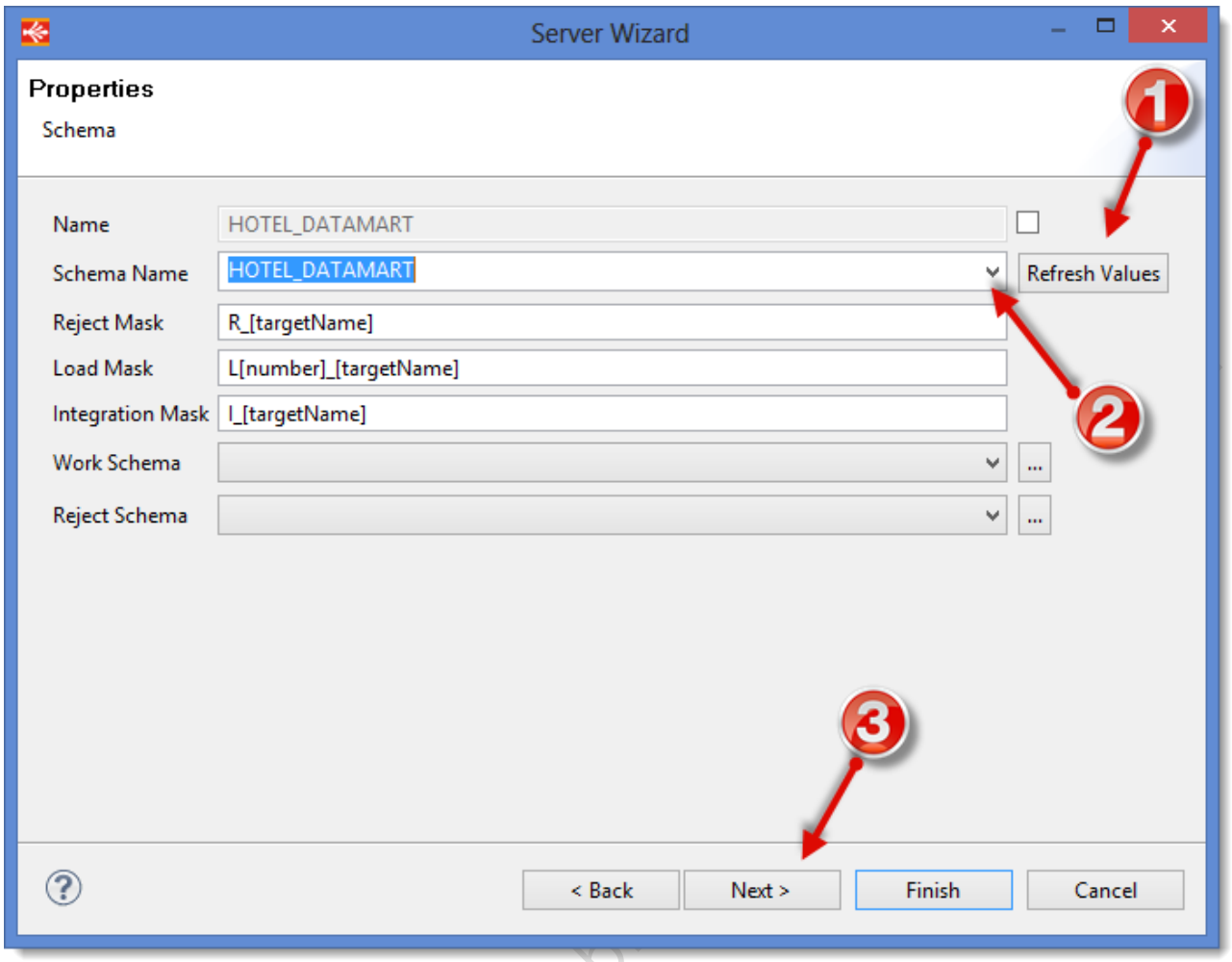
2. **User name is not required for this database** チェックボックスを外します。
3. **User** にユーザー名：sa を入力します。
4. **Connect** をクリックします。



これにより、**Stambia Designer** はデータベースへの JDBC 接続を開き、**Next** ボタンがクリック可能になります。**Next** をクリックして、スキーマ定義のウィンドウを開きます。

5.2.3 スキーマを定義

1. **Reflesh Values** ボタンをクリックし、**Stambia Designer** でデータベースの有効なスキーマリストを取得します。
2. **Schema Name** は HOTEL_DATAMART を選択します。
3. **Next** をクリックし、リバースを行いたい**データストア**を選択するためのウィンドウを開きます。

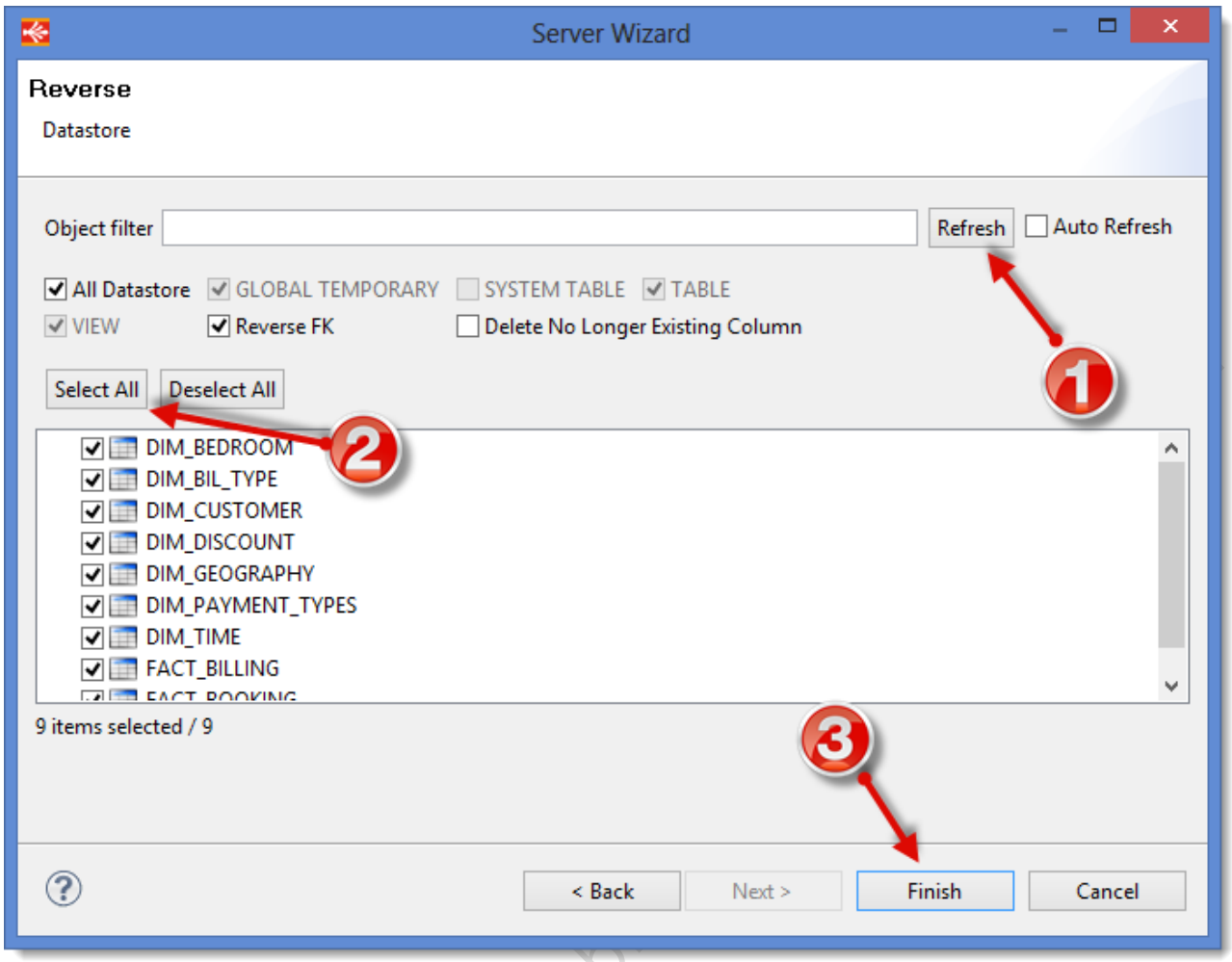


当チュートリアルでは、上記画面において **Work Schema** と **Reject Schema** は空欄になっています。これにより、**Stambia** が生成する技術的なテーブルは HOTEL_DATAMART スキーマに直接保存されます。しかし、**Stambia** を実際に使用するときは、work table と reject table に別個のスキーマを使用することが推奨されます。

以上により、HOTEL_DATAMART スキーマに含まれるテーブルから、リバースエンジニアリングの対象となるテーブルを選択することができます。

5.2.4 テーブルをリバースする

1. **Refresh** ボタンをクリックして、対象となるテーブルをロードします。
2. スキーマに含まれる全テーブルをリバースするため、**Select All** をクリックします。
3. **Finish** をクリックして、リバースエンジニアリングを開始します



5.2.5 メタデータファイルを保存

以上で、メタデータファイルを編集するウィンドウに戻ってきます。

メタデータファイルの先頭に付記された米印(*)に注目してください。

この米印は、メタデータファイルが変更されたが、未保存であることを示します。作業を進める前に、**Stambia Designer** がた正しいファイル内容を反映できるよう、ファイル保存をしてください。

ファイルを保存するには、アイコンをクリックするか、キーボードから **CTRL+S** を入力してください。

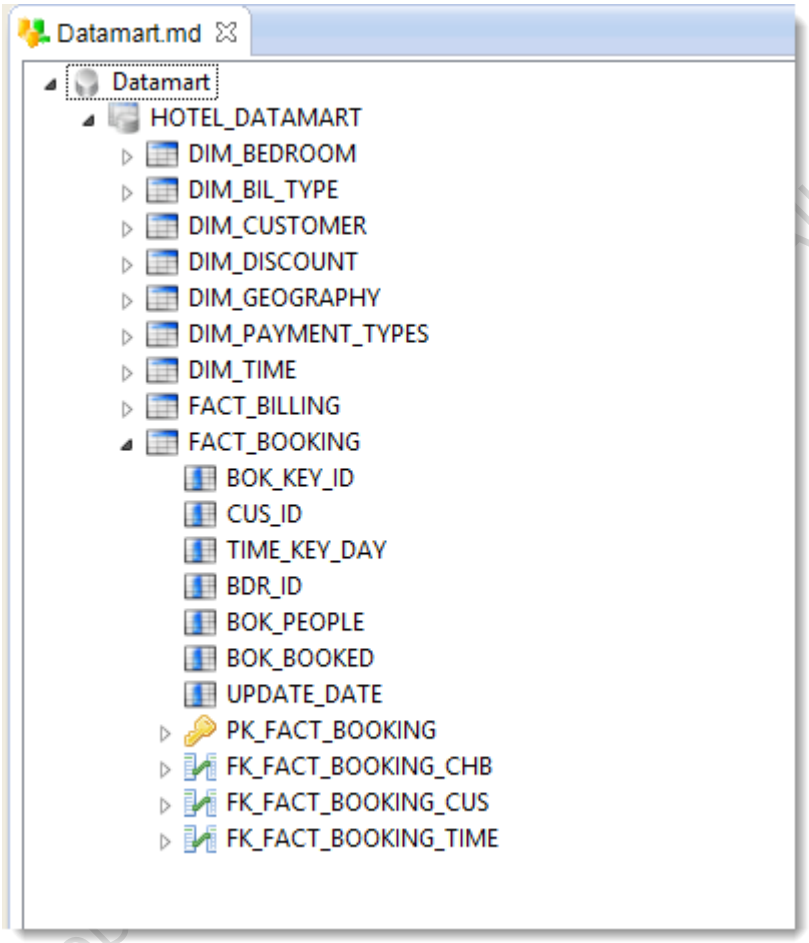
5.2.6 リバースされたテーブルを表示確認

データモデルを確認したい場合は、**Stambia Designer** でリバースエンジニアリングにより生成されたテーブルの構造を表示することできます。

1. データサーバーDatamart のノードを開きます。
2. スキーマ HOTEL_DATAMART のノードを開きます。

以上により、**Stambia Designer** でリバースエンジニアリングに選択した全テーブルをデータストアとして確認することができます。

各データストアのノードを展開すると、そのデータストアの構造（カラム、主キー、外部キーなど）にアクセスできます。



5.3 デリミテッドファイルのリバースする

5.3.1 メタデータファイルの作成

下記の通りに、新しいメタデータファイルを作成します。（方法は、データベースに対するメタデータファイルを作成するを参照してください。）

- 場所：Tutorial – Fundamentals プロジェクト内
- **メタデータモデル名**：Reference Files
- **メタデータタイプ**：File Server

Finish をクリックすると、データサーバーアシスタントが開きます。ただし、データベースに対してメタデータファイルを作成したときに開いたデータサーバーアシスタントと同一ではないことに注意してください。

5.3.2 ファイルを含むフォルダの定義

1. **Name** 欄に Reference Files Folder と記入します。
2. **Browse** をクリックし、**Stambia** のインストールディレクトリを探し、**stambiaRuntime** > **samples** > **files** ディレクトリを選択します。
3. **Next** をクリックします。

先に付けたディレクトリの名前と、実際の物理的ディレクトリ名（**files**）が一致しないことに注意してください。**Stambia** は、ユーザーにとって意味のある名前を使いながら、実際には、より複雑な物理的ディレクトリ名を持つことができます。同様に、異なる物理的値を同じ論理的フォルダ名にリンクさせることも可能です。（他のチュートリアルで解説されます。）

以上により、ファイル構造を定義するウィンドウが表示されます。

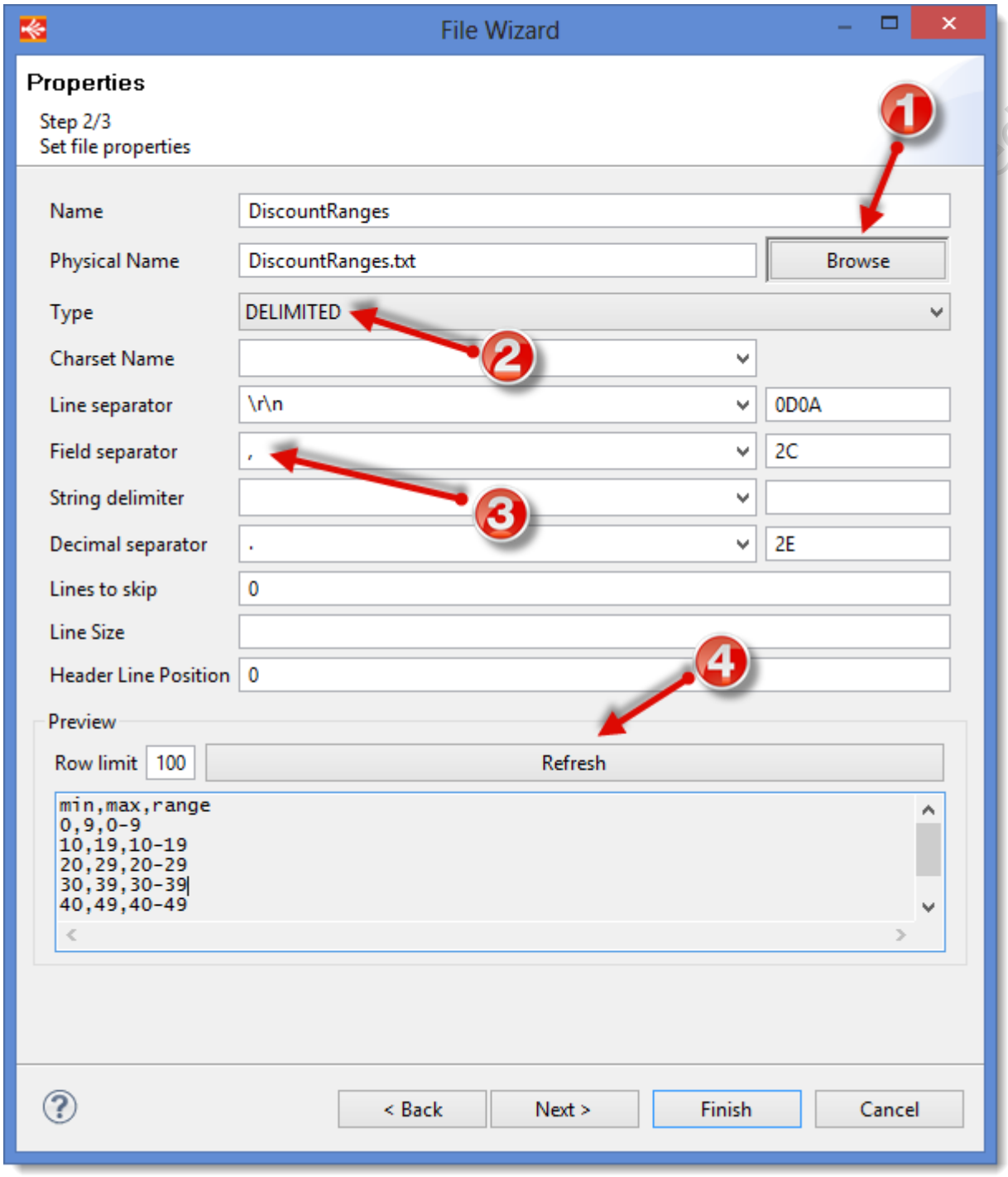
5.3.3 ファイル構造を定義

1. **Browse** をクリックして、DiscountRanges.txt ファイルを選択します。
2. **Type** が **DELIMITED** になっていることを確認します。DiscountRanges.txt ファイルは区切り文字でフィールドが区切られたファイルです。
3. DiscountRanges.txt ファイルの区切り文字はカンマ(,)です。**Field Separator** に **,** を選択します。

以上により、ファイル構造が定義されます。**Refresh** をクリックすることで、ファイルの内容がプレビューできます。この段階では、プレビューの 1 行目が各フィールド名になっています。

```
min,max,range
0,9,0-9
10,19,10-19
```

20,29,20-29
30,39,30-39
40,49,40-49
50,100,more than 50



Header Line Position に 1 を入力すれば、1 行目が列の見出しとして使用されます。
Header Line Position の値を変更した後、**Refresh** を再度クリックすると、1 行目がデータの一部ではなくなり、プレビューには表示されなくなります。

0,9,0-9

10,19,10-19
20,29,20-29
30,39,30-39
40,49,40-49
50,100,more than 50

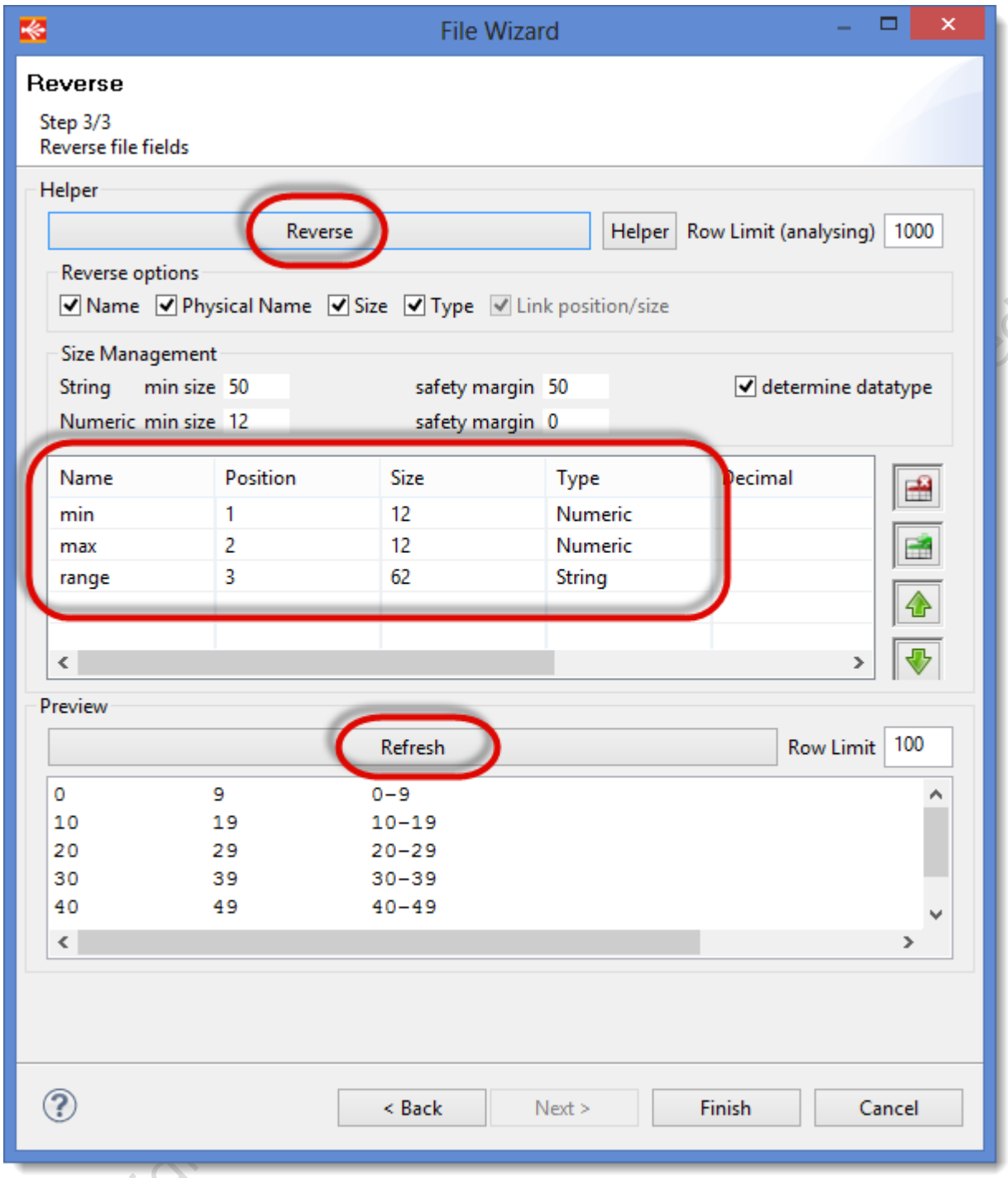
Next をクリックして、Datastore カラムの定義に進みます。

5.3.4 ファイルのカラムの定義

Reverse をクリックして、ファイルのカラムに対するリバースエンジニアリングを開始します。
その後、カラムが下記のプロパティ通りに定義されているか確認してください。

カラム名	ポジション	タイプ
min	1	Numeric
max	2	Numeric
range	3	String

ファイルの定義が正しいかどうかは、**Refresh** をクリックし、プレビューからデータがどのように解釈されているかを確認することにより判断することができます。



データストアの定義を終了するには、**Finish** をクリックしてください。

5.3.5 メタデータファイルの保存

この時点では、メタデータファイルはまだ保存されていません。

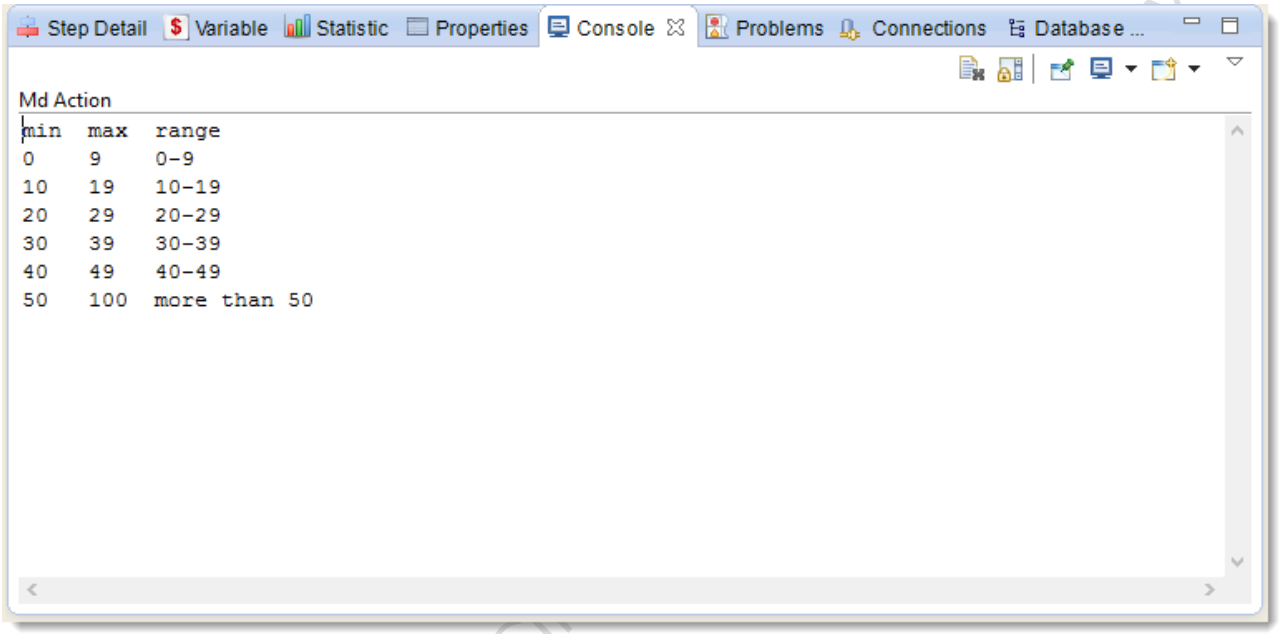
アイコンをクリックするか、キーボードから **CTRL+S** を入力し、保存してください。

5.3.6 ファイルデータの表示

Stambia Designer では、データストアのデータに簡単にアクセスでき、それらが **Stambia** にどのように読み込まれたのかを確認することができます。

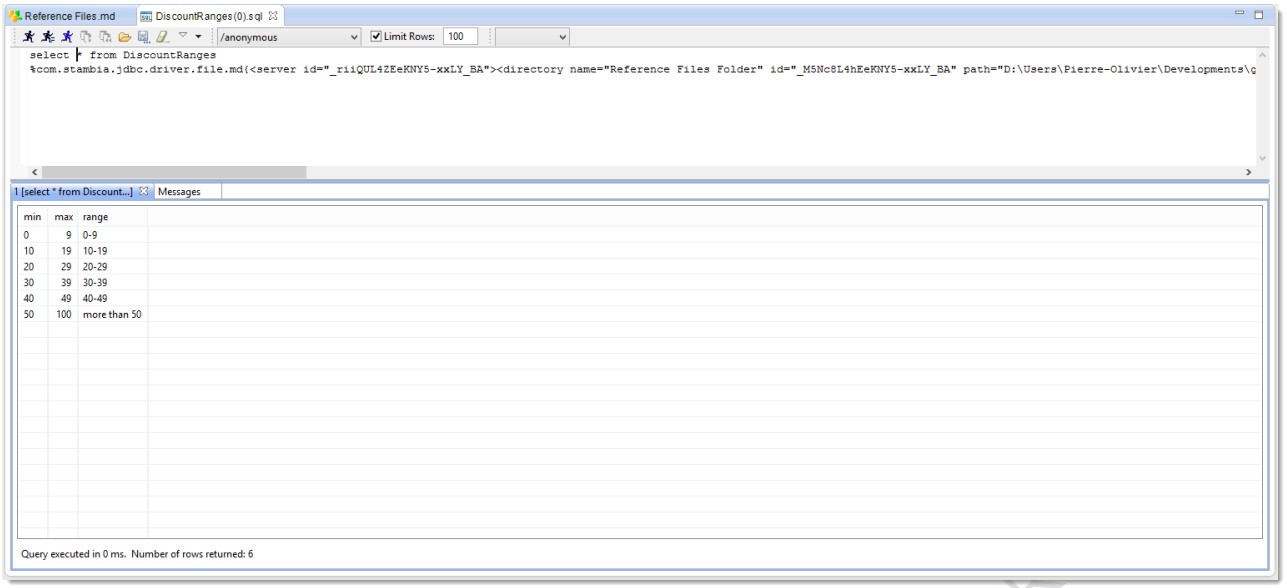
1. **Server** ノードを展開し、Reference Files Folder ノードを開きます。
2. DiscountRanges を右クリックし、**Actions > Consult Data (console)** を選択します。

Console ビューに表示されたデータを確認します。各列がタブで区切られ、1 行目に列の見出しが表示されます。



また、**Stambia Designer** は、SQL クエリを用いて、これらのデータを可視化する機能も備えています。

1. DiscountRanges を右クリックし、**Actions** > **Consult Data** を選択します。
2. SQL クエリエディタがデータを参照できるよう要求します。
3. をクリックするか、**CTRL + Enter** を入力してクエリを実行します。



データの照会はメタデータファイルに基づいています。メタデータファイルが保存されていない場合は、データストアに対してデータの照会を行うことはできません。

6 テンプレート

6.1 テンプレートとは

Stambia は、データ統合の **Processes** を生成するために、**Templates**（テンプレート）を用います。テンプレートが、ユーザーにより定義されたビジネスルールにもとづき、プロセスを生成します。

それにより、ユーザーは何をすべきか（ビジネスルール）を定義することに集中でき、どのようにするか（技術的プロセス）はテンプレートにより自動的に管理されます。

ここで、当チュートリアルを先に進める前に、**Stambia Designer** にテンプレートをインポートする必要があります。
s

6.2 テンプレートの取得する

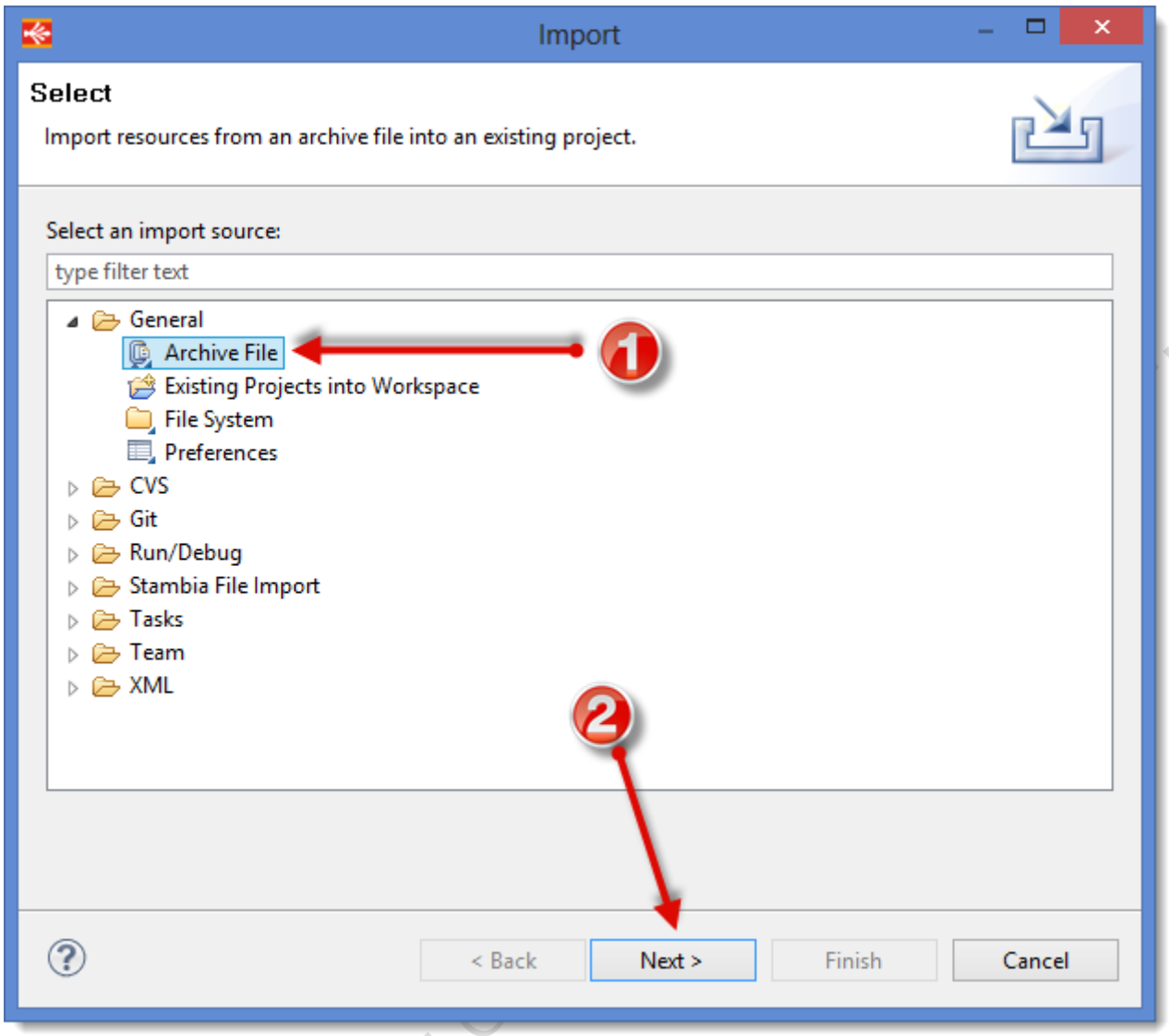
テンプレートは **Stambia** により、コミュニティウェブサイト：www.stambia.org を通じて提供されます。

1. ウェブブラウザから、**Stambia** の提供する接続情報をもとに www.stambia.org にアクセスします。
2. **Download** メニューを選択し、**Templates** を選択します。
3. **Generic** セクションの **Download** ボタンをクリックします。
4. ウェブブラウザから templates.generic.YYYY-MM-DD.zip というファイルをダウンロードできます。（YYYY-MM-DD はテンプレートの当該バージョンの製作日です）
5. ファイルを作業フォルダ（例：デスクトップ）に保存します。

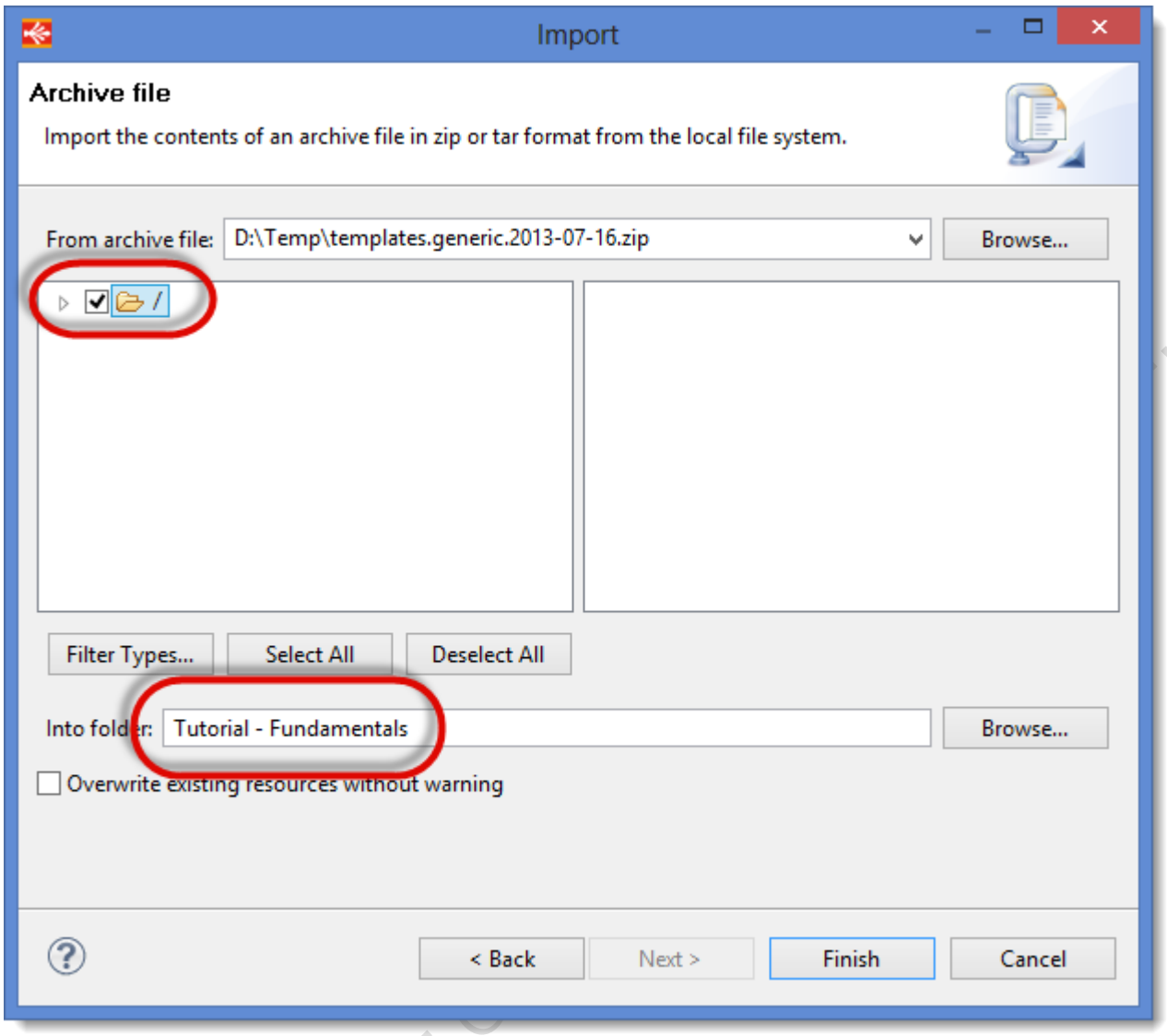
6.3 テンプレートのインポートする

Templates を含む ZIP 形式のアーカイブを以下の手順でインポートします。

1. **Stambia Designer** ウィンドウに戻ります。
2. **Project Explorer** ビューを開きます。
3. プロジェクト Tutorial – Fundamentals を右クリックし、表示されたメニューから **Import..**を選択します。
4. **Import source** として **General** > **Archive File** を選択します。
5. **Next** をクリックします。



1. ウィンドウ上部の **Browse** をクリックします。
2. テンプレートの ZIP ファイルを探し、選択します。
3. アーカイブのルートをインポートしているか確認してください（/ のボックスをチェックする必要があります）。
4. インポート先が Tutorial – Fundamentals プロジェクトになっていることを確認します。
5. **Finish** をクリックします。



このチュートリアルでは Generic テンプレートのみ必要になります。その他のテンプレートは他のチュートリアルで必要になるので、各チュートリアルの指示にしたがってインポートしてください。

7 データストアをロードする

7.1 マッピングとは？

Mapping（マッピング）とは、1 つ、あるいは複数の **Datastore**（データストア）から他のデータストアにデータを変換するルールを定義することです。変換元を **source**（ソース）、変換先を **target**（ターゲット）と呼びます。マッピングのテンプレートが、この変換のためのビジネスルールをもとに、最適化された完全なデータ統合プロセスを生成します。

7.2 シンプルマッピングの作成

7.2.1 DIM_DISCOUNT テーブルをロードするマッピングの作成

DIM_DISCOUNT テーブルをロードするためのマッピングファイルを作成します。

1. **Project Explorer** ビューにおいて、Tutorial – Fundamentals プロジェクトを右クリックします。
2. **New** を選択し、続いて **Mapping...**を選択します。
3. **File name** に Load DIM_DISCOUNT と記入し、**Finish** をクリックします。

上記の手順では、プロジェクト名と一致する親フォルダが自動的に使われます。プロジェクトを右クリックしてマッピングファイルを作成していない場合、親フォルダを手動で選択してください。

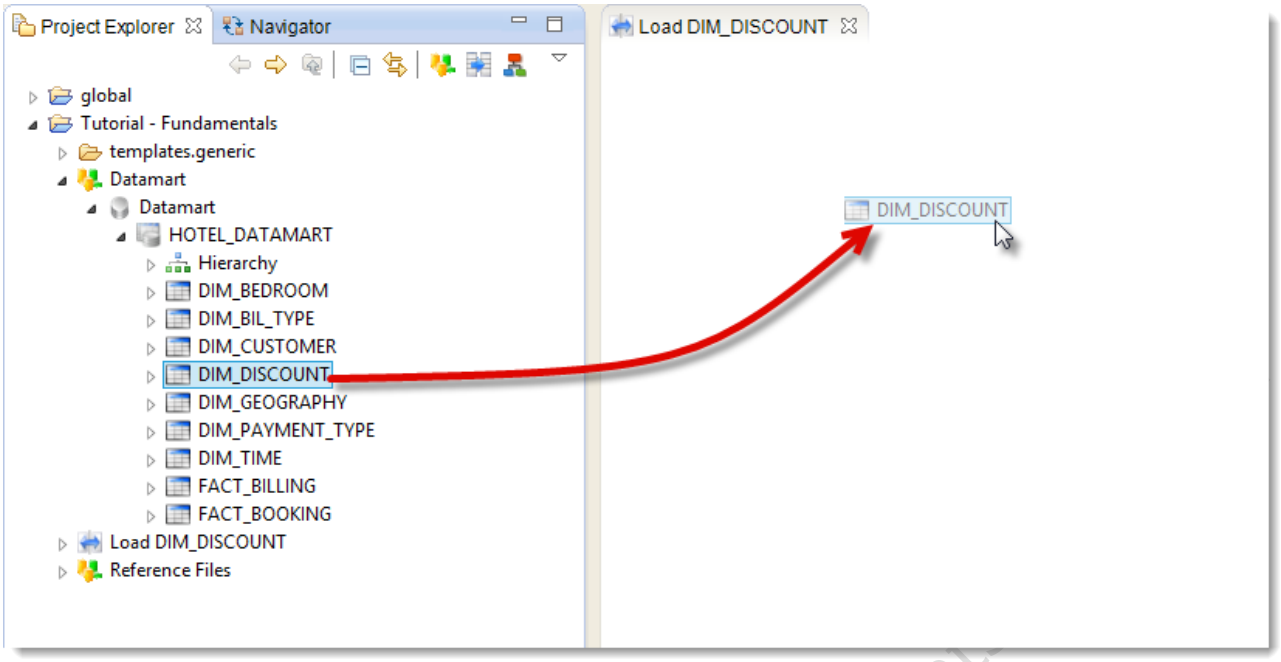
以上で、マッピングファイルが作成され、エディタが自動的に開くので、データ変化ルールの定義を開始することができます。

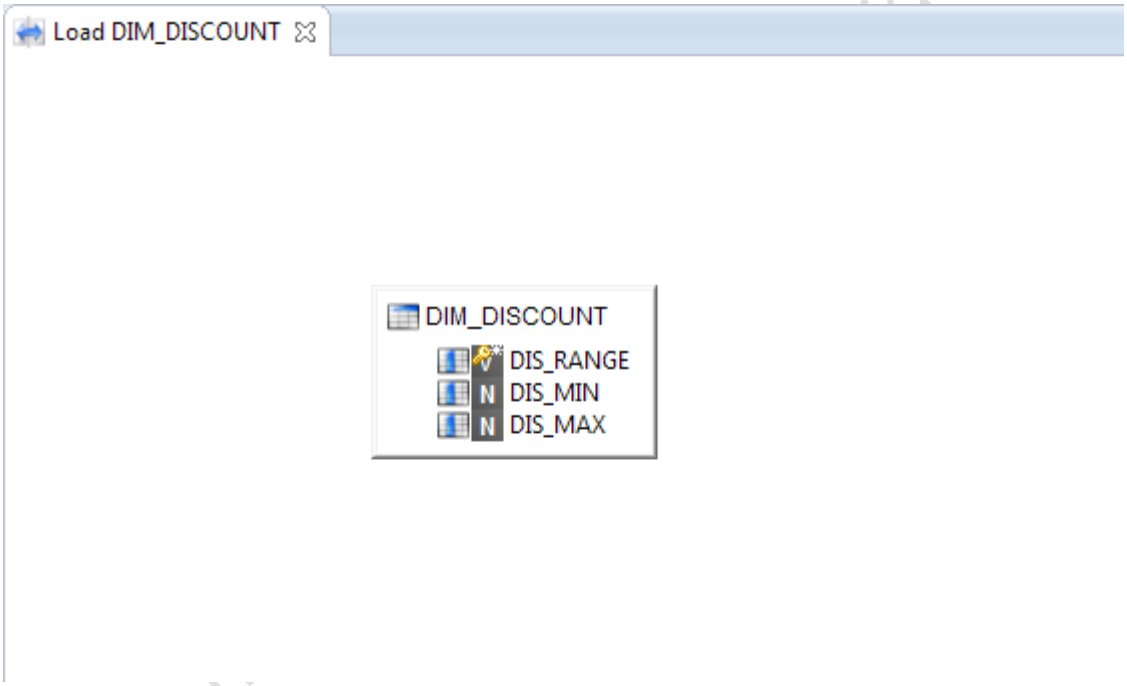
7.2.2 マッピングにデータストアを追加

最初のステップとして、必要なデータストアをマッピングに追加しなければなりません。

1. **Project Explorer** ビューにおいて、Tutorial – Fundamentals プロジェクトのノードを開きます。
2. 続いて、Datamart メタデータファイルノードを開きます。

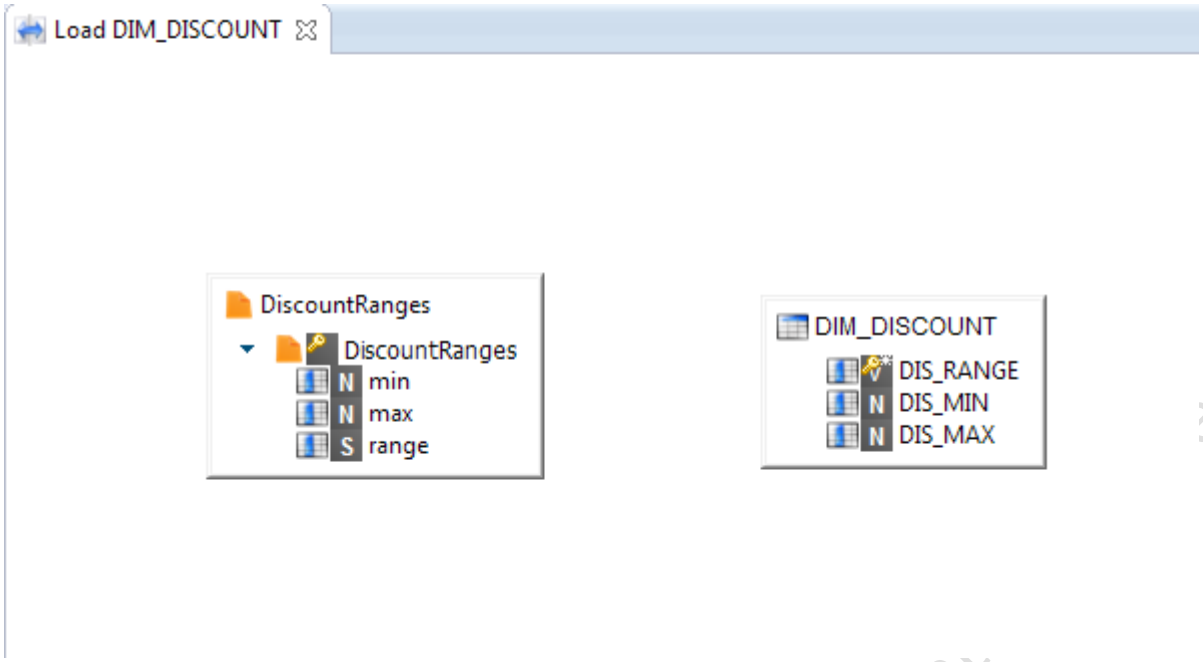
その下位ツリー構造の中からデータストア **DIM_DISCOUNT** を見つけ、それを **Project Explorer** ビューから Load DIM_DISCOUNT マッピングエディターにマウスでドラッグ&ドロップしてください。





上記の作業を次のデータストア DiscountRanges に対して繰り返します。

データストア DiscountRanges をメタデータファイルの Reference Files から、マウスでドラッグ&ドロップしてください。



このマッピングの目的は、DIM_DISCOUNT に DiscountRanges ファイルにあるデータをロードすることです。

この場合、DIM_DISCOUNT が**ターゲットデータストア**（ロードされる側）で、DiscountRanges が**ソースデータストア**に当たります。

マッピングにおいて、どのデータストアもターゲットあるいはソースになることがあります。

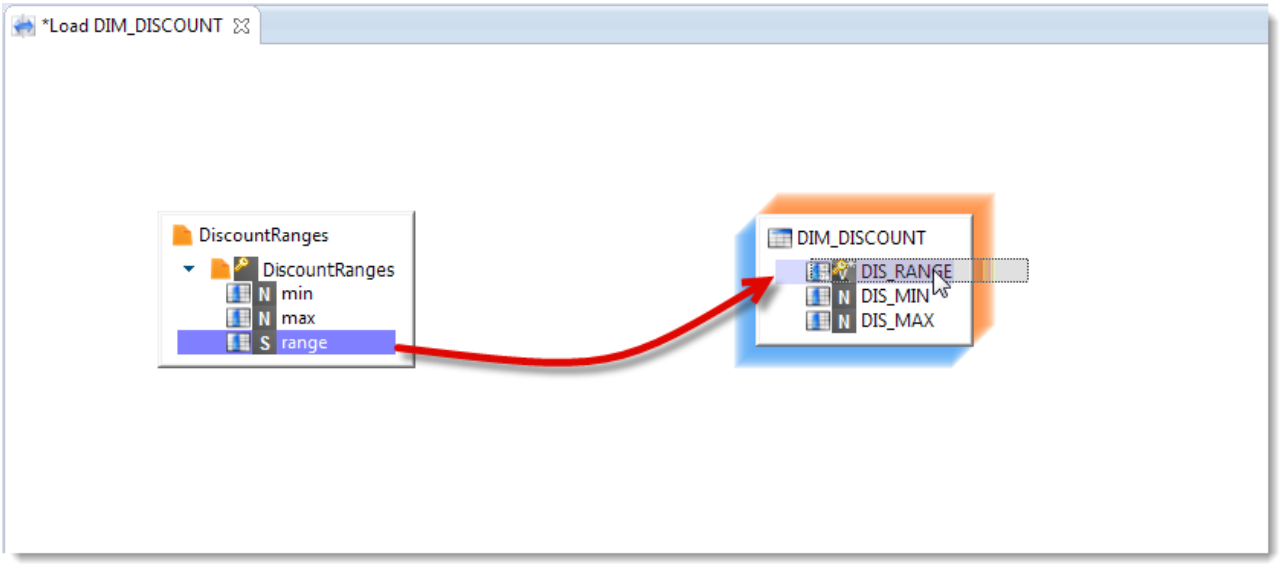
7.2.3 式を定義する

次に、**transformation expressions** を定義する必要があります。マッピングにもとづいてロードされた個々のカラムに対して、変換式を定義しなければなりません。

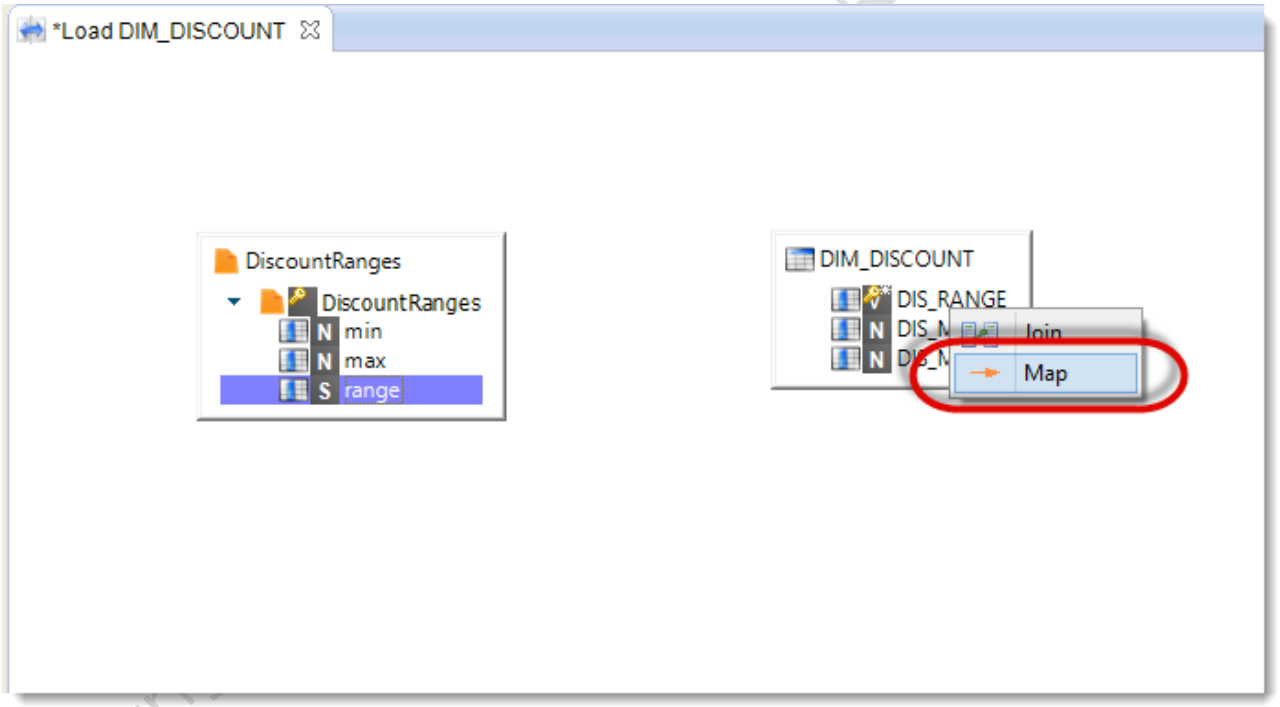
当チュートリアルで行う初めてのマッピングでは、簡単な変換式を取り扱います。

ターゲットカラム	変換式
DIS_RANGE	DiscountRanges.range
DIS_MIN	DiscountRanges.min
DIS_MAX	DiscountRanges.max

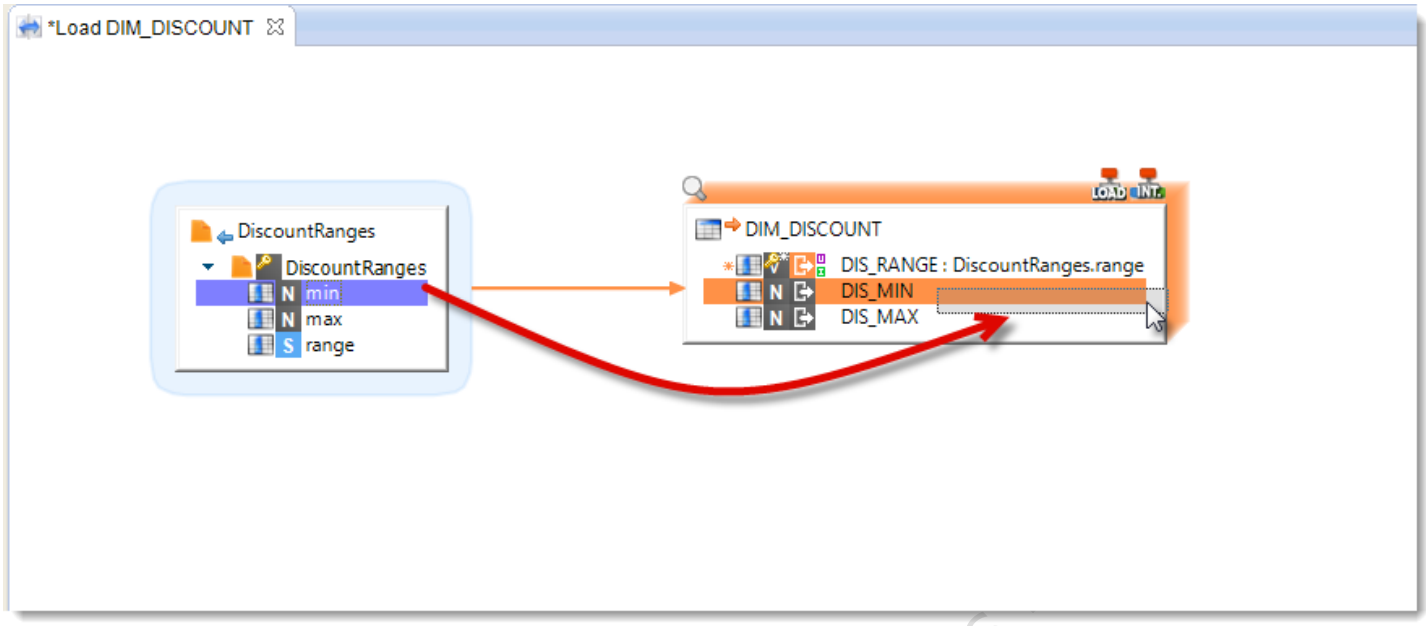
より複雑な変換式の定義方法も当チュートリアルで解説されますが、今回のマッピングでは、単にソースカラムを直接、対応するターゲットカラムにドラッグ&ドロップします。**Stambia Designer** がそれに応じ、対応する式を自動的に定義します。



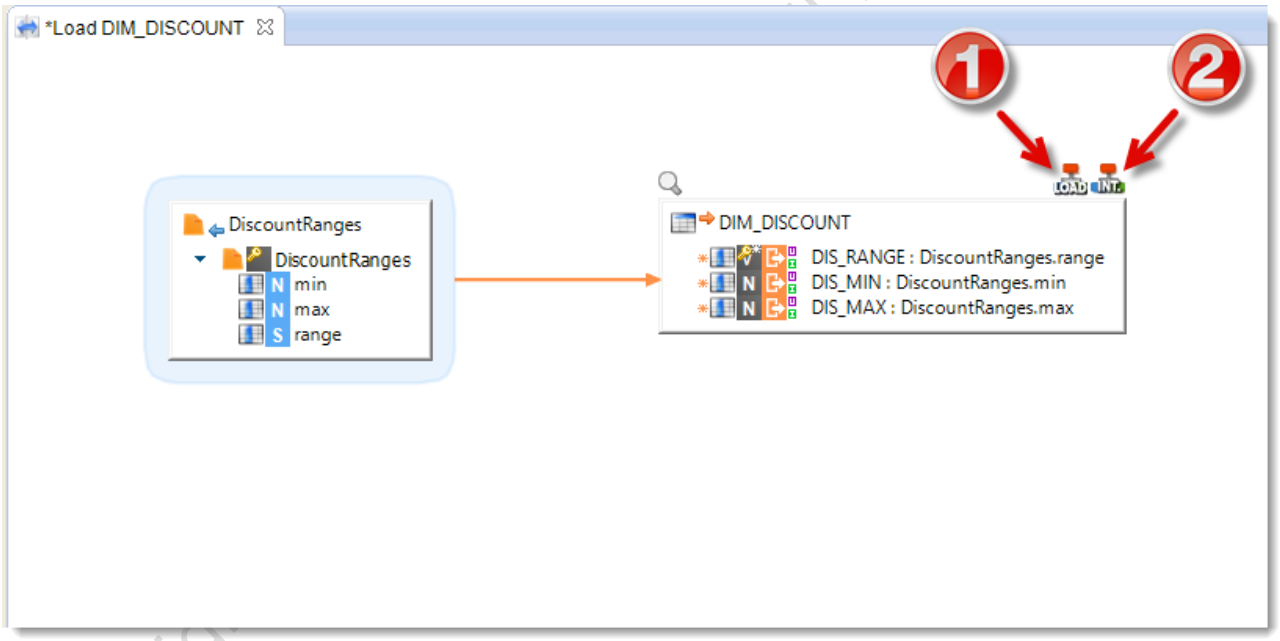
最初のドラッグ&ドロップ実行時に、両データストア間でどのような関係を定義すべきかが問われるので、**Map** を選択してください。



この作業を他のカラムに対しても繰り返してください。両データストア間の関係は最初に選択済みなので、2 回目以降は問われません。



7.2.4 統合およびロードステップ



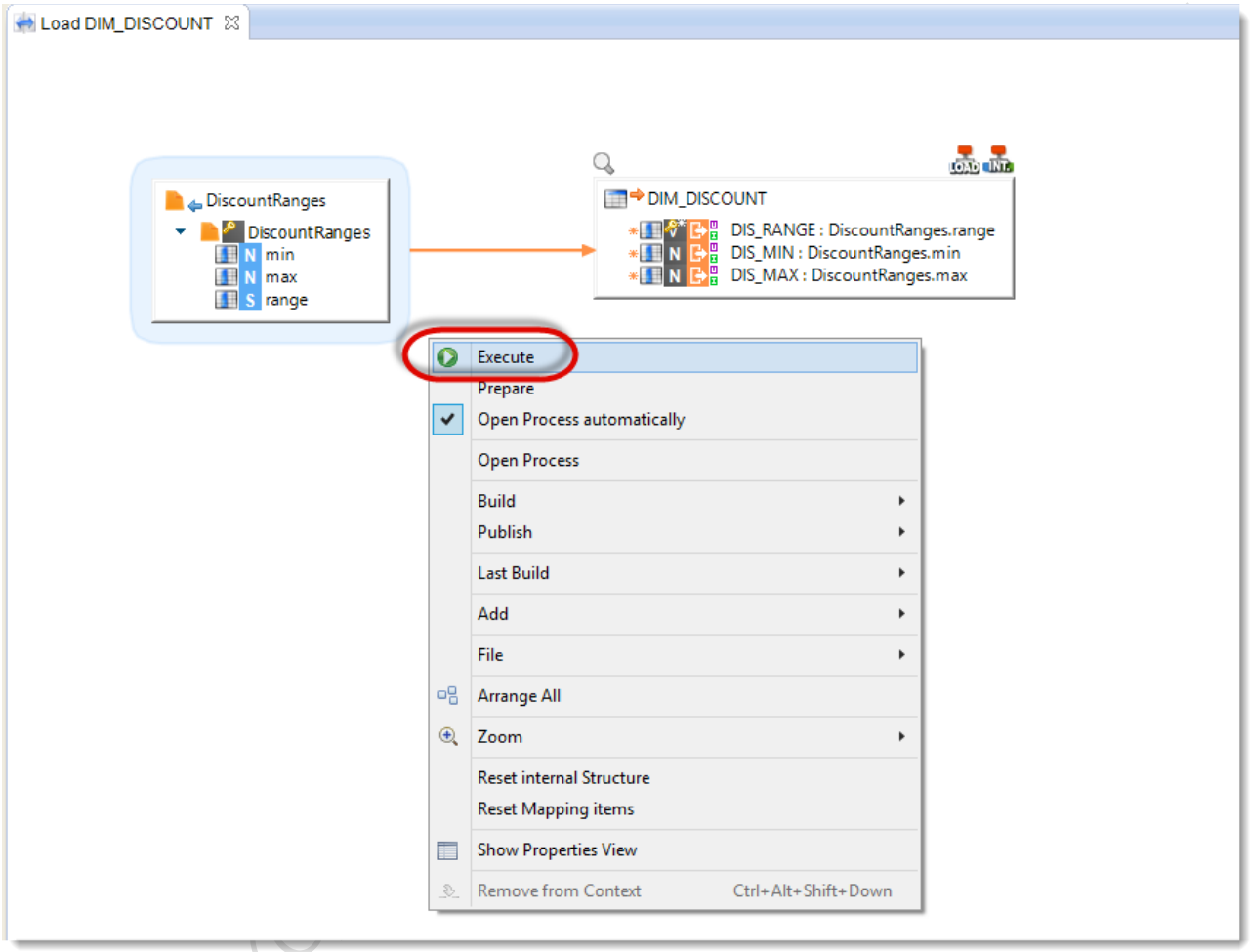
Stambia Designer はソース（例：ファイル）とターゲット（例：データベーステーブル）が異なる技術で保管されている場合に、それを検知し、自動的に **Load** ステップに組み込む機能を備えています。これにより、ソースをターゲットデータベースにロードするテンプレートの適切な定義および構成が可能になります。マッピングでは、デフォルト値を使用することができます。

Integration（統合）ステップが自動的に追加され、データをターゲットに統合し、コンフィギュレーションを行うテンプレートを定義します。よって、マッピングにデフォルト値をそのまま使用できます。（Load Template のコンフィギュレーション方法は、当チュートリアルにおいて、他のマッピング例で解説します。）

7.2.5 マッピングの実行

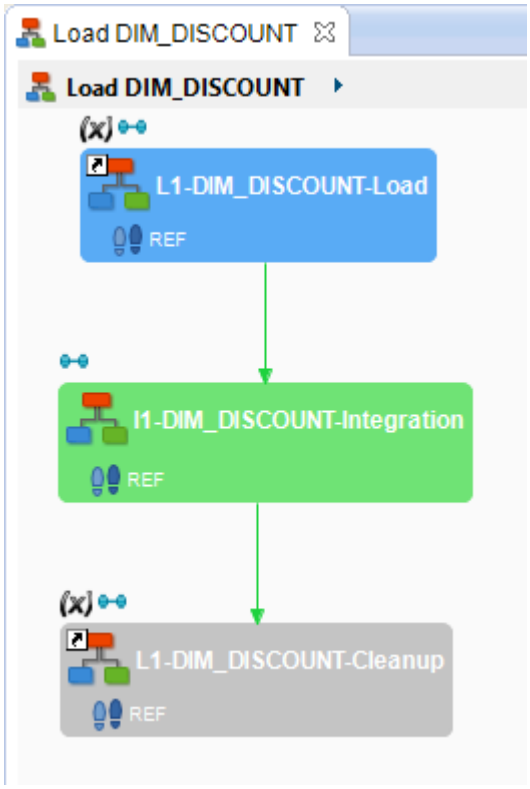
マッピングが完了したら、以下の手順を行い、実行します。

1. マッピングを保存します。
2. 次にマッピング表示の画面上を右クリックします。
3. **Execute** をクリックします。



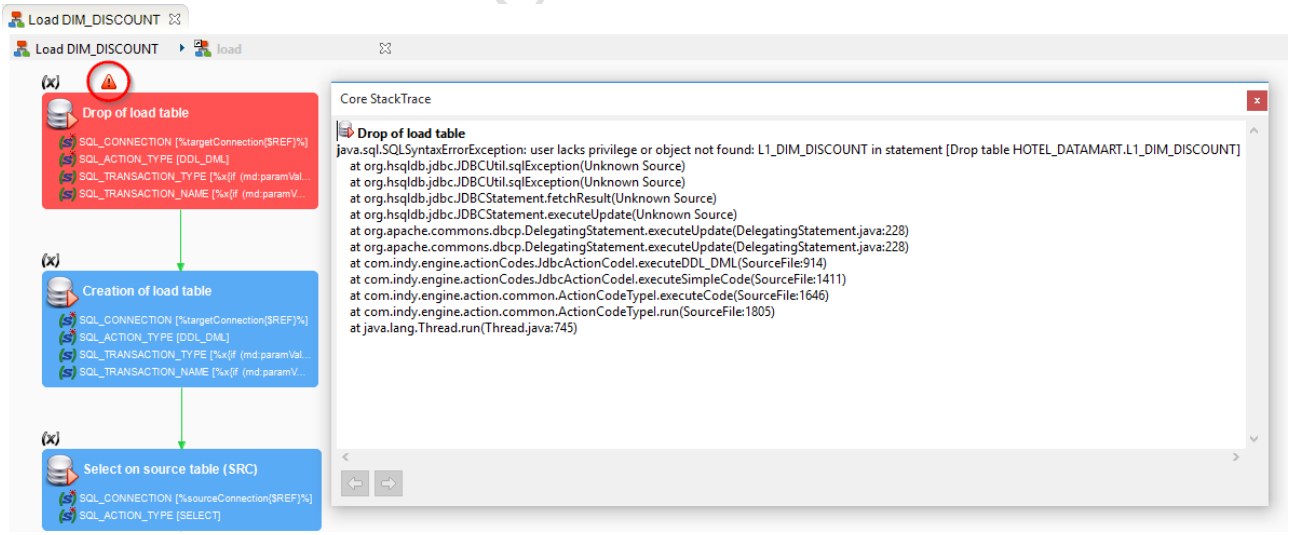
Stambia Designer によりマッピングの実行が開始され、マッピングに定義されたビジネスルールとテンプレートにもとづいて生成された **Process** を表示するウィンドウが開きます。同ウィンドウで、マッピングの実行状況を確認できます。

- 灰色で表示されたステップは、実行待ちのステップです。
- 緑色で表示されたステップは、実行中のステップです。
- 青色で表示されたステップは、実行完了のステップです。
- 赤色で表示されたステップは、実行失敗のステップです。



Runtime が起動されており、正しく接続されていることを確認してください。不確かな場合は[チュートリアル環境を起動する]をご参照ください。

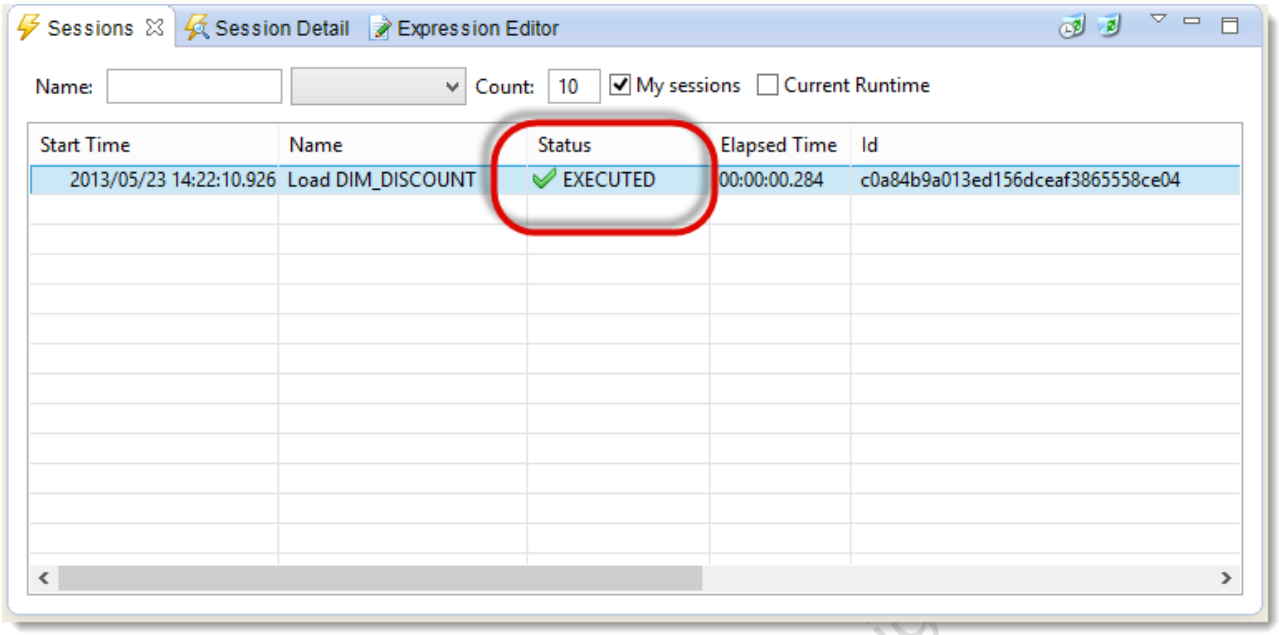
プロセスの実行中にエラーが生じた場合、特別なアイコンが表示されます。それをクリックすれば、エラーの詳細ログが表示されます。



7.2.6 マッピング実行結果の分析する

マッピングの実行により、**Stambia** に **Session**（セッション）が生成され、それが **Runtime**（ランタイム）に対して実行されます。

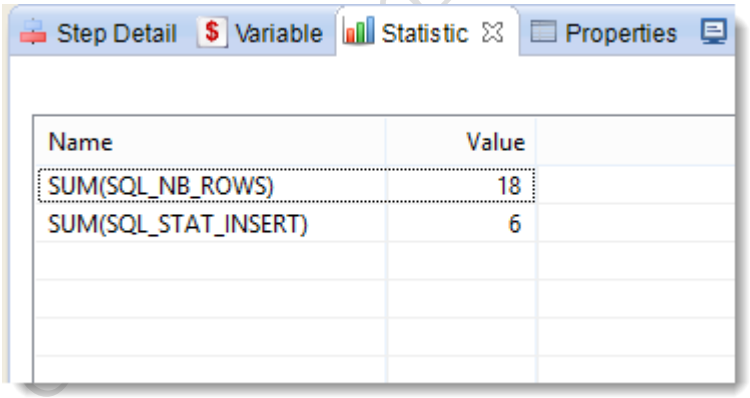
セッションのステータスは、**Sessions** ビューで確認できます。



また、実行結果の統計データが **Statistic** ビューで確認できます。同ビューには下記の情報がリスト表示されます。

演算	結果
SUM(SQL_NB_ROWS)	18
SUM(SQL_STAT_INSERT)	6
SUM(SQL_STAT_UPDATE)	0

上記のリストから、18 件のレコードが処理され、6 件が挿入されたことが判ります。



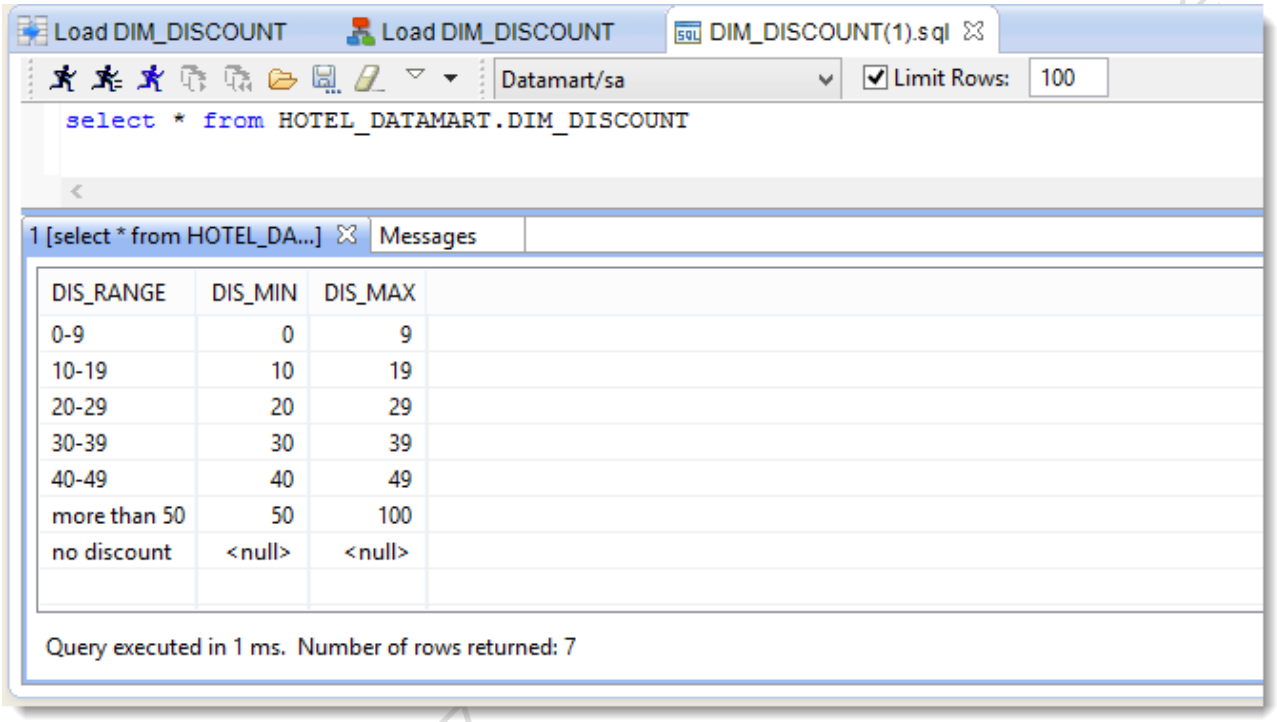
ここで、マッピングを再実行すれば、統計データが下記のように更新されます。

演算	結果
SUM(SQL_NB_ROWS)	18
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	0

すべてのレコードは前回の実行時に挿入されたため、挿入されたレコード数が 0 件に変わります。

次にデータがターゲットテーブルに正しく挿入されたかどうかを確認します。

1. **Mapping** ウィンドウにおいて、ターゲットデータストア(**DIM_DISCOUNT**)を右クリックし、**Action > Consult Data** を選択します。
2. SQL リクエストエディタが開かれます。をクリックするか、**CTRL + Enter** を入力してリクエストを実行します。



8 その他のメタデータを作成する

当チュートリアルを先に進めるために、ここで、使用される全データソースを定義する必要があります。

8.1 ソースモデルをリバースする

当チュートリアルで使用するデータベースには、各種マッピングのソースとして使用される Hypersonic SQL データベースがもう 1 つあります。対応するメタデータファイルを作成し、下記のプロパティにしたがって、リバースエンジニアリングを行ってください。

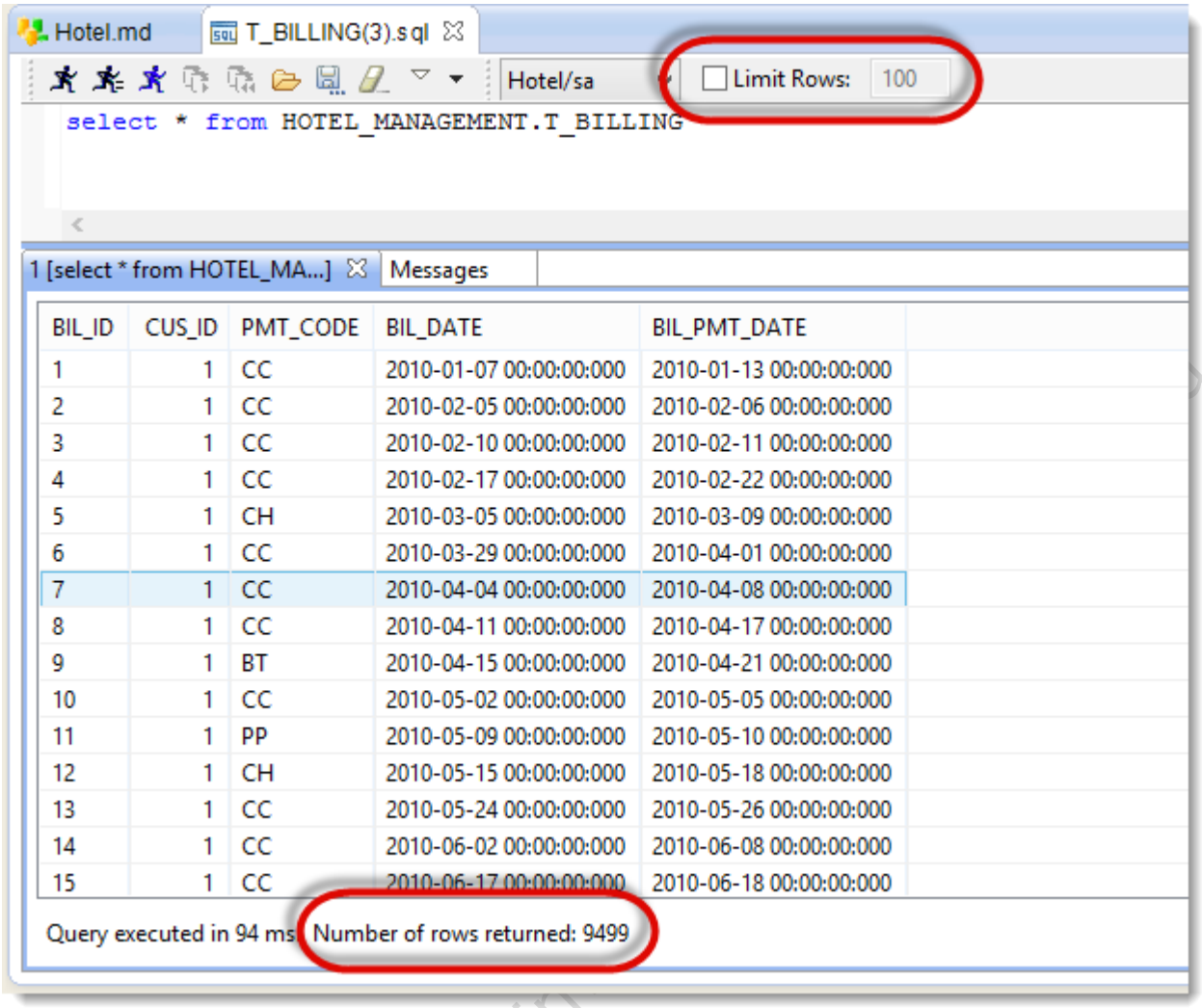
プロパティ	値
ディレクトリ	Tutorial – Fundamentals プロジェクトにメタデータファイルディレクトリを作成
技術	Hypersonic SQL
メタデータモデル	Hotel
URL	Jdbc:hsqldb:sql://localhost:62210
ユーザー名	sa
パスワード	
スキーマ	HOTEL_MANAGEMENT

上記以外のプロパティについてはデフォルト値を使用してください。

リバースエンジニアリングの手順で不明な点は、ターゲットデータベースのリバースを参照してください。

データベースのデータの詳細は各データストアを右クリックし、**Actions** > **consult data** を選択することで確認できます。

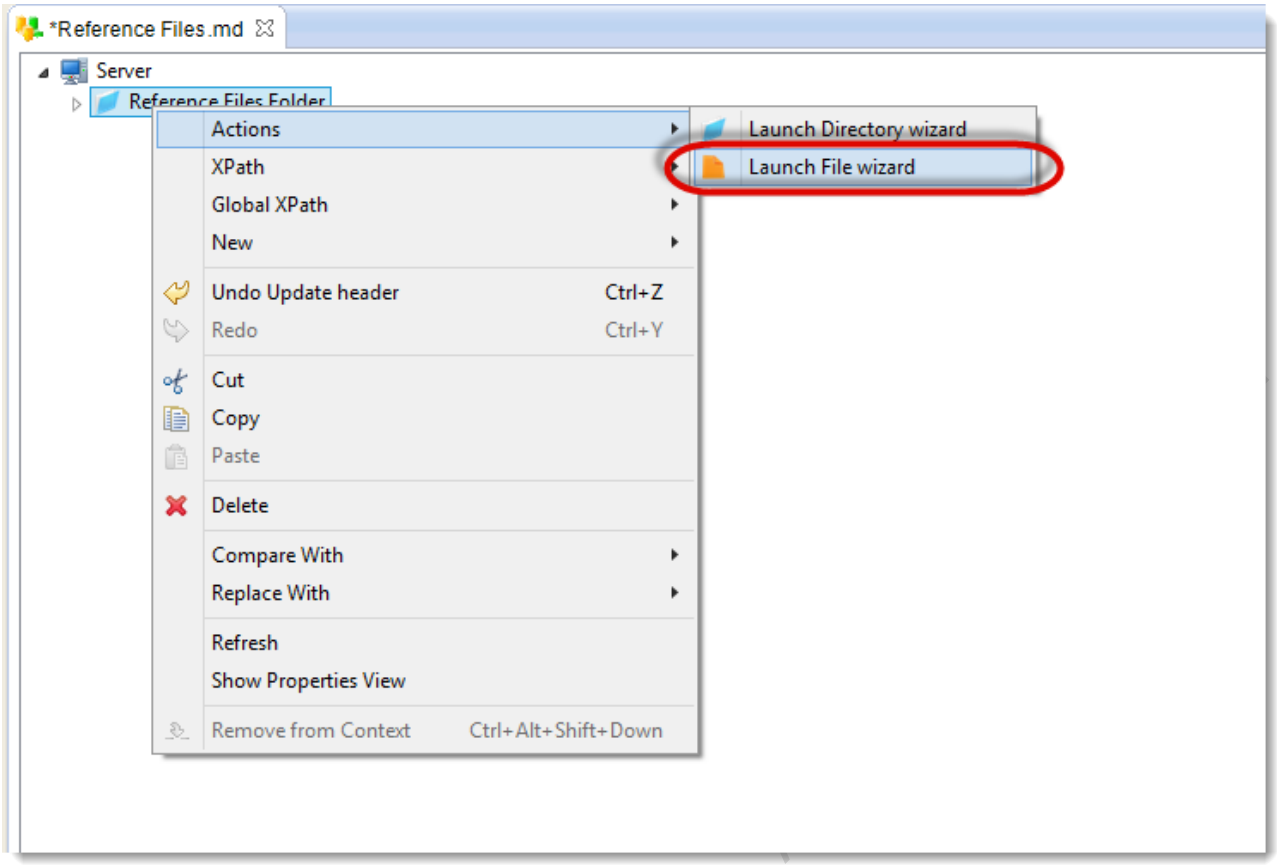
テーブルの中には、大量のレコードを含むものがあります（特に、T_BDR_PLN_CUS、T_BILLING、T_BILLING_LINES、T_PLANNING などのテーブル）。**Stambia Designer** のデフォルト設定では、処理速度を損なわないように、テーブルの読み取り行数が最大 100 までに制限されています。大きなテーブルの全レコードを見たいときは、**Limit Rows** チェックボックスの選択を外してください。



8.2 その他のデリミテッドファイルのリバースする

当チュートリアルでフラットファイルを含むフォルダは定義済みなので、新しいメタデータファイルを作成する必要はありません。

1. Reference Files ファイルをダブルクリックします。
2. **server** ノードを開きます。
3. Reference Files Folder を右クリックし、**Actions** > **Launch File wizard** を選択します。



8.2.1 Time.csv ファイルをリバースする

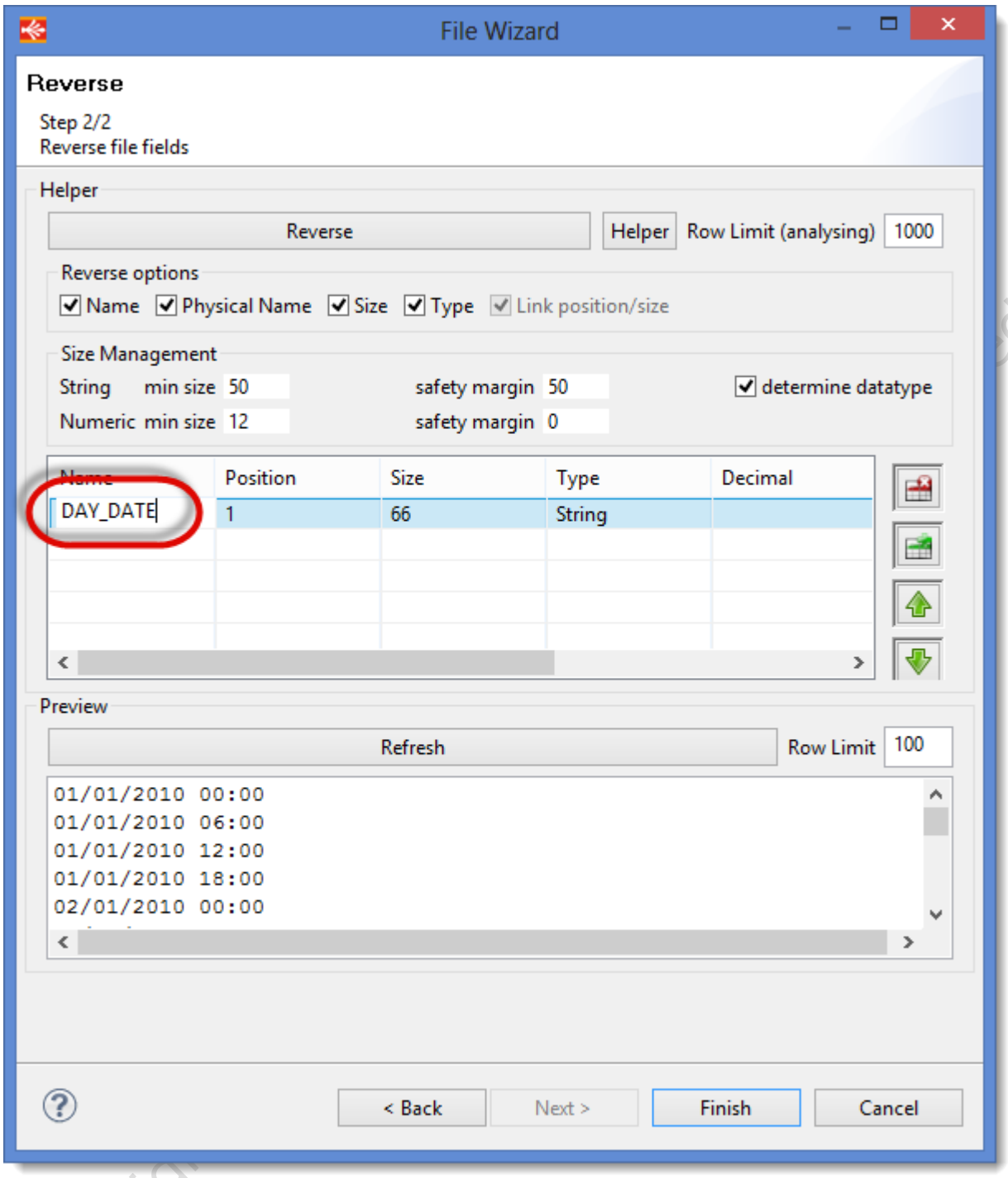
デリミテッドファイルを下記のプロパティで定義します。

プロパティ	値
Physical Name	Time.csv
Header Line Position	**0**

このファイルには下記のカラムが含まれます。

カラム	ポジション	タイプ
DAY_DATE	1	String

カラム名を更新するため、リバースエンジニアリングを行い、その後、テーブルのカラム名を直接変更します。



ファイル宣言が完了したら、**Stambia** 上で、データを閲覧、確認できます。

データを閲覧、確認する前に、メタデータファイルを必ず保存してください。

8.2.2 REF_US_STATES.csv ファイルをリバースする

デリミテッドファイルを下記のプロパティで定義します。

プロパティ	値
Physical Name	REF_US_STATES.csv
Field Separator	,
Header Line Position	1
Name	US_States

データストアの名前には、自動的に拡張子を除いたファイル名が使用されます。この自動生成された名前は、より分かりやすいもの（例：US_States）に手動で変更することができます。

このファイルには下記のカラムが含まれます。

カラム	ポジション	タイプ
STATE_UPPER_CASE	1	String
STATE	2	String
STATE_CODE	3	String

8.3 ポジショナルファイルをリバースする

レコードが区切り文字（デリミタ）で定義されているフラットファイルをデリミテッドファイルと呼ぶのに対し、各レコードの文字位置で定義するフラットファイルをポジショナルファイルと呼びます。ポジショナルファイルは、当チュートリアルでは、以下の方法で作成します。

1. File Reverse assistant を起動します。
2. **Browse** をクリックし、ref_us_cities.txt ファイルを選択します。
3. **Name** に US_Cities と入力します。
4. **Type** を修正し、**POSITIONAL** を選択します。

Refresh をクリックして、ファイルの内容を確認することができます。

Next をクリックすれば、カラム定義を行うウィンドウが表示されます。

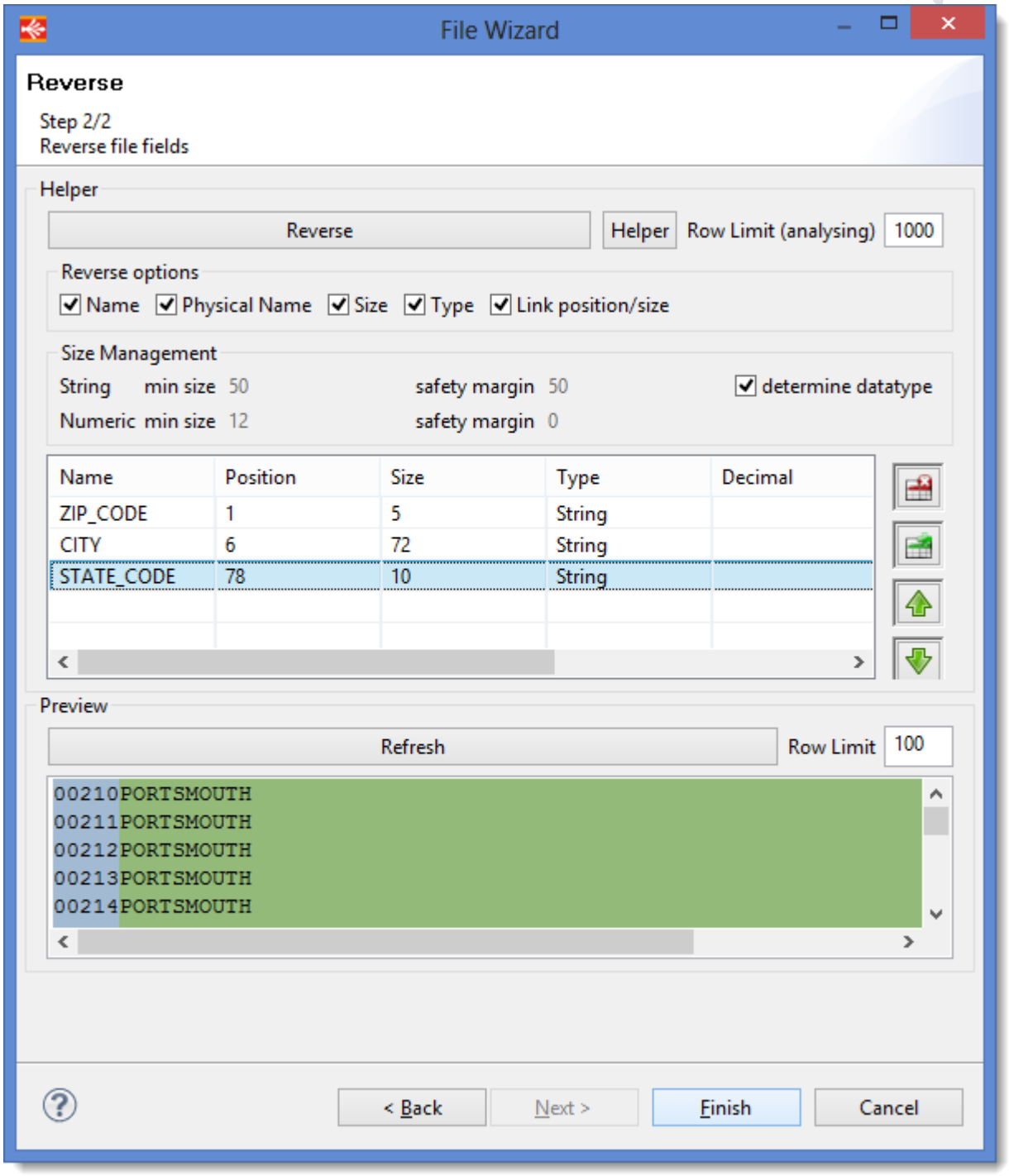
ポジショナルファイルのリバースエンジニアリングでは、複数カラムが連結している場合、それらは異なるカラムとして検知されません。手動で定義しなおしてください。下記のカラムに以下の作業を行い、定義を変更します。

1.  をクリックして、新しいカラムを宣言します。
2. テーブルにおいて、カラムのプロパティを変更します。

カラム名	ポジション	サイズ	タイプ
ZIP_CODE	1	5	String
CITY	6	72	String
STATE_CODE	78	10	String

デリミテッドファイルのポジションはカラムの番号を表しますが、ポジショナルファイルのポジションは各カラムの最初の文字の位置を表します。

ファイルのレイアウトを確認するには、**Refresh** をクリックします。



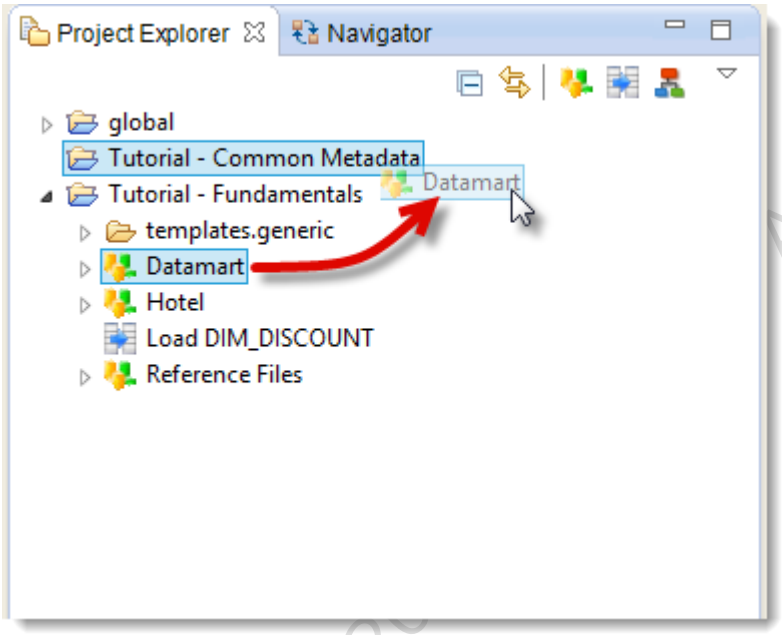
9 開発要素の整理・再編成する

9.1 メタデータを単一プロジェクトにまとめる

一般的に、メタデータファイルは多くのプロジェクトに共有されます。メタデータファイルをそれ専用のプロジェクトにまとめ、管理および共有しやすくすることを推奨します。

さらに、当チュートリアルで使用されるメタデータファイルには、その他の全チュートリアルと共有されるものもあります。そのようなメタデータファイルを 1 つのプロジェクトにまとめておきましょう。

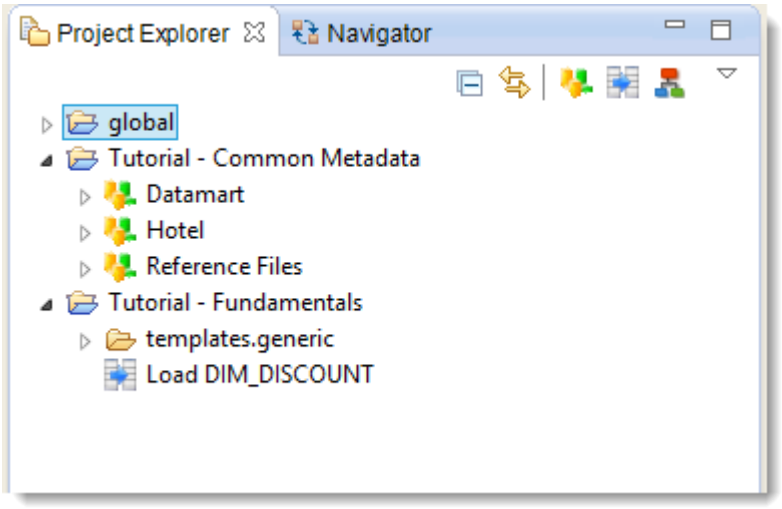
1. Tutorial – Common Metadata プロジェクトを作成します。
2. メタデータ Datamart ファイルを Tutorial – Common Metadata プロジェクトにドラッグ&ドロップします。



複数選択によって時間を短縮することができます。

1. Hotel を選択します。
2. CTRL キーを押しながら、Reference Files を選択します。
3. 全体を Tutorial – Common Metadata プロジェクトにドラッグ&ドロップします。

以上でメタデータ専用プロジェクトが完成します。



上記のファイル整理は、Load DIM_DISCOUNT のマッピングに影響も及ぼしません。マッピングを開き、再実行すれば、何も変わっていないことが判ります。したがって、**Stambia Designer** では、開発要素の整理・再編成が非常に手軽に行えます。

9.2 テンプレートをグローバルオブジェクトに移動

テンプレートは **global** プロジェクトにまとめるのが一般的です。これにより、テンプレートの更新が、すべてのプロジェクトに反映されるようになります。

- **Templates.generic** フォルダをドラッグ&ドロップして **global** プロジェクトに移動してください。

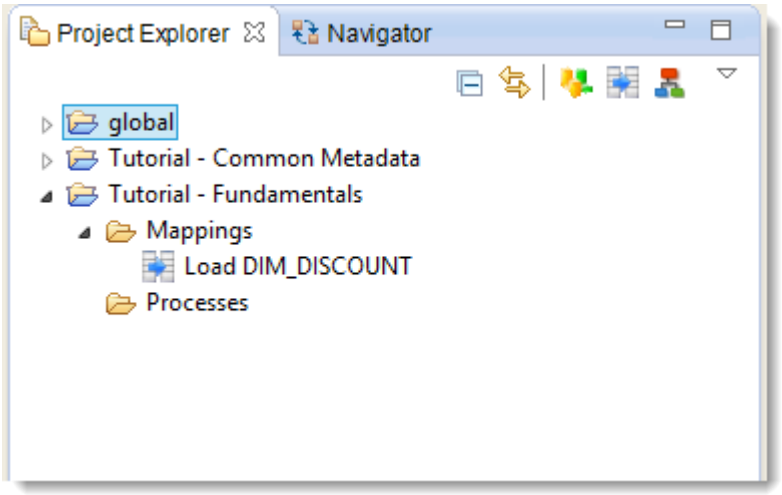
テンプレートは **global** 以外のプロジェクトに保存することも可能です。しかし、他のプロジェクトから閲覧可能にするには、両プロジェクト間で依存性を指定しなければなりません（**Project references** セクションにおいて、プロジェクトの **Properties** を定義します）。一方、**global** プロジェクトにインポートされたテンプレートは、そのまま全プロジェクトから閲覧可能です。

9.3 開発要素をフォルダごとにまとめる

マッピングをまとめるためのフォルダを作成します。

1. Tutorial – Fundamentals プロジェクトを右クリックし、**New > Folder** を選択します。
2. フォルダ名を Mapping と入力します。
3. **Finish** をクリックします。
4. Mappings フォルダに Load DIM_DISCOUNT をドラッグ&ドロップします。

上記と同様に、Processes フォルダも作成してください。当チュートリアルで後ほど使用します。



10 フィルタ付きマッピングを作成する

10.1 DIM_PAYMENT_TYPE テーブルをロードするマッピングの作成

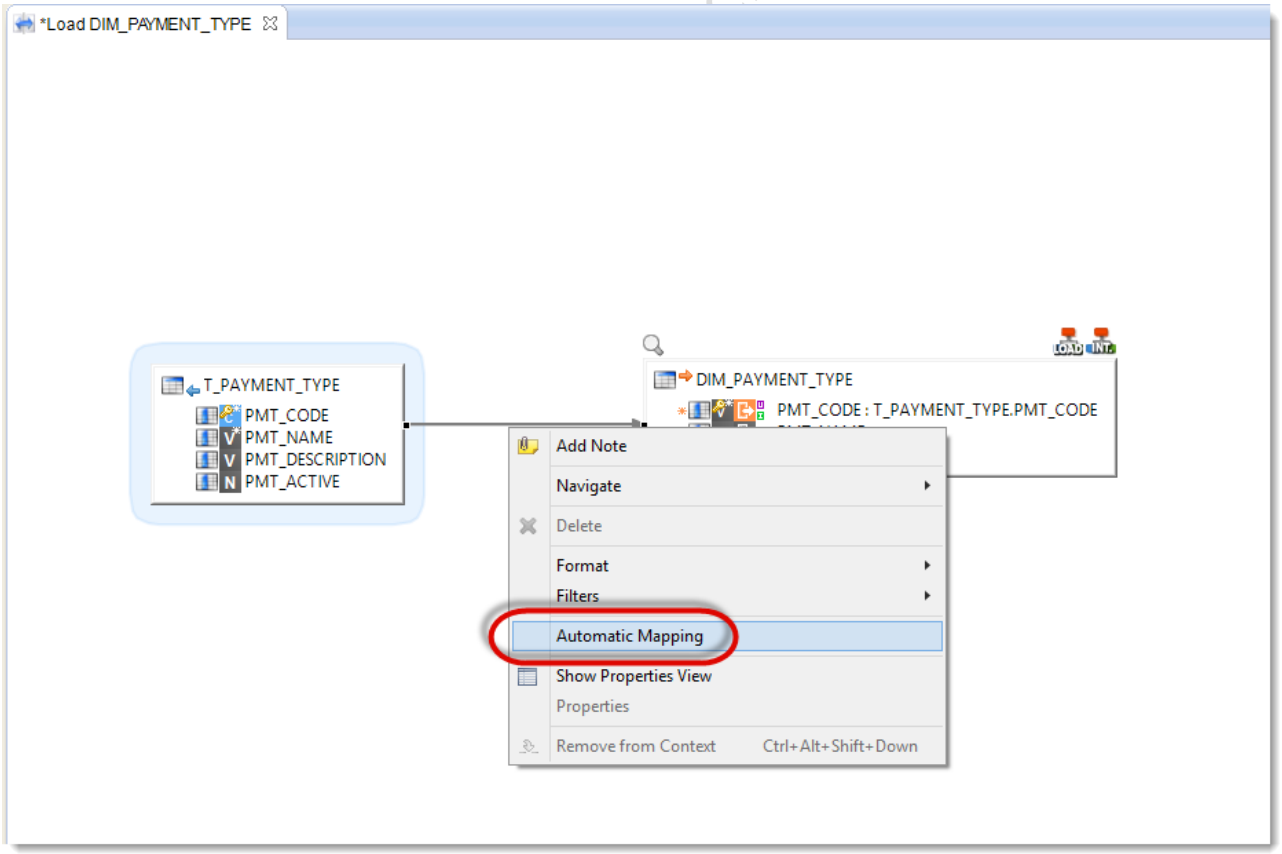
Mappings フォルダに下記のプロパティに基づき新しいマッピングを作成します。

プロパティ	値
マッピング名	Load DIM_PAYMENT_TYPE
ターゲットデータストア	DIM__PAYMENT_TYPE
ソースデータストア	T_PAYMENT_TYPE

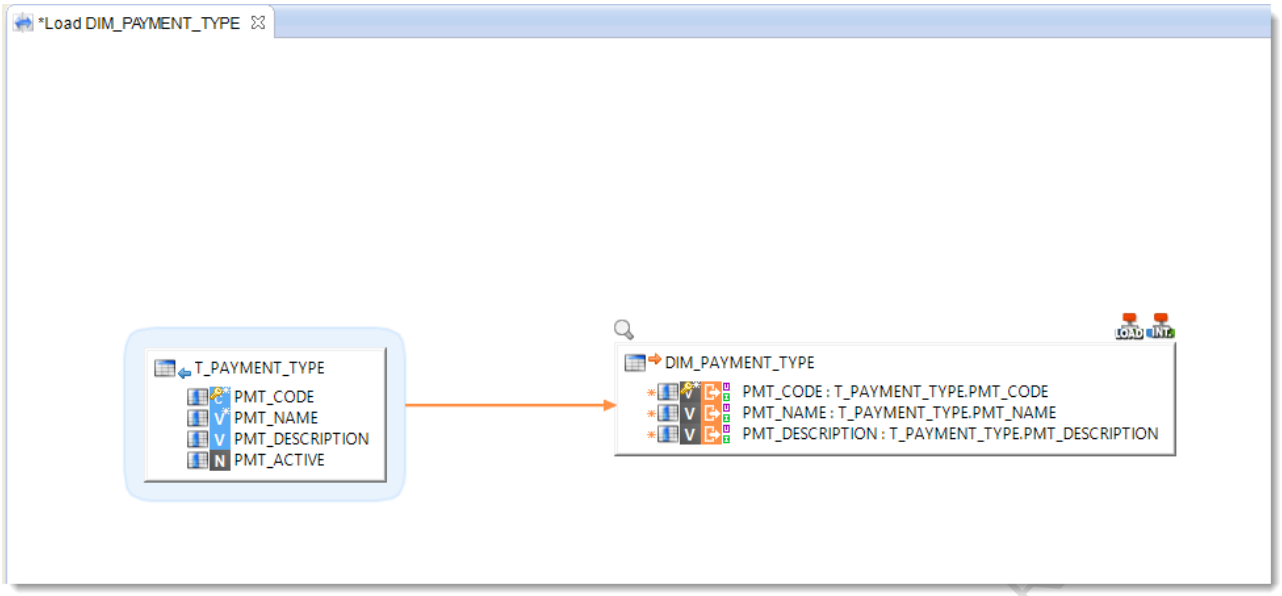
メタデータファイルは現在 Tutorial – Common Metadata プロジェクトに属することに注意してください。

2 つのデータストア間で少なくとも 1 件の変換式が定義されていれば、**Stambia Designer** は同じ名前を持つターゲットとソースを自動的にマッピングします。それに相当しないケースでは、手動で各種フィールドのマッピングを起動することも可能です。

そのためには、まず、ツールバーの  アイコンをクリックして、データストア間のリンクを表示させます。その後、データストア間のリンクを右クリックし、**Automatic Mapping** を選択します。



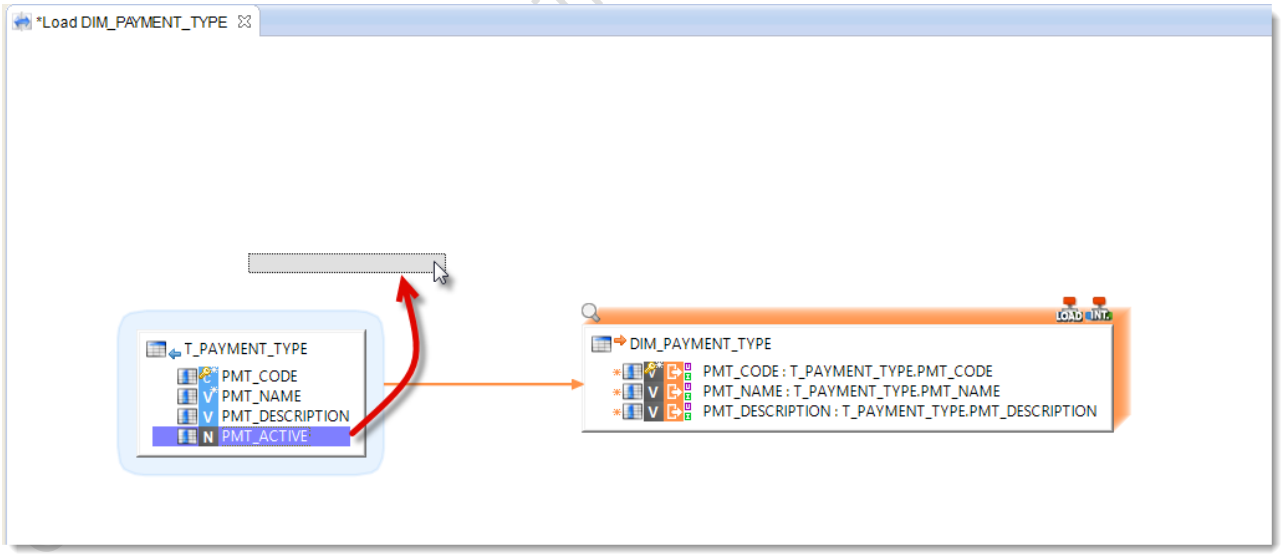
これにより、すべてのカラムに対してマッピングが完了します。



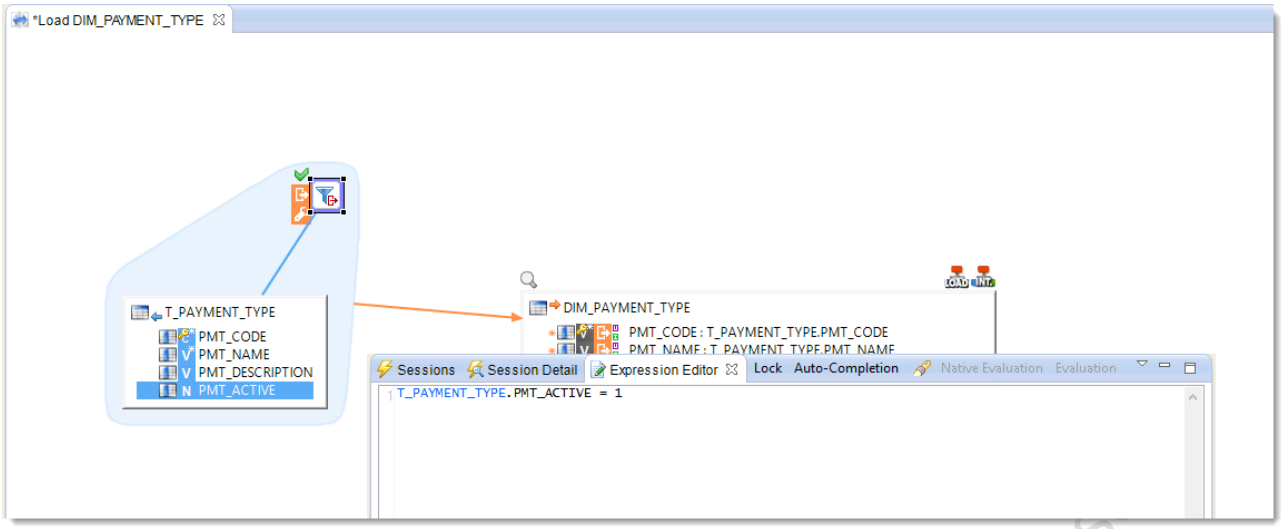
10.2 フィルタを追加する

DIM__PAYMENT_TYPE をロードするためには、T_PAYMENT_TYPE テーブルの古いデータを選別する必要があります。保持するべき有効データは PMT_ACTIVE の値を 1 にセットします。**Filter**（フィルタ）はソースデータストアに作成します。

- **PMT_ACTIVE** カラムを画面の背景にマウスでドラッグ&ドロップします。



- **Expression Editor** ビューを開きます。
- フィルタの式が T_PAYMENT_TYPE.PMT_ACTIVE=1 であることを確認します。



10.3 マッピングを実行する

マッピングが完了したので、そのマッピングを実行します。

1. マッピングを保存します。
2. 画面のマッピング図の背景を右クリックします。
3. **Execute** をクリックします。

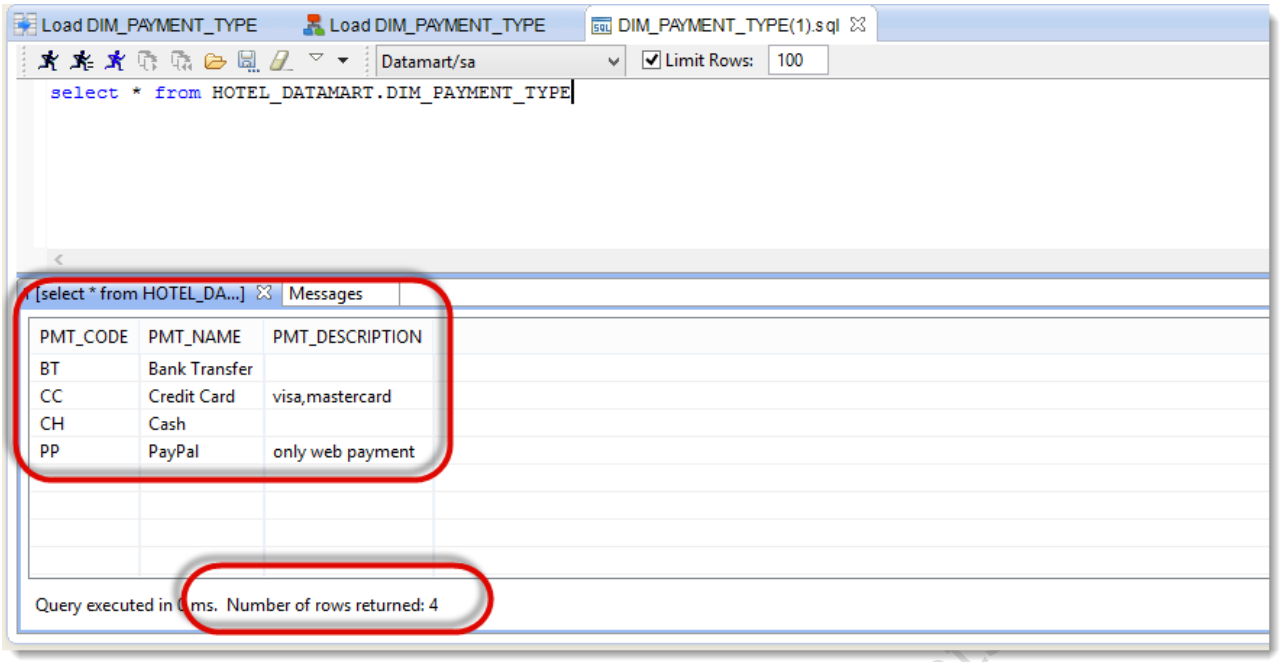
10.4 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	12
SUM(SQL_STAT_INSERT)	4
SUM(SQL_STAT_UPDATE)	0

ターゲットテーブルに下記の 4 レコードが含まれていることを確認します。

PMT_CODE	PMT_NAME	PMT_DESCRIPTION
BT	Bank Transfer	
CC	Credit Card	Visa,mastercard
CH	Cash	
PP	PayPal	Only web payment



データが仕様と合わない場合、フィルタ定義が適切でなかった可能性があります。ターゲットテーブルを再設定するには、空のテーブルからやりなおすのが便利です。

1. ターゲットデータストア **DIM_PAYMENT_TYPES** を右クリックします。
2. **Action > Consult data** を選択します。
3. 次の SQL コマンドを入力します： delete from HOTEL_DATAMART.DIM_PAYMENT_TYPE
4. Click on をクリックするか、**CTRL+Enter** を入力します。
5. フィルタを修正し、実行しなおします。

10.5 生成コードを確認

Stambia Designer によって生成されたプロセスのウィンドウに戻ります。このプロセスは下記の 3 種類のブロックから構成されます。

- L1_DIM_PAYMENT_TYPE-Load
- I1_DIM_PAYMENT_TYPE-Integration
- L1_DIM_PAYMENT_TYPE-Cleanup

各ブロックは Load DIM_PAYMENT_TYPE マッピングに使用されるテンプレートに由来します。

L1_DIM_PAYMENT_TYPE-Load と L1_DIM_PAYMENT_TYPE-Cleanup は、ロードステップに使用されるテンプレートに由来します。L1_DIM_PAYMENT_TYPE-Load がソースからターゲットにデータを移す作業を担い、L1_DIM_PAYMENT_TYPE-Cleanup が L1_DIM_PAYMENT_TYPE-Load ブロックの実行中に生じた一時オブジェクトのクリーンアップに使用されます。

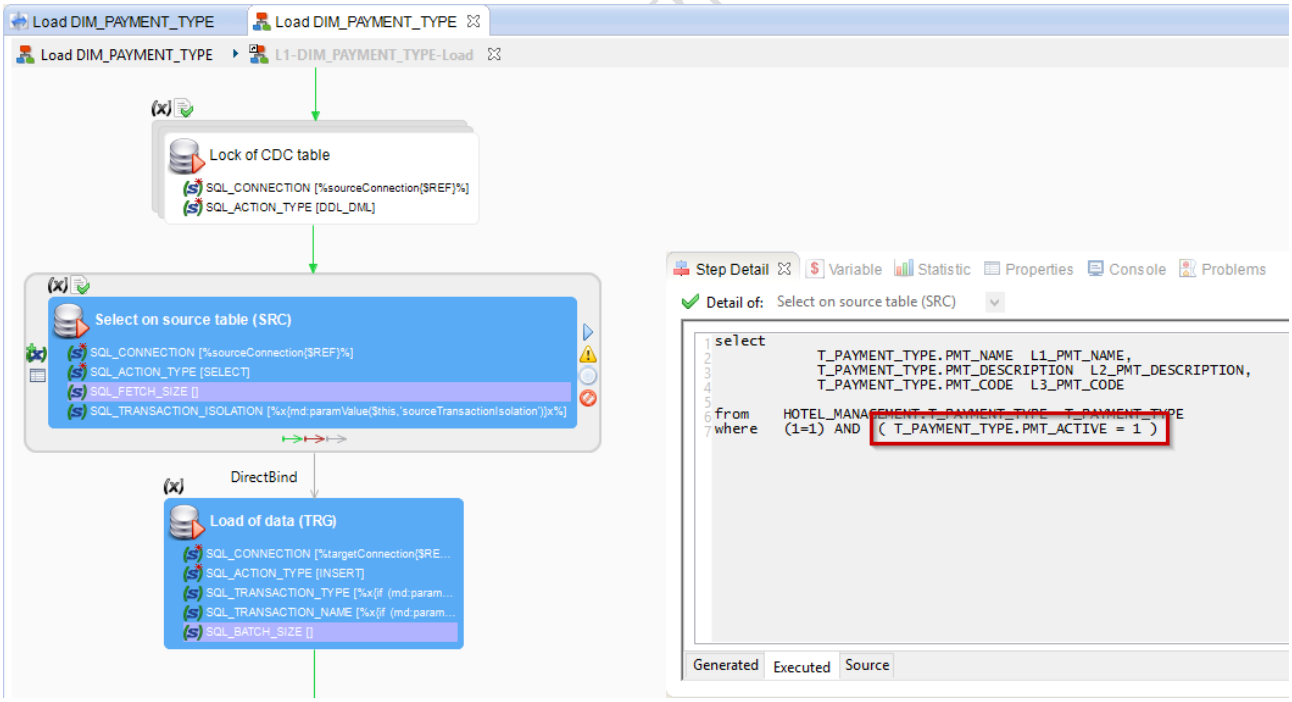
I1-DIM_PAYMENT_TYPE-Integration ブロックをダブルクリックして内容を確認すると、さらに 2 つのブロック、I1-DIM_PAYMENT_TYPE-Prepare および I1-DIM_PAYMENT_TYPE-Integration から構成されているのが判ります。

I1-DIM_PAYMENT_TYPE-Prepare と I1-DIM_PAYMENT_TYPE-Integration は、統合ステップに使用されるテンプレートに由来します。I1-DIM_PAYMENT_TYPE-Prepare が初期ステップ（一時オブジェクトの作成、ソースデータの変換など）を担い、I1-DIM_PAYMENT_TYPE-Integration がデータをターゲットテンプレートに統合する最終的な統合ステップを担います。

各ブロックを開いて、内容を確認してみてください。L1_DIM_PAYMENT_TYPE-Load をダブルクリックすれば、ターゲットデータベースにデータをロードするロードテンプレートにアクセスでき、**Actions** から構成されていることがわかります。**Actions** は処理実行時に **Stambia** に使用される不可欠な要素です。色分けはより上位のブロックレベルと変わりありません。進出の色分けは下記の通りです（既出の色分けはマッピングの実行を参照してください）。

- 白色表示のステップ：当該マッピングに有効ではないので使用されなかったことを示します。
- 黄色表示のステップ：失敗に終わったステップ。エラーはテンプレートにより許容されたことを示します。

Select on source table(SRC)アクションをクリックし、ソースベースに実行されたコードを **Step Detail** ビューで確認してください。特に注目すべきは、先に作成したフィルタを適用するコードが確認できるはずです。



Step Detail ビューの **Source** タブから、テンプレートの適合するコードにアクセスすることができますが、当チュートリアルの目的から外れるため、ここでは取り扱いません。

11 複雑な変換式をともなうマッピングを作成

11.1 DIM_BEDROOM テーブルをロードするマッピングの作成

下記のプロパティに基づき、新しいマッピングを作成します。

プロパティ	値
親フォルダ	Mappings
マッピング名	Load DIM_BEDROOM
ターゲットデータストア	DIM_BEDROOM
ソースデータストア	T_BEDROOM

11.2 ビジネスルールを定義する

たとえソースとターゲットのカラム名が同一でも、データがターゲットに統合される前に、その変換方法を指定しなければなりません。下記の各カラムに対して、ビジネスルールを定義します。

1. ターゲットカラムをクリックします。
2. **Expression Editor** ビューにおいて、変換式を記入します。

ターゲットカラム	ビジネスルール
BOR_ID	T_BEDROOM.BOR_ID
BDR_NUMBER	T_BEDROOM.BOR_NUMBER
BDR_FLOOR	case when lower(T_BEDROOM.BDR_FLOOR) = 'gf' then 0 when lower(T_BEDROOM.BDR_FLOOR) = '1st' then 1 when lower(T_BEDROOM.BDR_FLOOR) = '2nd' then 2 end
BDR_BATH	case when T_BEDROOM.BDR_BATH = 'true' then 1 else 0 end
BDR_SHOWER	case when T_BEDROOM.BDR_SHOWER = 'true' then 1 else 0 end
BDR_BAR	case when T_BEDROOM.BDR_SHOWER = 'true' then 1 else 0 end

BDR_BED_COUNT	Convert(T_BEDROOM.BDR_BED_COUNT,NUMERIC)
BDR_PHONE_NUMBER	T_BEDROOM.BDR_PHONE_NUMBER
BDR_TYPE	T_BEDROOM.BDR_TYPE
UPDATE_DATE	current_timestamp

以上の設定でマッピングを実行します。

11.3 マッピング実行結果を検証

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	60
SUM(SQL_STAT_INSERT)	20
SUM(SQL_STAT_UPDATE)	0

11.4 マッピングの最適化

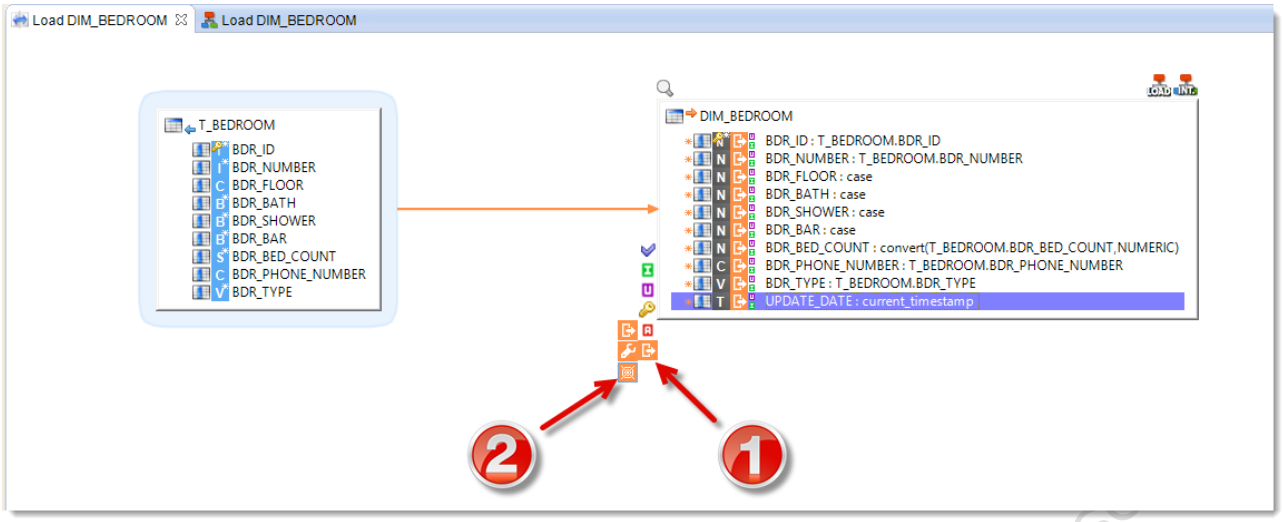
上記のマッピングをデータ変更せずに再実行すれば、下記の結果が得られます。

演算	結果
SUM(SQL_NB_ROWS)	80
SUM(SQL_STAT_INSERT)	00
SUM(SQL_STAT_UPDATE)	20

更新件数（STAT_UPDATE）が 20 となっているのは、**UPDATE_DATE** カラムの式 current_timestamp に由来します。**Stambia** は、デフォルト設定では、式をソースに対して実行します。つまり、何も変更せずにマッピングを実行した場合、ソースからの個々のデータに対し、**UPDATE_DATE** カラムが新しい値で取得されます。マッピングはデフォルト設定では«incremental»モード（データを比べるモード）なので、ターゲットの UPDATE_DATE とは異なるデータとなり、結果的に、ターゲットのデータが更新されることになります。

これを防ぐには、**UPDATE_DATE** フィールドにリンクされたビジネスルールの定義を変更し、変換が実行される場所をソースからターゲットに変更します。それにより、変換式が最後に実行されるようになり、データの比較処理から除外されるようになります。修正手順は以下の通りです。

1. ターゲットカラム **UPDATE_DATE** を選択します。
2. **Execution Location** において、**Datastore** 左横のアイコン をクリックします。
3. **Target** を示すアイコン をクリックします。



Execution Location は、ターゲットカラムを右クリックして、**Execution Location** > **Target** を選択することにより、**Properties** ビューから変更することも可能です。

以上により、マッピングの最適化が完了したので、もう一度マッピングを実行します。

変換処理の実行場所はテーブル内に、Execution Location カラムのビジネスルールとして定義されています。

11.5 マッピング実行結果を再検証

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	60
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	0

変換場所をソースからターゲットに変更後のマッピング再実行結果では、更新件数（STAT_UPDATE）が 0 になっています。ソースのデータが更新された場合のみ、更新件数の数値が変わります。技術フィールドが故意にデータを加算するようなことはありません。

12 結合を伴うマッピングを作成する

12.1 DIM_GEOGRAPHY テーブルをロードするマッピングの作成

下記のプロパティに基づき、新しいマッピングを作成します。

プロパティ	値
親フォルダ	Mappings
マッピング名	Load DIM_GEOGRAPHY
ターゲットデータストア	DIM_GEOGRAPHY
ソースデータストア	US_States、US_Cities

変換ビジネスルールは下記のとおりです。

ターゲットカラム	ビジネスルール	実行場所
GEO_KEY_ID	HOTEL_DATAMART.SEQ_GEO_KEY_ID の次の値	ターゲット
GEO_ZIP_CODE	US_Cities.ZIP_CODE	ソース
GEO_CITY	US_Cities.CITY	ソース
GEO_STATE_CODE	US_Cities.STATE_CODE	ソース
GEO_STATE	US_States.STATE_UPPER_CASE	ソース

GEO_KEY_ID のビジネスルールはターゲットデータベースに固有の連続オブジェクトを使用します。マッピングの最適化で解説した **UPDATE_DATE** のように、ビジネスルールをターゲットで実行し、データフローの不必要な負荷を避けます。

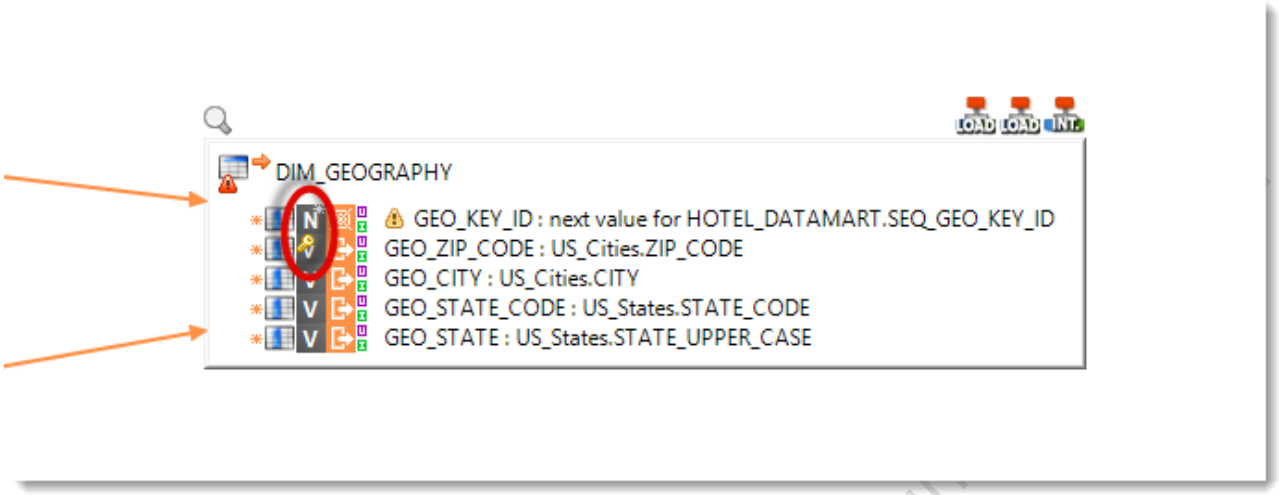
12.2 マッピングのファンクショナルキーを定義する

DIM_GEOGRAPHY テーブルをロードするマッピングの定義において、最初にすべきことは **functional key(ファンクショナルキー)**を定義することです。ファンクショナルキーにより、レコードをここに特有の方法で識別することが可能になります。特に、ソースのレコードがターゲットにすでに挿入済みであるのか、新しいレコードなのかの識別に有効です。

デフォルト設定では **Stambia Designer** は Datastore の主キーに基づき、自動的にファンクショナルキーを選択します。DIM_GEOGRAPHY テーブルの主キーは GEO_KEY_ID です。しかし、GEO_KEY_ID にはソースのデータフローに依存しない（ターゲットを変換場所とする）ビジネスルールが設定されています。これにより、新旧のレコードの識別を GEO_KEY_ID に基づいて行うことができません。当マッピングに対しては、下記の手順に従い、ほかのファンクショナルキーを選択する必要があります。

1. **GEO_KEY_ID** カラムを選択します。
2. アイコンをクリックして、同カラムのファンクショナルキーを無効化します。
3. **GEO_ZIP_CODE** カラムを選択します。
4. アイコンをクリックして、同カラムをマッピングのファンクショナルキーとして定義します。

以上により、**Stambia** で使用するファンクショナルキーが **GEO_ZIP_CODE** に変わります。それにより、ソースレコードの GEO_ZIP_CODE に、ターゲットテーブルに存在しない値がある場合のみ、ターゲットに新しいレコードが作成されます。ターゲットに GEO_ZIP_CODE のソースと同じ値がある場合は、そのレコードの全データが比較され、ターゲットに違う値があれば、ソースの値で更新されます。



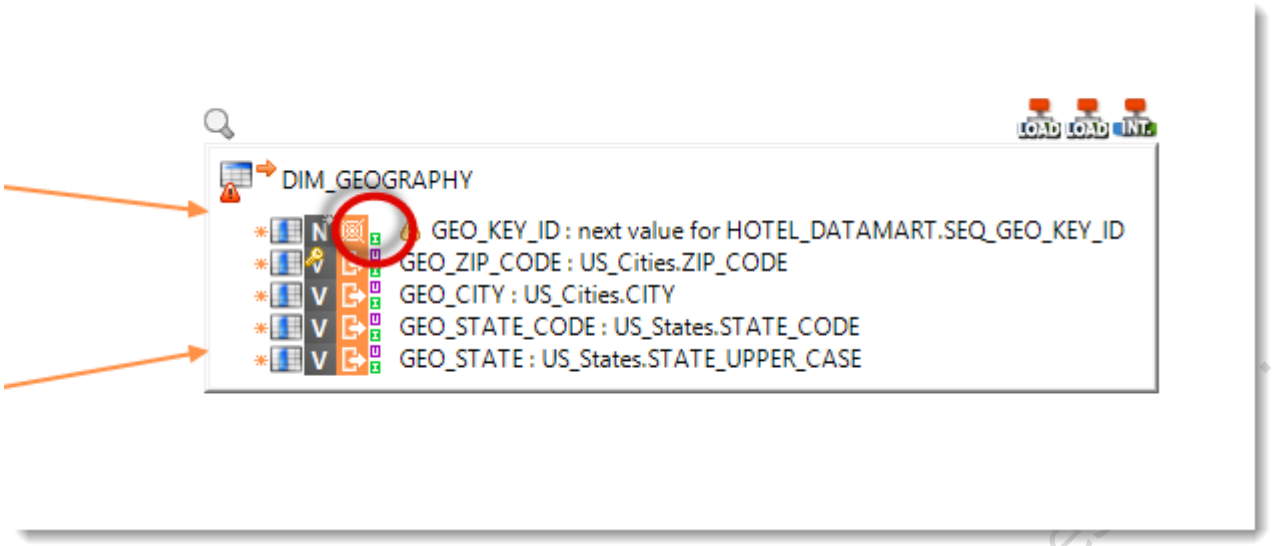
ファンクショナルキーを定義しなおすと、鍵マークのアイコンが移動します（上の例では、GEO_KEY_ID カラムから GEO_ZIP_CODE カラムへ移動）。それにより、どのカラムがマッピングのファンクショナルキーであるかが、ひと目で判ります。

ファンクショナルキーのカラムは変換ビジネスルールを特定するテーブルの Characteristics（特性）カラムに定義されています。

12.3 ビジネスルールを挿入のみに実行する

ターゲットの **GEO_KEY_ID** カラムはデータベースの連続オブジェクトとしてロードされます。つまり、呼び出されるたびに新しい値が送り返されます。例えば、連続するファイル US_Cities の町名に誤表記があり、それが 2 回目のマッピング実行時に修正されたとします。そのレコードはデータフローに既存するので、町名が更新されなければなりません。しかし、ビジネスルールはロードされるたびに **GEO_KEY_ID** を追加するので、ターゲットに新しいレコードが追加されてしまいます。
これを防ぐには、そのビジネスルールがターゲットへのレコード挿入でのみ適用され、更新では無視されるようにします。

1. **GEO_KEY_ID** カラムを選択します。
2. U アイコンをクリックして、カラムの更新を無効化します。



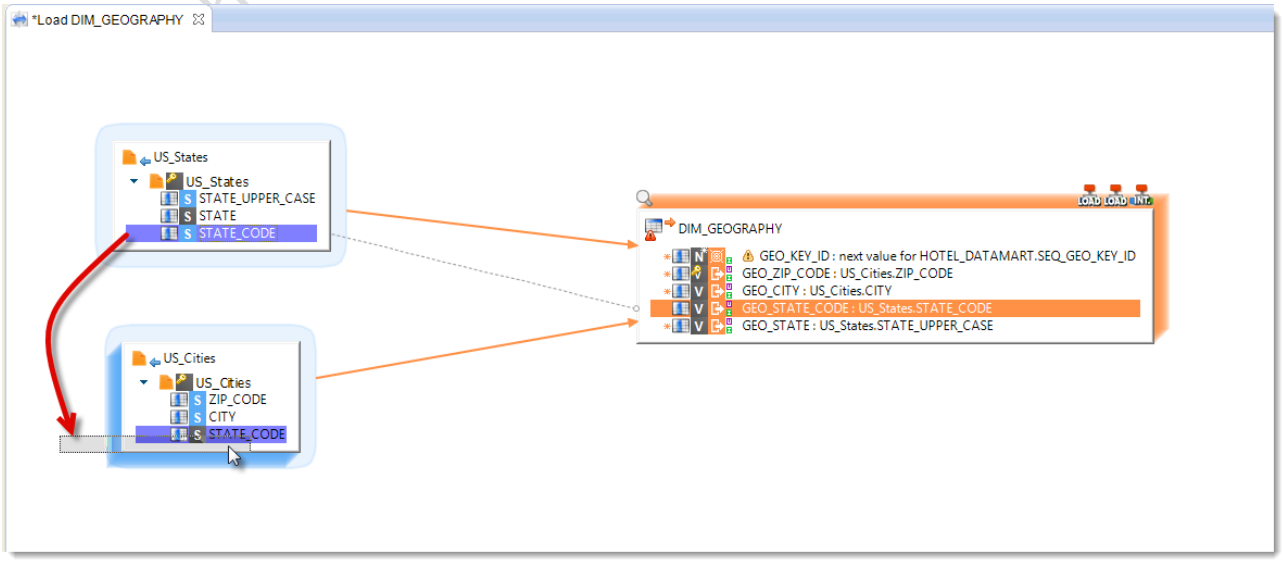
上記の結果、更新を表す U（Update）アイコンの表示が消えます。それにより、更新時にどのビジネスルールが実行されなかったのか、簡単に判るしくみになっています。ちなみに、I（Insert）アイコンは挿入時に実行されるビジネスルールを表します。

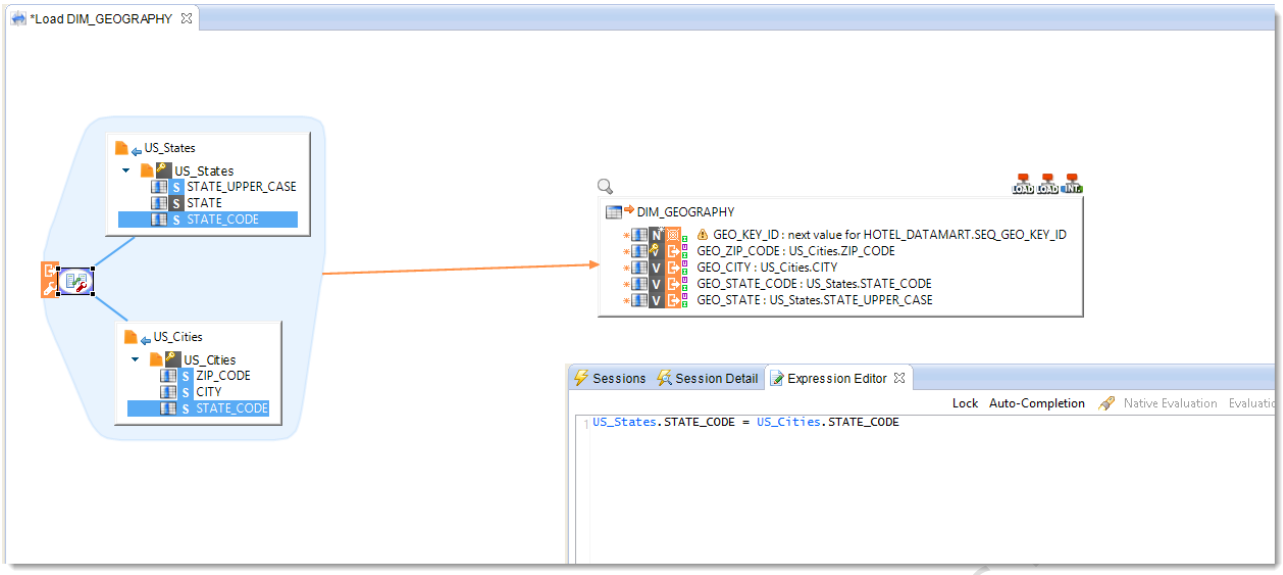
レコードの挿入または更新時に実行されるビジネスルールは、変換ビジネスルールを特定するテーブルの Characteristics カラムに定義されています。

12.4 ソースファイル間の結合を定義する

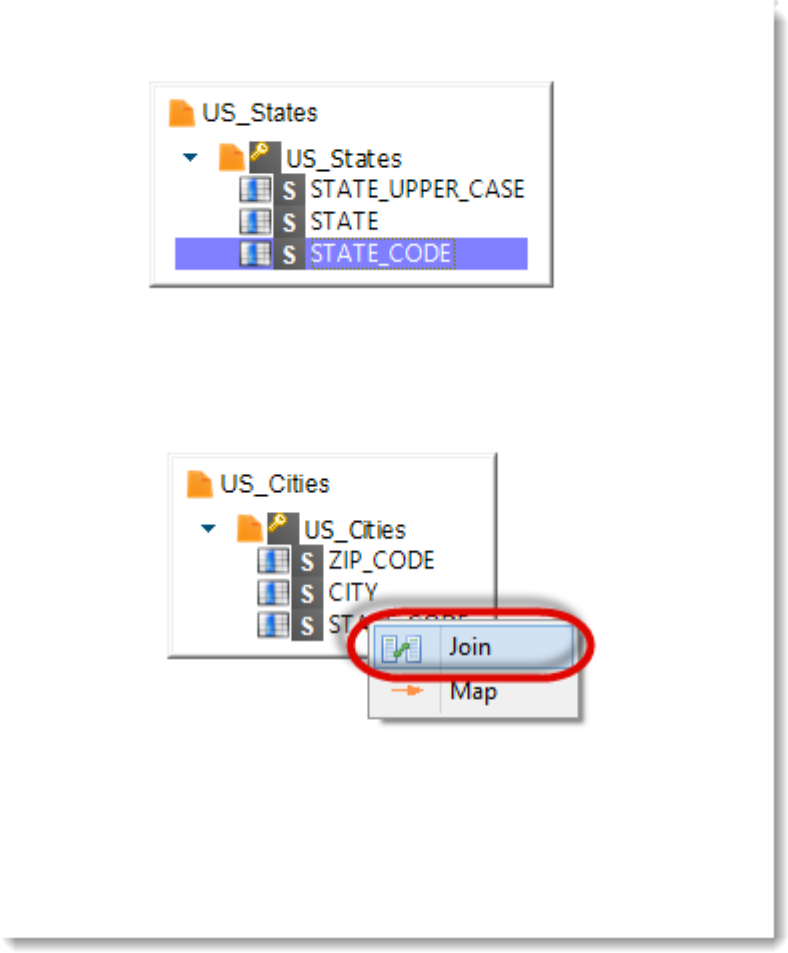
DIM_GEOGRAPHY テーブルをロードするマッピングにおいて、2 つのデータセット US_States と US_Cities は結合されていません。よって、**Stambia** は最初のセットの全行を 2 番目のセットの各行とまとめながら読み取りを繰り返し、直積集合を生成します。それを防ぐには、2 つの Datastore に join（結合）を設定しなければなりません。

1. US_States の **STATE_CODE** カラムを US_Cities の **STATE_CODE** カラムに、ドラッグ&ドロップします。
2. 生成されたビジネスルールが正しいかどうか、**Expression Editor** ビューで確認します。





変換ビジネスルールはターゲットに定義されているので、2 つのデータストア US_States と US_Cities の間に設定し得る関係は結合のみです。よって、上記の手順により、**Stambia** は自動的に結合を生成します。仮に曖昧な点があれば、上記の手順でポップアップメニューが表示され、そこから **Join** を選択することになります。



以上により、マッピングの定義がすべて完成し、実行する準備が整います。

12.5 マッピング実行結果を検証する

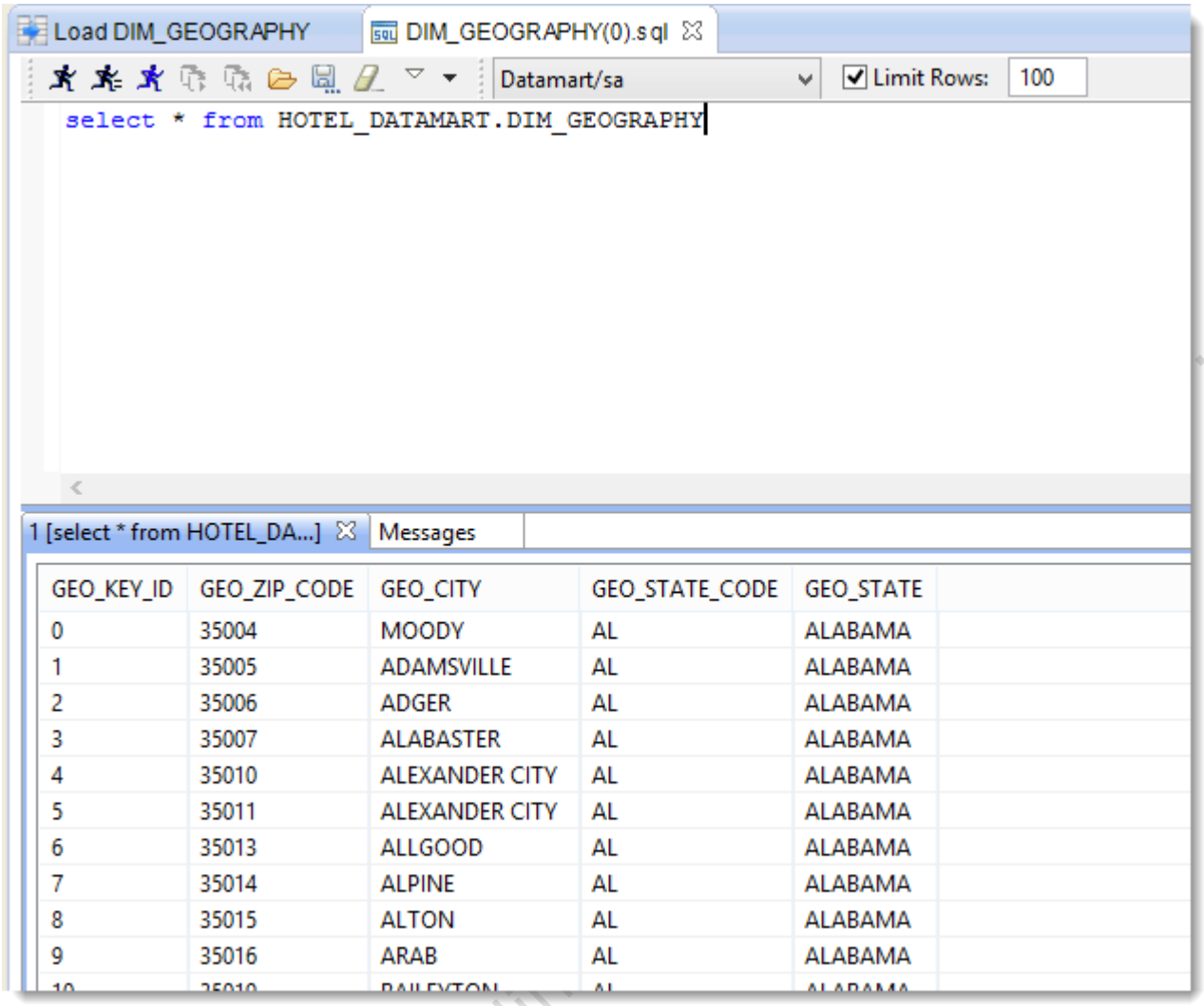
Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	125 628
SUM(SQL_STAT_INSERT)	41 693
SUM(SQL_STAT_UPDATE)	0

当マッピングを再度実行すると、次の結果が得られます。

演算	結果
SUM(SQL_NB_ROWS)	125 628
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	0

ターゲットテーブルのデータも確認すると以下のようになります。



GEO_KEY_ID	GEO_ZIP_CODE	GEO_CITY	GEO_STATE_CODE	GEO_STATE
0	35004	MOODY	AL	ALABAMA
1	35005	ADAMSVILLE	AL	ALABAMA
2	35006	ADGER	AL	ALABAMA
3	35007	ALABASTER	AL	ALABAMA
4	35010	ALEXANDER CITY	AL	ALABAMA
5	35011	ALEXANDER CITY	AL	ALABAMA
6	35013	ALLGOOD	AL	ALABAMA
7	35014	ALPINE	AL	ALABAMA
8	35015	ALTON	AL	ALABAMA
9	35016	ARAB	AL	ALABAMA

13 外部結合を伴うマッピングを作成する

13.1 DIM_CUSTOMER テーブルをロードするマッピングの作成

下記のプロパティに基づき、新しいマッピングを作成します。

プロパティ	値
親フォルダ	Mappings
マッピング名	Load DIM_CUSTOMER
ターゲットデータストア	DIM_CUSTOMER
ソースデータストア	T_CUSTOMER、T_TITLE、T_ADDRESS、DIM_GEOGRAPHY

変換ビジネスルールは下記のとおりです。

ターゲットカラム	ビジネスルール	実行場所	有効化
CUS_ID	T_CUSTOMER.CUS_ID	ソース	I/U
CUS_TITLE	T_TITLE.TIT_NAME	ソース	I/U
CUS_NAME	rtrim(T_CUSTOMER.CUS_FIRST_NAME) + ' ' + upper(rtrim(T_CUSTOMER.CUS_LAST_NAME))	ソース	I/U
CUS_COMPANY	T_CUSTOMER.CUS_COMPANY	ソース	I/U
GEO_KEY_ID	DIM_GEOGRAPHY.GEO_KEY_ID	ソース	I/U
UPDATE_DATE	current_timestamp	ターゲット	I/U
CUS_VIP	（このカラムのロードは当チュートリアル）		

Join のビジネスルールは下記のとおりです。

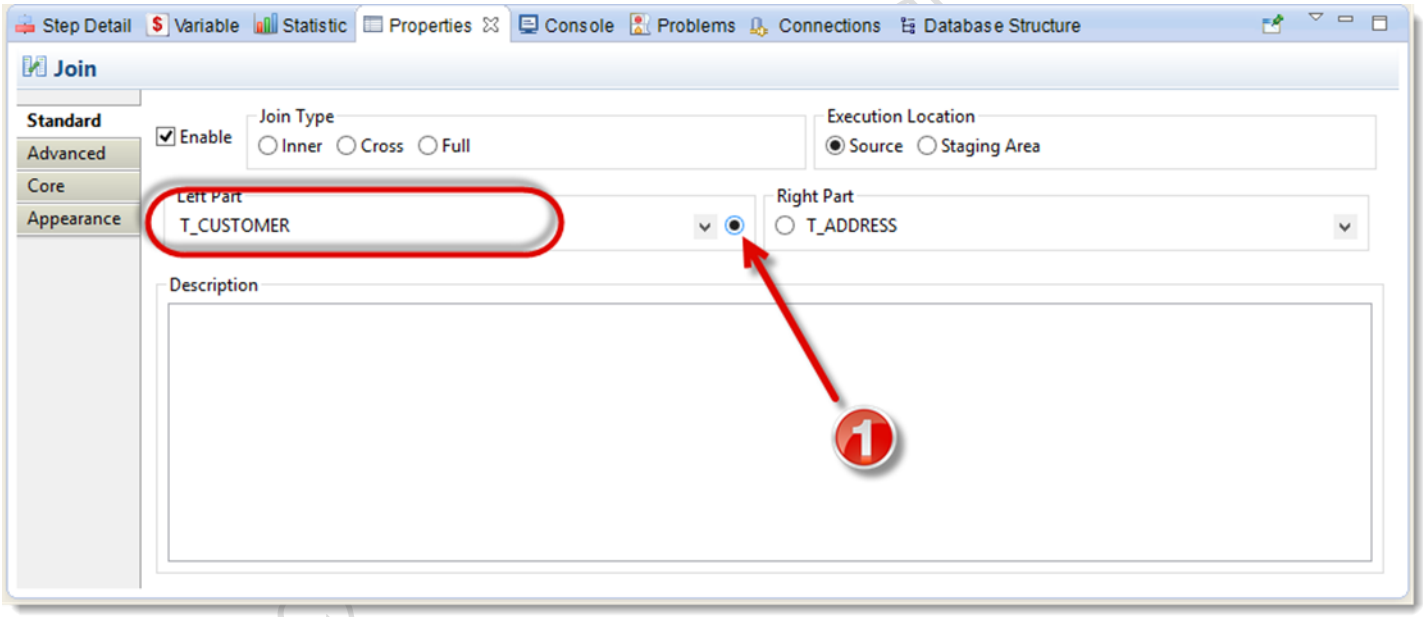
第 1 データストア	第 2 データストア	ビジネスルール	実行場所
T_CUSTOMER	T_TITLE	T_CUSTOMER.TIT_CODE=T_TITLE.TIT_CODE	ソース
T_CUSTOMER	T_ADDRESS	T_CUSTOMER.CUS_ID=T_ADDRESS.CUS_ID	ソース
T_ADDRESS	DIM_GEOGRAPHY	DIM_GEOGRAPHY.GEO_ZIP_CODE=T_ADDRESS.ADR_ZIP_CODE	ステージ領域

Join を追加したことにより、ロードステップが 3 から 1 に減ったことに注目してください。**Stambia Designer** は、その **E-LT architecture**（抽出、ロード、変換アーキテクチャ）により、ビジネスルールをソースにすばやく実行することができ、マッピングを最適化する機能を備えています。そのため、ソースとターゲット間でやりとりされるフローが常に最小限に抑えられます。

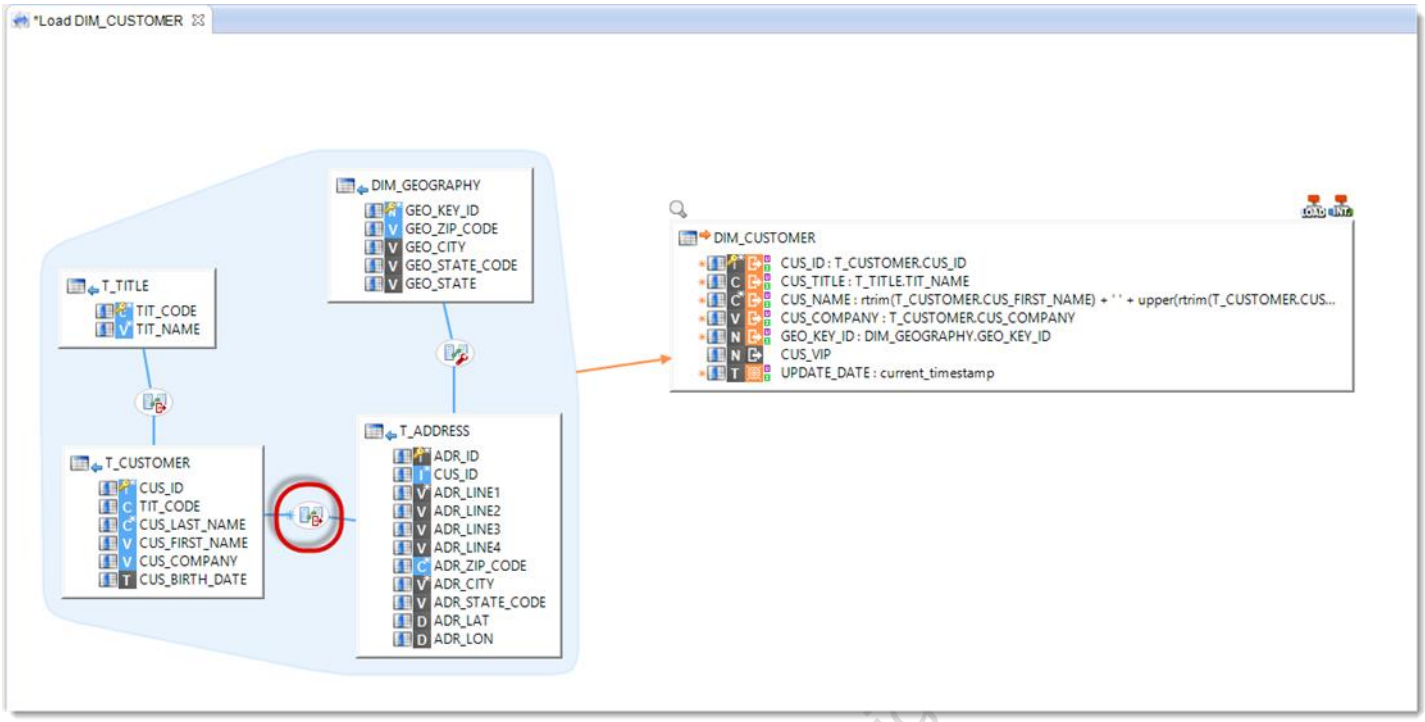
13.2 外部結合を定義する

DIM_CUSTOMER テーブルをロードするマッピングでは、T_CUSTOMER にある全レコードを取得しなければならない制約があります。しかし、join（結合）が一部レコードを除外する場合があります。例えば、T_CUSTOMER 上のあるレコードが T_ADDRESS に対応する住所データを持たないとき、そのまま処理すれば、inner join（内部結合）によって、その顧客レコードは対象外となります。（内部結合は、キーとなる値が両テーブルに存在するときのみ、リクエストにデータが送り返されるので、strict join とも呼ばれます。）そこで、outer join（外部結合）を定義し、T_CUSTOMER の全レコードが（たとえ T_ADDRESS に対応する住所を持たなくても）処理対象となるように設定しなければなりません。

1. T_CUSTOMER と T_ADDRESS の間の join をクリックします。
2. **Properties** ビューを開きます。
3. **T_CUSTOMER** の横のラジオボタンを選択し、同テーブルを outer join（外部結合）の **master table**（main table）として定義します。



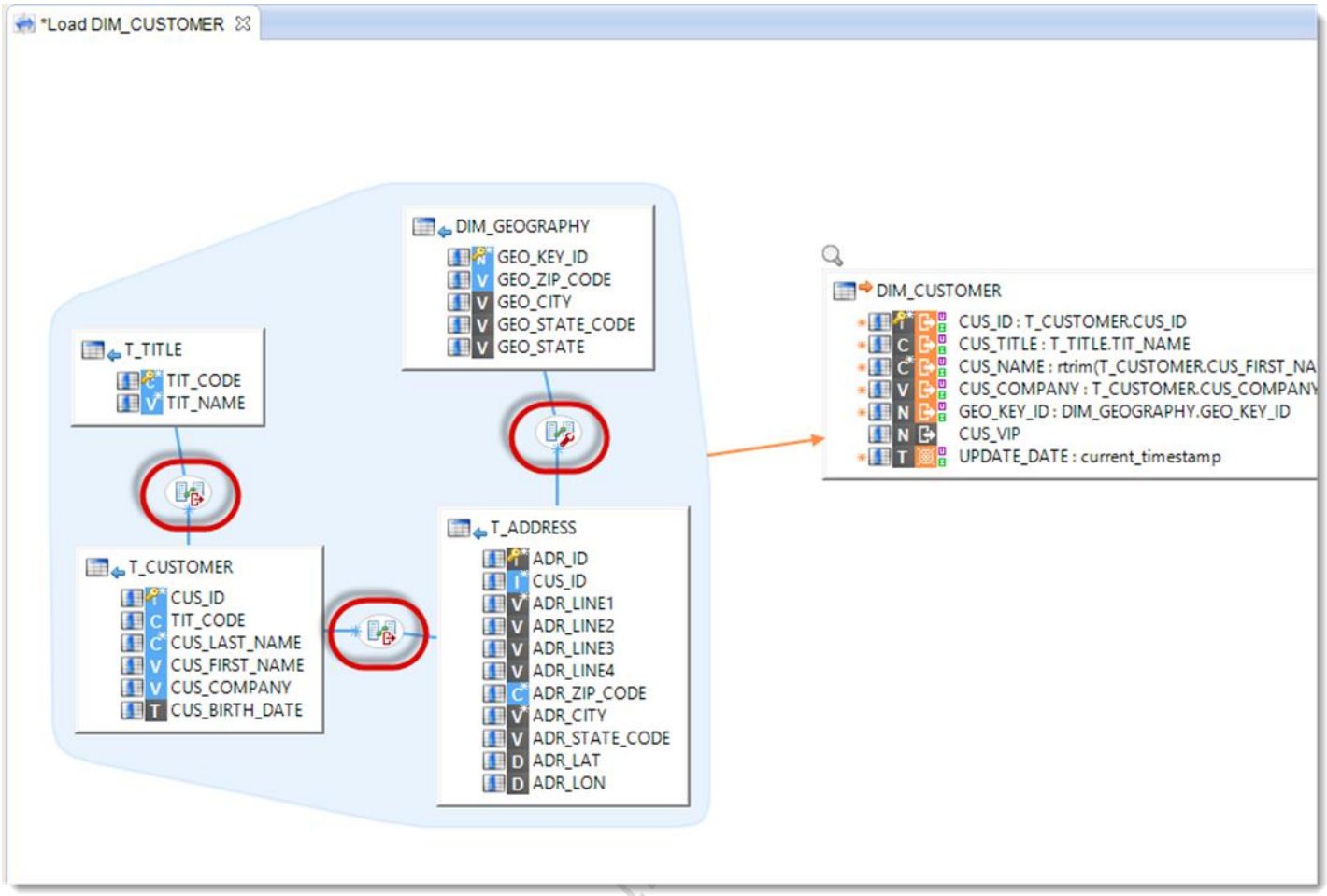
以上により、結合アイコンと T_CUSTOMER の間に星印が表示されるはずです。それにより、外部結合の設定が視覚的に確認できます。



上記と同じ手順で、下記の外部結合を定義してください。

第 1 データストア	第 2 データストア	マスターテーブル
T_CUSTOMER	T_ADDRESS	T_CUSTOMER
T_CUSTOMER	T_TITLE	T_CUSTOMER
T_ADDRESS	DIM_GEOGRAPHY	T_ADDRESS

各々の外部結合に対して、T_CUSTOMER 側に星印が表示され、外部結合が定義されたことが視覚的に確認できます。



以上により、マッピングの定義がすべて完成し、実行する準備が整いました。

13.3 マッピング実行結果を検証

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	300
SUM(SQL_STAT_INSERT)	100
SUM(SQL_STAT_UPDATE)	0

上記に加え、下記のリクエストを実行すると、11 という値が得られるはずです。データストアを右クリックし、**Action**、次いで **Consult data** を選択して、リクエストを下記のように修正してください。

```
select count(*) from HOTEL_DATAMART.DIM_CUSTOMER where GEO_KEY_ID is null
```

このリクエストを実行すると、11 が結果欄に表示されます。実際、ソースデータには、住所記録が存在しない、あるいは正しくない顧客レコードが 11 件あります。当マッピングに定義された外部結合は、これらの顧客レコードを、住所を NULL にセットしながら挿入します。

14 リジェクト検知

14.1 リジェクト検知とは

データ統合において、データのクオリティはすべての統合プロセスに共通する最大の課題の 1 つに挙げられます。まず、ターゲットデータベースの技術的要件（外部キーや固有キーなど）に合わせ、データの整合性が確立されていなければなりません。同時に、エンドユーザーのニーズも満たさなければなりません。

Stambia は、この 2 つのニーズに応えることができます。**Stambia** 独自のツールである **reject detection**（リジェクト検知）メカニズムを利用しながら、その一方で、ユーザー定義の constraint（制約）を追加してメタデータをより完全にすることができます。
それらの制約に対し、技術的なものか機能的なものかにかかわらず、**Stambia** は以下の処理を行うことができます。

- Reject（リジェクト）されるべきデータを検知する。
- Reject（リジェクト）されたデータを主要データフローから選り分ける

14.2 DIM_CUSTOMER にユーザー定義の制約を作成する

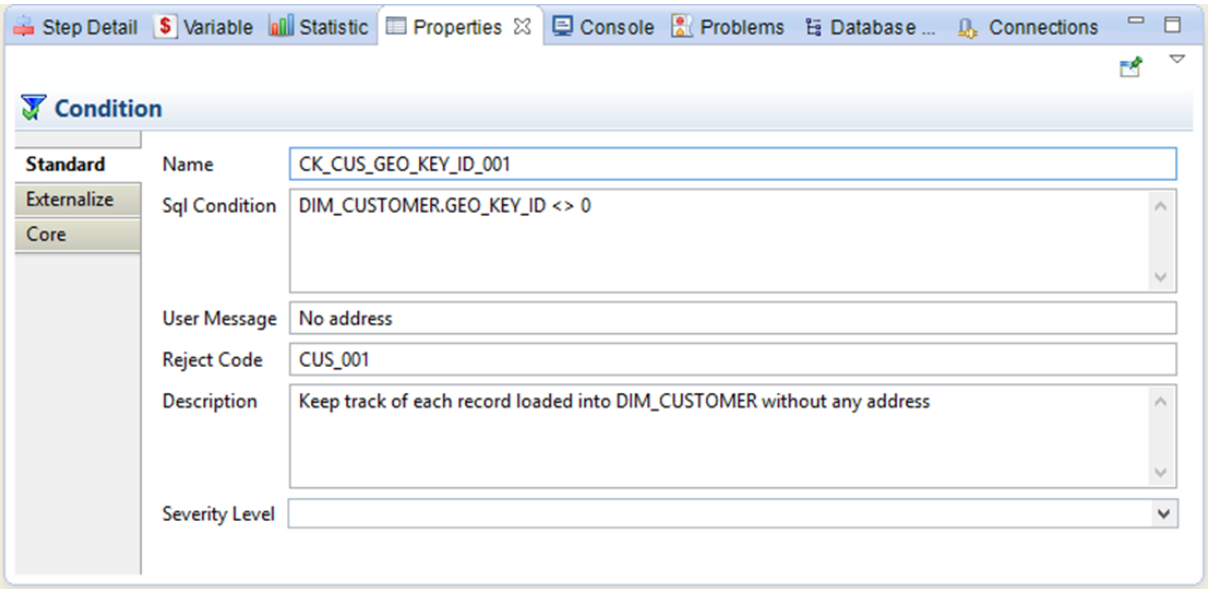
ターゲットデータベースの DIM_GEOGRAPHY テーブルには、下記の特定の値が記録されています。

GEO_KEY_ID	GEO_ZIP_CODE	GEO_CITY	GEO_STATE_CODE	GEO_STATE
0	<null>	No Address（住所なし）	<null>	<null>
1	?	Unknown Zip Code（郵便番号不明）	?	?

各統合プロセスにおいて、上記の値を追跡調査する必要があると仮定します。その場合、DIM_CUSTOMER の GEO_KEY_ID の値が 0 または 1 ではないことを確認するために、2 種類の constraint（制約）を DIM_CUSTOMER に作成します。制約は、メタデータファイルに対し、既存の制約をオーバーロードすることにより、追加できます。

1. メタデータファイル Datamart を開きます。
2. **DIM_CUSTOMER** ノードを開きます。
3. **DIM_CUSTOMER** を右クリックします。
4. **New**、次いで、**Condition** を選択します。
5. **Properties** ビューを開きます。
6. **Name** フィールドに新しい制約の名前 CK_CUS_GEO_KEY_ID_001 を記入します。
7. **Sql Condition** フィールドに、値を検証するための下記 SQL コードを記入します。
   DIM_CUSTOMER.GEO_KEY_ID <> 0
8. **User Message** フィールドにメッセージ”No address”と記入します。

9. リジェクト理由が技術的に識別できるように、**Reject Code** フィールドに識別コード（CUS_001）を設定します。
10. **Description** フィールドには、この制約を説明するための記述を自由に記入します。例： Keep track of each record loaded into DIM_CUSTOMER without any address（DIM_CUSTOMER にロードされるレコードで住所のないものを証跡）



同様に、**DIM_CUSTOMER** に 2 つめの制約を、下記のプロパティにしたがって作成します。

プロパティ	値
Name	CK_CUS_GEO_KEY_ID_002
Sql Condition	DIM_CUSTOMER.GEO_KEY_ID <> 1
User Message	Unknown Zip Code（郵便番号不明）
Reject Code	CUS_002
Description	Keep track of each record loaded into DIM_CUSTOMER with an address containing an unknown Zip Code（DIM_CUSTOMER にロードされるレコードで郵便番号が判らないものを選別）

時間節約のために、最初に作成した条件をコピーし、DIM_CUSTOMER にペーストしてから値を変えることもできます。

制約を 2 種類に分ける理由は、ユーザーメッセージをより解りやすくするためです。次の **Sql Condition** を用いて、制約を 1 つにまとめることも可能です。DIM_CUSTOMER.GEO_KEY_ID not in (0,1)

14.3 Load DIM_CUSTOMER のデフォルト値を使用する

DIM_CUSTOMER に設定された constraint（制約）の条件を実際に利用するには、まず DIM_CUSTOMER をロードするビジネスルールを修正する必要があります。現時点では、outer join（外部結合）により、ビジネスルールは

GEO_KEY_ID を DIM_GEOGRAPHY に見つかった値か、もし見つからなかった場合は NULL でロードするようになっています。

1. マッピング Load DIM_CUSTOMER を開きます。
2. ターゲットカラム **GEO_KEY_ID** をクリックします。
3. **Expression Editor** ビューにおいて、既存の式を下記の式に置き換えます。

```
case
        when T_ADDRESS.ADR_ID is null then 0
        when T_ADDRESS.ADR_ID is not null and DIM_GEOGRAPHY.GEO_KEY_ID
is null then 1
        else DIM_GEOGRAPHY.GEO_KEY_ID
end
```

GEO_KEY_ID カラムに対してビジネスルールを変更する際、**Stambia Designer** の開発補助機能に注目してください。当ビジネスルールは、デフォルト設定でソースに実行されますが、ソースが 2 つ存在することになるので、赤色のアイコンによりルールが適正ではないことが示されます。マウスのポインタをアイコンに 2、3 秒合わせたままにすると、ツールチップが表示されます。

上記の変更後、ビジネスルールが **Staging Area**（ステージ領域）において実行されます。

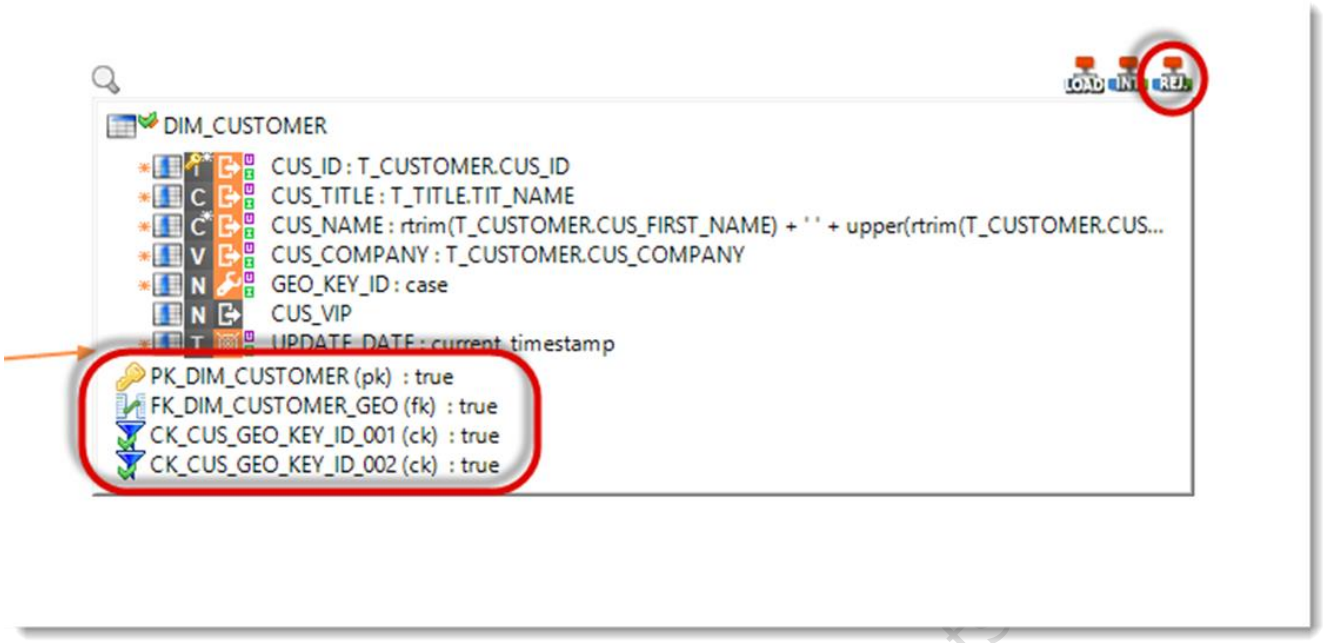
14.4 Load DIM_CUSTOMER にリジェクト検知を設定する

マッピング Load DIM_CUSTOMER の reject detection（リジェクト検知）を有効にする準備がすべて整いました。リジェクト検知（つまり、不適格データの選別）は下記の手順で施行します。

1. データストア DIM_CUSTOMER を選択します。
2. 同データストアの左側に表示されたアイコン✔をクリックして、リジェクト管理を有効にします。

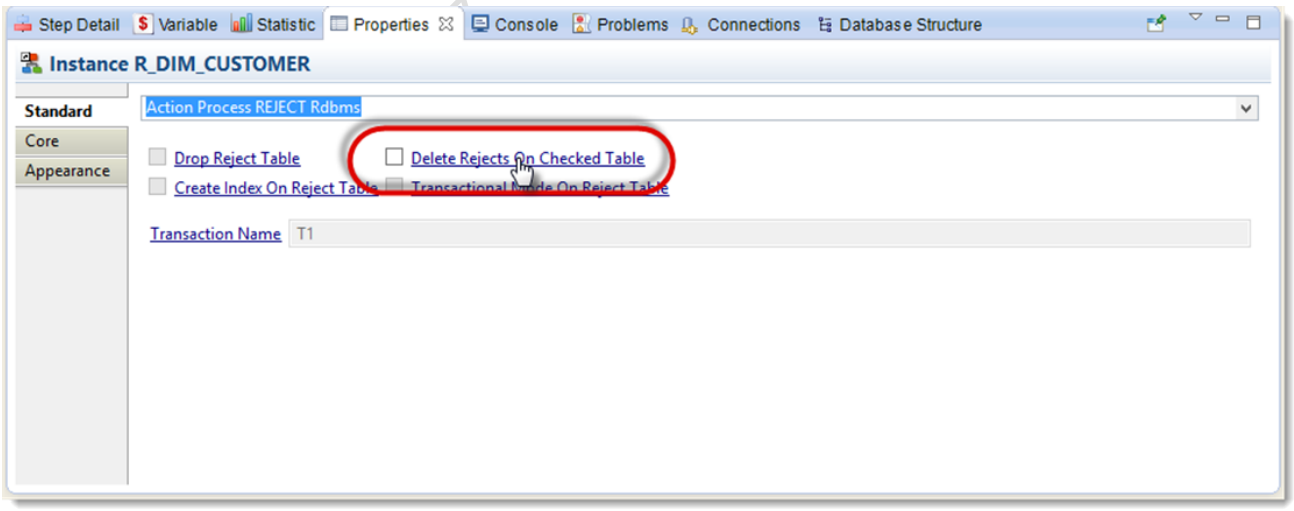
リジェクト管理は、ターゲットデータストアのクリック後、あるいは右クリックして **Enable Rejects** を選択後、**Properties** ビューからも有効にできます。

以上により、マッピングに新しいステップが追加されます。リジェクトを検知するための Check ステップと言います。さらに、DIM_CUSTOMER データストアは制御可能な constraint（制約）を以下のように表示します。



このマッピングでは、リジェクトされたレコードをデータフローから削除せずに選別することを目指しています。そのための手順は下記の通りです。

1. **Check: DIM_CUSTOMER** ステップをクリックします。（先に表示された«REJ»アイコン）
2. **Properties** ビューを開きます。
3. **Delete Rejects On Checked Table** （選択されたテーブルのリジェクトを検知する）と書かれたリンクをクリックし、テーブル記入欄を有効化します。
4. チェックボックスの選択は必ず外しておきます。



以上により、マッピングに対し、リジェクト検知が正しく設定され、リジェクトされたレコードはデータフローからは削除されずに、証跡されます。

14.5 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

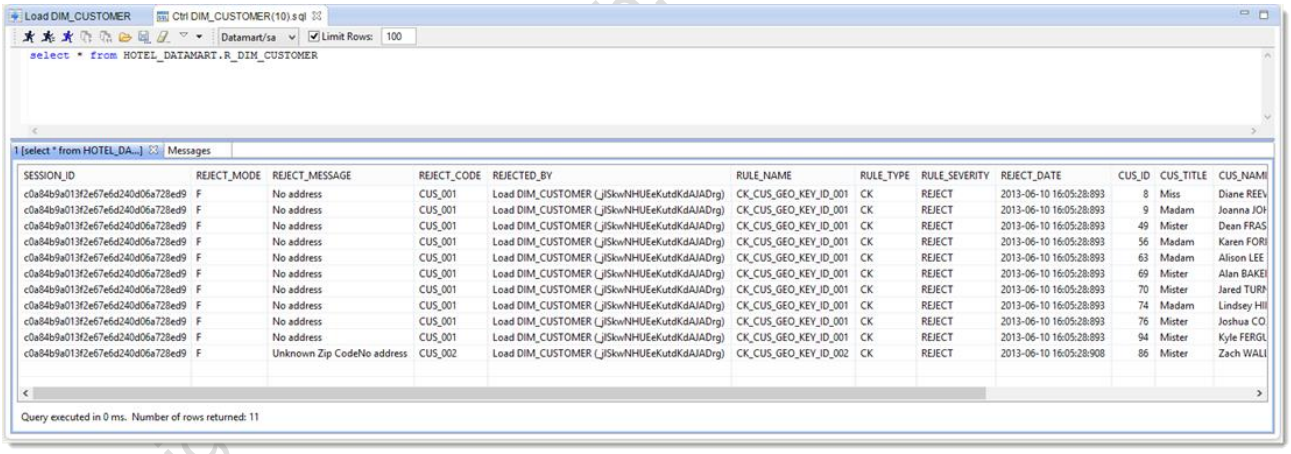
演算	結果
SUM(SQL_NB_ROWS)	322
SUM(SQL_STAT_ERROR)	11
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	11

新しい統計結果が追加表示されていることに注意してください。SUM(SQL_STAT_ERROR)として、検知されたリジェクト数 11 が表示されています。

マッピング実行中にリジェクトされたレコード数は、リジェクトテーブルから確認することも可能です。

1. ターゲットデータベース DIM_CUSTOMER を右クリックします。
2. **Action**、次いで **Consult Reject Table** を選択します。
3. リクエストを実行します。

以上により、11 件のエラーが一覧表示されます。



上記に加え、下記のリクエストを実行すると、10 という値が得られるはずです。ターゲットデータストアを右クリックし、**Action**、次いで **Consult data** を選択して、リクエストを下記のように修正してください。

```
select count(*) from HOTEL_DATAMART.DIM_CUSTOMER where GEO_KEY_ID = 0
```

このリクエストを実行すると、10 が結果欄に表示されます。

同様に、下記のリクエストは 1 を返します。

```
select count(*) from HOTEL_DATAMART.DIM_CUSTOMER where GEO_KEY_ID = 1
```

15 重複削除を伴ともなうマッピングを作成する

15.1 DIM_TIME テーブルをロードするマッピングの作成

下記のプロパティにもとづいて、新しいマッピングを作成します。

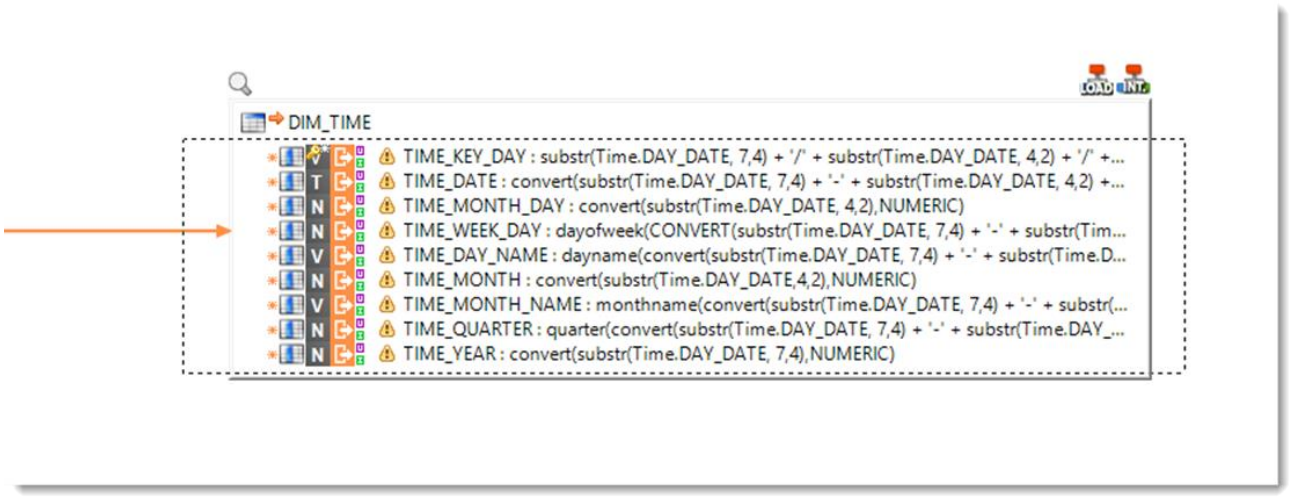
プロパティ	値
親フォルダ	Mappings
マッピング名	Load DIM_TIME
ターゲットデータストア	DIM_TIME
ソースデータストア	Time

変換ビジネスルールは下記の通りです。

ターゲットカラム	ビジネスルール	特性		
TIME_KEY_DAY	substr(Time.DAY_DATE, 7,4) + '/' + substr(Time.DAY_DATE, 4,2) + '/' + substr(Time.DAY_DATE, 1,2)	ステージ領域	I/U	ファンクショナルキー
TIME_DATE	convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP)	ステージ領域	I/U	
TIME_MONTH_DAY	convert(substr(Time.DAY_DATE, 1,2),NUMERIC)	ステージ領域	I/U	
TIME_WEEK_DAY	dayofweek(CONVERT(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP))	ステージ領域	I/U	
TIME_DAY_NAME	dayname(convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00', TIMESTAMP))	ステージ領域	I/U	
TIME_MONTH	convert(substr(Time.DAY_DATE,4,2),NUMERIC)	ステージ領域	I/U	
TIME_MONTH_NAME	monthname(convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP))	ステージ領域	I/U	
TIME_QUARTER	quarter(convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE,	ステージ領域	I/U	

	4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP))			
TIME_YEAR	convert(substr(Time.DAY_DATE, 7,4),NUMERIC)	ステージ領域	I/U	

実行場所をすべてのビジネスルールに対して一括して定義するには、選択したいカラムの周囲にマウスで四角形を描くことにより複数選択することができます。あるいは、**CTRL** を押しながら個々のターゲットカラムをクリックするか、最初のカラムを選択後に **CAPS** キーを押したまま最後のカラムをクリックする方法もあります。対象となる全カラムが選択されたら、実行場所（上記の例ではステージ領域）を定義します。



15.2 重複削除を設定する

Time ファイルのデータを確認すると、6 時間ごとに 1 件のレコードが作成されていることに気付くはずです。1 日 4 行、生成されることになります。

01/01/2010 00:00
01/01/2010 06:00
01/01/2010 12:00
01/01/2010 18:00
02/01/2010 00:00
02/01/2010 06:00
02/01/2010 12:00
02/01/2010 18:00
03/01/2010 00:00
03/01/2010 06:00

テーブルは 1 日 1 行しか受け付けません。そこで、データフローから重複レコードを削除（**deduplication**）しなければなりません。

1. **Integration: DIM_TIME** テンプレートをクリックします。
2. **Properties** ビューを開きます。
3. **Use distinct** リンクをクリックしてチェックボックスを有効にします。
4. チェックボックスが選択されていることを確認します。

以上により、重複レコードを削除するテンプレートのコンフィギュレーションが完了しました。
マッピングを実行してください。

15.3 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	6 576
SUM(SQL_STAT_INSERT)	1 096
SUM(SQL_STAT_UPDATE)	0

16 変換ビジネスルールの相互利用を可能にする

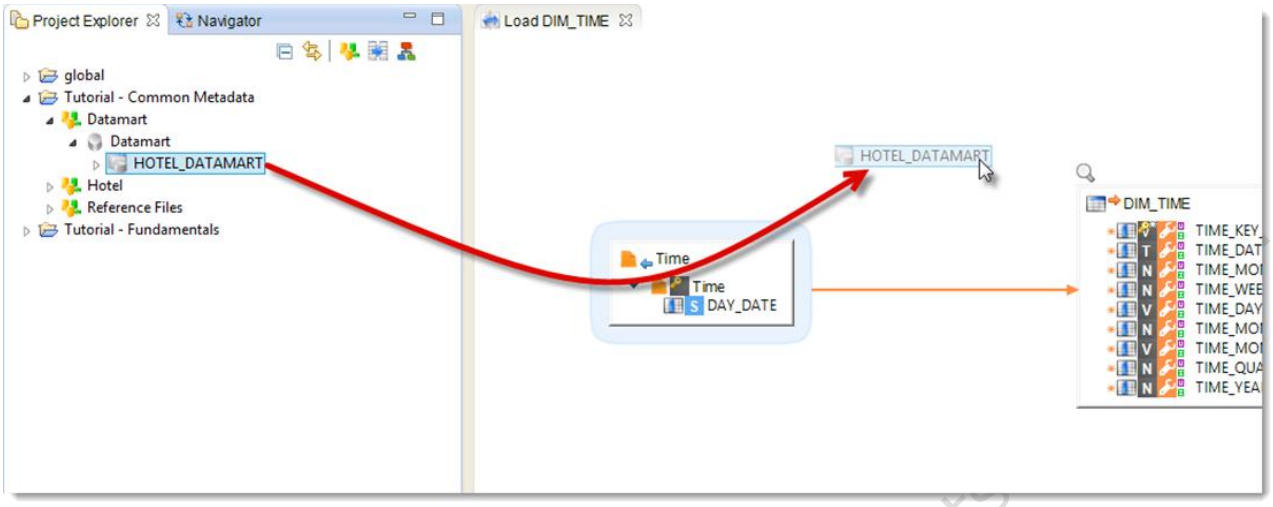
16.1 DIM_TIME テーブルをロードするマッピングの修正

これまで、変換ビジネスルールに同じ式が何度も再利用されていることに気付いたでしょうか。例えば、下記を参照してください。

ターゲットカラム	ビジネスルール	特性		
TIME_KEY_DAY	substr(Time.DAY_DATE, 7,4) + '/' + substr(Time.DAY_DATE, 4,2) + '/' + substr(Time.DAY_DATE, 1,2)	ステージ領域	I/U	ファンクショナルキー
TIME_DATE	convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP)	ステージ領域	I/U	
TIME_MONTH_DAY	convert(substr(Time.DAY_DATE, 1,2),NUMERIC)	ステージ領域	I/U	
TIME_WEEK_DAY	dayofweek(convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP))	ステージ領域	I/U	
TIME_DAY_NAME	dayname(convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00', TIMESTAMP))	ステージ領域	I/U	
TIME_MONTH	convert(substr(Time.DAY_DATE,4,2),NUMERIC)	ステージ領域	I/U	
TIME_MONTH_NAME	monthname(convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP))	ステージ領域	I/U	
TIME_QUARTER	quarter(convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP))	ステージ領域	I/U	
TIME_YEAR	convert(substr(Time.DAY_DATE, 7,4),NUMERIC)	ステージ領域	I/U	

Stambia Designer では、**Stage** を活用して、上記のような式を mutualize（相互利用）することができ、デー

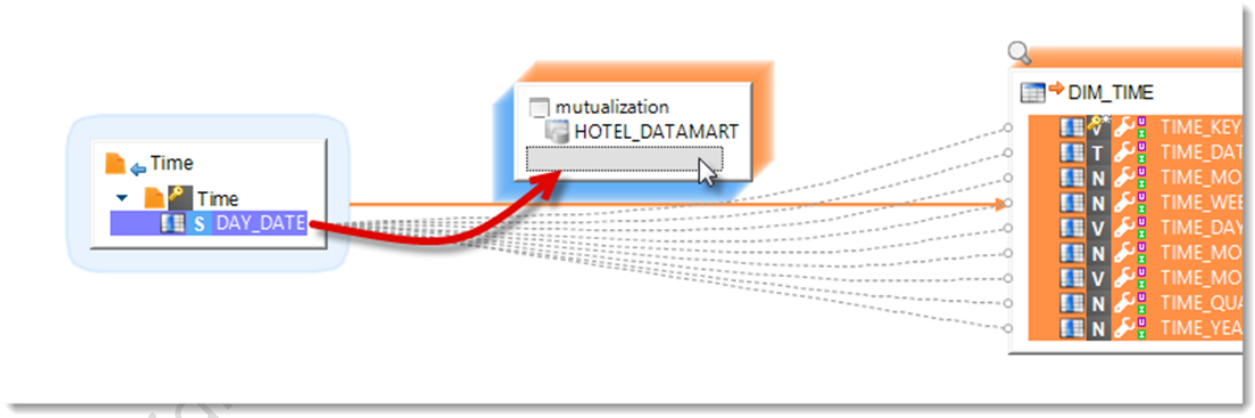
タストアのロード時に再利用を可能にできます。空の **Stage** を作成するには、HOTEL_DATAMART スキーマをマッピングにマウスでドラッグ&ドロップします。



空の **Stage** を作成したら、下記の手順で名前を変更します。

1. **Stage** を選択します。
2. **Properties** ビューにおいて、**Alias** フィールドに mutualization と入力します。

次に、ソースファイル Time から DAY_DATE カラムをドラッグ&ドロップして、最初のフィールドを追加します。



フィールドは、アイコンをクリックしても追加できます。その後、**Properties** ビューで **Alias** を変更して名前を付け、最後に、ターゲットカラムにするのと同じ方法で変換式を入力します。

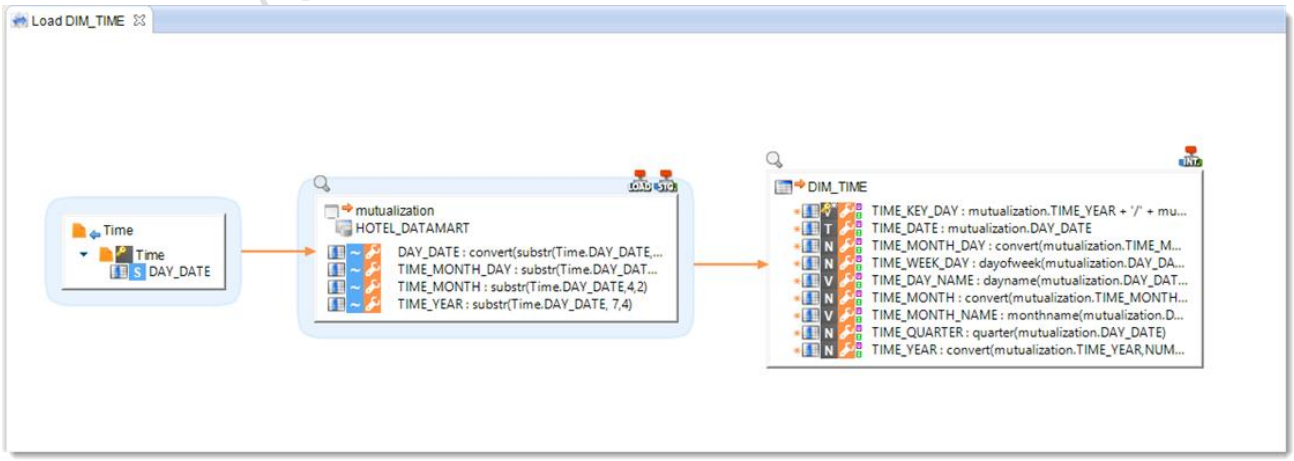
Stage を以下のカラムで完成させてください。それにより、以下のビジネスルールが再利用可能になります。

ターゲットカラム	ビジネスルール
DAY_DATE	convert(substr(Time.DAY_DATE, 7,4) + '-' + substr(Time.DAY_DATE, 4,2) + '-' + substr(Time.DAY_DATE, 1,2) + ' 00:00:00',TIMESTAMP)

TIME_MONTH_DAY	substr(Time.DAY_DATE, 1,2)
TIME_MONTH	substr(Time.DAY_DATE,4,2)
TIME_YEAR	substr(Time.DAY_DATE, 7,4)

さらに、**Stage** に設定されたカラムを使用可能するために、DIM_TIME テーブルをロードするビジネスルールを下記の通りに修正します。

ターゲットカラム	**ビジネスルール**	**特性**		
TIME_KEY_DAY	mutualization.TIME_YEAR + '/' + mutualization.TIME_MONTH + '/' + mutualization.TIME_MONTH_DAY	ステージ領域	I/U	ファンクショナルキー
TIME_DATE	mutualization.DAY_DATE	ステージ領域	I/U	
TIME_MONTH_DAY	convert(mutualization.TIME_MONTH_DAY,NUMERIC)	ステージ領域	I/U	
TIME_WEEK_DAY	dayofweek(mutualization.DAY_DATE)	ステージ領域	I/U	
TIME_DAY_NAME	dayname(mutualization.DAY_DATE)	ステージ領域	I/U	
TIME_MONTH	convert(mutualization.TIME_MONTH,NUMERIC)	ステージ領域	I/U	
TIME_MONTH_NAME	monthname(mutualization.DAY_DATE)	ステージ領域	I/U	
TIME_QUARTER	quarter(mutualization.DAY_DATE)	ステージ領域	I/U	
TIME_YEAR	convert(mutualization.TIME_YEAR,NUMERIC)	ステージ領域	I/U	



以上により、マッピングを実行する準備が整いました。

16.2 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	6 576
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	0

上記の結果を見れば、変換ビジネスルールの mutualize（相互利用）化がテーブルの内容を変えてないことが解ります（DIM_TIME に挿入や更新は行われていません）。つまり、mutualize は、ビジネスルールに影響を与えずに再利用を可能にします。

17 複数ソースの和集合からデータソースをロードする

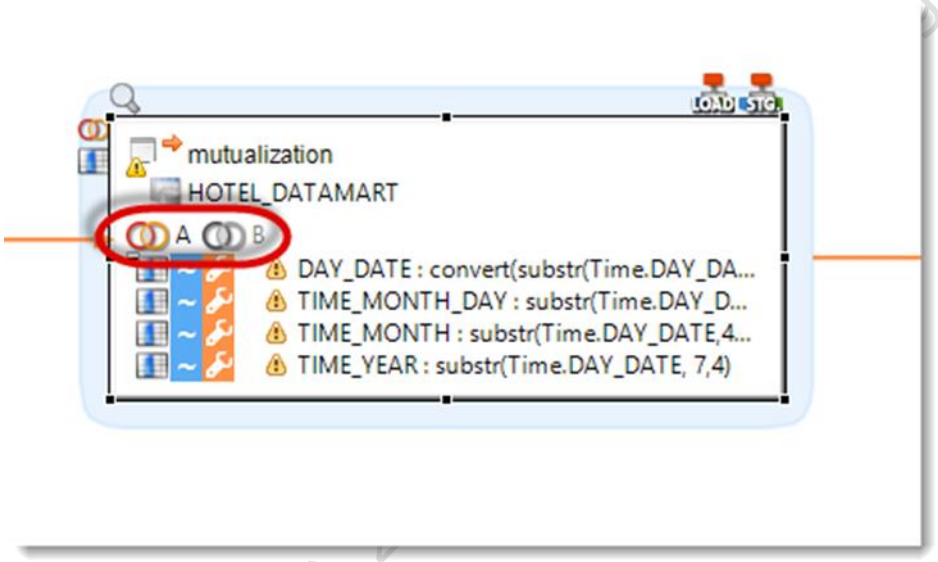
17.1 DIM_TIME テーブルをロードするマッピングの修正

DIM_TIME データストアは、Time.csv ファイルと T_PLANNING テーブルの 2 つのソースからロードされなければなりません。**Stambia Designer** では、同一テーブルに 2 つのロードマッピングを作成する代わりに、2 つのソースからデータの union（和集合）を作成することができます。

和集合のデータセットは **Stage** で作成されます。

1. mutualization **Stage** を選択します。
2. Stage の左横の アイコンをクリックします。

以上により、**Stage** に A と B の両方のデータセットが表示されます。



表示を見やすくするために、それぞれのデータセット名を変更します。

1. データセット A を選択します。
2. **Properties** ビューにおいて、**Alias** フィールドに File と入力します。

同様に、データセット B の名前を Rdbms に変更します。
ここで、2 つのデータセット間の union（和集合）を指定します。

1. mutualization **Stage** を選択します。
2. **Expression Editor** において、次のビジネスルールを入力します。
   [File] union [Rdbms]

上記で設定された新しいデータセットが T_PLANNING テーブルからロードされます。T_PLANNING テーブルには、Time.csv ファイルのデータ以外の日付が含まれます。まず、T_PLANNING データストアをマッピングに追加します。次に、2 番目のデータセットに対するビジネスルールを指定するために、Rdbms セットを選択して下記のビジネスルールを入力します。

ターゲットカラム	ビジネスルール
DAY_DATE	T_PLANNING.PLN_DAY
TIME_MONTH_DAY	lpad(day(T_PLANNING.PLN_DAY), 2, '0')
TIME_MONTH	lpad(month(T_PLANNING.PLN_DAY), 2, '0')
TIME_YEAR	convert(year(T_PLANNING.PLN_DAY), VARCHAR)

以上の設定により、マッピングを実行する準備が整いました。

17.2 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	9 132
SUM(SQL_STAT_INSERT)	730
SUM(SQL_STAT_UPDATE)	0

18 練習課題：単純なマッピングを作成する

18.1 FACT_BOOKING をロードするマッピングの作成

下記のマッピングを作成し、実行してください。

プロパティ	値
親フォルダ	Mappings
マッピング名	Load FACT_BILLING
ターゲットデータストア	FACT_BOOKING
ソースデータストア	DIM_TIME, T_BDR_PLN_CUS

変換ビジネスルールは下記の通りです。

ターゲットカラム	ビジネスルール	特性		
BOK_KEY_ID				
CUS_ID	T_BDR_PLN_CUS.CUS_ID	ソース	I/U	ファンクショナルキー
TIME_KEY_DAY	DIM_TIME.TIME_KEY_DAY	ソース	I/U	ファンクショナルキー
BDR_ID	T_BDR_PLN_CUS.BDR_ID	ソース	I/U	ファンクショナルキー
BOK_PEOPLE	T_BDR_PLN_CUS.PLN_CUS_PERS_COUNT	ソース	I/U	
BOK_BOOKED	case when T_BDR_PLN_CUS.PLN_CUS_BOOKED = 'true' then 1 else 0 end	ステージ領域	I/U	
UPDATE_DATE	current_timestamp	ターゲット	I/U	

BOK_KEY_ID カラムは«auto-incremented»フィールドで、すべてのレコード挿入に ID 番号を付けるための連番フィールドです。そのため、データベース自体がロードするので、同フィールドのマッピングは不要です。

Join（結合）のビジネスルールは下記の通りです。

第 1 データストア	第 2 データストア	ビジネスルール	実行場所
DIM_TIME	T_BDR_PLN_CUS	T_BDR_PLN_CUS.PLN_DAY=DIM_TIME.TIME_DATE	ステージ領域

18.2 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	43 650
SUM(SQL_STAT_INSERT)	14 550
SUM(SQL_STAT_UPDATE)	0

19 集計計算をともなうマッピングを作成する

19.1 FACT_BILLING をロードするマッピングの作成

下記のマッピングを作成してください。

プロパティ	値
親フォルダ	Mappings
マッピング名	Load FACT_BILLING
ターゲットデータストア	FACT_BOOKING
ソースデータストア	T_BILLING, T_BILLING_LINES, DIM_TIME, DIM_DISCOUNT

変換ビジネスルールは下記の通りです。

ターゲットカラム	ビジネスルール	特性		
BOK_KEY_ID				
BIL_ID	T_BILLING.BIL_ID	ソース	I/U	ファンクショナルキー
CUS_ID	T_BILLING.CUS_ID	ソース	I/U	
TIME_KEY_DAY	DIM_TIME.TIME_KEY_DAY	ソース	I/U	
DIS_RANGE	DIM_DISCOUNT.DIS_RANGE	ソース	I/U	ファンクショナルキー
PMT_CODE	T_BILLING.PMT_CODE	ソース	I/U	
BIL_AMOUNT	sum(T_BILLING_LINES.BLL_AMOUNT)	ステージ領域	I/U	
BIL_QTY	sum(T_BILLING_LINES.BLL_QTY)	ステージ領域	I/U	
UPDATE_DATE	current_timestamp	ターゲット	I/U	

Join（結合）のビジネスルールは下記の通りです。

第 1 データストア	第 2 データストア	ビジネスルール	実行場所
T_BILLING	DIM_TIME	T_BILLING.BIL_DATE=DIM_TIME.TIME_DATE	ステージ領域
T_BILLING	T_BILLING_LINES	T_BILLING_LINES.BIL_ID=T_BILLING.BIL_ID	ソース
T_BILLING_LINES	DIM_DISCOUNT	round(case when T_BILLING_LINES.BLL_DISCOUNT_AMOUNT != 0 then	ステージ領域

		(T_BILLING_LINES.BLL_DISCOUNT_AMOUNT / (T_BILLING_LINES.BLL_AMOUNT + T_BILLING_LINES.BLL_DISCOUNT_AMOUNT))*100 else T_BILLING_LINES.BLL_DISCOUNT_RATE end) between DIM_DISCOUNT.DIS_MIN and DIM_DISCOUNT.DIS_MAX	

マッピングを実行する前に、どのカラムに対して aggregation（集約）のビジネスルールが適用されるのかを、**Stambia Designer** に指示しなければなりません。

1. ターゲットカラム **BIL_AMOUNT** を選択する。
2. データストアの左横の アイコンをクリックして、カラムの aggregation（集約）を有効にします。

BIL_QTY カラムにも同じ処理を行います。
以上により、マッピングを実行する準備が整いました。

19.2 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	53 314
SUM(SQL_STAT_INSERT)	12 107
SUM(SQL_STAT_UPDATE)	0

20 新たな制約を追加する

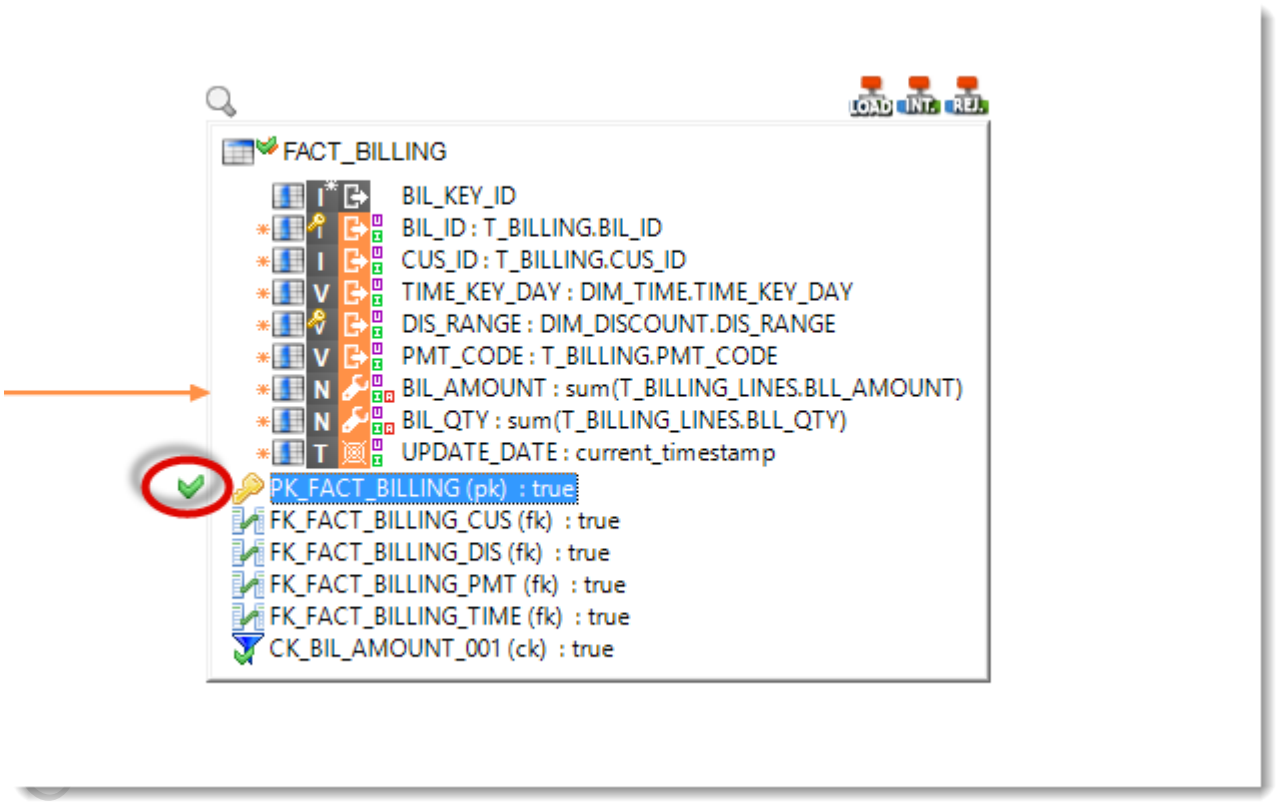
20.1 練習課題

BIL_AMOUNT カラムの値が 10 より大きいことを確認する constraint（制約）を **FACT_BILLING** に追加してください。
FACT_BILLING.BIL_AMOUNT > 10

さらに、Load FACT_BILLING マッピングの reject detection（リジェクト検知）を有効にして、追加された制約を検証してください。フローからデータを削除せずに、単に Reject テーブル上で証跡することが、この課題の目的です。

手順に不明な点があれば、Load DIM_CUSTOMER にリジェクト検知を設定するを参照してください。

BIL_KEY_ID カラムはマッピングによりロードされるのではなく、データベースにより自動的にロードされます。よって、リジェクト検知のステップにおいて、**FACT_BILLING** の主キーを確認することはできません。これを防ぐには、**PK_FACT_BILLING (pk) : true** を選択し、データストアの左横の✔アイコンをクリックしてください。



20.2 マッピング実行結果を検証する

マッピングを reject detection（リジェクト検知）を適用するように修正した後、実行したら、Statistic ビューを開いて実行結果の統計を確認してください。

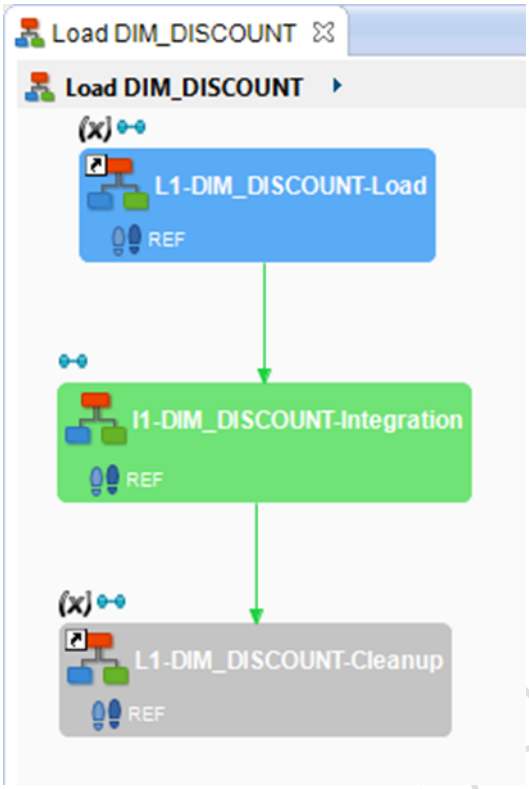
演算	結果
SUM(SQL_NB_ROWS)	53 316
SUM(SQL_STAT_ERROR)	2
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	0

上記に加え、Reject テーブルにおいても証跡結果（**BIL_ID** の 224 と 5994）を確認できます

21 DATAMART のローディングを自動化する

21.1 プロセスを構成する

Runtime（ランタイム）に実行される **Actions** のセットを **Process** と呼びます。マッピングが実行されるたびに、**Stambia** は個々のマッピングからプロセスを生成します。そのプロセスが下記のように、マッピング実行ごとに **Stambia Designer** で確認できます。



プロセスは手動で作成することも可能です。さらに、上位のコンポーネント（マッピングや他のプロセス）の統括や、下位レベルのアクション（SQL コマンド、ファイル処理、各種ツール）を管理することができます。

21.2 アクションを統括するプロセスの作成

21.2.1 すべての Datamart をロードするプロセスを作成する

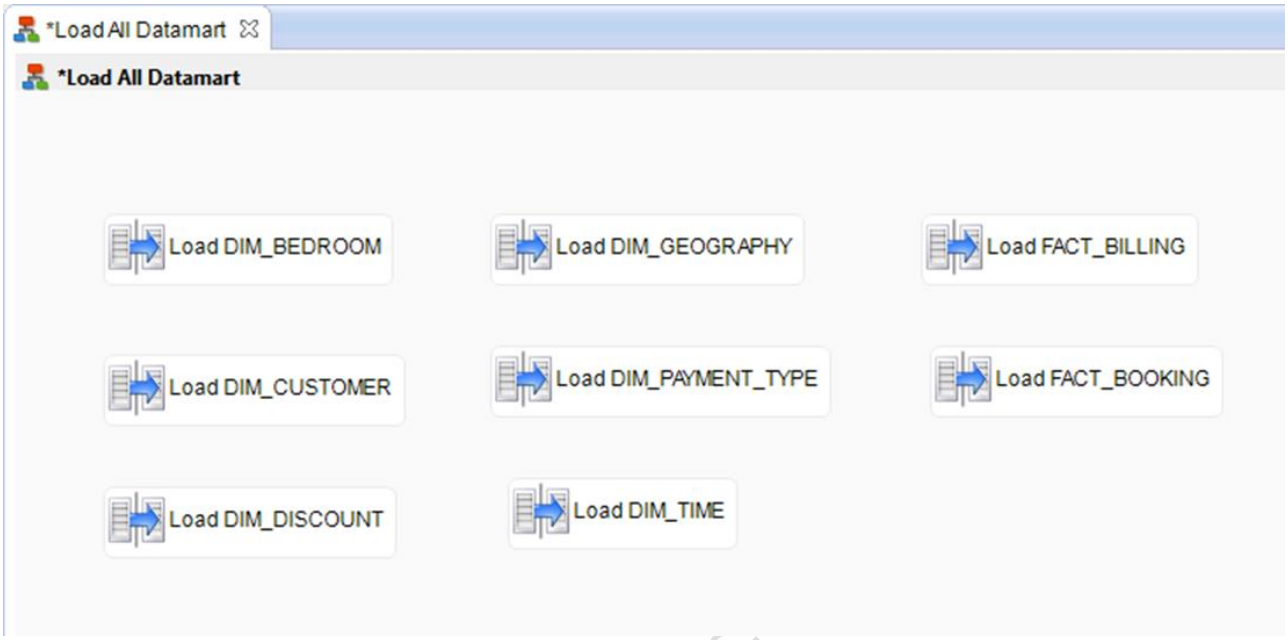
Datamart の全ローディングを実行するプロセスを、下記の手順で作成します。

1. Tutorial – Fundamentals プロジェクトを開きます。
2. Processes フォルダを右クリックします。
3. **New**、次いで **Process** を選択します。
4. **File name** フィールドに Load All Datamart と入力します。

21.2.2 マッピングをプロセスに追加する

これまで作成した個々のマッピングを追加します。

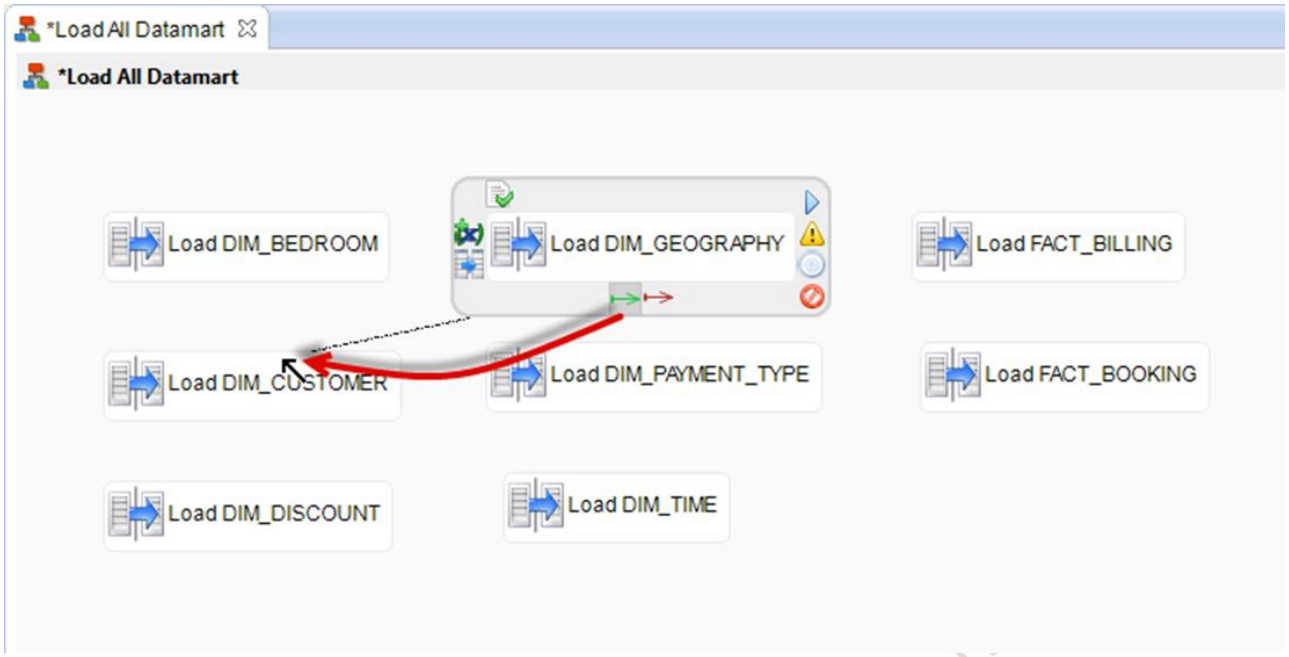
1. **Project Explorer** ビューにおいて、Mapping を選択します。
2. そのマッピングを Process エディターにマウスでドラッグ&ドロップします。



21.2.3 リンクを定義する

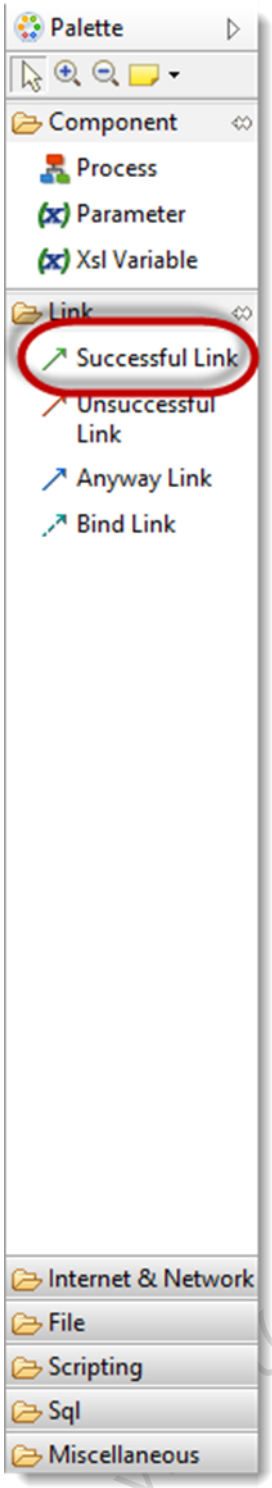
デフォルト設定では、Process の各ステップは、プロセスの最初から並行して一斉に実行されます。しかし、ターゲットモデルの外部キーに対するアクセスの関係上、特定マッピングの実行順序を設定する必要があります。ステップの実行順を設定するには、マッピング間にリンクを作成します。

1. 元となるステップを選択します。
2. 緑色の矢印を選択し、後のステップまでマウスでドラッグします。



また、リンクは **Palette** を使用しても定義できます。

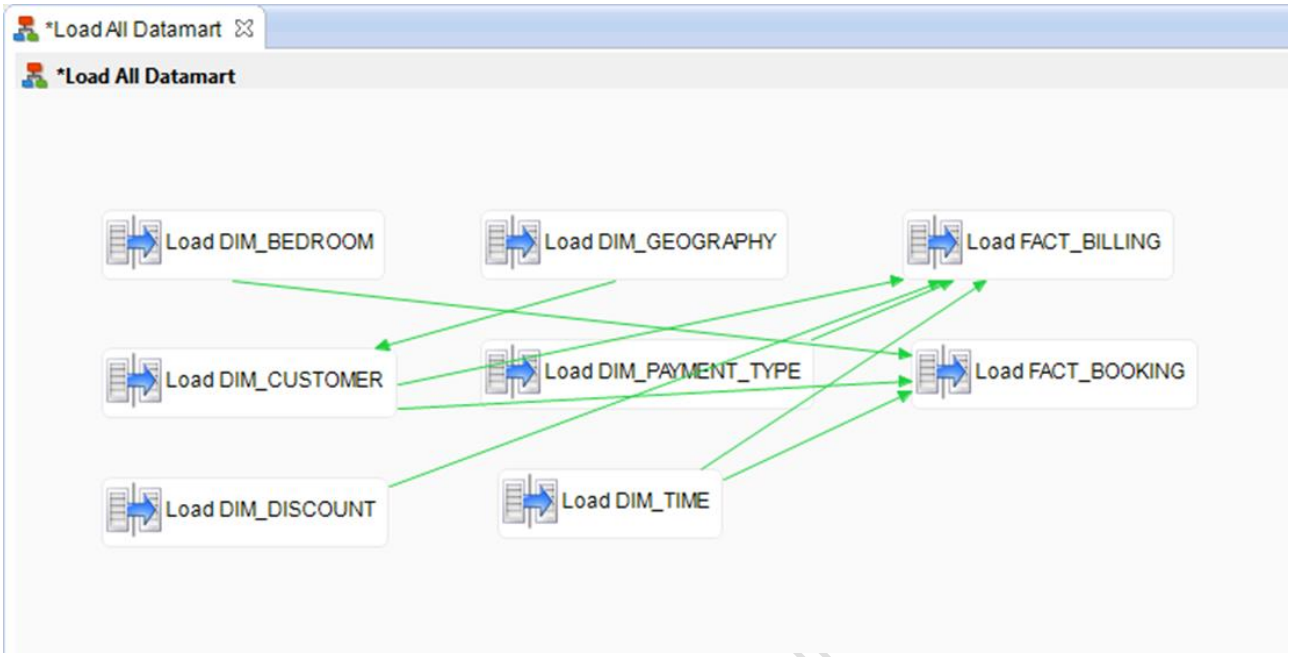
1. コンポーネントの **Palette** から **Link** を開きます。
2. **Successful Link** を選択します。
3. リンクすべきマッピング間に矢印を引きます。最初に実行されるべきマッピングから、その後に実行されるべきマッピングに向かって、マウスでドラッグ&ドロップしてください。



当プロセスが正しく実行されるために必要なステップ間のリンクは、下記の通りです。

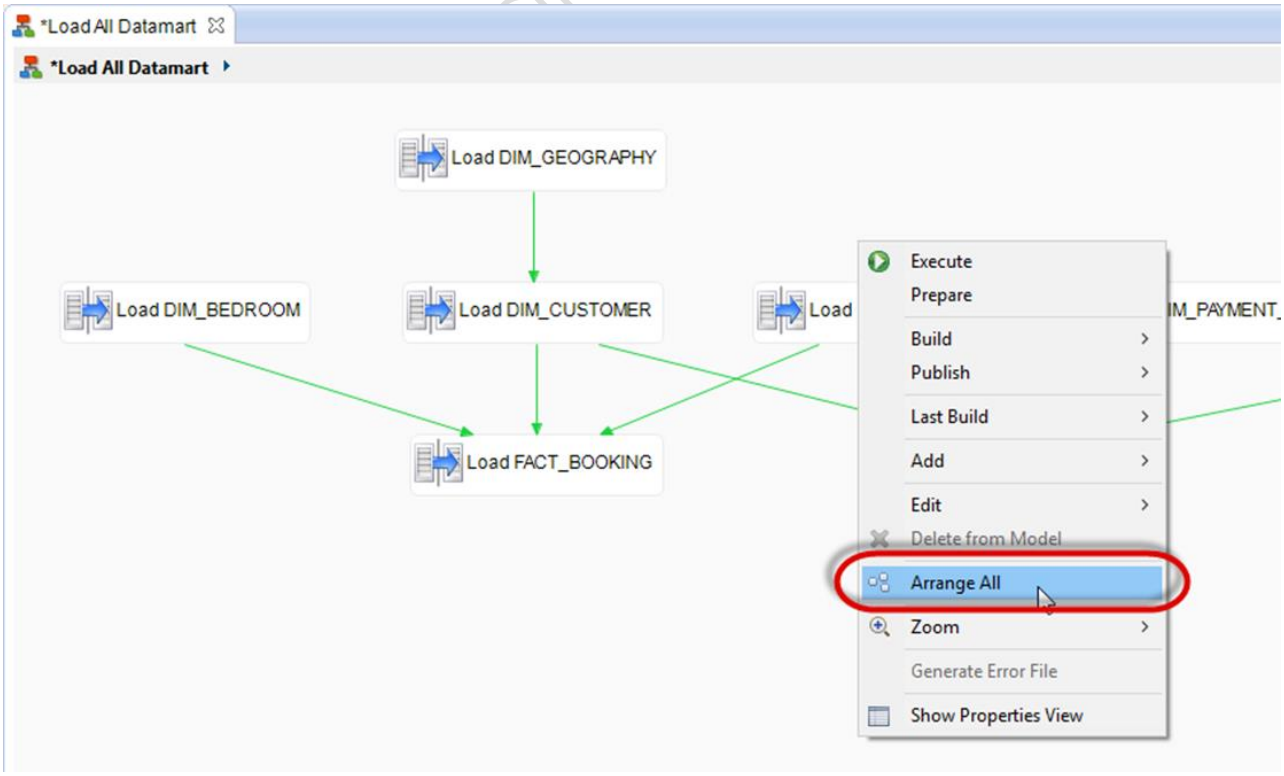
元のステップ	後のステップ
DIM_GEOGRAPHY	DIM_CUSTOMER
DIM_CUSTOMER	FACT_BOOKING
DIM_TIME	FACT_BOOKING
DIM_BEDROOM	FACT_BOOKING
DIM_CUSTOMER	FACT_BILLING
DIM_TIME	FACT_BILLING

DIM_DISCOUNT	FACT_BILLING
DIM_PAYMENT_TYPE	FACT_BILLING



最後に、画面上のプロセス構成図を自動調整機能で見やすくすることができます。

1. プロセス図の背景を右クリックします。
2. **Arrange All** を選択します。



21.2.4 プロセスを完了する

以上の設定が完了したら、Process を実行してください。

1. プロセス図の背景を右クリックします。
2. **Execute** を選択します。

21.2.5 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

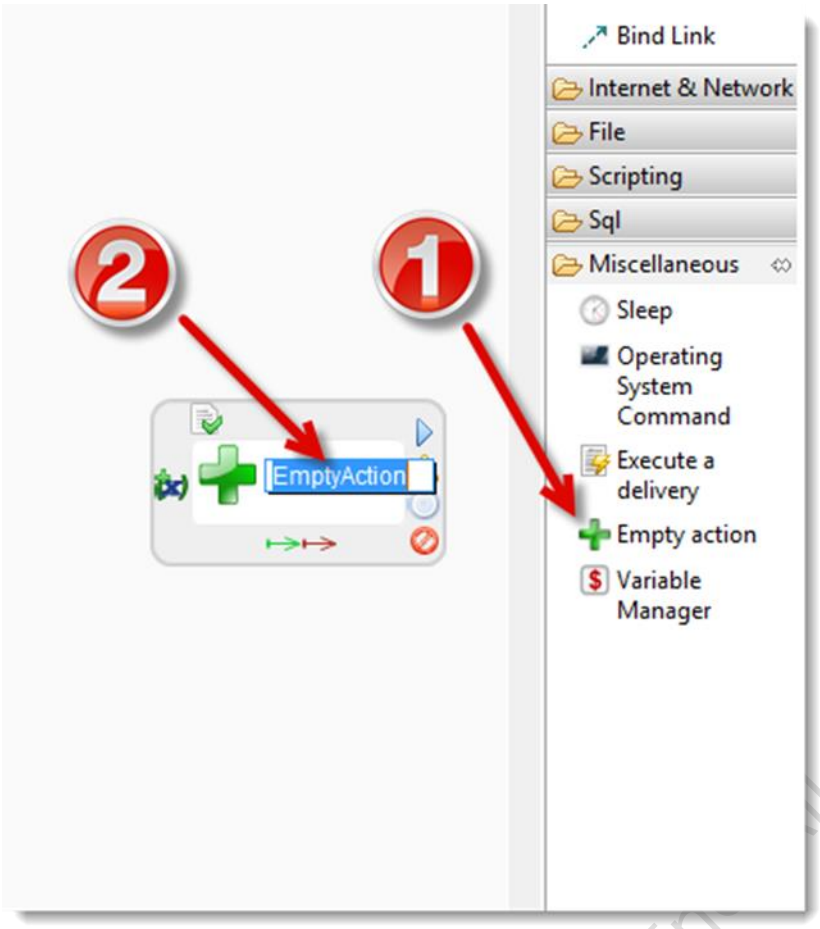
演算	結果
SUM(SQL_NB_ROWS)	232 140
SUM(SQL_STAT_ERROR)	13
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	0

21.3 空のステップを追加する

プロセス構成図をさらに見やすくするために、あるいは同時進行タスクの編成を機能化するためには、空のステップを追加して同期化ポイントを定義するのが有効な場合があります。

ここでは、ローディングの最初のステップの前と最後のステップの後に新たなステップを追加します。そのためには、まず、空のステップをプロセスの最初に追加します。

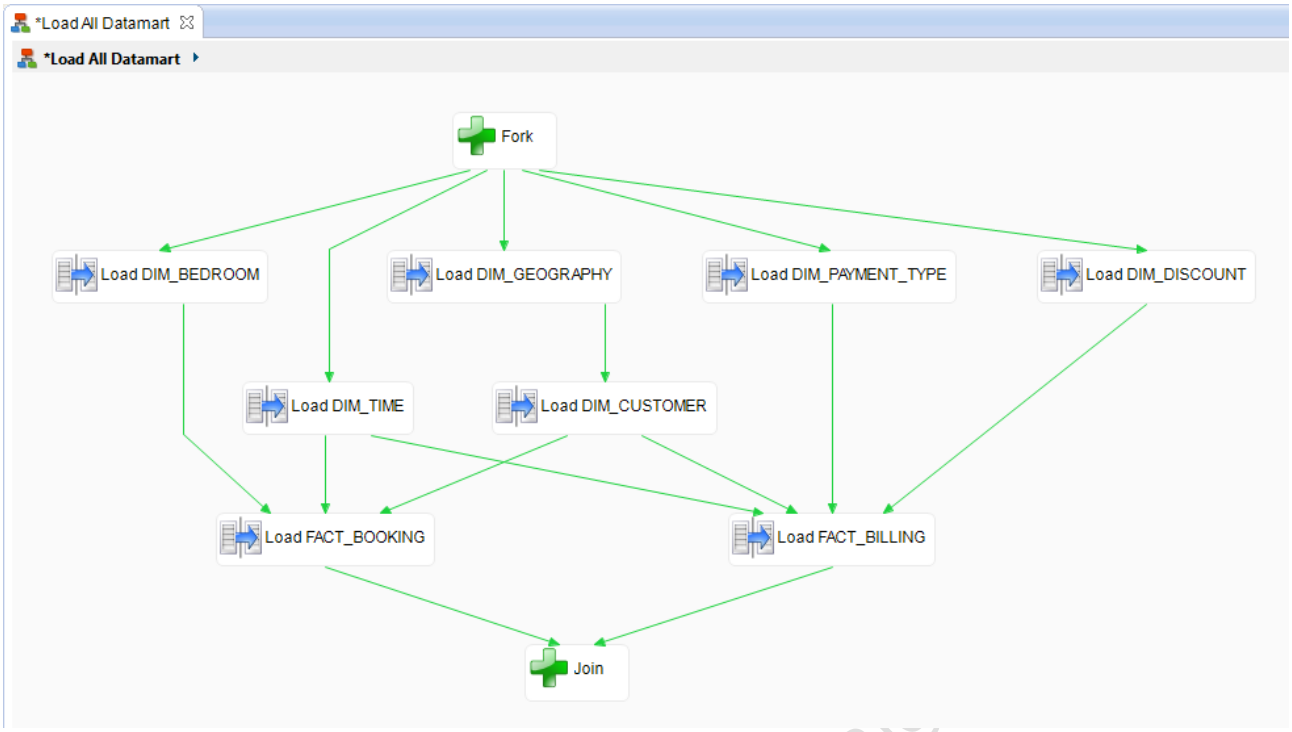
1. コンポーネントの **Palette** から **Miscellaneous** を開きます。
2. **Empty action** を選択します。
3. 表示された図の内側をクリックして、空のステップを作成します。
4. アクション名を Fork と記入します。



空のステップが作成されたら、ローディングの最初のステップとリンクします。

元のステップ	後のステップ
Fork	DIM_GEOGRAPHY
Fork	DIM_TIME
Fork	DIM_BEDROOM
Fork	DIM_DISCOUNT
Fork	DIM_PAYMENT_TYPE

同様に、ローディングの最終地点に Join という空のステップを追加します。



21.4 制約の無効化/有効化

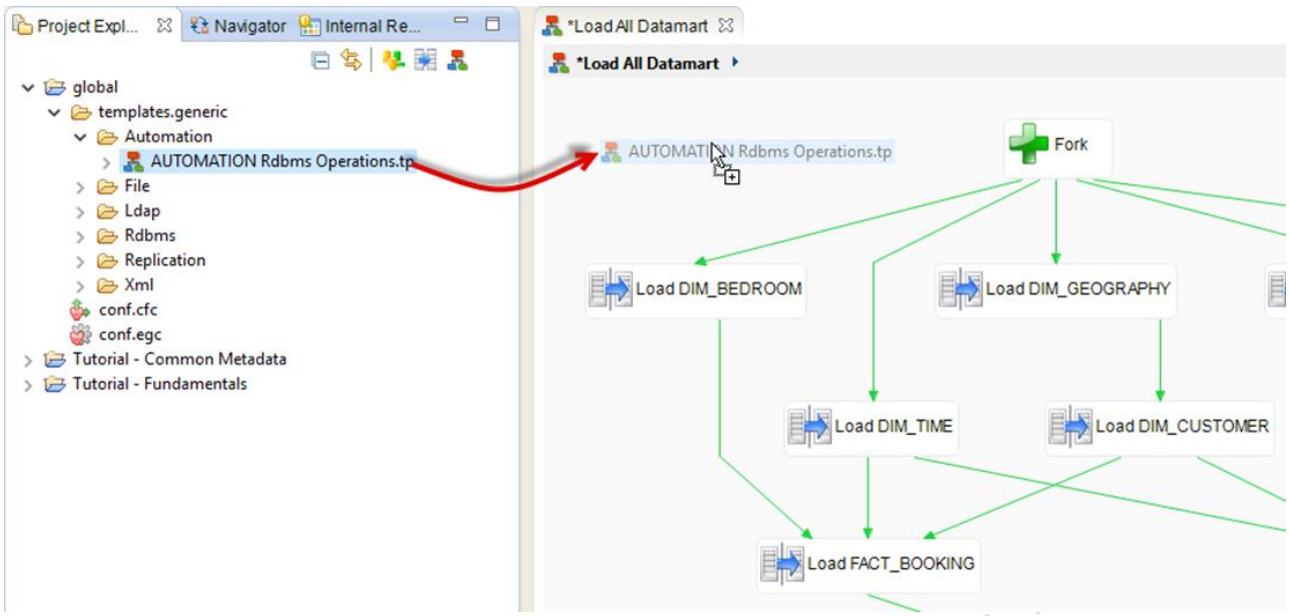
21.4.1 スキーマの制約をすべて無効にするステップの追加

Stambia のテンプレートはメタデータのアクションを自動化して、Processes の自動生成を可能にし、開発を効率化します。例えば、テンプレートにより、データベーススキーマのテーブルリストを巡回して演算を端から順次実行していくこともできます。

この機能を利用して、ローディング処理の最初に constraint（制約）を無効化し、最後に再生成することができます。つまり、データベースはローディング処理中は制約を確認する必要がなくなり、処理速度が向上します。

制約を無効化するステップは下記の手順で作成します。

1. **Project Explorer** ビューにおいて、**global** プロジェクトを開きます。
2. 次に、**template.generic** のノード、さらに **Automation** のノードを開きます。
3. プロセス図に **AUTOMATION Rdbms Operations.tp** テンプレートをマウスでドラッグ&ドロップします。

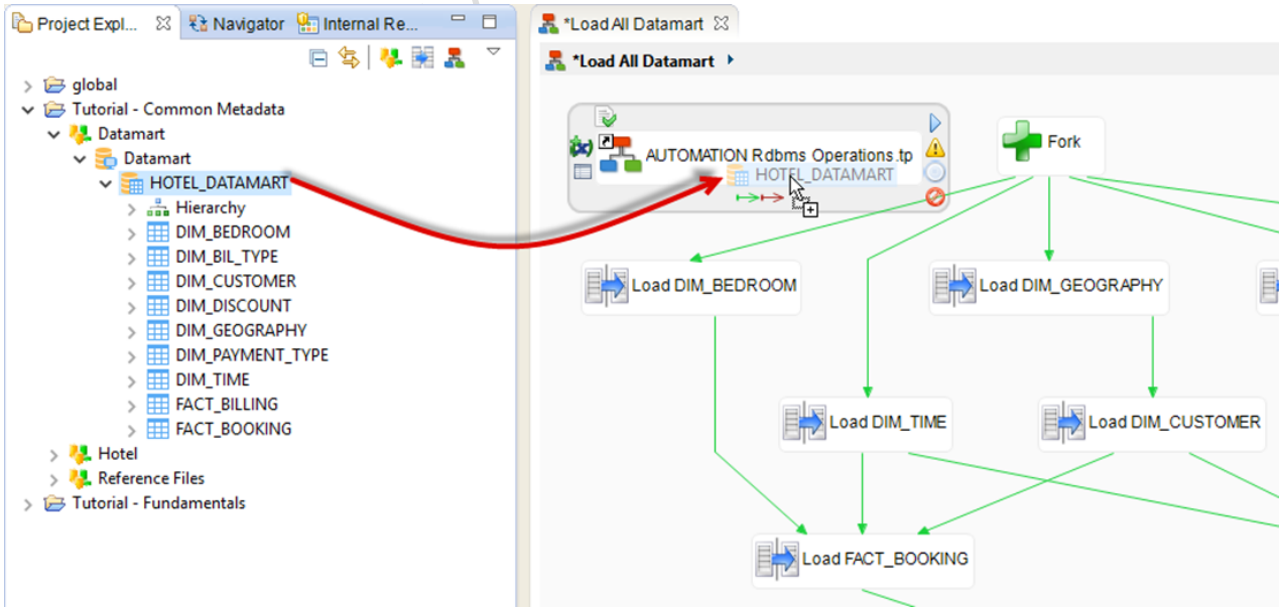


作成されたステップを下記の手順でコンフィギュレーションします。

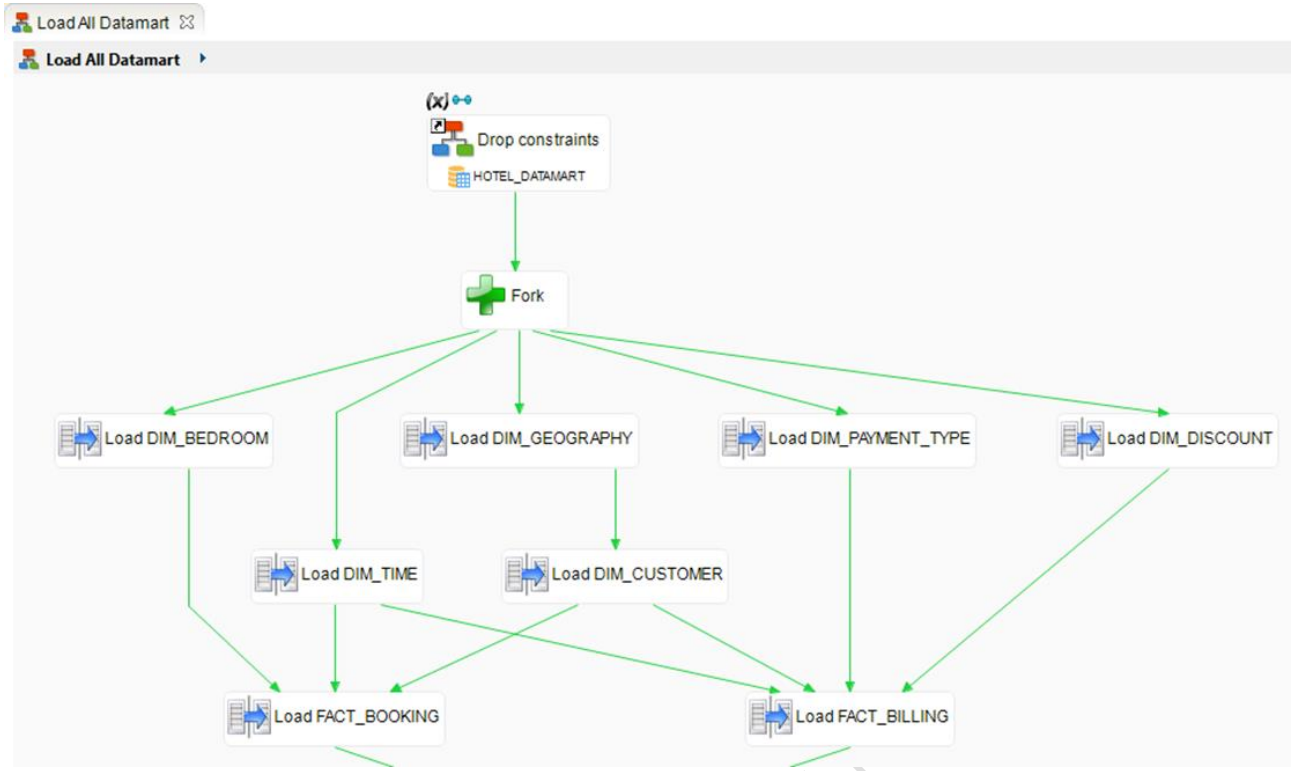
1. 作成されたステップをクリックし、**Properties** ビューを開きます。
2. **Name** フィールドに Drop constraints と記入します。
3. **Drop Fk** リンクをクリックし、チェックボックスが選択されていることを確認します。

次に、対象となるテーブルのスキーマを、テンプレートに認知させる必要があります。

1. メタデータファイル Datamart のノードを開き、次に Datamart サーバーのノードを開きます。
2. スキーマ **HOTEL_DATAMART** を Drop constraints ステップのタイトルにマウスでドラッグ&ドロップします。



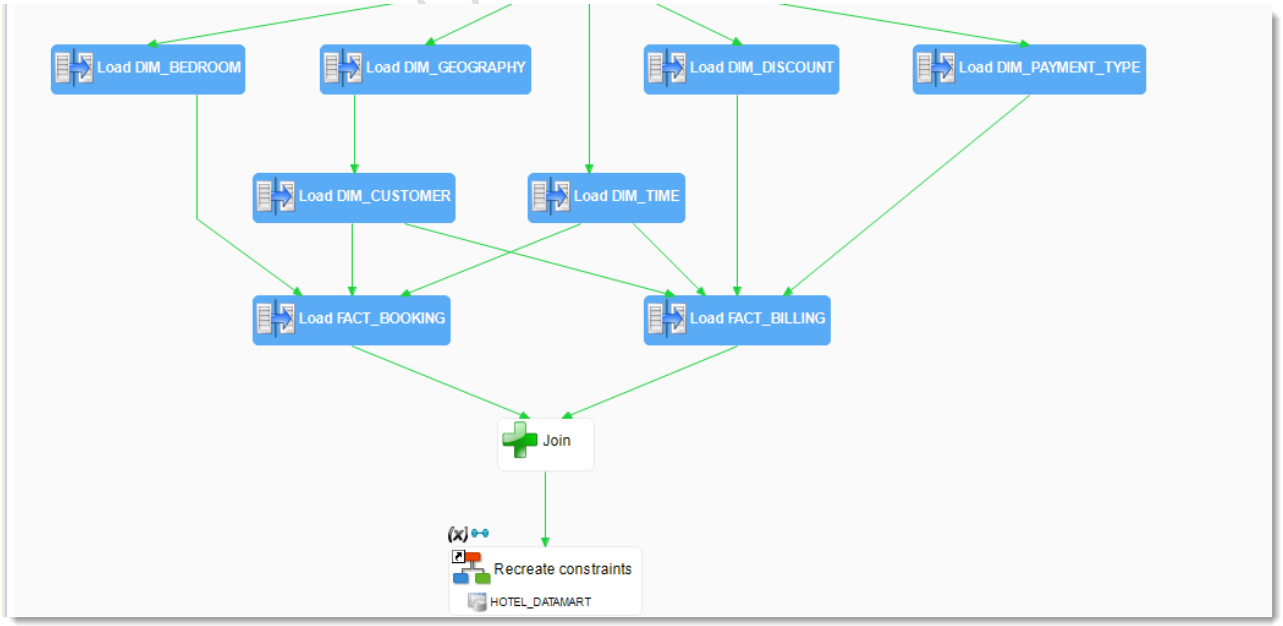
最後に、このステップを、現在の第 1 ステップ Fork にリンクさせます。



21.4.2 スキーマの制約をすべて再有効化するステップの追加

スキーマのすべての制約を再生成するステップを、制約の無効化ステップと同じ方法で作成します。ここでは、**Drop Fk** オプションを選択せずに、**Create Fk** オプションを選択します。

ステップ名は Recreate constraints とします。



以上の設定で、プロセスを実行する準備が完了です。

21.4.3 プロセス実行結果を検証する

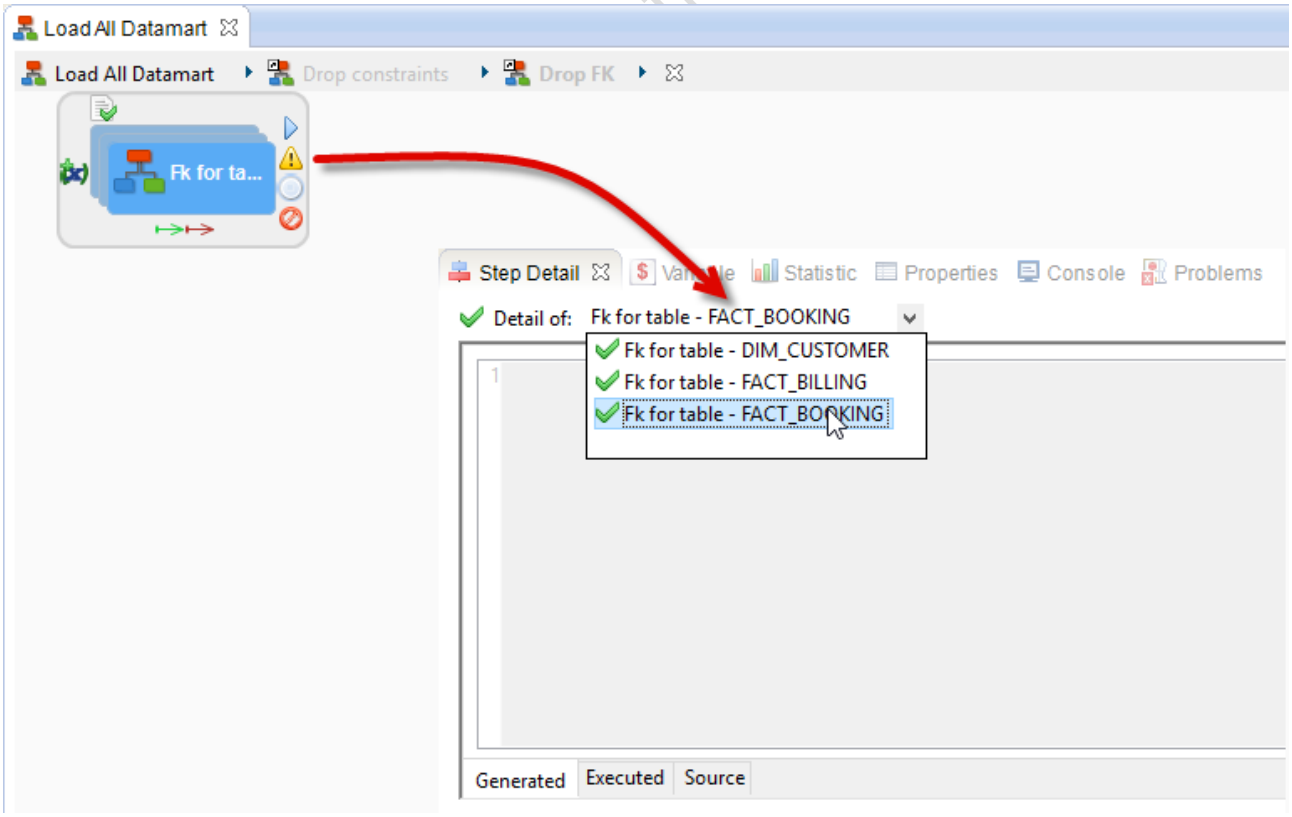
Statistic ビューを開き、プロセス実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	232 140
SUM(SQL_STAT_ERROR)	13
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	0

さらに、生成されたコードを確認することもできます。例えば、Drop constraints ステップから Drop FK を開きます。表示される under-process（下位プロセス）は、Fk for table の 3 回のインスタンスから成ります。この下位プロセスは、外部キーを持つテーブルに対して実行されるようにコンフィギュレーションされています。Fk for table 3/3 という名前が付けられているのは、そのためです。

このようなステップでは、正しいインスタンスを選択することが重要になります。

1. 下位プロセス Fk for table 3/3 を選択します。
2. **Step Detail** ビューを開きます。
3. **Detail of** のドロップダウンリストから、確認したいインスタンスを選択することができます。
4. **Fk for table – FACT_BILLING** を選択します。



以上により、Fk for table 3/3 をダブルクリックすれば、下位プロセスの該当するインスタンスを開くことができます。
テーブルには 4 つの外部キーが含まれます。そのため、アクション名は Drop FK **4/4** となっています。
以上の手法を用いて、それぞれの外部キーに対して実行されたコードを確認してください。

21.5 デフォルト値を追加する

21.5.1 データパージを追加する

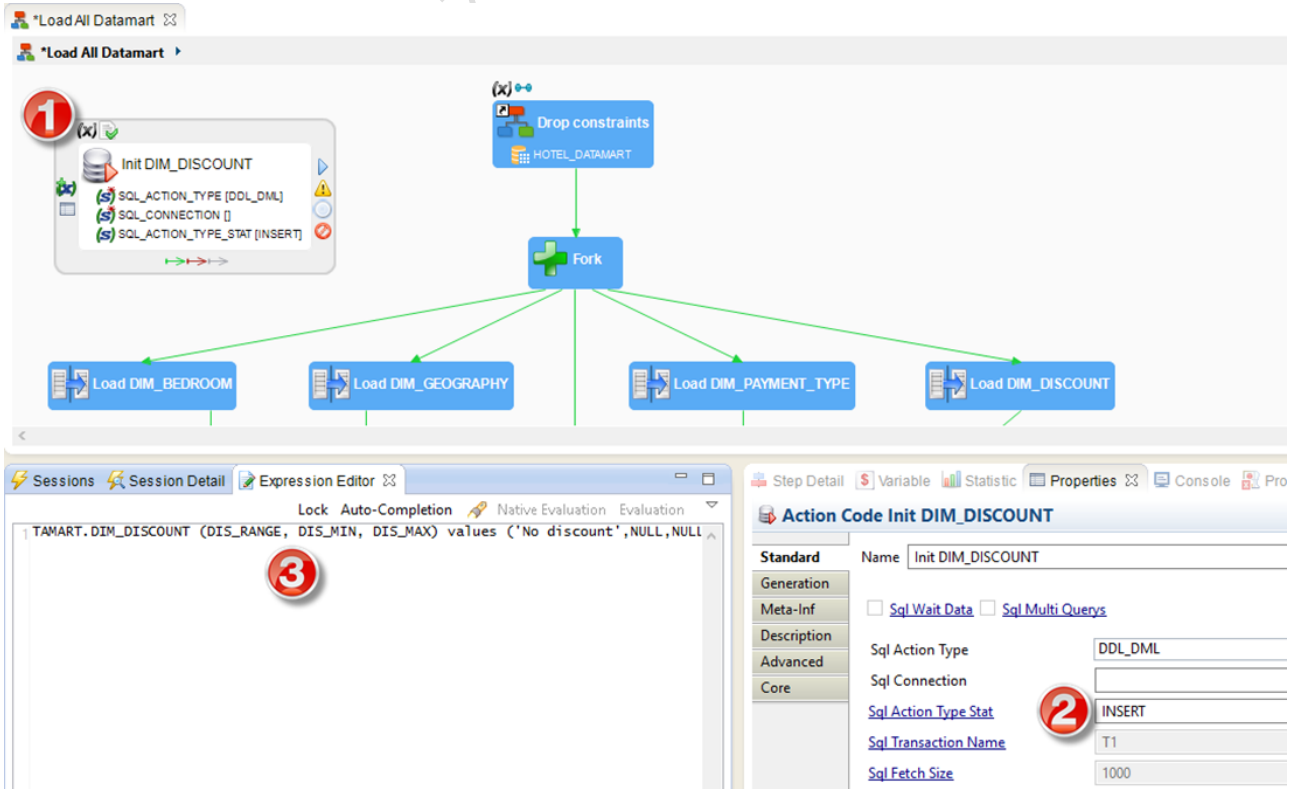
Process の処理で次に行うステップは、全テーブルにおける全データのパージ（除去）ならびに初期データのリセットです。（**HOTEL_DATAMART** ベースのテーブルには自動パージで削除されるレコードを含むものがあります。）
テーブルからデータをパージするには、Drop constraints ステップの編集により、テンプレートオプション **Delete Tables**、さらには **Drop Fk** を選択します。このオプションにより、**Stambia** はターゲットテンプレートの全データをパージします。

21.5.2 テーブルを初期化する

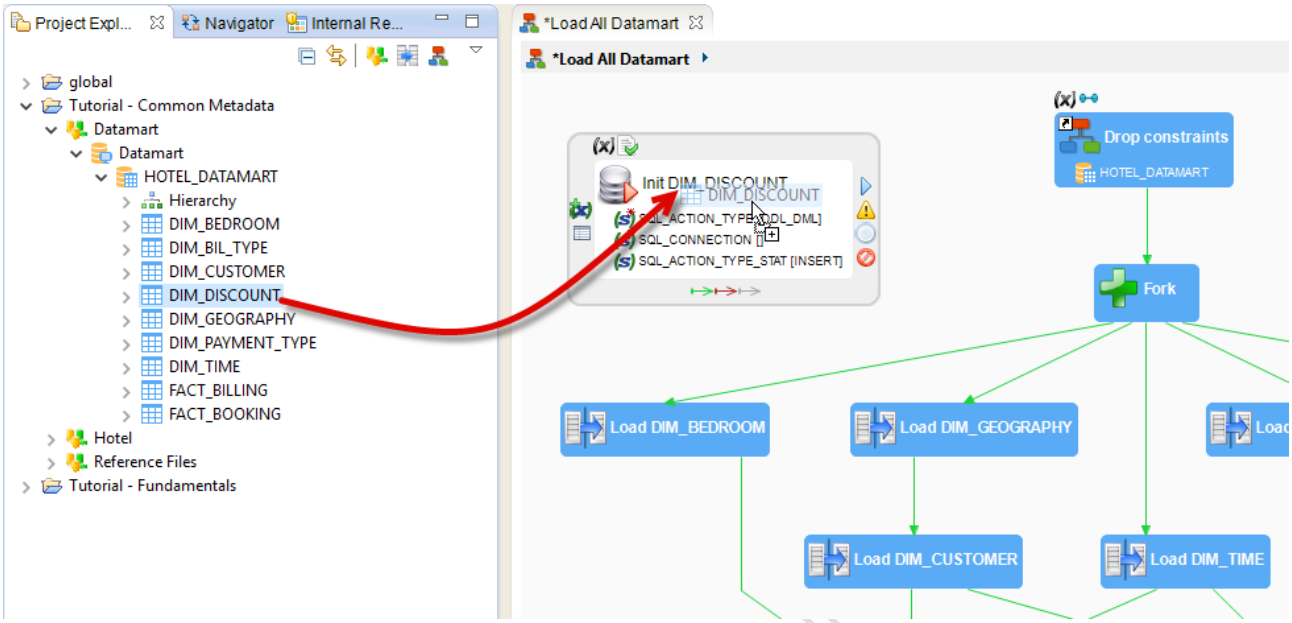
最後に、データベースに不足するデータを再挿入するアクションを作成する必要があります。

1. コンポーネントの **Palette** から **Sql** を開きます。
2. **Sql Operation** を選択します。
3. 表示された図の内側をクリックして、空のステップを作成します。
4. アクション名 Init DIM_DISCOUNT を入力します。
5. **Properties** ビューにおいて、**Sql Action Type Stat** リンクをクリックし、`INSERT` と入力します。
6. **Expression Editor** ビューにおいて、下記の SQL コマンドを入力します。

```
insert into HOTEL_DATAMART.DIM_DISCOUNT (DIS_RANGE, DIS_MIN, DIS_MAX)
values ('No discount',NULL,NULL);
```



さらに、この SQL コマンドをどの JDBC 接続に対して実行すべきかを指定するために、メタデータリンクを使用します。メタデータリンクを作成するには、**DIM_DISCOUNT** データストアを当該ステップにマウスでドラッグ&ドロップします。



上記の **Sql Action Type Stat** オプションをパラメータ化すれば、**Stambia** は挿入されたレコード件数をカウントし、その結果を INSERT (statistic SQL_STAT_INSERT)のセッション統計に統合することができます。

さらに、次の SQL アクションを下記のプロパティにもとづいて作成してください。

プロパティ	値
名前	Init DIM_GEOGRAPHY
SQL コード	insert into HOTEL_DATAMART.DIM_GEOGRAPHY (GEO_KEY_ID, GEO_ZIP_CODE, GEO_CITY, GEO_STATE_CODE, GEO_STATE) values (0, NULL, 'No Address', NULL, NULL); insert into HOTEL_DATAMART.DIM_GEOGRAPHY (GEO_KEY_ID, GEO_ZIP_CODE, GEO_CITY, GEO_STATE_CODE, GEO_STATE) values (1, '?', 'Unknown Zip Code', '?', '?');
Sql Action Type Stat	INSERT
Sql Multi Queries	<true>

パラメータ Sql Multi Queries を用いれば、複数の SQL コマンドを同じアクション内で実行することができます。同パラメータは、各種 SQL 指示の区切り文字を定義する Sql Multi Queries Separator パラメータと同時に使用されます（区切り文字のデフォルトはセミコロンです）。それら SQL の結果統計は、同ステップで実行される全 SQL の結果に統合されます。

上記により作成された 2 つのアクションはリンクでつなぎ、Drop constraints と Fork の間で並行して実行されるようにしなければなりません。

以上で、プロセスを実行する準備が完了しました。

21.5.3 プロセス実行結果を検証する

Statistic ビューを開き、プロセス実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	302 452
SUM(SQL_STAT_DELETE)	70 309
SUM(SQL_STAT_ERROR)	13
SUM(SQL_STAT_INSERT)	70 309
SUM(SQL_STAT_UPDATE)	0

21.6 レポートファイルを作成する

21.6.1 セッション変数の使用

Stambia はセッションを実行する際、下記のような情報を大量に処理します。

- 各ステップの実行時間
- ランタイムの一時フォルダ
- ランタイムのホスト名とポート
- セッション名
- 各ステップ名
- その他

これらの情報はすべて **Session Variables**（セッション変数）の形式で保存されています。

- セッション変数のうち、セッションに対してグローバルな情報はセッションレベルで直接アクセスできます。それらは **Session Detail** ビューにおいて、**Variables** タブで参照できます。
- その他のセッション変数は、プロセスの各ステップ処理時に動的に生成され、値が割り当てられます。それらは **Variables** ビューで確認できます。プロセス内で選択されたオブジェクトの変数が、**Variables** ビューに表示されます。

変数の名前は、その保管場所に拠って決まります。変数を参照するには、次にようにパスを指定する必要があります。
`/VAR` がグローバルセッション変数を表します。«/»はセッションのルートを表します。
ステップに対してローカルな変数を参照するには、当該ステップまでのパスと変数名を指定しなければなりません。
`../Step/VAR` によって、親ステップ下の現行ステップが検索され、その中の変数が照会されます。`Step` は現行ステップを表します。

セッション実行中はいつでも、下記のシンタックスで変数の値を照会することができます。
${Variable}$
変数の値がテキストと置き換えられて返されます。シンタックスの詳細は、**Stambia** ヘルプのプロセスに関する章を参照してください。

例：

変数	値
${/CORE_DURATION}$	セッション実行時間
${../Init/CORE_DURATION}$	Init ステップ実行時間
${../Init/First Value/CORE_DURATION}$	Init ステップ内の First Value アクション実行時間

21.6.2 実行レポートファイルの作成

下記の手順で、自由形式でファイルにテキストを書き込むためのステップが作成できます。

1. コンポーネントの **Palette** から **File** を開きます。
2. **Write a File** を選択します。
3. アクション名 Write Execution Report を入力します。
4. **Properties** ビューにおいて、**Txt Write Filename** プロパティに下記の値を指定します。
   ${/CORE_TEMPORARY_FOLDER}$/Load_All_Datamart_Report.txt

以上の手順は、下記の場所のファイルにテキストを書き込むことを **Stambia** に指示するものです。
${/CORE_TEMPORARY_FOLDER}$/Load_All_Datamart_Report.txt

このパスは、ランタイムの一時フォルダを保持するセッション変数 **CORE_TEMPORARY_FOLDER** にもとづいています。（セッションを選択し、**Session Detail** ビューの **Variables** タブを参照すれば、ランタイムのデフォルト値が判ります。）

次に、ファイルに書き込むべきテキストを指定します。

1. **Expression Editor** ビューを開きます。
2. 下記のテキストを入力します。

```
Session started on: ${/CORE_BEGIN_DATE}$

Steps duration:
    Load FACT_BILLING: ${../Load FACT_BILLING/CORE_DURATION}$
    Load FACT_BOOKING: ${../Load FACT_BOOKING/CORE_DURATION}$
    Recreate Fk : ${../Recreate constraints/Create FK/CORE_DURATION}$
```

当ステップは、制約の再生成後に実行され、その後にプロセスが実行されるようにリンクする必要があります。

21.6.3 プロセス実行結果を検証する

Statistic ビューを開き、プロセス実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	302 452
SUM(SQL_STAT_DELETE)	70 309
SUM(SQL_STAT_ERROR)	13
SUM(SQL_STAT_INSERT)	70 309
SUM(SQL_STAT_UPDATE)	0

Step Detail view. Write Execution Report ステップに実行されたコードを確認してください。変数値を含むテキストが、**Step Detail** ビューの **Executed** タブをクリックすることにより参照できます。

また、Stambia のインストール場所、stambiaRuntime フォルダの temp では、Load_All_Datamart_Report.txt ファイルが確認できます。

22 スクリプトの使用

22.1 STAMBIA スクリプト言語

Stambia では、スクリプト言語により、一連のセッションを通じて Runtime（ランタイム）に働きかけ、連携することができます。

Stambia でスクリプトを活用するための一般的なシンタックスは下記の通りです。

```
%e(language){ .... SCRIPT .... }e(language)%
```

利用可能なスクリプトには以下の言語があります。

- groovy
- jython（Python のシンタックス）
- rhino（JavaScript のシンタックス）

さらに、**Stambia** は Runtime と直接連携できる API も提供します。この API__ctx__は上記のいずれの言語でも利用できます。
※あらかじめ定義され、開発用に準備されたメソッドを持つ Java クラスです。

Stambia でのスクリプトに関するさらに詳しい情報は、**Stambia Designer** のユーザーガイドを参照してください。

22.2 レポートファイルに統計データを追加する

__ctx__.sumVariable() は、**Stambia** で使用できる関数の 1 つで、Runtime（ランタイム）の変数から数値を合計します。
__ctx__.sumVariable(<VAR>)が、VAR という変数の、セッション全体からの数値合計を送り返します。
__ctx__.sumVariable(<VAR>, <STEP>)は、VAR という変数の、STEP というステップのみでの数値合計を送り返します。

その他の関数は **Stambia** のヘルプに解説されています。

Write Execution Report ステップを修正して、プロセスの特定ステップの統計データを追加してください。

```
Total Rows: %e(rhino){__ctx__.sumVariable("SQL_NB_ROWS")}e(rhino)%
Total Inserts: %e(rhino){__ctx__.sumVariable("SQL_STAT_INSERT")}e(rhino)%
Inserts into FACT_BILLING: %e(rhino){__ctx__.sumVariable("SQL_STAT_INSERT","../Load FACT_BILLING")}e(rhino)%
```

23 練習課題

23.1 テーブルのデータを拡充する

当練習課題では、DIM_CUSTOMER テーブルの既存レコードの更新を、既存する値を修正せずに CUS_VIP カラムに値を追加することにより、行います。目的は、ホテルの開設以来、通算で 64000 ドルを超える額を支払った VIP 顧客を見つけることです。

23.1.1 DIM_CUSTOMER テーブルを拡充するマッピングの作成

下記のプロパティにもとづいて、新しいマッピングを作成してください。

プロパティ	値
親フォルダ	Mappings
マッピング名	Load DIM_CUSTOMER.CUS_VIP (1)
ターゲットデータストア	DIM_CUSTOMER
ソースデータストア	T_BILLING, T_BILLING_LINES

変換ビジネスルールは下記の通りです。

ターゲットカラム	ビジネスルール	特性		
CUS_ID	T_BILLING.CUS_ID	ソース	I/U	ファンクショナルキー
CUS_TITLE				
CUS_NAME				
CUS_COMPANY				
GEO_KEY_ID				
UPDATE_DATE	current_timestamp	ターゲット	U	
CUS_VIP	case when sum(T_BILLING_LINES.BLL_QTY*((T_BILLING_LINES.BLL_AMOUNT- T_BILLING_LINES.BLL_DISCOUNT_AMOUNT) - (T_BILLING_LINES.BLL_AMOUNT * T_BILLING_LINES.BLL_DISCOUNT_RATE / 100))) > 64000 then 1 else 0 end	ソース、集約	U	

CUS_VIP カラムのマッピングは **Updates**（更新）のみに有効となっており、特に他のターゲットカラムはマッピングされていない点に注意してください。つまり、他のターゲットカラムはマッピングの実行に影響を受けません。

Join（結合）のビジネスルールは下記の通りです。

第 1 データストア	第 2 データストア	ビジネスルール	実行場所
T_BILLING	T_BILLING_LINES	T_BILLING_LINES.BIL_ID=T_BILLING.BIL_ID	ソース

23.1.2 マッピング実行結果を検証する

Statistic ビューを開き、マッピング実行結果の統計を確認します。

演算	結果
SUM(SQL_NB_ROWS)	400
SUM(SQL_STAT_INSERT)	0
SUM(SQL_STAT_UPDATE)	100

DIM_CUSTOMER テーブルのデータが下記の情報と一致するかどうかを確認してください。

- VIP ではない顧客は 95 名：
  select count(*) from HOTEL_DATAMART.DIM_CUSTOMER where CUS_VIP != 1
- VIP 顧客の CUS_ID は **7**、**18**、**75**、**92**、**96**：
  select CUS_ID from HOTEL_DATAMART.DIM_CUSTOMER where CUS_VIP = 1

23.2 テーブルを部分的に拡充する

23.2.1 DIM_CUSTOMER テーブルを拡充するマッピングの作成 2

次の練習課題として、CUS_VIP フィールドを下記のルールで更新してください。

- 顧客が既に VIP として識別される場合、その顧客は VIP の立場を維持する
- 顧客が'suite'タイプの客室に 30 日を超えて宿泊したと請求されている場合、その顧客を VIP とする。
  T_BILLING_LINES のレコードおよび DIM_BEDROOM.BDR_TYPE カラムを確認する
- UPDATE_DATE カラムには、レコードが更新されるたびに現在の日時がセットされなければならない

23.2.2 実行結果を確認する

当課題の答えを導く方法は何通りかあり、それぞれ異なる統計データを生成するので、結果は DIM_CUSTOMER テーブル内容で確認します。

- VIP ではない顧客は 92 名：

select count(*) from HOTEL_DATAMART.DIM_CUSTOMER where CUS_VIP != 1

- VIP 顧客の CUS_ID は **7**、**18**、**31**、**53**、**75**、**89**、**92**、**96** :
  select CUS_ID from HOTEL_DATAMART.DIM_CUSTOMER where CUS_VIP = 1

実行結果が正しくなかった場合、再実行する前に初期段階に戻る必要があります。まず Load All Datamart プロセスを実行し、それから Load DIM_CUSTOMER.CUS_VIP (1)マッピングを実行してください。

23.2.3 よくある間違い

当トレーニング課題でもっとも間違えやすい点をいくつか紹介します。

スイート 30 日超宿泊分請求済みの顧客を見つける方法
'suite'タイプの客室に 30 日を超えて宿泊したと請求されている顧客を選択するもっとも簡単な方法は、マッピングにフィルタを作成することです。

count(T_BILLING_LINES.BLL_ID) > 30

SQL エラー：Not a condition
SQL 関数 count()は集約（aggregation）関数です。フィルタ作成時に、そのフィルタが集約関数を使用することを **Stambia Designer** に知らせなければなりません。

1. フィルタ count(T_BILLING_LINES.BLL_ID) > 30 を右クリックします。
2. **Aggregate** を選択します。

No VIP customer detected（VIP 顧客が見つかりません）
フィルタは、実行場所によっては期待通りの結果が得られない場合があります。フィルタ count(T_BILLING_LINES.BLL_ID) > 30 がソースに実行されたときにも、表題のエラーが生じます。

Stambia では、**source**、**staging area**、**target** の 3 通りの実行場所を指定することができます。例えば、DIM_BEDROOM テーブルをロードする際、ターゲットでタイムスタンプのカラムに対して関数を実行しています。これは、データがターゲットテーブルに統合された際に、その時点のタイムスタンプを記録したいためです。また、**Stambia** がターゲットで式を評価できるもう 1 つの場所は **staging area** になります。

ソースからのすべてのデータが、ターゲットに統合される前にまとめられる場所を **staging area** と呼びます。

以下に、単純化されたデータを例に見て行きましょう。

BDR_ID	BDR_TYPE
1	suit
2	suit
3	suit

4	standard
5	standard

BDR_ID	CUS_ID	BDR_TYPE
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	2	1
7	2	1
8	2	1
9	2	1
10	3	5
11	3	5
12	3	5
13	3	5

次のフィルタを使用します： count(T_BILLING_LINES.BLL_ID) > 2

上記のデータを見れば、CUS_ID=1 の顧客は、タイプ 1、2、3 の客室に宿泊しているので、選択対象となるのが判ります。CUS_ID=2 の顧客もタイプ 1 の客室に 4 回宿泊しているので、選択対象です。

上記のフィルタがソースに実行されたら、**Stambia** はステージ領域での結合に BDR_ID を必要とするので、ソースで実行されるリクエストは下記の通りになります。

```
select BDR_ID, CUS_ID
from T_BILLING_LINES
group by BDR_ID, CUS_ID
having count(BLL_ID) > 2
```

結果のデータは以下の通りになります。

CUS_ID	BDR_ID	Count(BLL_ID)
~~1~~	~~1~~	~~1~~
~~1~~	~~2~~	~~1~~
~~1~~	~~3~~	~~1~~
~~1~~	4	~~1~~
~~1~~	~~5~~	~~1~~
2	1	4

3	5	4

棒線で消されたデータは、フィルタ `count(BLL_ID) > 2` によって消去され、ステージ領域には存在できないデータです。

ここで、DIM_BEDROOM との結合が実行されたら、suite タイプの客室だけが対象となるので、残る顧客は `CUS_ID=2` のみになります。

BDR_ID	CUS_ID	BDR_TYPE
2	1	suit
3	5	~~s~~

上記の通り、最後のレコードも客室タイプ **standard** なのでフィルタにより消去されます。

以上の例により、フィルタがソースで実行されたときに誤った結果が生じていることが判ります。しかし、**staging area** で実行したらどうなるでしょうか。ソースデータがステージ領域にまとめられ、そこで結合とフィルタが実行されます。以下が、実行されるリクエストです。

```
select L.CUS_ID
from T_BILLING_LINES L, DIM_BEDROOM B
where T.BDR_ID=B.BDR_ID
and B.BDR_TYPE='suite'
group by CUS_ID
having count(BLL_ID) > 2
```

CUS_ID	BDR_ID	Count(BLL_ID)
1	suit	4
1	~~S~~	2
2	suit	4
3	~~S~~	4

選択されるべき顧客が 2 名とも正しく選択されています。

したがって、マッピングを下記のように修正します。

- フィルタ count(T_BILLING_LINES.BLL_ID) > 30 を右クリックします。
- **Execution Location** を選択し、次いで **Staging Area** を選択します。

SQL エラー : Violation of unique index

ソースデータが count(T_BILLING_LINES.BLL_ID) > 30 でフィルタされていないときに、下記のような式で変換されると生じるエラーです。

```
case
when count(T_BILLING_LINES.BLL_ID) > 30 then 1
else 0
end
```

変換式が **Staging Area** ではなく **Source** で実行されてことが理由で生じる問題です。

実行場所をステージ領域とすることの必要性については、No VIP Customer Detected に詳しく解説されています。

マッピングを下記のように修正する必要があります。

- 変換式 CUS_VIP を右クリックします。
- **Execution Location** を選択し、次いで **Staging Area** を選択します。

I’ve got only 6 VIP customers（VIP 顧客が 6 名のみ）

ソースデータにフィルタを使用する代わりに、以下の変換式が使用された可能性があります。

```
case
when count(T_BILLING_LINES.BLL_ID) > 30 then 1
else 0
end
```

その過程で、前の課題で計算された VIP 顧客が修正されたしまったと考えられます。

マッピングのソースに DIM_CUSTOMER テーブルを追加し、顧客を下記の条件でフィルタしてください。

```
DIM_CUSTOMER_2.CUS_VIP!=1
```

24 更新履歴

版	修正日	修正者	内容