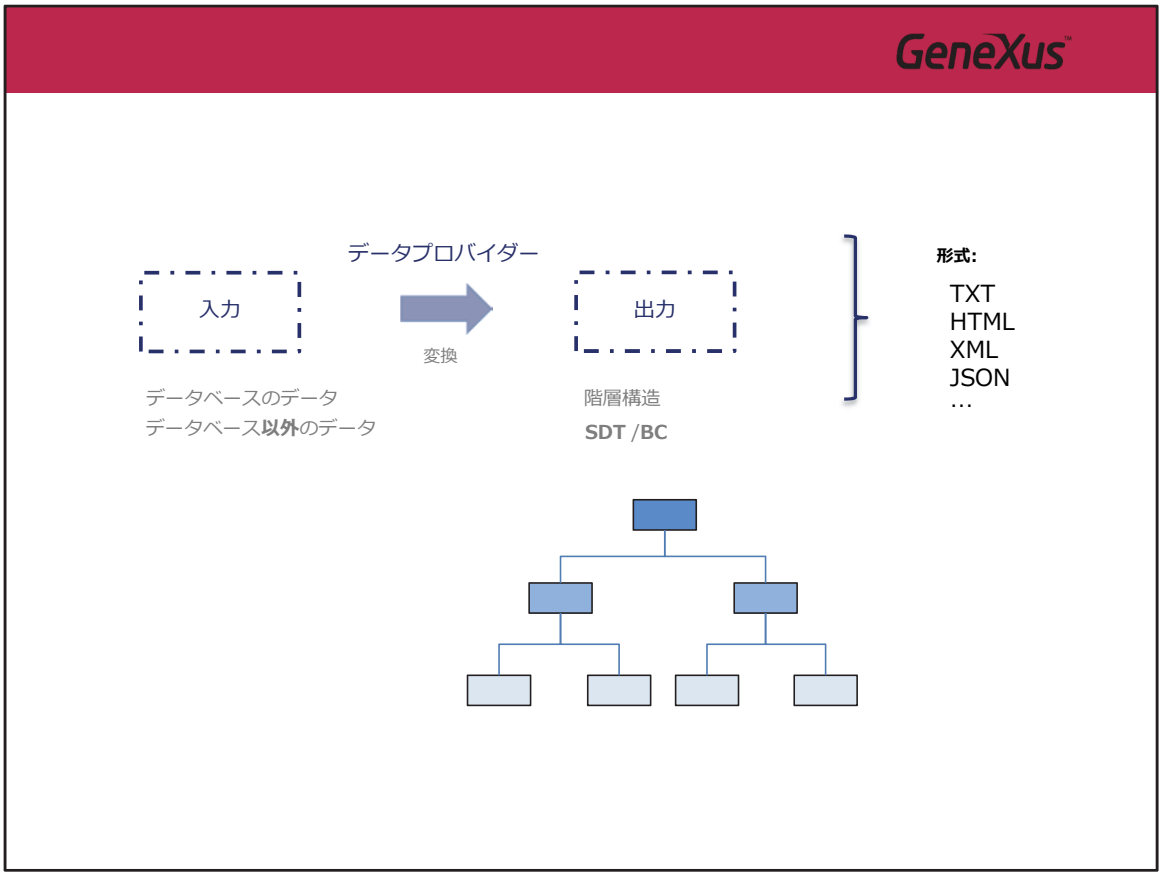
データプロバイダー

言語とその例

GeneXus™



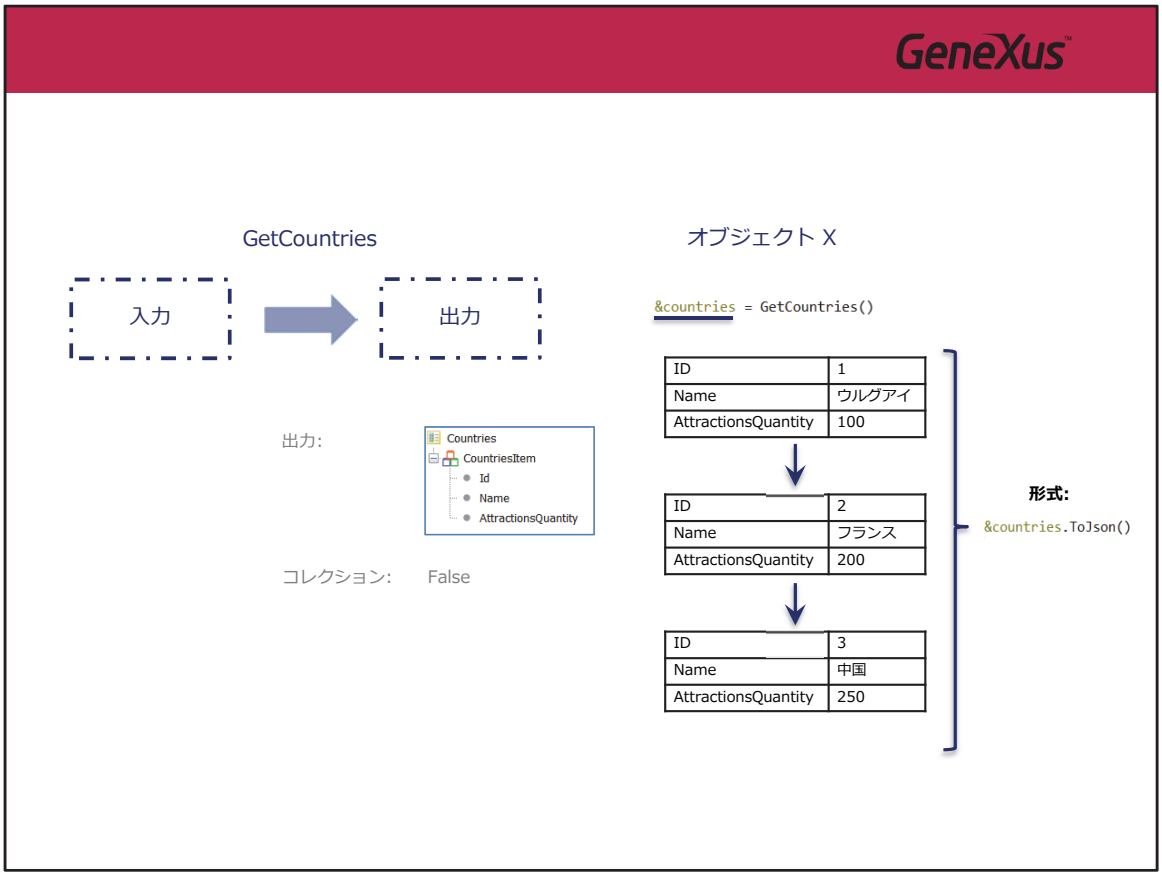
データプロバイダーの目的は、階層的な情報を取得して、それを必要とする人が後で使用できるようにすることです。

データプロバイダーでは出力言語に重点が置かれていることを思い出してください。その出力がどのように設計されているかを示すのが階層構造です。入力データをこの構造化された出力に変換するプロセスを説明するのはそのためです。データは、データベースから取得される場合とそうでない場合があります。

GeneXus では、階層構造を SDT オブジェクトを使用して表します。コレクションを定義することも可能です。もちろん、構造的に、ビジネスコンポーネントを SDT として考えることもできます。データプロバイダーの [Output] プロパティで SDT とビジネスコンポーネントの両方を指定できるのはそのためです。また、出力が指定されたデータタイプのコレクションになるか、単一のアイテムになるかを示す [Collection] プロパティもあります。

そのため、データプロバイダーは必ず、呼び出し元に階層を返します (SDT、SDT のコレクション、ビジネスコンポーネント、またはビジネスコンポーネントのコレクション)。

したがって、これを呼び出す人は、その階層情報に対して必要なことを行う必要があります。たとえば、階層データを表現するために、サードパーティと情報をやり取りするのに便利な XML や JSON などの形式に変換します。



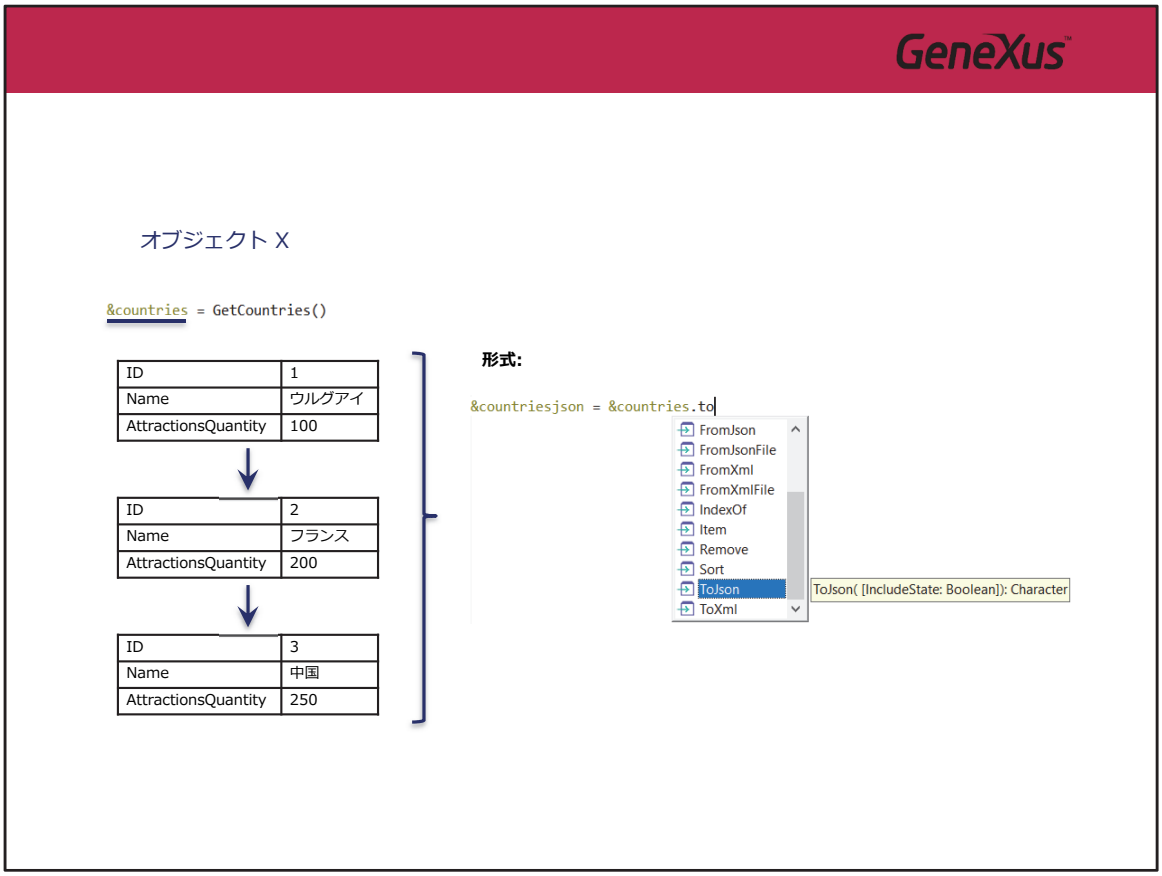
この例には GetCountries というデータプロバイダーがあり、Countries というデータタイプを返します。ご覧のとおり、単純な SDT のコレクションであり、各アイテムが CountriesItem という名前を使用します。

データプロバイダーのプロパティには次のものがあります。

[Output] プロパティは、Countries という名前の SDT オブジェクトです。
[Collection] プロパティは False に設定されています。これは Countries のコレクションが必要ないためです。True に設定されていた場合はコレクションになります。

この例では、データプロバイダーの呼び元が、出力と同じデータタイプの Countries 変数に結果を割り当てています。この結果は、そのデータプロバイダー内で計算された値を持つ、メモリー内の特定のコレクションになります。

この変数を使用するプログラムは、コンテンツを JSON 形式に変換するなど、さまざまなことを実行できます。

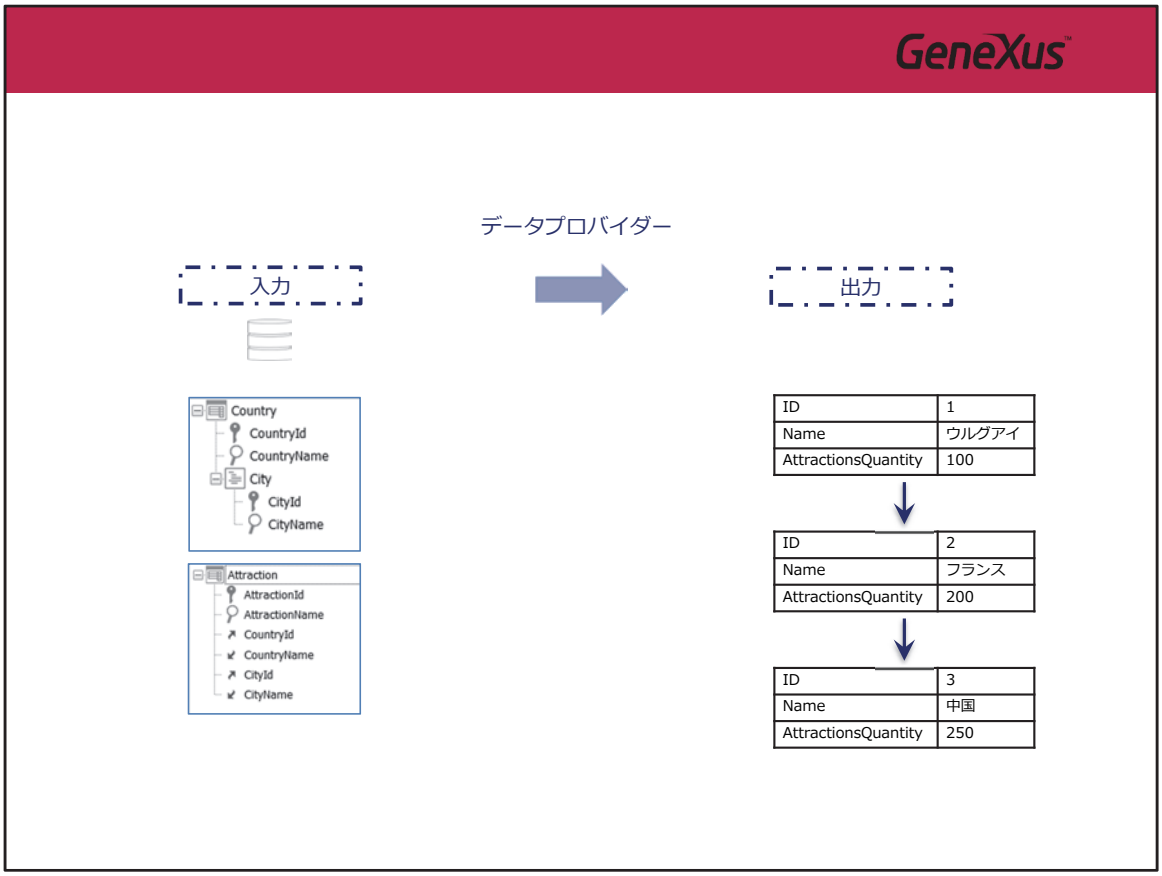


GeneXus には、SDT とその他の形式を変換する方法が複数あります。

将来、構造化された情報を表す新しい形式が使用できるようになったとしても、データプロバイダーは変更されません。その形式への変換方法が GeneXus に実装され、それを使用するようになるだけです。

SDT から別の形式への変換も、その逆の変換にも対応します。

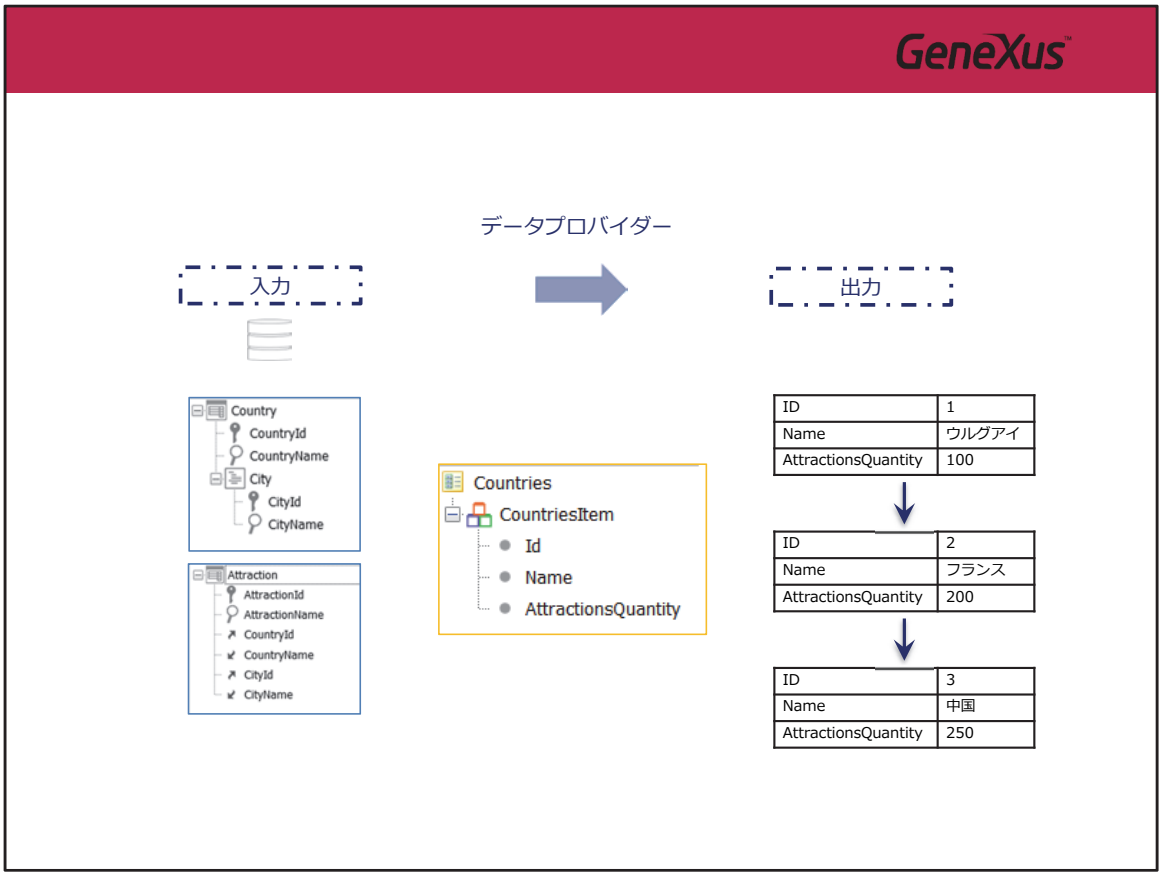
これは、データプロバイダー自体とは関係のない、構造化データタイプの問題です。国のコレクションは、データプロバイダーではなくプロシージャーで取得される場合もあります。変換の処理は同様に行われます。



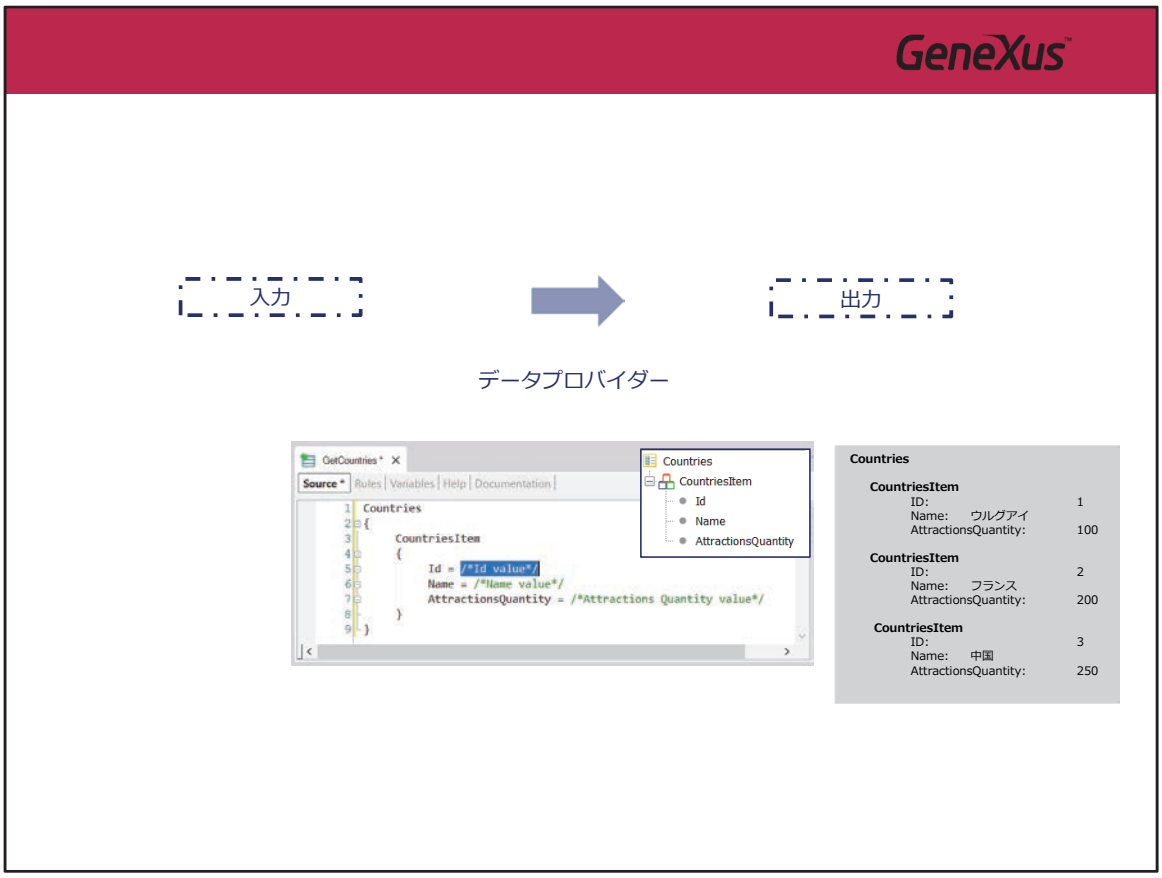
例を見てみましょう。旅行代理店向けのアプリケーションで、国を観光名所の数が多い順に画面に表示する必要があるとします。

ここでは Country トランザクションと Attraction トランザクションがあり、図に示す項目属性があります。
この処理を行う最もシンプルな方法は、国のコレクションをそれぞれの名前、ID、観光名所の数とともに返すデータプロバイダーを宣言することです。その後、返されたコレクションを観光名所の数の多い順に並べ替えます。

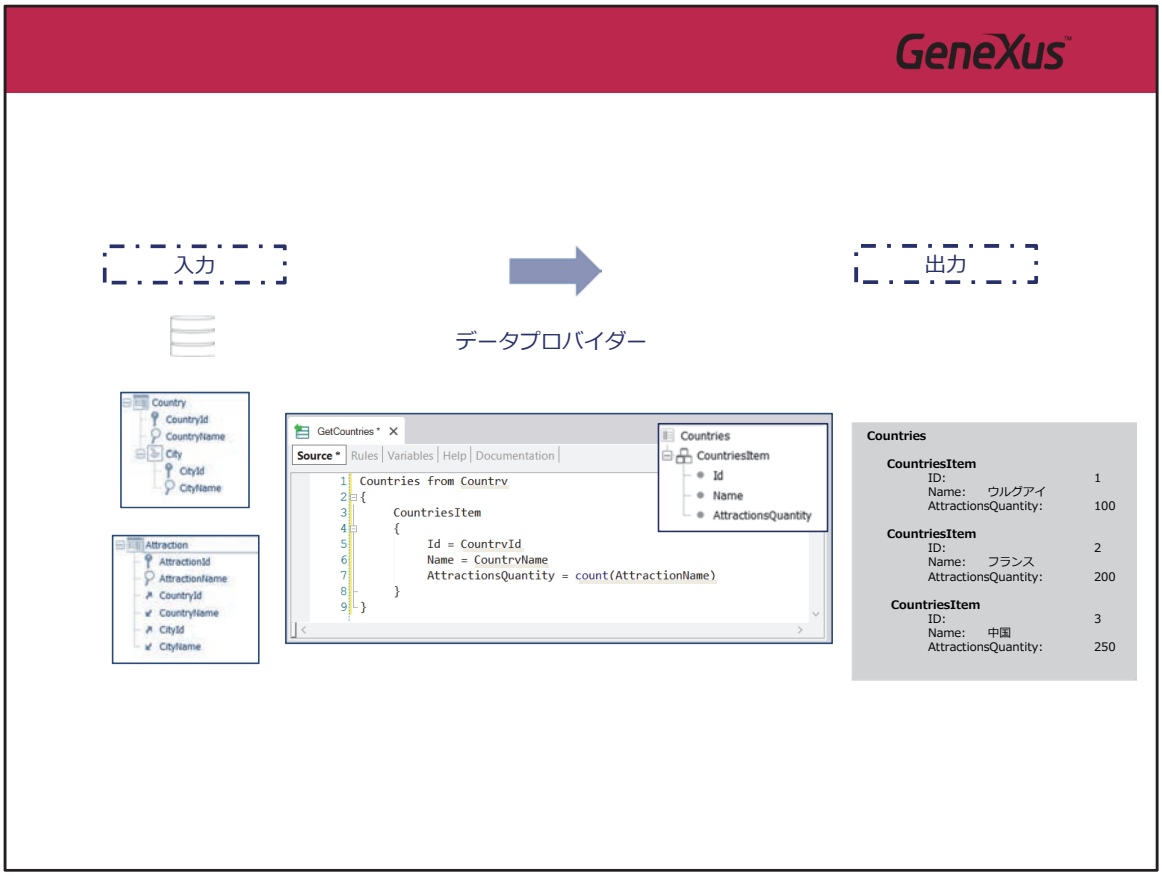
前に言ったように、データプロバイダーの言語は出力に重点を置いており、エレメントは、結果となる階層の観点から計算されます。



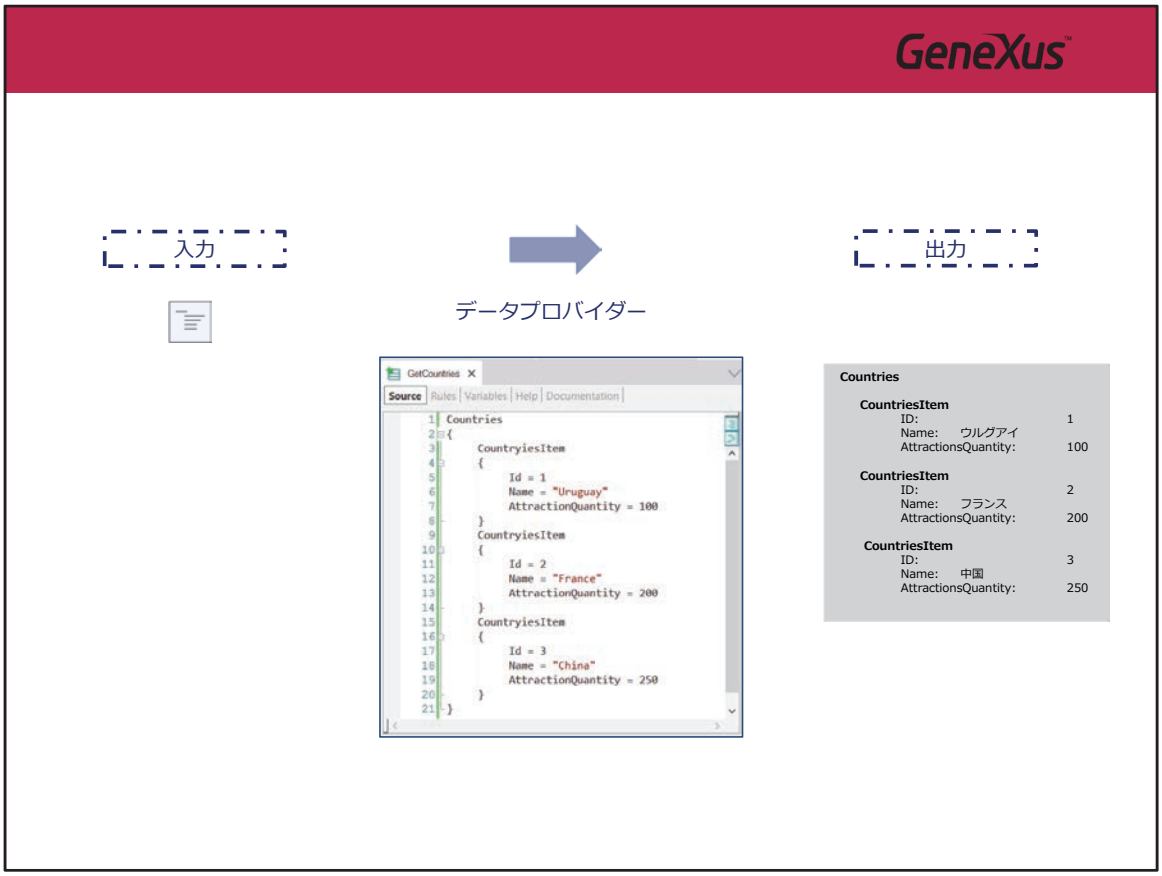
この例を表すために、後でデータプロバイダーから返されるデータ構造を作成します。次に、この SDT オブジェクトをデータプロバイダーのソースで読み込みます。



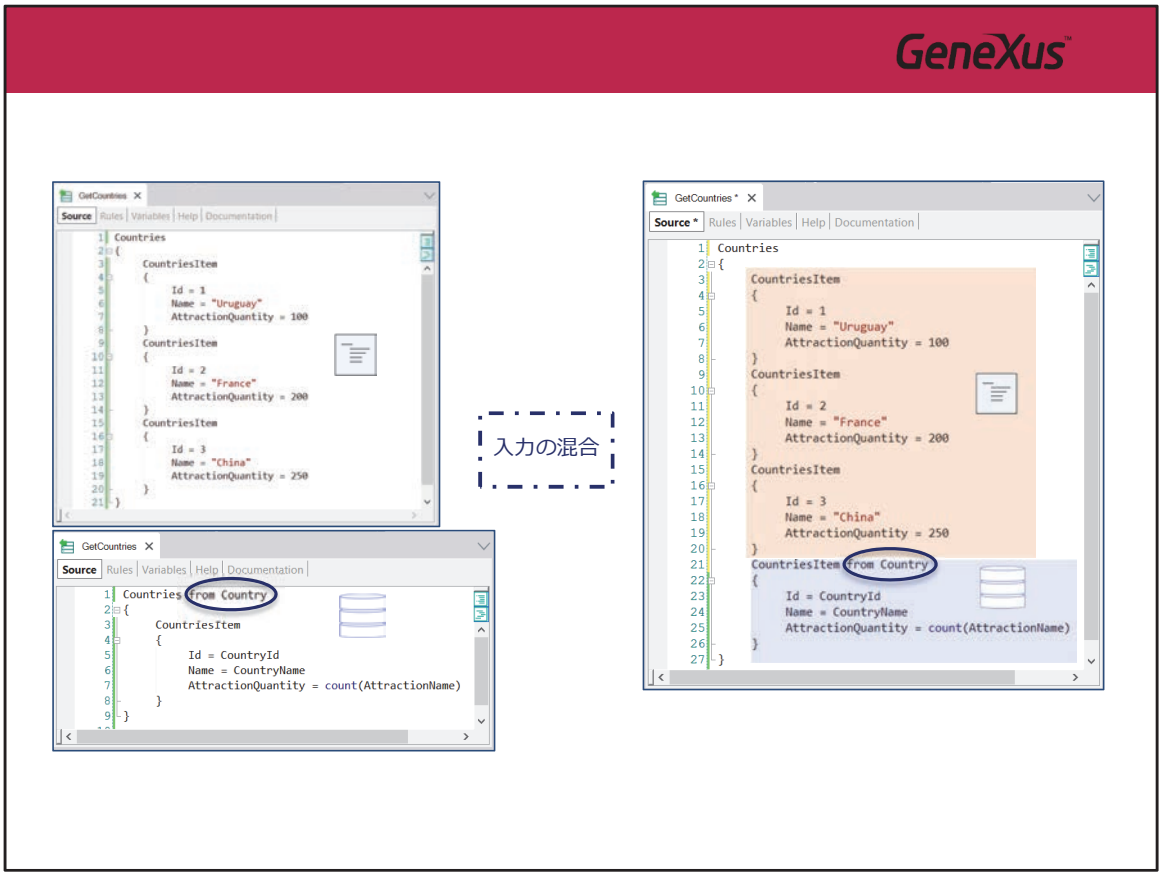
SDT をドラッグすると、データプロバイダーの出力として、読み込む構造が表示されます。その言語が出力ステートメント向けであることがよく分かります。



ここに示されているのは、データプロバイダーの入力、つまり、データが取得される場所です。ここでは、ベーストランザクション、項目属性、インライン式が指定されています。データはデータベースから取得され、階層的なデータに変換されます。



データがソースに静的にロードされていた、つまり、入力をデータベースから取得するのではなく手動でコーディングした場合も、同じ結果になります。ここではベーストランザクションや項目属性が指定されていないため、GeneXus ではデータベースから情報が取得されません。



入力を混合させることも可能です。つまり、1 つのパートは静的にコーディングし、もう 1 つのパートはデータベースから取得する方法です。

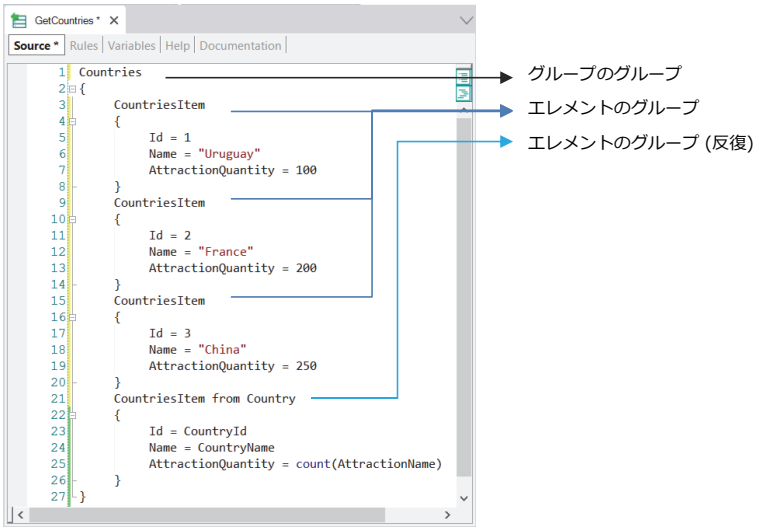
この例では、データプロバイダーのソースで、静的にロードされたコレクションから 3 つのアイテムが出力に表示されること、また、データベースの Country テーブルのレコードから N 個のアイテムがロードされることが分かります。

from 節は、すべての CountriesItem サブグループではなく、最後の CountriesItem サブグループに適用されるように移動する必要がある点に留意してください。先ほど説明したとおり、手動でコーディングした静的なパートでは、レコードはデータベースからは取得されません。

GeneXus

データプロバイダーの言語

- グループ
- エレメント
- 変数



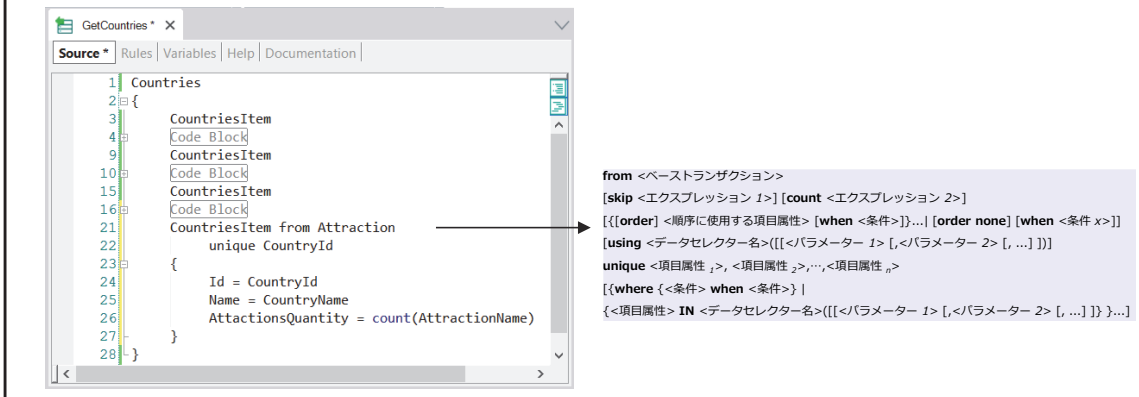
次に、データプロバイダーのソース言語の主なコンポーネントを確認します。

グループとエレメントのほか、変数も使用できます。

エレメントは 1 つの SDT のメンバーと似ています。階層をツリーとして考えると、グループはブランチで、エレメントはリーフになります。つまり、グループは複合エレメントであり、ほかのグループやエレメントで構成することが可能です。

グループは、静的なものも、動的にロードされるものもあります。これらは反復グループと呼ばれます。この例では、最初の 3 つのグループが静的グループで、固定データがロードされます。最後のグループはベーステーブルが関連付けられているグループであり、ベーステーブルのレコードごとに 1 つずつ、N 個のアイテムが出力に表示されます。

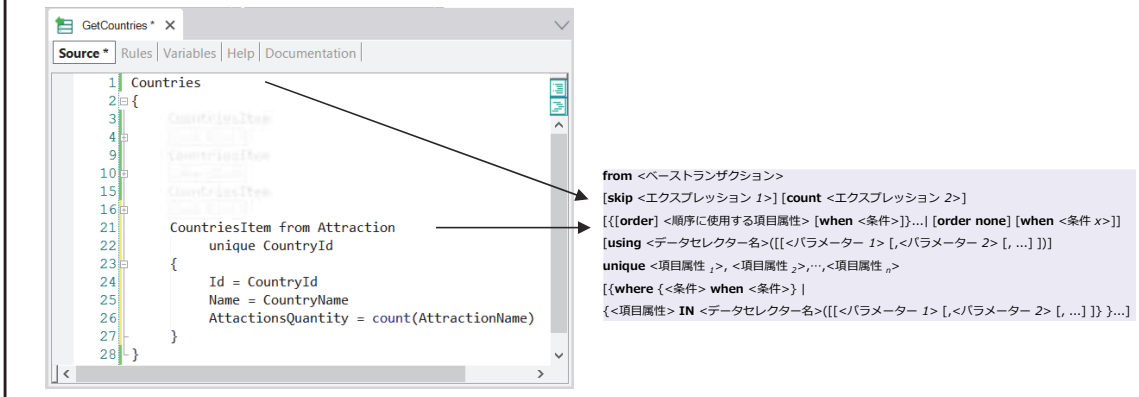
ベーステーブルが関連付けられたグループは、For Each コマンドと同等です。



グループではベーストランザクションを指定することができますが、For Each とは異なり、ベーストランザクションの名前またはレベルの前に「from」を置いて指定します。

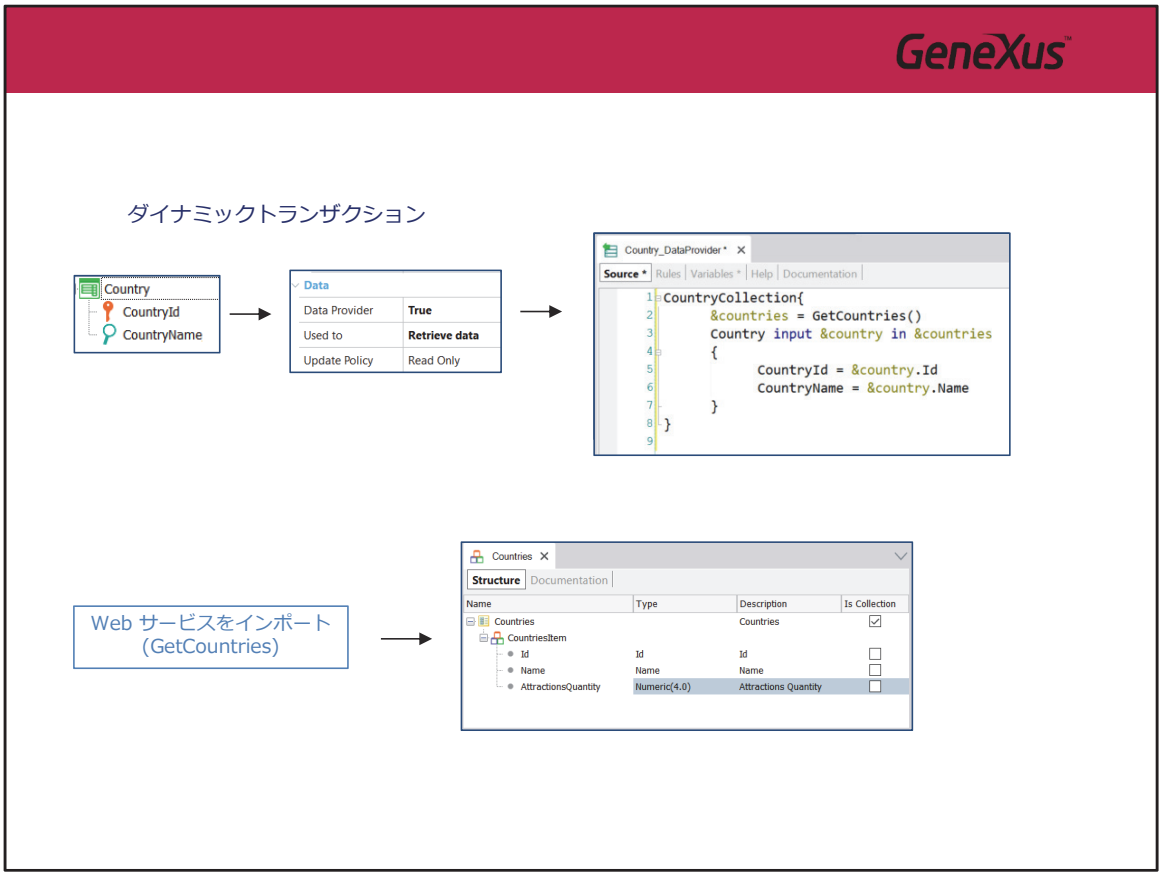
この例では、COUNTRY ベーステーブルを参照するのではなく、ATTRACTION を参照しています。また、unique 節を使用することで、1 つの国に複数の観光名所がある場合、1 つのみを考慮し、それに対して、同じ国にあるすべての観光名所をカウントします。このようにして、観光名所のレコードがある国のみが出力にリストされます。

For Each コマンドで説明したすべての処理を行うことができます。



静的グループが不要な場合は、節を CountriesItem サブグループレベルでも Countries グループでも指定できます。

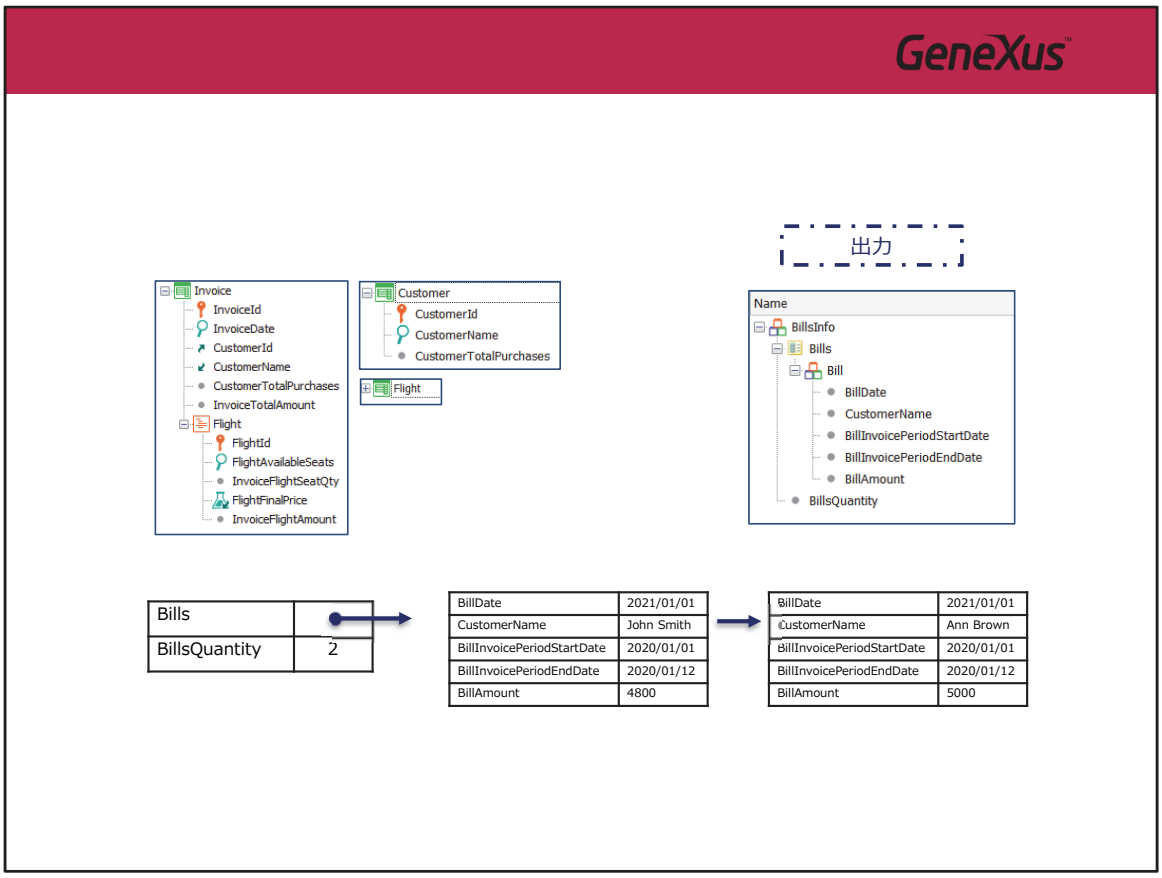
この場合、節を親グループのレベルで宣言しても、子グループのレベルで宣言しても同じです。



別の例を見てみましょう。国のデータを、データベースには格納しないが、何らかのサービスで取得するとします。そのために、動的な Country トランザクションを作成します。つまり、別の場所からデータを取得し、[Data Provider] プロパティを True に設定すると自動的にビルドされるデータプロバイダーで指定します。

ここでは、Web サービスが GetCountries であり、それをインポートすることでナレッジベースに Countries SDT が作成されるとします。

ソースで、まず国のコレクションを取得し、Country グループで INPUT 節を使用して参照します。コレクションの各アイテムを参照し、アイテムに対するグループで、そのエレメント (ダイナミックトランザクションの項目属性) に対応する値を割り当てます。



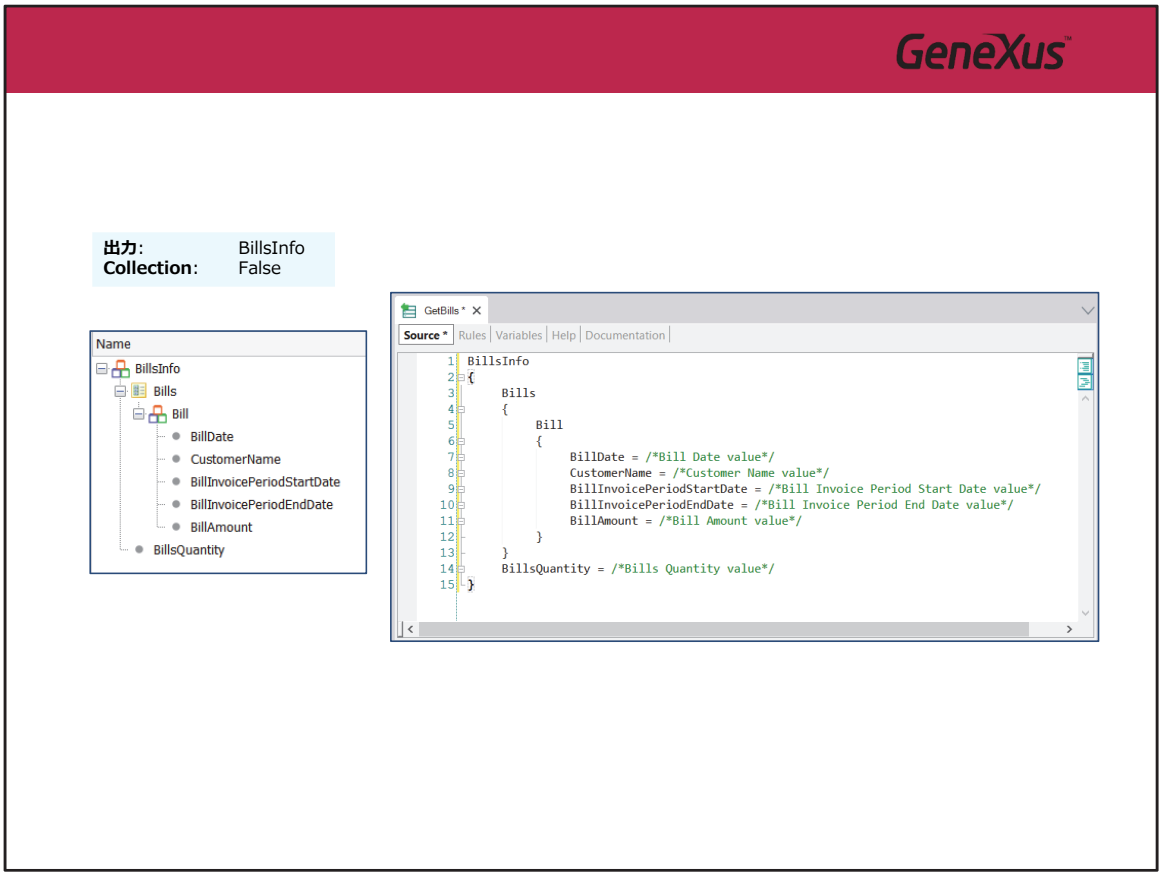
次に、少し複雑なケースを見てみましょう。
Bills のコレクションと Quantity エレメントを持つ構造を SDT として返すデータプロバイダーがあります。

このアプリケーションには、第 2 レベルと、図に示す項目属性を持つ Invoice トランザクション、スタンドアロンの Flight トランザクションと Customer トランザクションがあります。

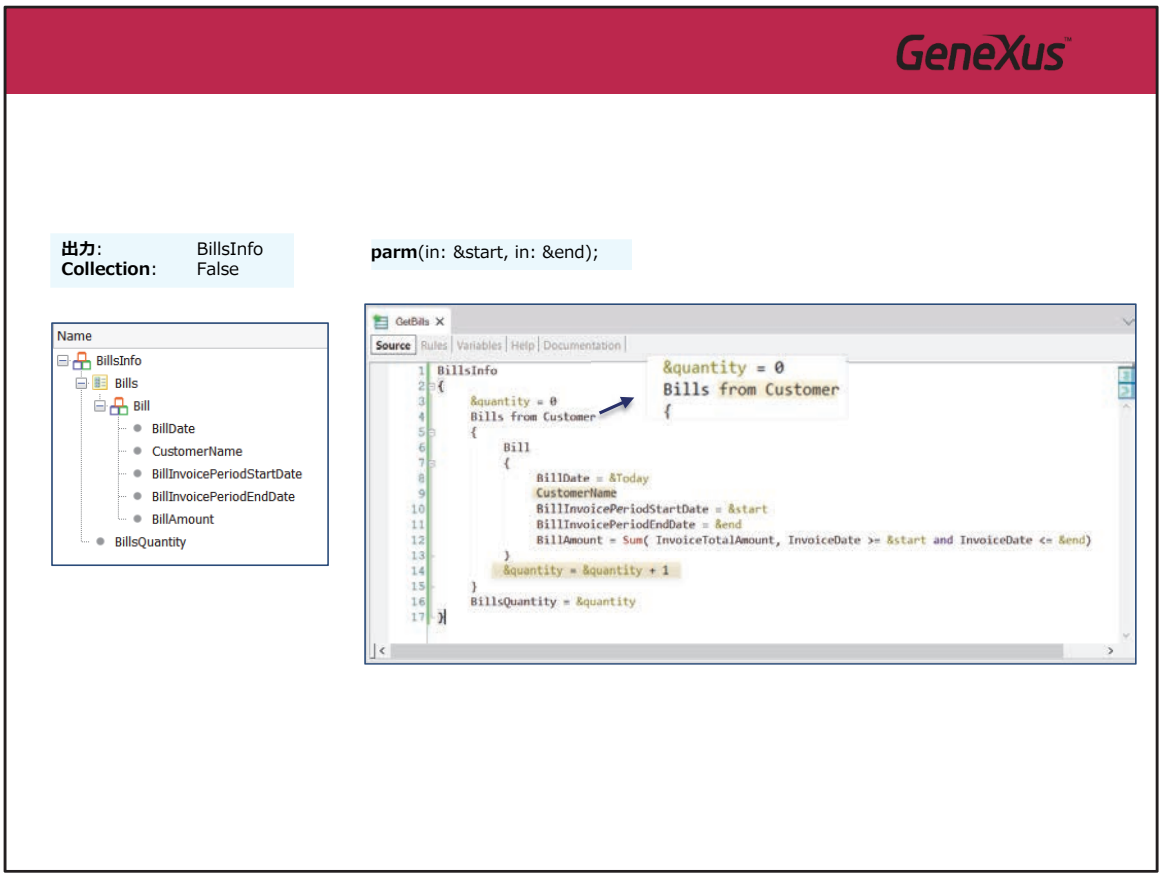
指定した 2 つの日付の間に各顧客宛てに生成された請求書に基づいて、すべての請求書の総計に対する領収書を作成します。この 2 つの日付の間に、すべての顧客に対して請求が発生しているとは限らないため、領収書が作成されない顧客もいます。返される構造の中では、計算によって得られる領収書の枚数を知る必要があるため、BillsQuantity メンバーが設定されています。

2020 年 1 月 1 日から 2020 年 1 月 12 日までの日付範囲内には John Smith と Ann Brown 宛ての請求書のみがある場合、出力はこの図に示すように 2 つのメンバーを持つ SDT 変数になります。メンバーの 1 つは Collection タイプで、もう 1 つは Numeric タイプです。コレクションには 2 つのアイテムがあります。

つまり、データプロバイダーでは 2 つのエレメントを持つ構造が返されます。1 つは領収書のコレクションで、もう 1 つはそのコレクション内のアイテムの数です。



SDT をデータプロバイダーのソースにドラッグすると、どのように初期化されるかが分かります。[Collection] プロパティは既定値の False のままになります。BillsInfo のコレクションを返すのではなく、該当するタイプの 1 つのエレメントのみを返すので、これは適切な処理です。返されるエレメントには Bills のコレクションが含まれます。



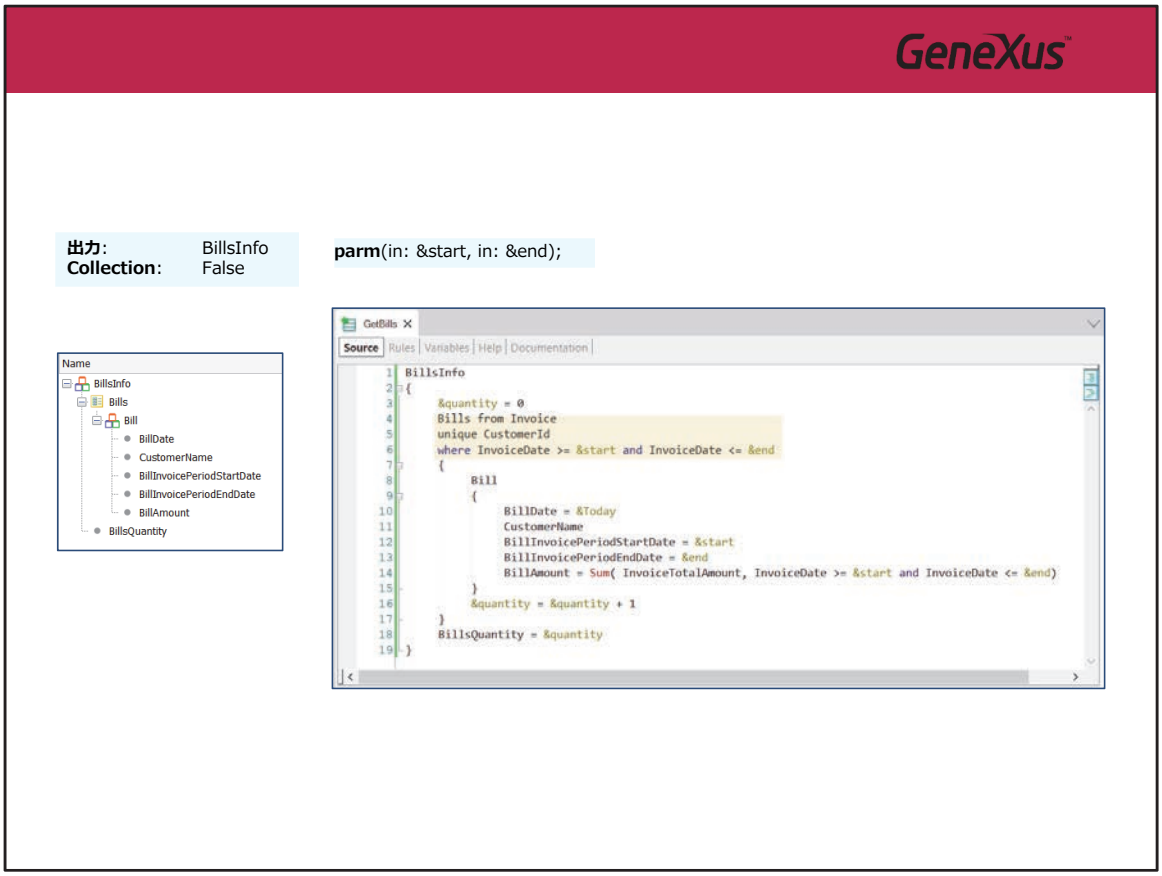
parm ルールをプログラミングして、請求日の範囲をパラメーターで受け取るようにします。

コレクションを表す Bills グループにはベーストランザクションを指定します。

値を割り当てずに CustomerName を記述するのはなぜでしょうか。これは、名前が CustomerName 項目属性と同じであり、Customer テーブルがナビゲーションされているためです。このように省略して表記できるのはそのためです。これは CustomerName = CustomerName と記述するのと同等です (左側は SDT メンバー、右側は Customer テーブルの項目属性)。

変数の使用については、For Each コマンドの場合と同じです。

ただし、この場合、パラメーターで受け取った範囲内に請求書がない顧客に対しても、出力で Bill アイテムが返されるという問題があります。この問題は、どうしたら回避できるでしょうか。



その方法の 1 つは、ベーストランザクションを Invoice に変更し、CustomerId を重複させず、希望する日付範囲内で Invoice レコードを保持することです。

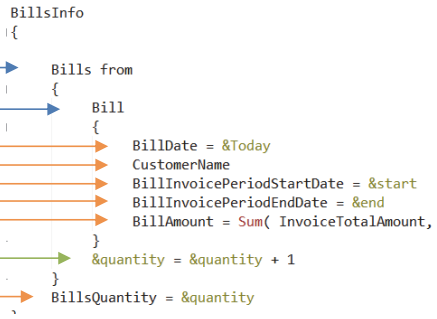
また、内部に指定した Sum では、同じ日付範囲内にある、顧客のすべての請求書の合計をカウントします。説明したとおり、これは For Each のロジックと同じです。

GeneXus

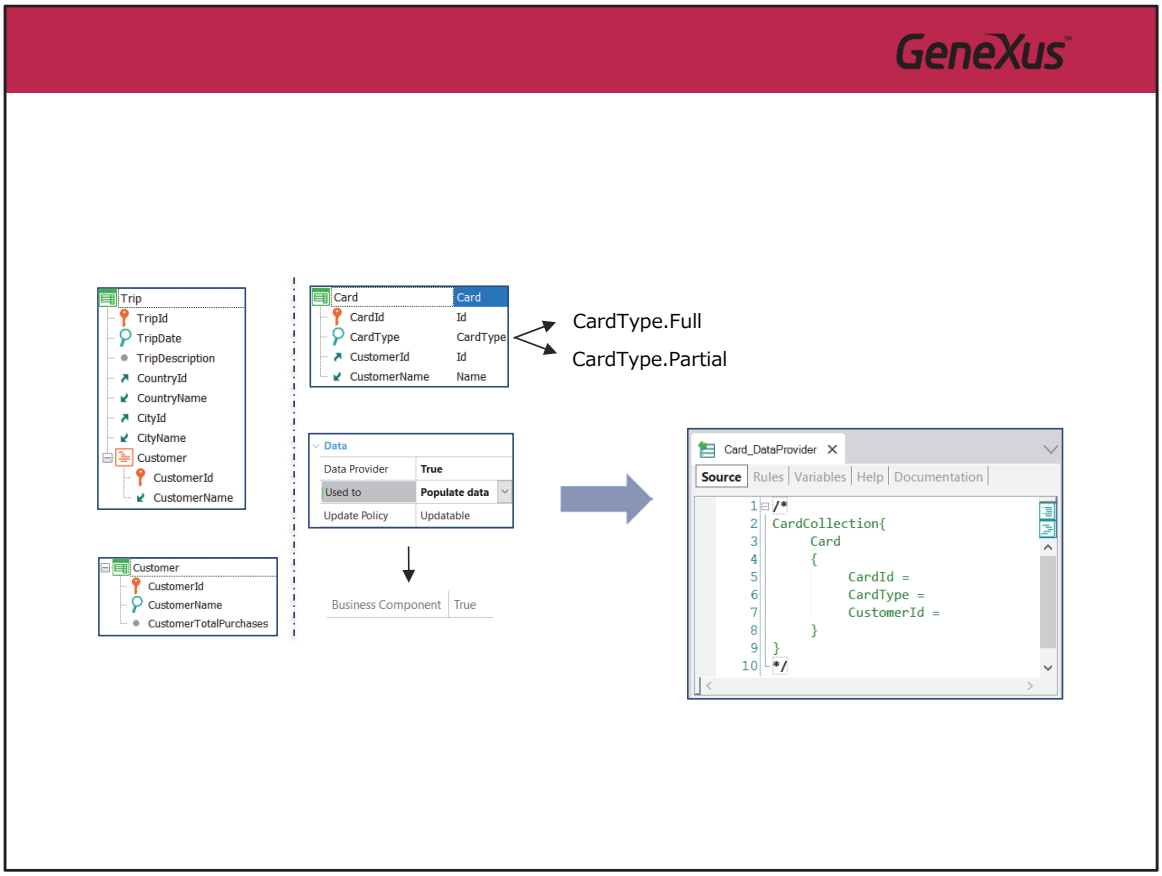
SDT 言語

標準的なコンポーネント

- グループ
- エレメントまたはメンバー
- 変数



これは、データプロバイダーの言語の標準的なコンポーネントを示しています。



別の例を見てみましょう。Card という新しいトランザクションと関連付けられているテーブルにデータを設定するとします。これを行うには、関連付けられたデータプロバイダーと、データベース内の他のテーブルのデータを使用します。

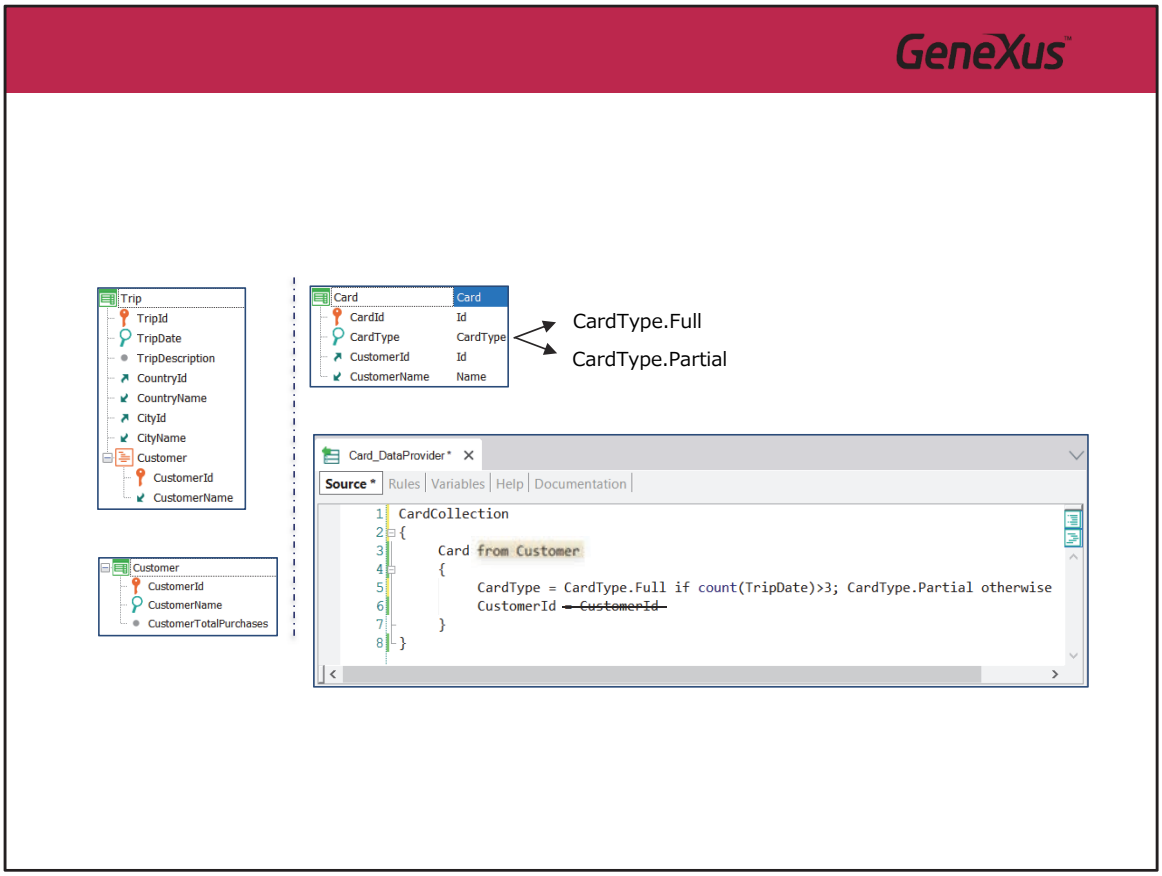
旅行代理店の顧客を記録する Customer トランザクションと、この旅行代理店が特定の都市で提供するツアーを記録する Trip トランザクションがあります。ツアーに登録されている顧客のサブレベルがあります。

この旅行代理店が、3 つ以上のツアーを予約した顧客全員に、すべてのサービスを無料で利用できる「Full サービス」タイプの特別なカードを提供することを決めたとします。また、予約したツアーの数が 3 つを下回る場合は、「Partial サービス」タイプのカードを提供します。

このカードには、自動採番された ID 番号と、顧客、カードタイプが設定され、「Full」または「Partial」の値のみをサポートする列挙型ドメインが定義されています。

ここで、Card トランザクションに関連付けられたテーブルを、適切な情報を使用して初期化する必要があります。そのため、[Data Provider] プロパティを有効にし、[Used to] の値を [Populate data] のままにします。こうすることで、表示されているデータプロバイダーが自動的に作成されます。また、初回実行時に自動的に実行されるときにそのデータプロバイダーによって返されるカードを挿入するため、[Business Component] プロパティを有効にします。

データプロバイダーのソースはどのように宣言されるのでしょうか。



各顧客の Card を作成するため、コレクション内で Card ビジネスコンポーネントに値を格納するための記述を行います。Card グループに Customer ベーストランザクションが指定されているのはそのためです。これで、ベーステーブルを持つグループであることが分かります。

トランザクションの CardId 項目属性の ID ドメインは自動採番されるため、Card グループから CardId エレメントを削除します。

次に、インライン条件式を使用して CardType エレメントの値をロードする方法を確認します。インライン式 count(TripDate) の実行結果が 3 を上回る場合は、列挙型の CardType.Full の値になります。そうでない場合は、値 CardType.Partial が割り当てられます。

この式は、TripCustomer テーブルのレコードを CustomerId でフィルタしてカウントします。

次に、ビジネスコンポーネントに対応する CustomerId エレメントに、Customer グループのベーステーブルの CustomerId 項目属性の値が割り当てられます。省略表記を使用して、割り当てを削除することができます。
この例では、データプロバイダーが、別のテーブルからデータを取得したビジネスコンポーネントのコレクションを返します。

これも、For Each コマンドの場合と同じです。