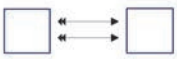
サブタイプのさまざまな
ユースケース

GeneXus™

GeneXus

複数参照

直接

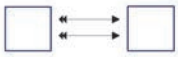


Basic コースでは、1 つのテーブルから直接関係がある別のテーブルへの複数参照のケースを学習しました。

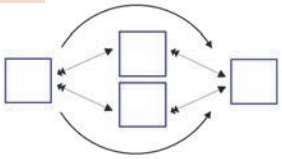
GeneXus

複数参照

直接



間接

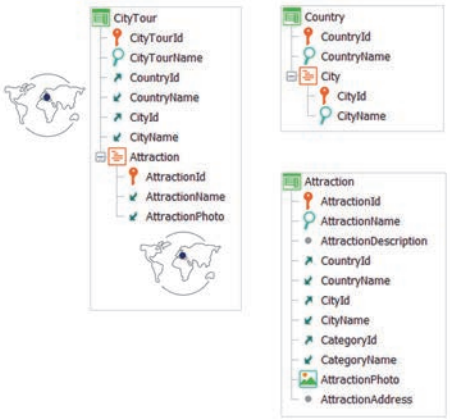


また、それらの参照が間接的に関係するケースについても学習しました。1 つのテーブルから別のテーブルへの 2 つのパスがあるため、サブタイプを使用してあいまいさを解消する必要が生じることがよくあります。

この章では、間接的な複数参照の別のケース、その問題点、考えられるソリューションについて学習します。

GeneXus

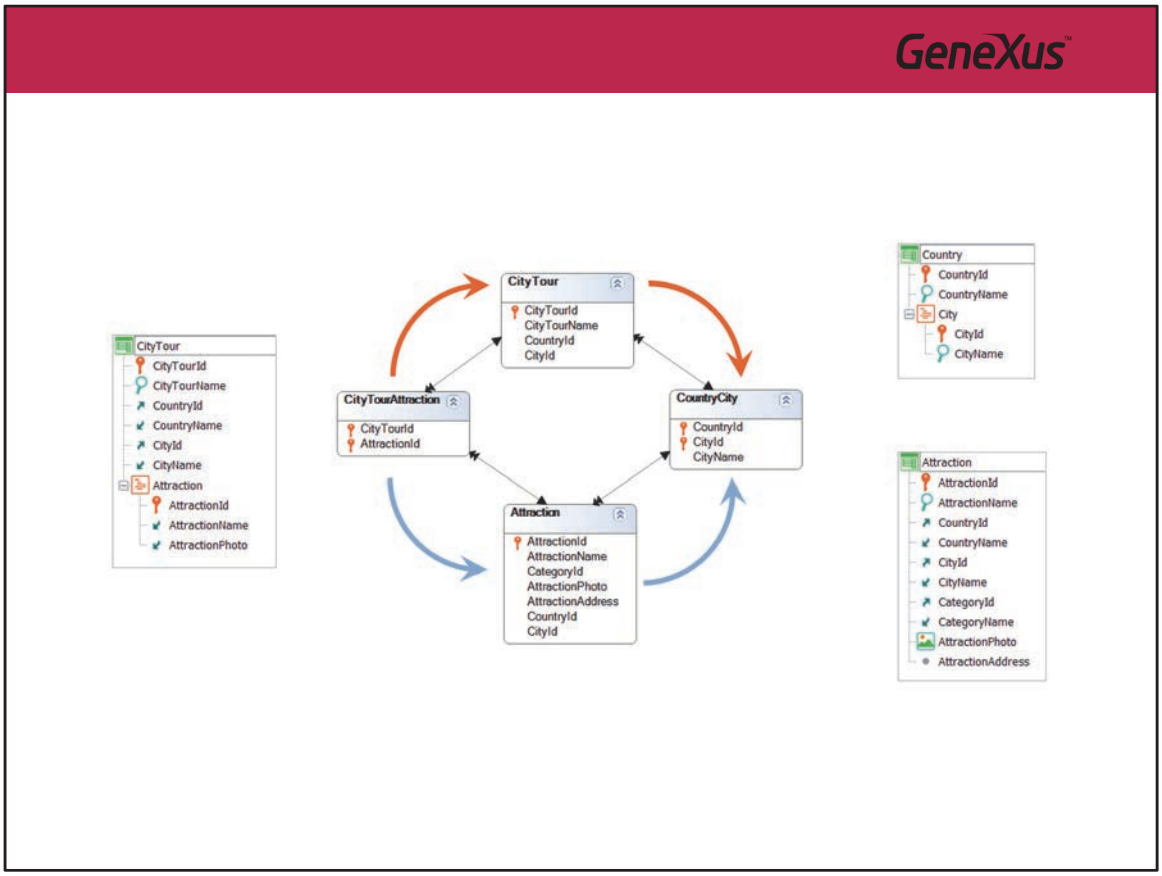
間接複数参照



旅行代理店の顧客に提供する、特定の都市の複数の観光名所を訪問するツアーを登録する必要があるとします。

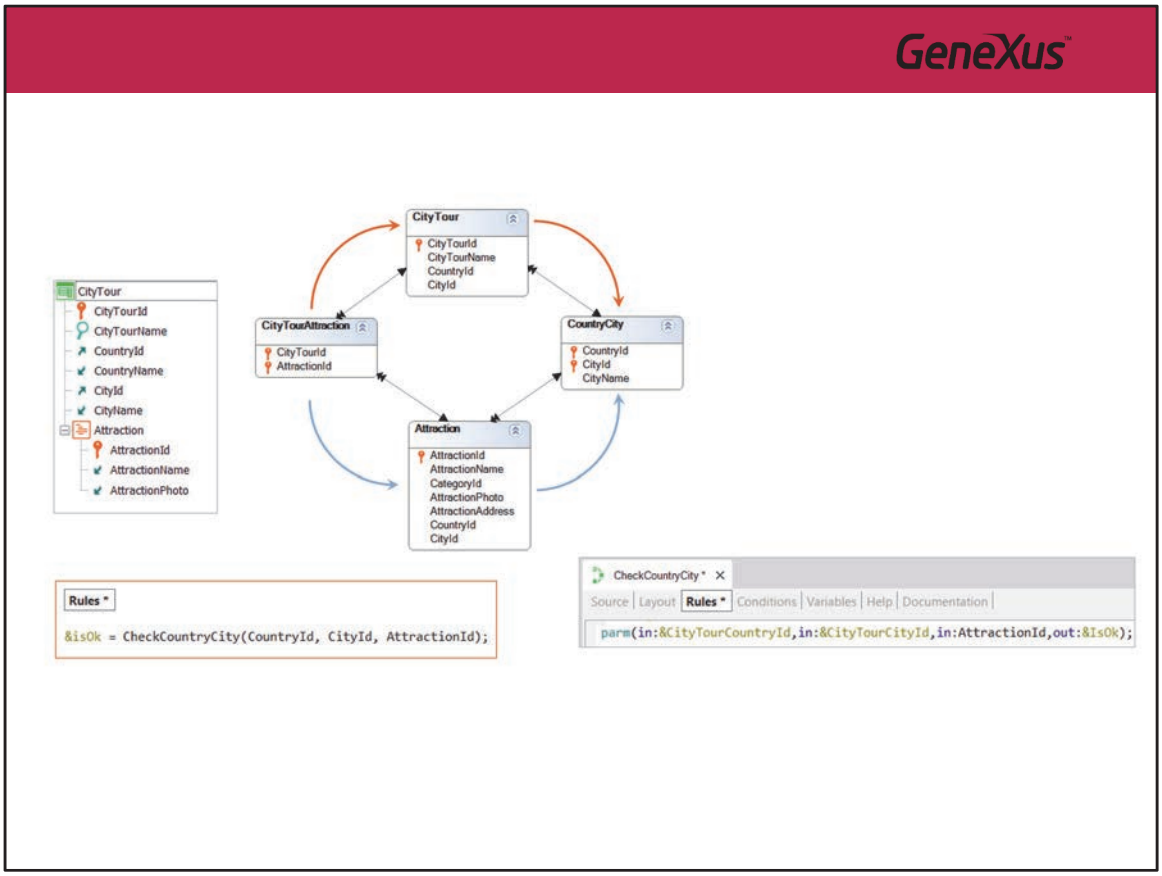
これを行うため、CityTour トランザクションを作成します。最初のレベルでは、ツアーの名前を登録するだけでなく、国および都市も指定します。2 番目のレベルでは、ツアー中に訪れる観光名所を指定します。

各観光名所には国および都市が定義されています。そのため、何もしない場合、ユーザーが該当する国や都市にはない観光名所を市内観光に入力する可能性があります。



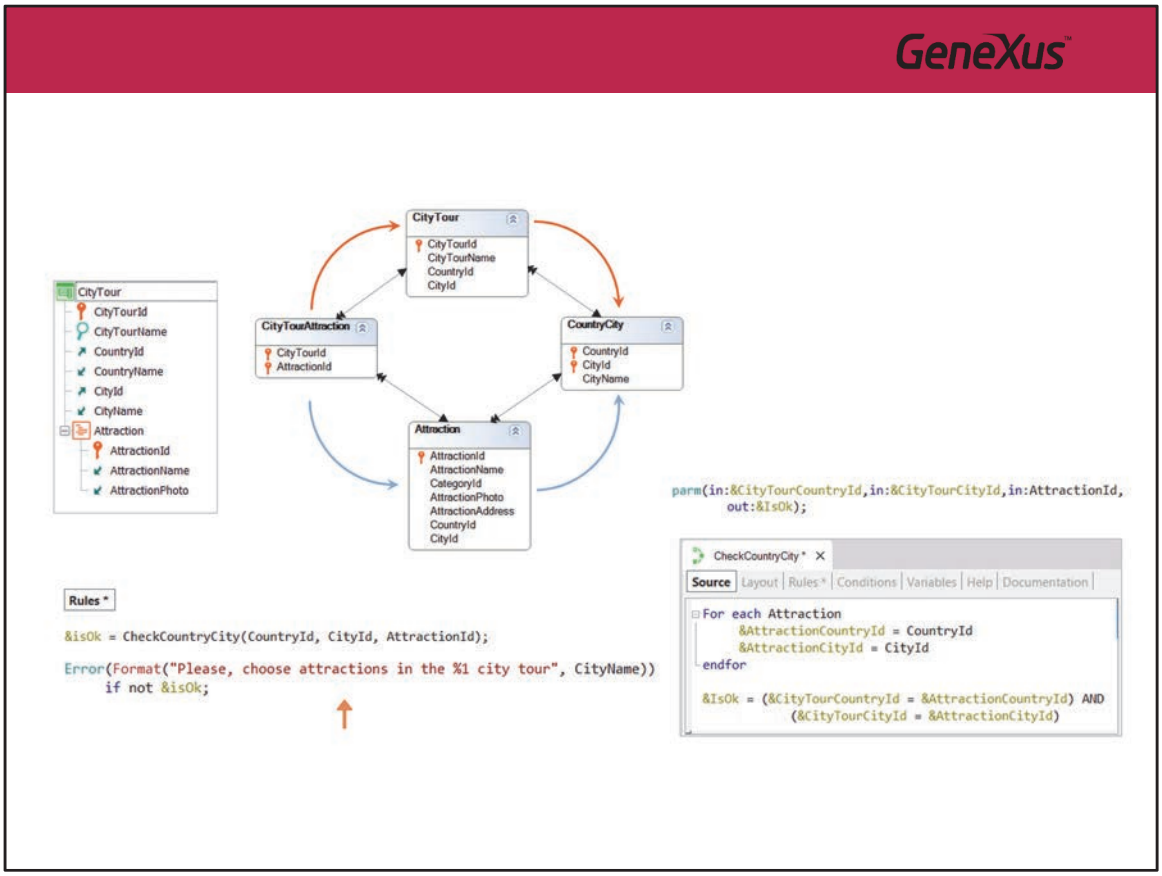
このテーブルダイアグラムを見ると、CityTour の 2 番目のレベルから都市のテーブルまで、2 つの異なる方法で到達できることがわかります。つまり、CityTourAttraction の拡張テーブルに都市のテーブル CountryCity がありますが、そこには 2 つの異なる方法で到達できます。CityTourAttraction レコードから開始すると、市内観光の都市が観光名所の都市と一致するかどうかわかりません。

市内観光を追加または変更するときに両方のパスを確実に一致させる場合は、サブタイプの使用は必須ではありません。



たとえば、CityTour トランザクションのルールでプロシージャーを呼び出し、パラメーターで項目属性 CountryId、CityId、および AttractionId を渡すとします。ここで行うのは、CountryId および CityId のペア (CityTour のペア) が、市内観光に追加する観光名所に基づいて Attraction レコードにアクセスする際に見つかるペアと一致するかどうかのチェックです。

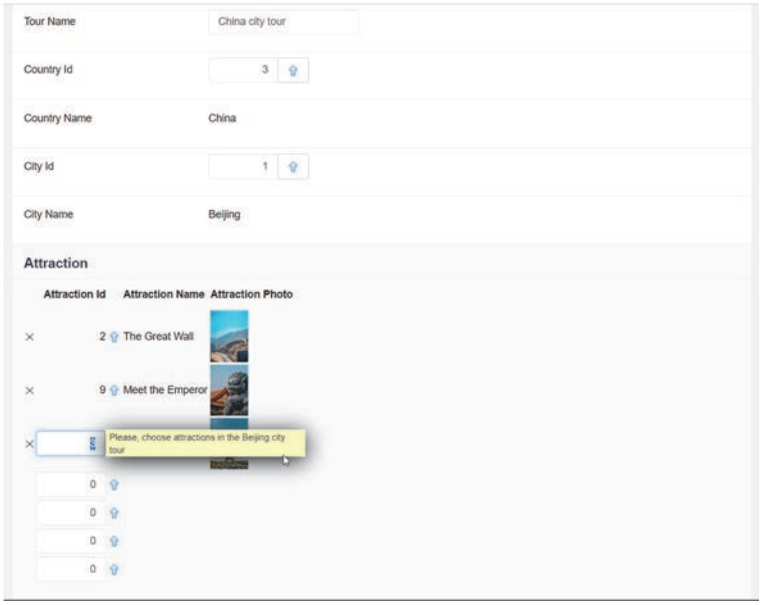
このプロシージャーは、変数で CityTour の国および都市の ID を受け取り、項目属性でチェックしたい行の AttractionId を受け取ります。また、国と都市が一致するかどうかを示すブール値を返します。



[Source] エレメントで Attraction トランザクションのテーブルにアクセスし、新しい 2 つの変数に、観光名所の国および都市をロードします (自動フィルタを使用)。

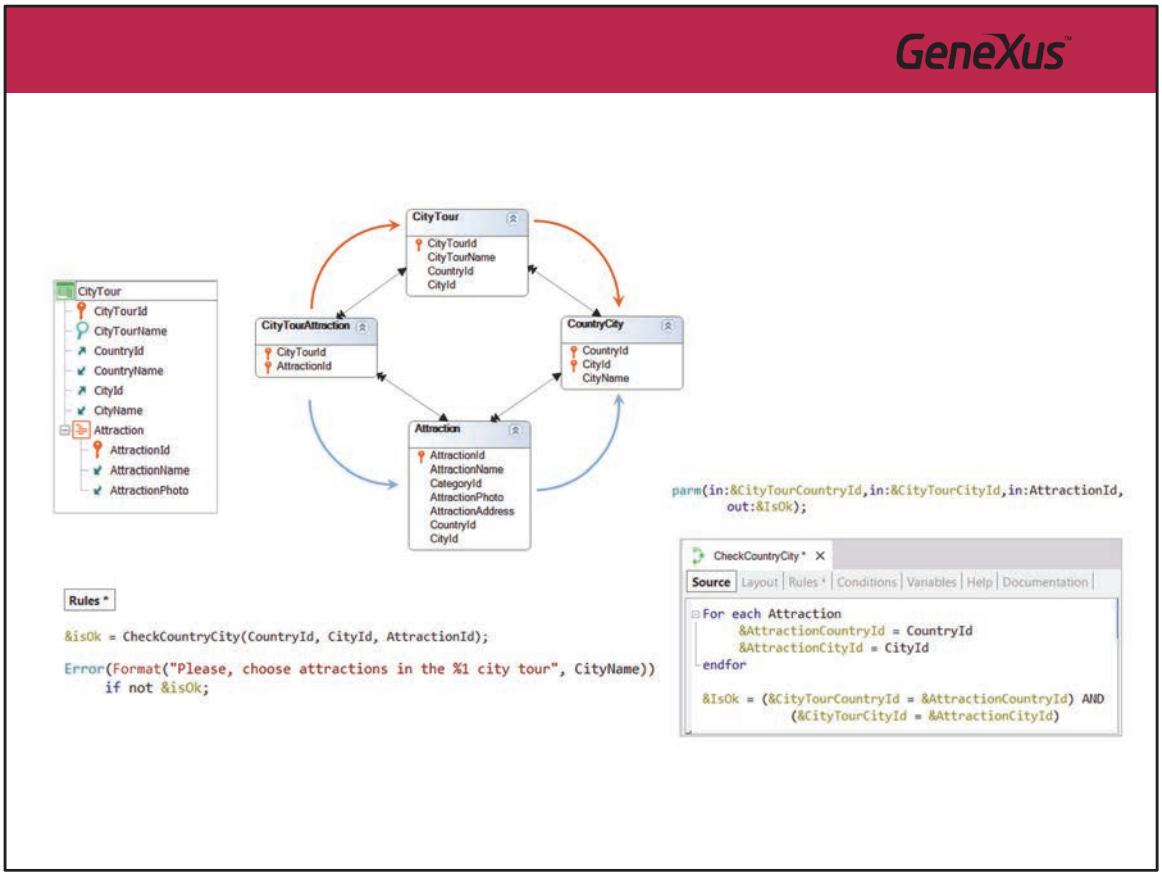
パラメーターで受け取った国名が観光名所の国名と一致し、パラメーターで受け取った都市名も観光名所の都市名と一致する場合は、ブール値 True が返されます。そうでない場合は False が返されます。

このようにトランザクションで条件を指定し、プロシージャーで False が返された場合には Error ルールをトリガーします。



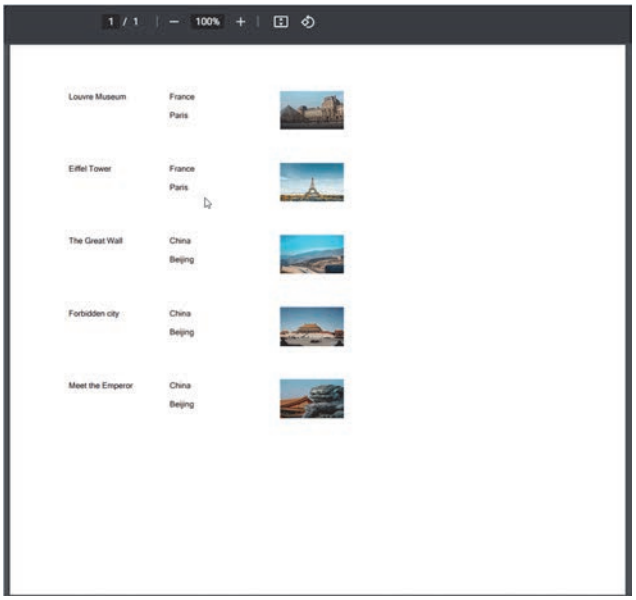
これを GeneXus で見てみましょう。
既にロードされた北京で行われる市内観光を使用してトランザクションを実行するとします。これには北京にある 2 つの観光名所が含まれます。ここで、エッフェル塔など、北京にはない別の観光名所を追加すると、エラーが正しくスローされることがわかります。

また、紫禁城など、北京にある観光名所を追加した場合は、問題なく保存できます。



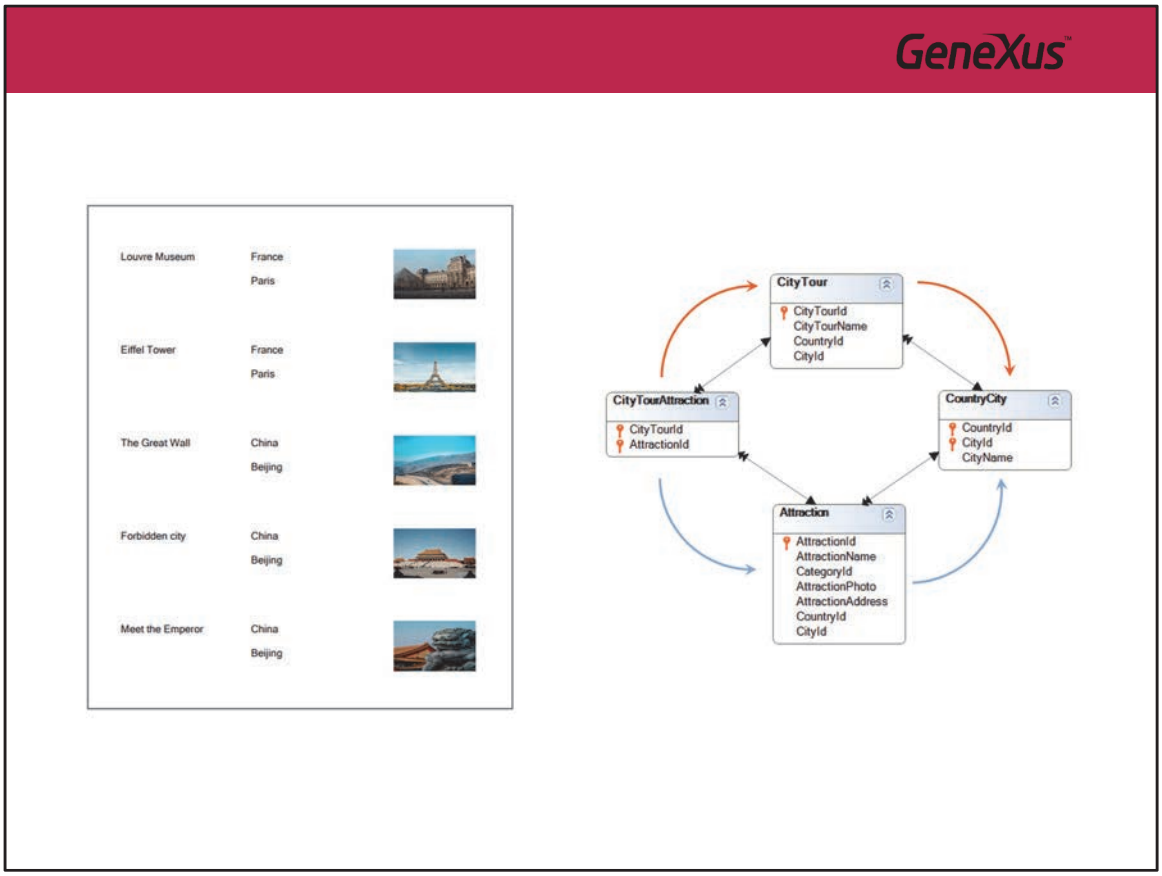
このように、トランザクションを介してこのチェックを行うときに、サブタイプを使用する必要はありませんでした。しかし、観光名所の CountryId および CityId 項目属性にあいまいさを排除してアクセスし、2 つの変数にそれらの項目属性を手動でロードして、市内観光の国や都市と混同されないようにするには、プロシージャーを呼び出す必要がありました。

CityTourAttraction テーブルのレコードに対して何かの処理を行うたびに、この問題が発生します。また、国および都市のペアを取得する必要もあります。これは、CityTour のペアか Attraction のペアかを確認するためです。



たとえば、CityTour の Attraction レベルのテーブルを読み込んで処理し、観光名所の名前、国名、都市名、写真を表示する必要があるとします。CountryName および CityName 項目属性が Attraction テーブルの CountryId および CityId から取得されるか、CityTour テーブルの CountryId および CityId から取得されるか、どのように確認できるでしょうか。ここにはあいまいさがあり、いずれかから取得することになります。どちらから取得するかを知るには、ナビゲーションリストを確認します。ナビゲーションリストでは、CityTourAttraction から CityTour にアクセスして国および都市の ID を取得し、Attraction にアクセスしてその他の項目属性、つまり写真および名前を取得することが示されます。Attraction の国および都市を表示するのではなく、CityTour の国および都市を表示することがわかります。

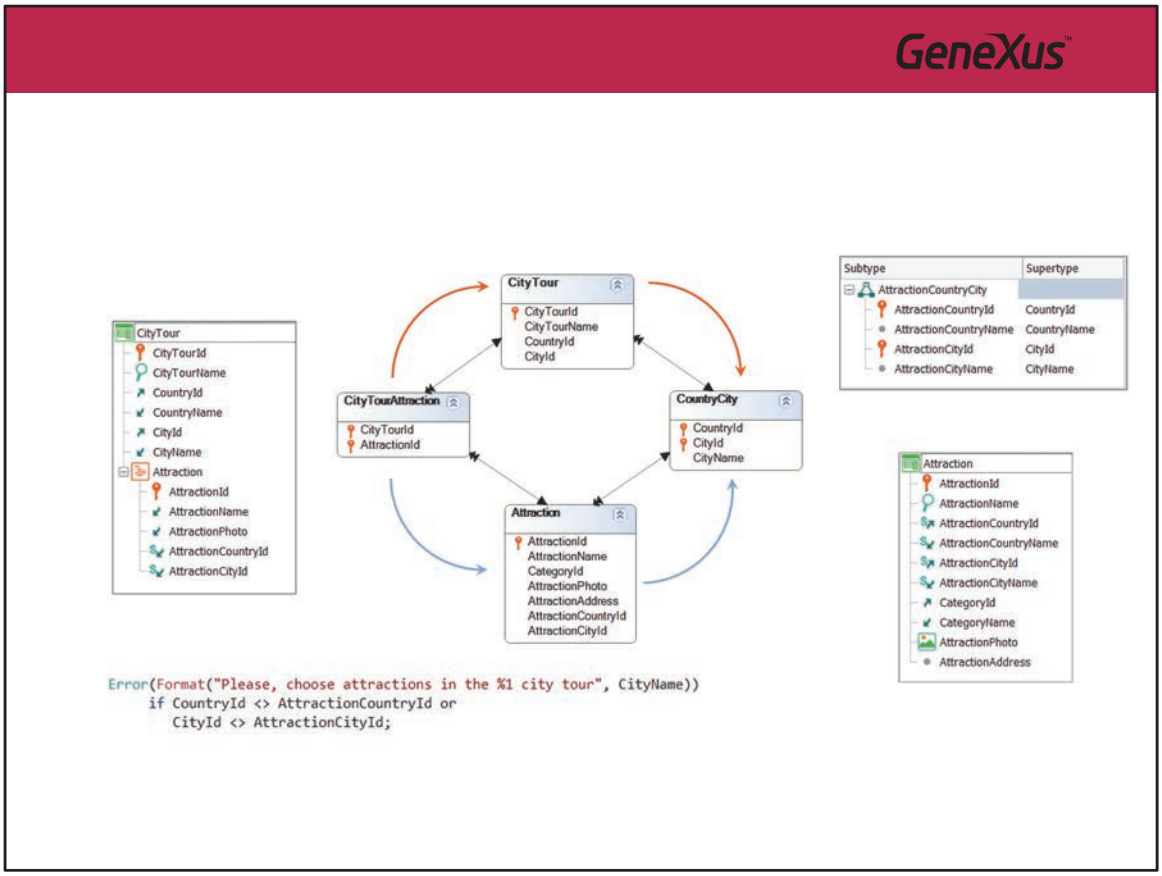
この場合は、観光名所が入力されるたびにこれらの値が一致するかどうかを確認するため、あいまいさの問題はありません。リストに市内観光の国および都市ではなく、観光名所の国および都市が表示されているのは、それが理由です。
しかし、このデータチェックを上書きし、たとえば、Attraction トランザクションでエッフェル塔の都市をパリではなくニースに変更しても、リストにはパリと表示されたままになります。これは、エッフェル塔が含まれる CityTour の情報であり、観光名所自体の情報ではないためです。CityTour トランザクションでのみ定義されており、変更は Attraction で行ったため、ルールを破ることができます。また、トランザクションを開いても、その行に対して何も行っていないため、エラーにはなりません。
そのため、このデータの変更が可能なあらゆる場所をチェックする必要があります。



あらゆる場所でチェックが行われ、両方のパスでデータが必ず一致すると確信できる場合は、データの取得にどちらのパスを選択するかは問題になりません。ただし、パフォーマンスの面では、どちらを選択するかが問題になる場合があります。市内観光に含まれる観光名所のリストを示すこの例では、Attraction にのみアクセスして CountryCity および Country を取得するほうが、Attraction にアクセスして名前および写真を取得してから CityTour にアクセスして CountryCity および Country を取得するよりも効率的です。

2 つのパスは同じではないため、特定のタイミングで 2 つのパスのいずれかを指し示す必要がある場合は、サブタイプを使用できます。

最初に明確な 2 つの可能性を分析し、最後に最も不明確な 1 つの可能性を分析して、3 つの可能性を分析します。



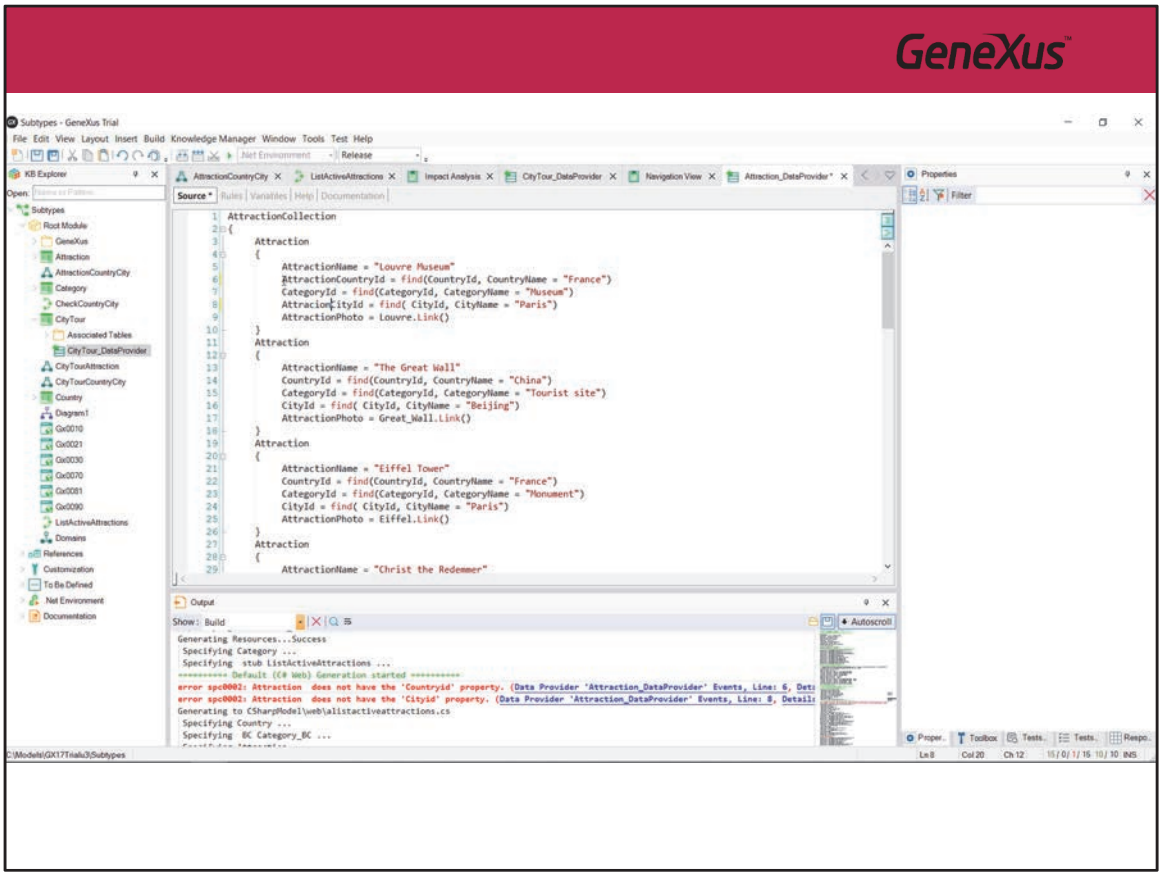
最初は、このパスの国および都市の項目属性の名前を変更して、識別可能にします。

これにより、観光名所の国および都市のサブタイプのグループを定義します。

このグループには 2 つの主項目属性があります。AttractionCountryId および AttractionCityId です。これらは、次のスーパータイプに従い、CountryCity テーブルの主キーに対応しています: {CountryId, CityId}。

これらのスーパータイプを Attraction トランザクション内のサブタイプで置き換えます。

これにより、下のパスが識別可能になります。
また、AttractionId から推論された国および都市の項目属性を CityTour トランザクションに追加して、Error ルールを通じてチェックを直接実装することも可能です。



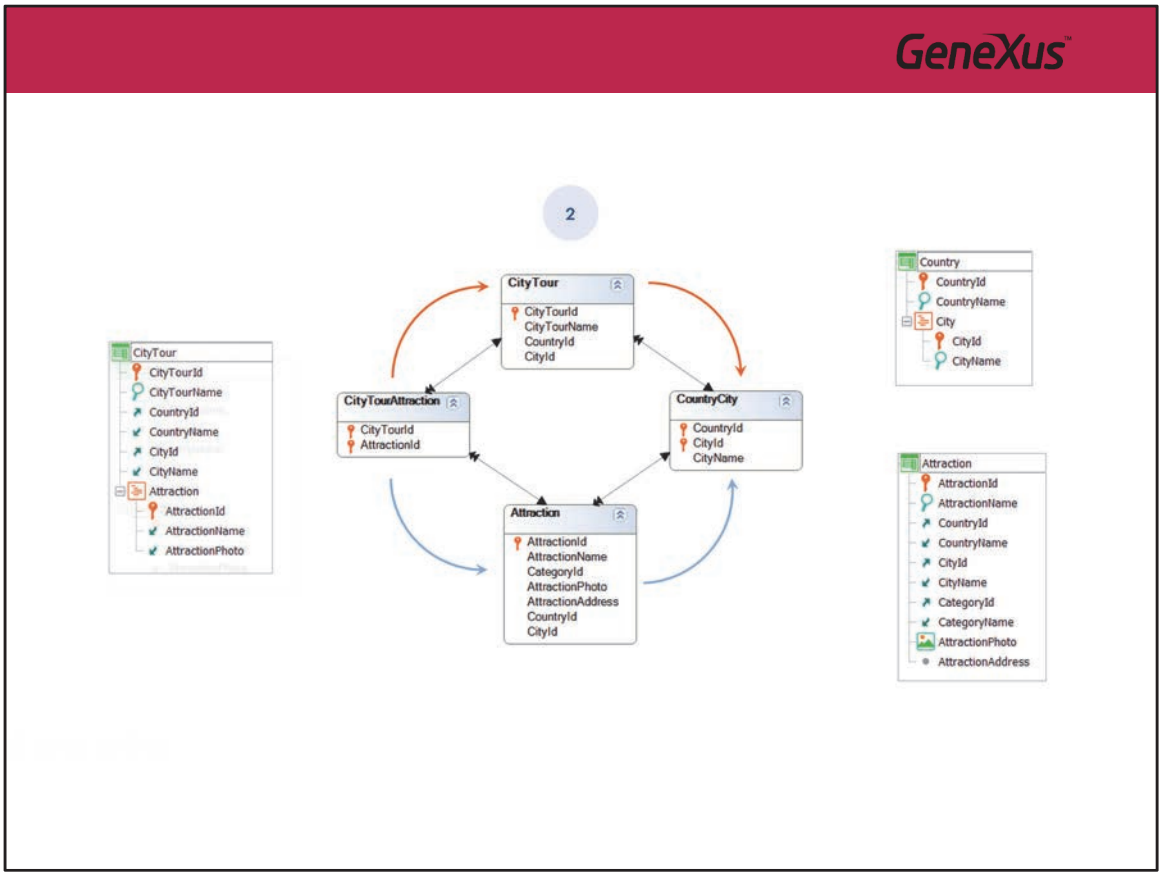
これらの変更を行った後は、前に見たリストのあいまいさを容易に解消できます。ここでサブタイプの CountryName および CityName を変更するだけです。

ただし、テーブル内に既にデータが設定されているため、特に Attraction テーブルについて、再編成する必要があると示されます。新しい項目属性 (サブタイプ) を配置し、古い項目属性 (スーパータイプ) を削除する必要があります。古い項目属性 (スーパータイプ) から、新しい項目属性 (サブタイプ) へと、データが適切に変換されるように見えます。
しかし、仕様解析完了後に再編成を行うと、エラーが発見されます。特に、データプロバイダーでテーブルに観光名所のデータを設定するように示されます。前からあるデータプロバイダーでは CountryId 項目属性が使用されていますが、この項目属性は Attraction にはもう存在しません。CityId も同様です。このソリューションのデメリットはこの点です。前に Attraction テーブルにアクセスしていたオブジェクトすべてで、古い項目属性を新しい項目属性に 1 つずつ置き換える必要があります。

CityTour を考慮せずにアプリケーションの開発を既に開始していた場合、あいまいさの問題に気付かずに、Attraction の CountryId および CityId にアクセスするほかの多くのオブジェクトを設定している可能性があります。

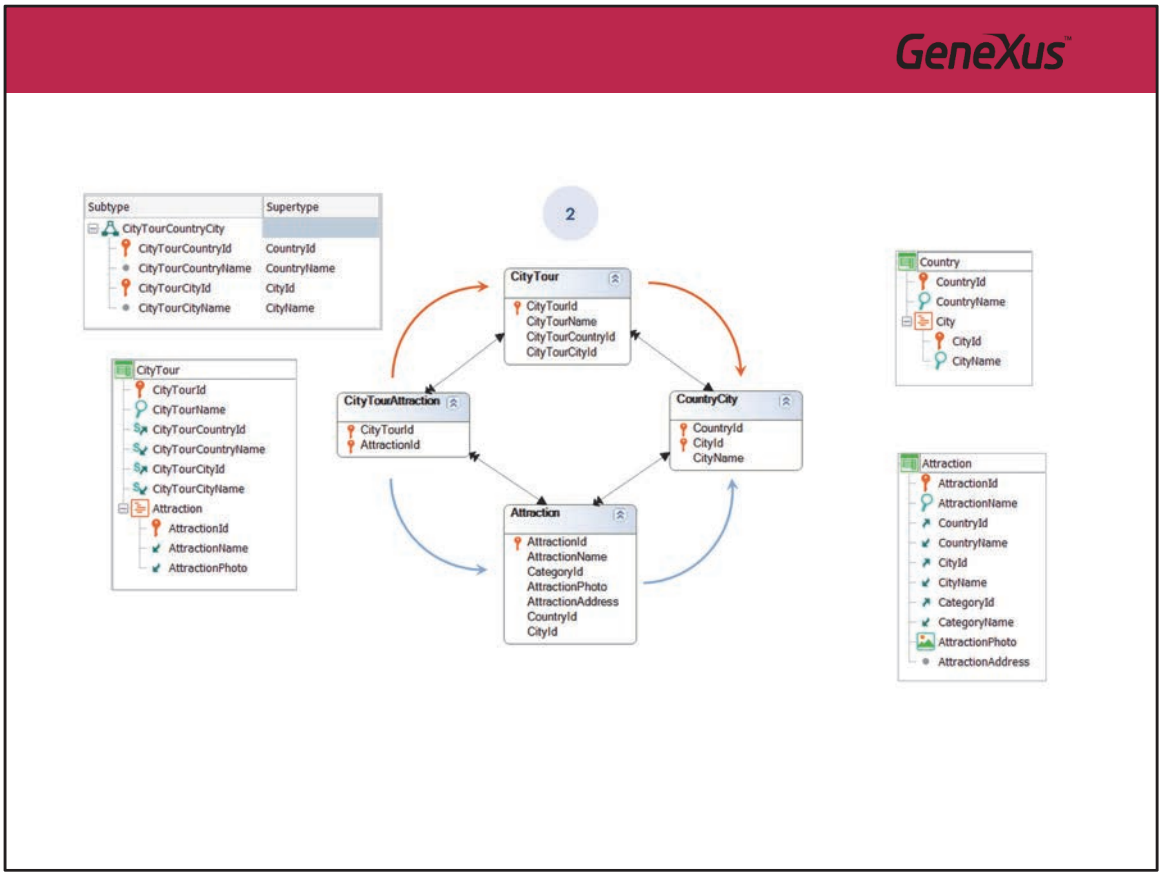
このソリューションを選択した場合は、これらの問題をすべて解決する必要があります。

この時点でナビゲーションリストを見ると、CityTourAttraction の各レコード用に国および都市を取得するために、希望に応じて Attraction テーブルが使用されています。あいまいさを解消し、希望するパスを明示的に選択しています。



次に、既に存在していたオブジェクトに関して、より複雑でない別の方法を考えてみます。

この 2 番目のソリューションでは、別のパスで国および都市の項目属性の名前を変更することであいまいさを解消します。

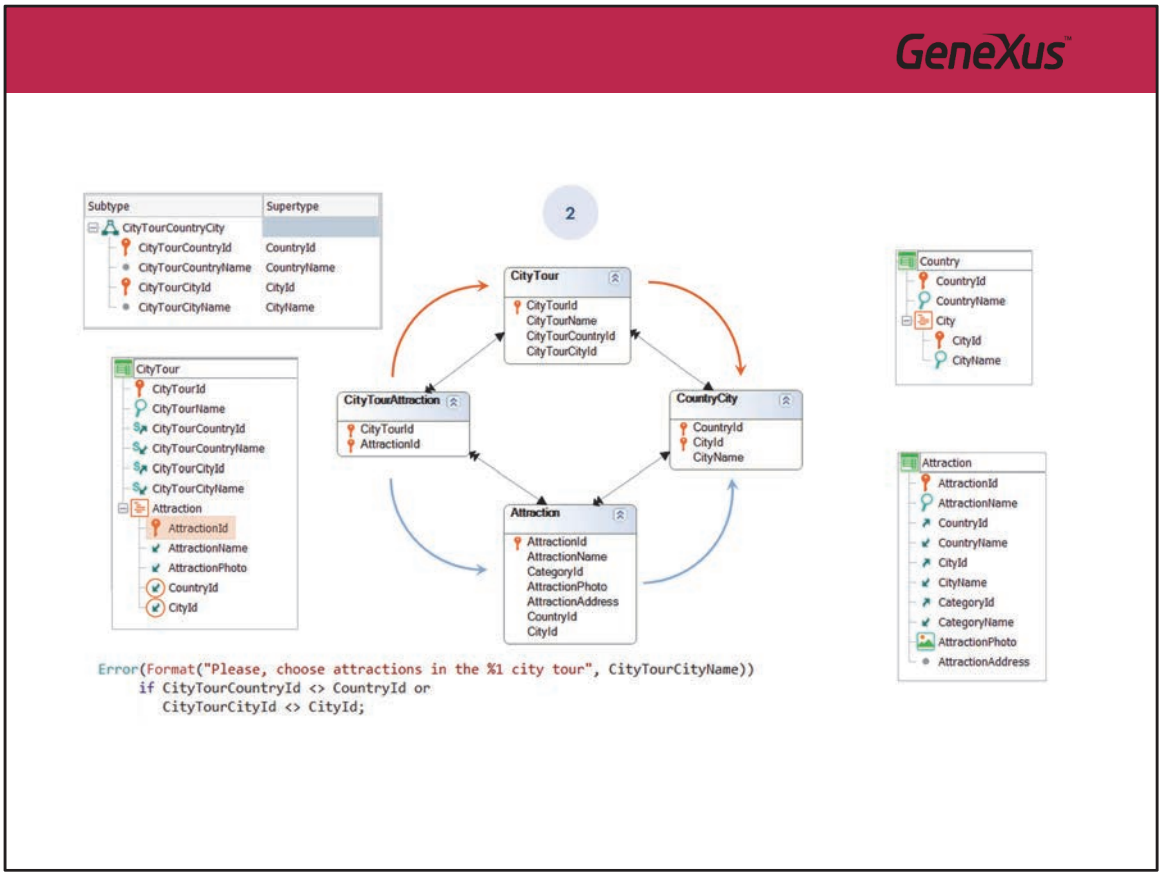


これを行うには、国および都市のサブタイプグループを定義し、CityTour トランザクションのヘッダーでそれを直接使用します。それにより、CityTour テーブルが変更されます (スーパータイプからサブタイプへの変更)。

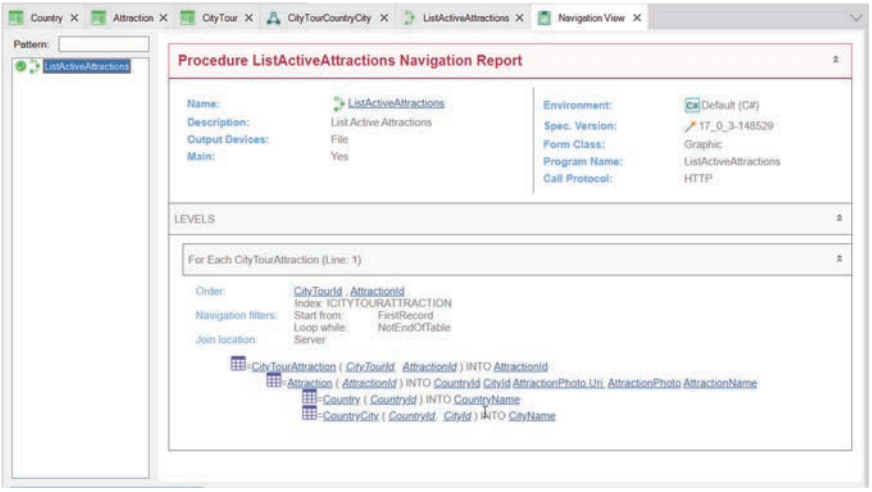
このソリューションと前のソリューションの違いは、2 番目のレベルがない単層のトランザクションがまず構築されてしまう (CountryCity への 2 つのパスが設定される) 可能性が低くなることです。そのため、2 番目のレベルの追加を検討する前に、ほかのオブジェクトが CityTour をナビゲートし、最初のレベルで項目属性をサブタイプに変更する必要が生じる可能性を考える必要はありません。

つまり、まず CityTour を作成し、それにデータをロードしてから (2 つのパスとこのソリューションを生成する) 2 番目のレベルも必要だと気付くのではなく、CityTour テーブルおよび CityTourAttraction テーブルが同時に作成されると考えられます。なぜなら、1 番目のレベルにスーパータイプではなく、サブタイプを利用しているためです。

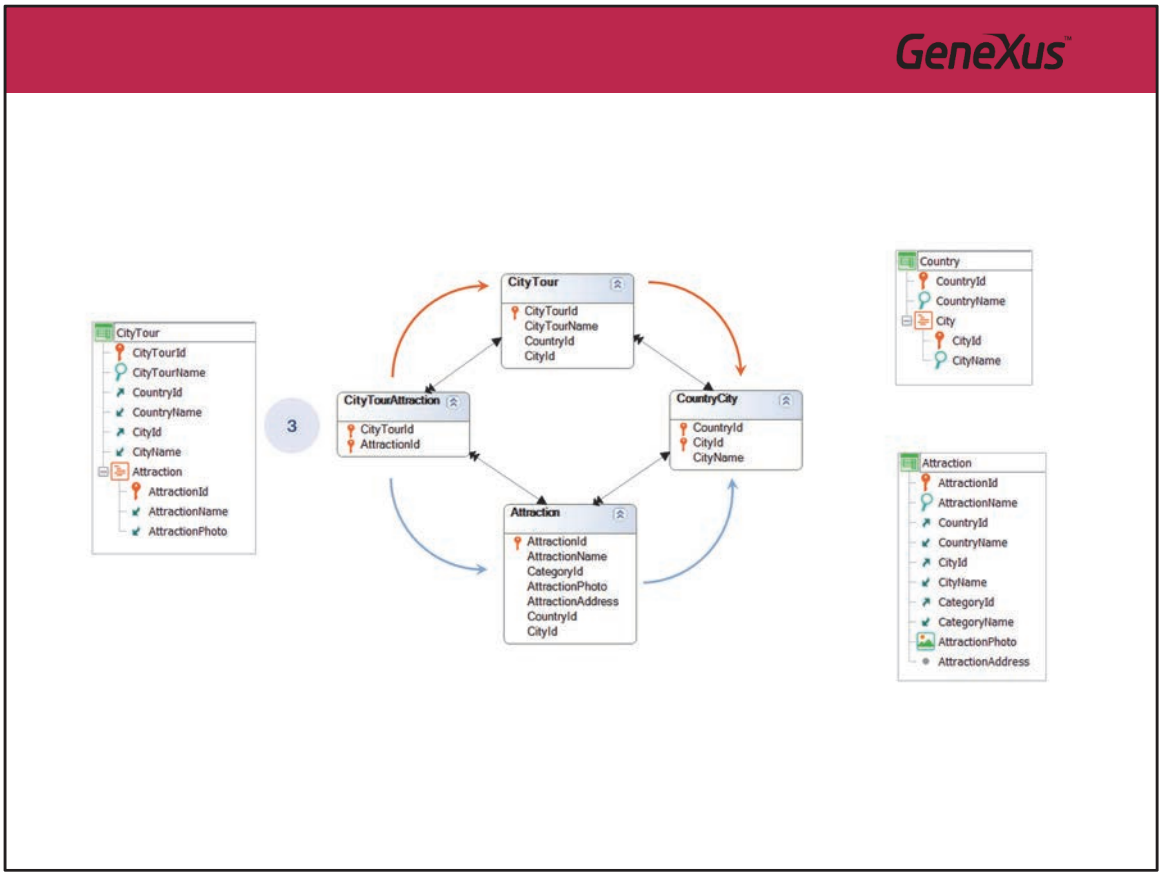
このソリューションでは、ここに示すパスが識別可能になります。そのため、



たとえば、プロシージャーを使用する必要なく、Error ルールを使用して一致のチェックを直接実装できるようになります。
これを行うには、CountryId および CityId を構造に追加して、それらをルールで使用できるようにする必要があります。これらは AttractionId から推論されるものであることに注意してください。

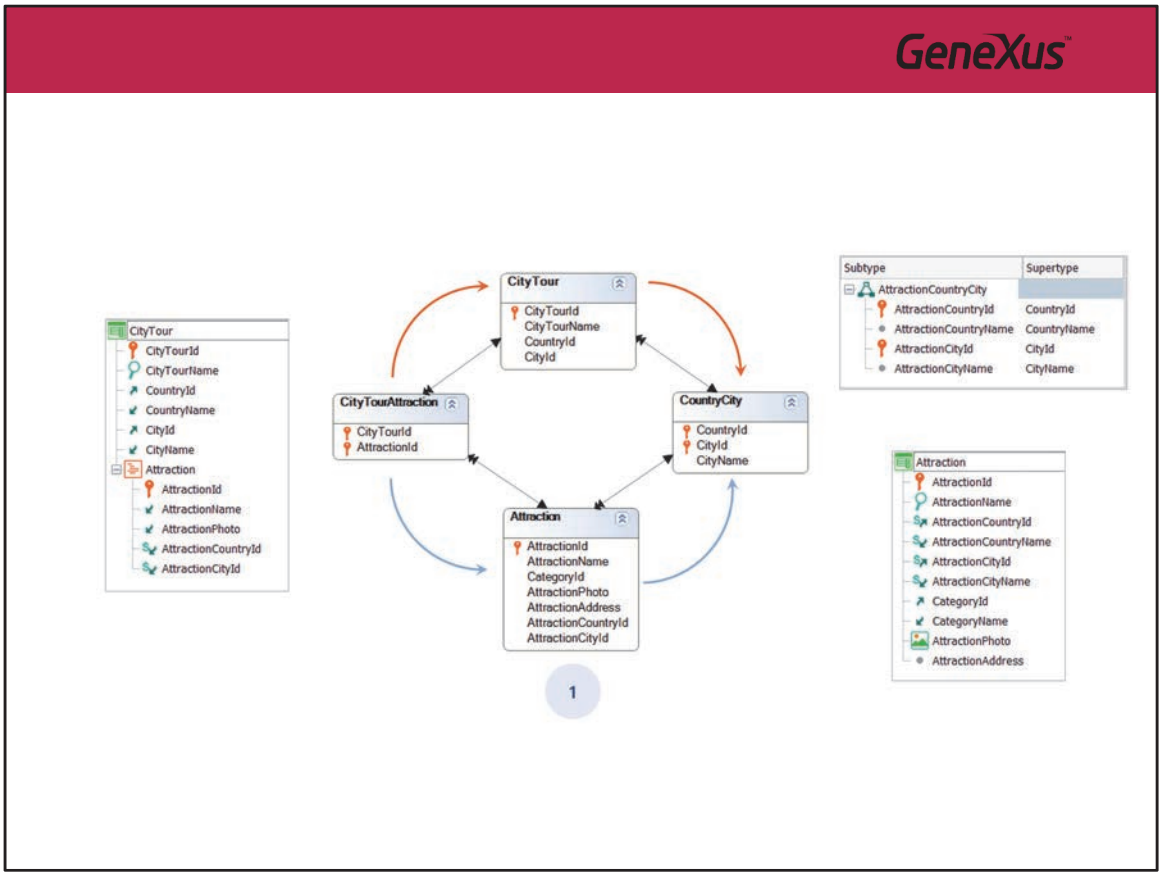


ソリューションが GeneXus で実装されたため、分析しているリスト内にスーパータイプ CountryName および CityName を残すと、Attraction テーブルの CountryId および CityId から取得されます。CityTour の別のパスは使用されなくなります。ナビゲーションリストでこれを確認してみましょう。リストでは想定したとおりに示されます。あいまいさは解消されています。

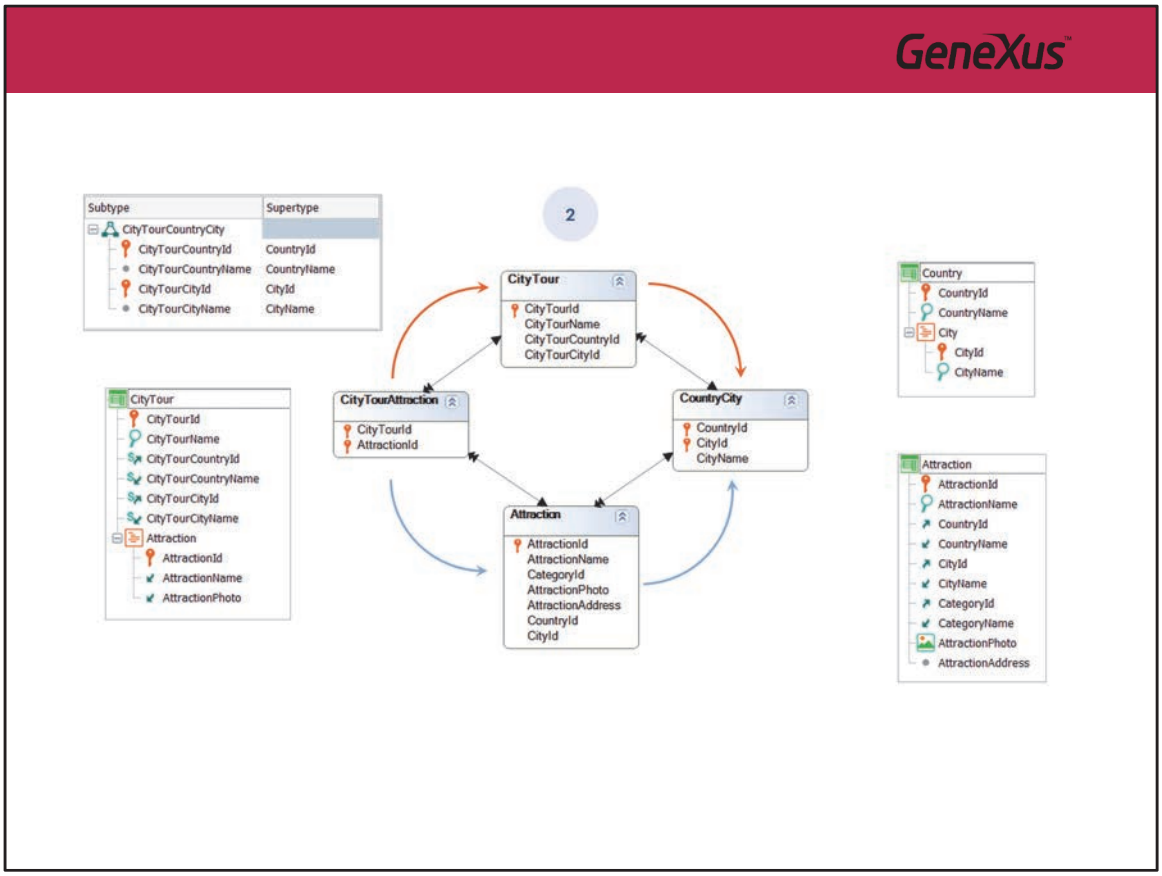


次に、サブタイプを使う最後の方法について説明します。あまり直観的ではありませんが、あいまいさが発生するテーブル自体であいまいさを解消するという、大きなメリットがあります。つまり、2 つのパスが発生するテーブルでの話です。

前のソリューションでは、テーブル内で Country および City 項目属性の名前を変更して、その拡張テーブルであいまいさが生じないようにしました。

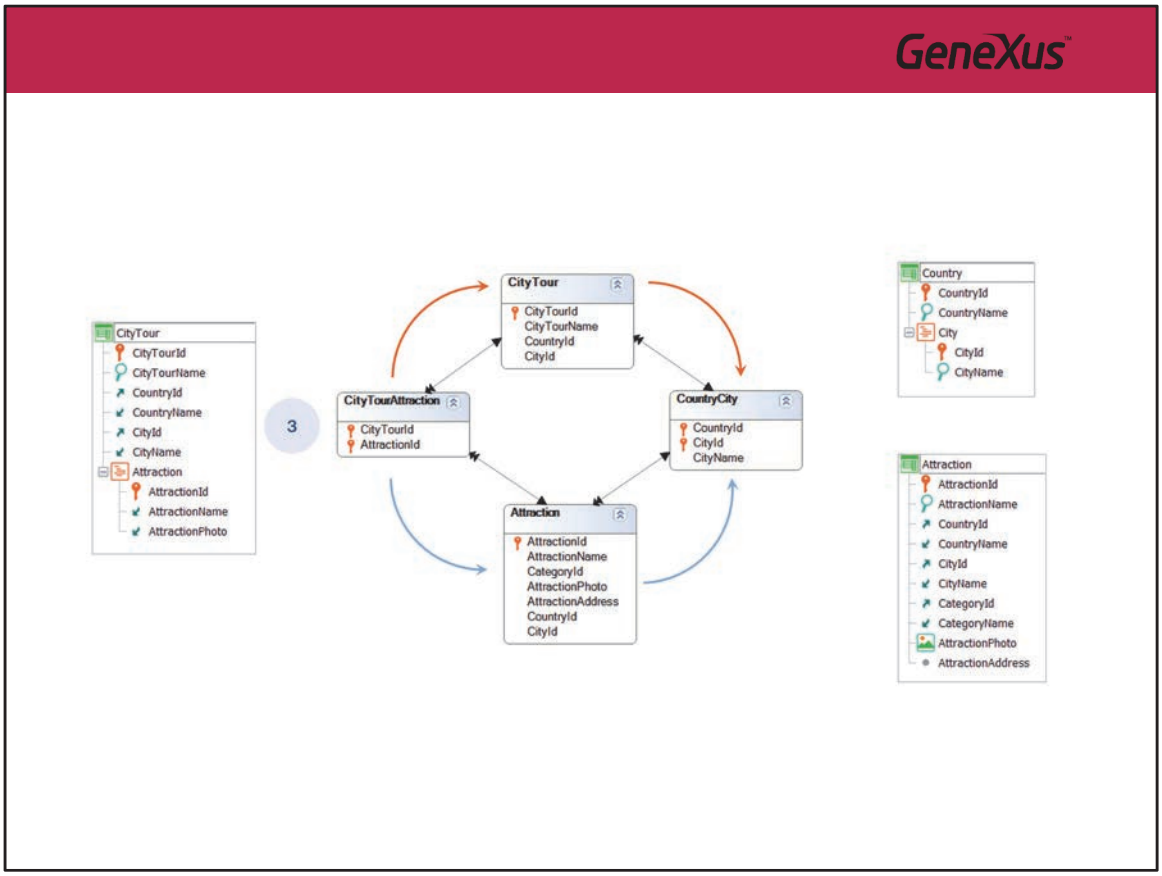


ソリューション 1 で、たとえば観光名所を国および都市の情報とともに表示する Web パネルを開発したいと考える場合、テーブルにスーパータイプではなくサブタイプが設定されている理由を理解できません。Attraction から見ると、それらを定義する必要はありません。



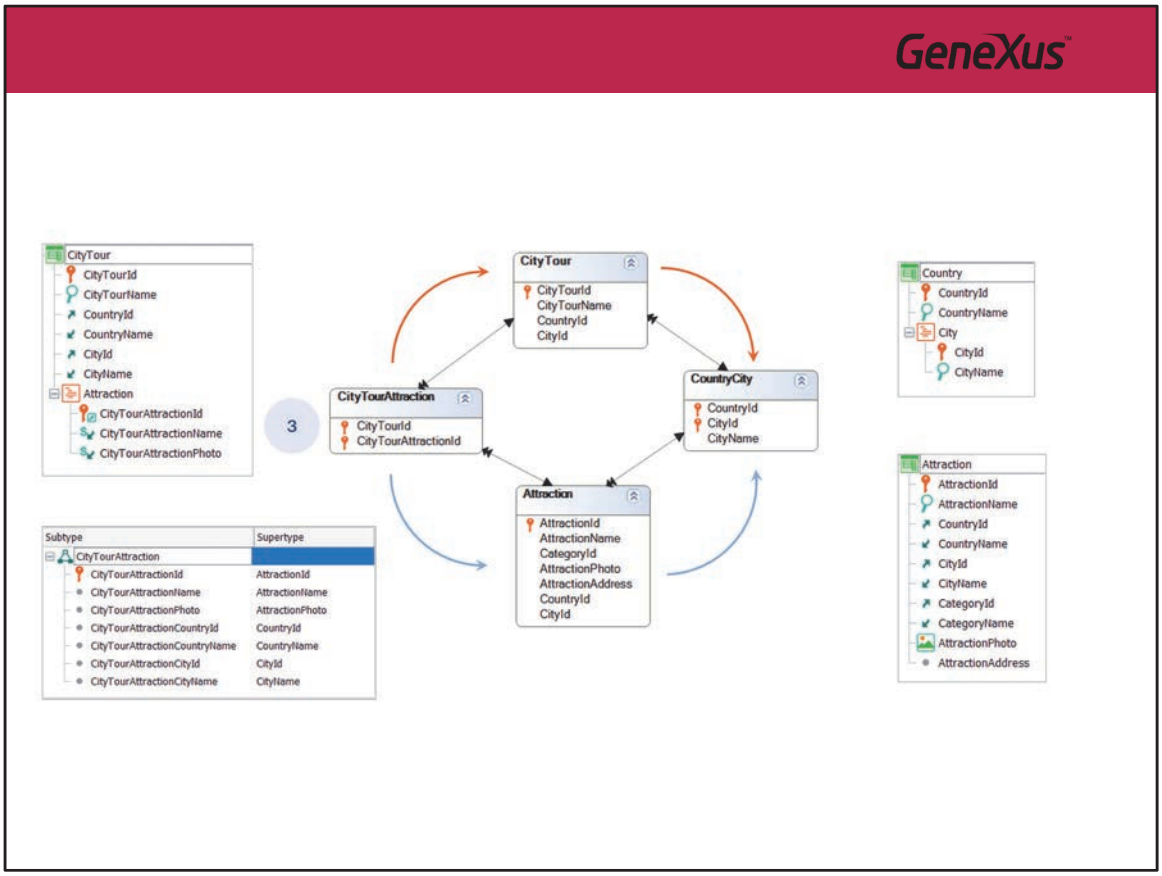
ソリューション 2 で CityTour に基づいて考えた場合も同様です。

市内観光をベーステーブルにある国および都市とともにリストしたい場合、サブタイプが使用されている理由を理解できません。しかし、この場合、CityTourAttraction テーブルは CityTour テーブルを生成するトランザクションの 2 番目のレベルから取得されるため、より関連性が高く、これらのサブタイプを使用する理由がより明白です。



2 つのパスを生むテーブル内であいまいさを解消できるのはメリットです。少なくとも、あいまいさがこのテーブルに固有のものであることを考えると、ここにサブタイプがあるのは筋が通っていると言えます。

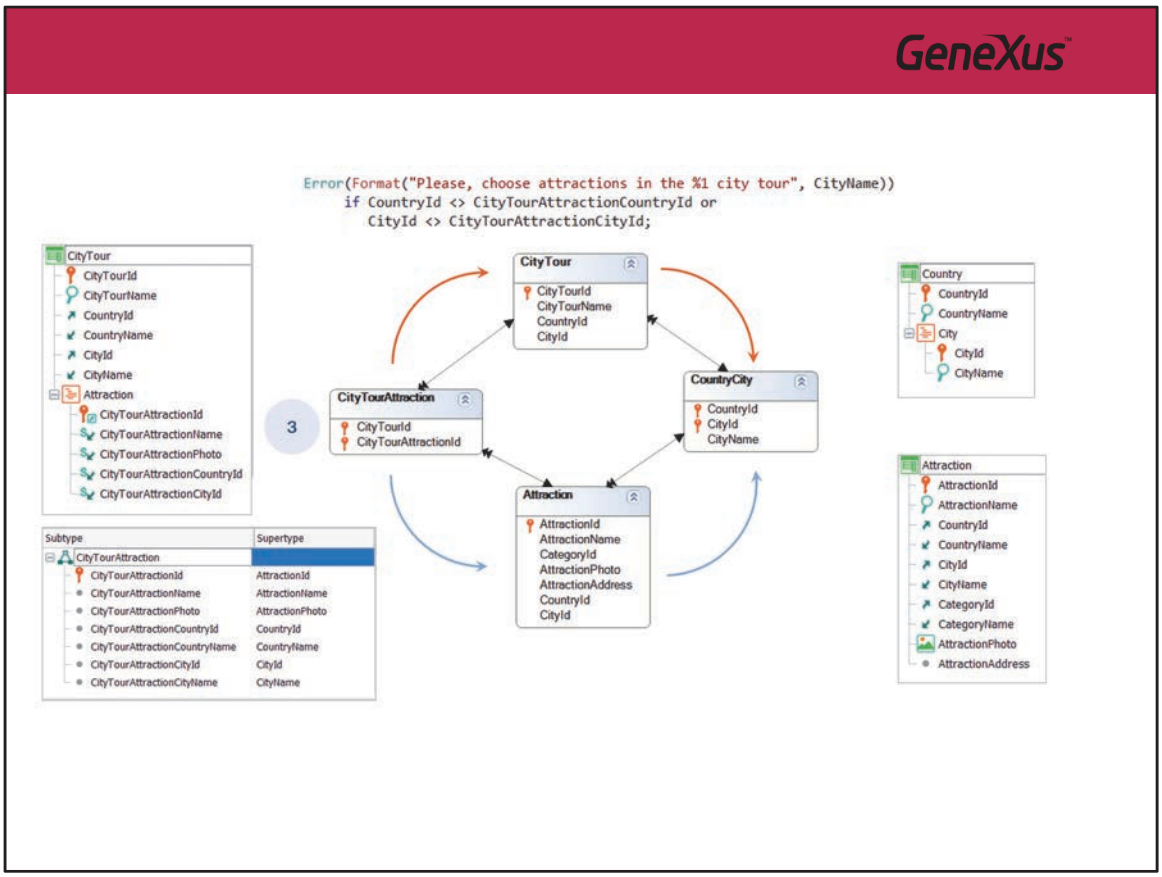
これら 2 つのパスが生じるテーブルであいまいさを解消するには、どうしたらよいでしょうか。この例の CityTourAttraction テーブル自体で考えてみましょう。



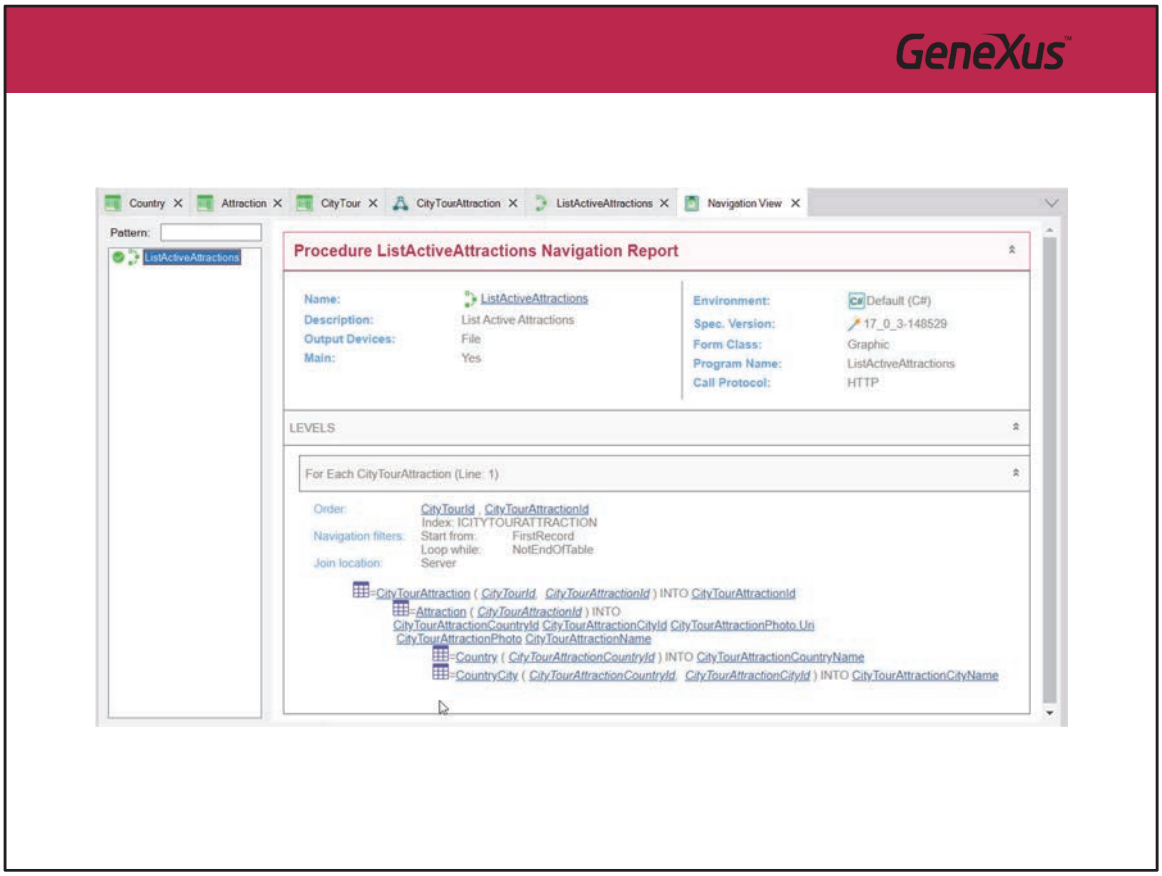
サブタイプのグループを定義して、外部キーとして表示される AttractionId の名前およびそれから推論されるすべての項目属性を変更できるようにするとします。

また、CityTour で、AttractionId 項目属性ではなく、そのサブタイプを使用するとします。もちろん、観光名所の名前および写真も、そのグループのサブタイプで推論する必要があります。

これを行うと、テーブルダイアグラムはこのようになります。ただし、ほかのテーブルはまったく変更せず、それらのテーブルにアクセスしていたオブジェクトすべてが引き続き問題なく使用できるようにする必要があります。



市内観光の国および都市が観光名所の国および都市と同一であることを確認するには、推論された 2 つの項目属性 CityTourAttractionCountryId および CityTourAttractionCityId をトランザクション構造に追加し、ここに示すように Error ルールを記述すれば済みます。

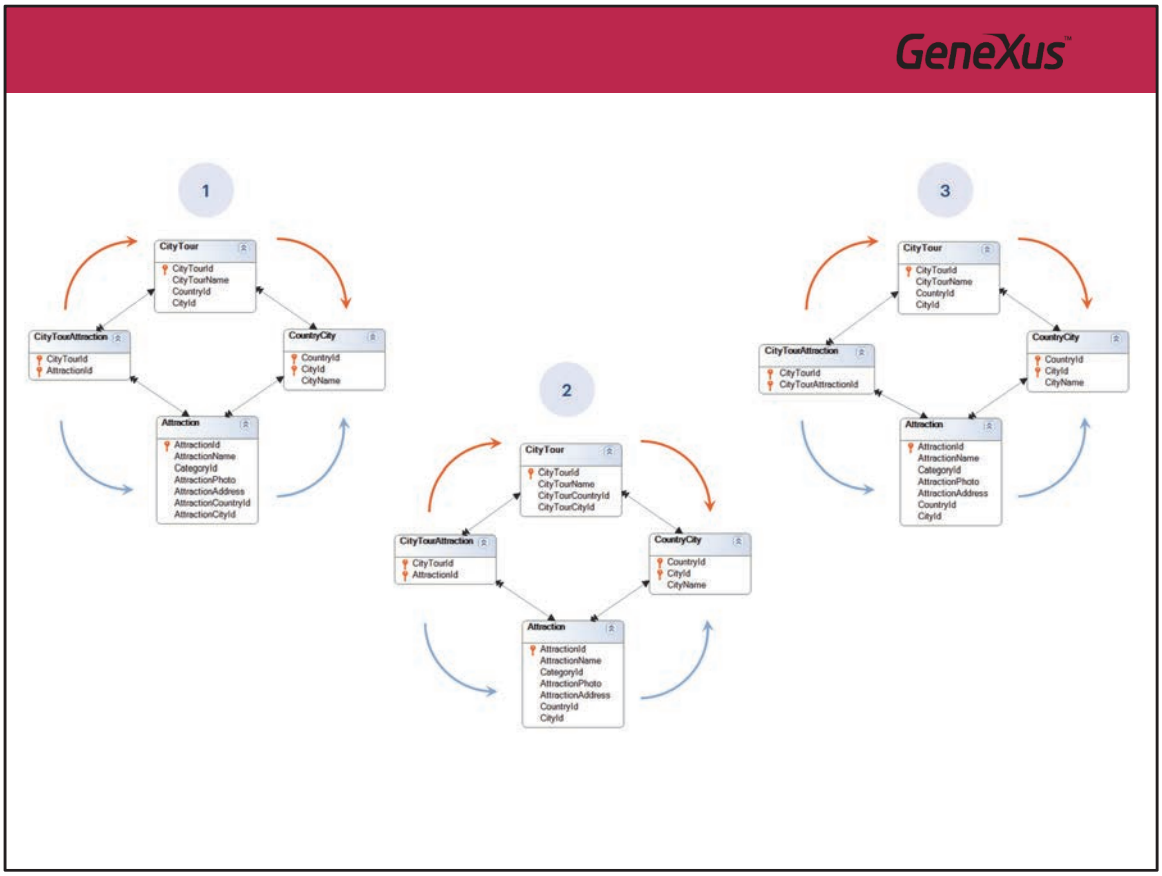


このソリューションでは、最初からあいまいさを解消する必要があるリストが非常に明白です。
For each を使用して、CityTour トランザクションの Attraction レベルを読み込んで処理します。ここでは、すべて観光名所から推論された項目属性を変更する必要がありますが、AttractionId ではなく、そのサブタイプが使われます。すべての項目属性をそのサブタイプに変更します。特に、国名と都市名です。しかし、ナビゲーションリストを見ると、これら 2 つの項目属性に到達できないことがわかります。

なぜでしょうか。グループ内にこれら 2 つのサブタイプは定義したが、For each で読み込んで処理するトランザクションのレベルでそれらを指定していなかったためです。それらを追加し (推論されたものであるため、悪影響は生じない)、リストを再度ナビゲートすると、問題がなくなったことがわかります。

ナビゲートしたテーブル内のサブタイプから情報を正しく取得し、Attraction テーブルにアクセスして、そこから CountryCity および Country に到達していることが示されます。

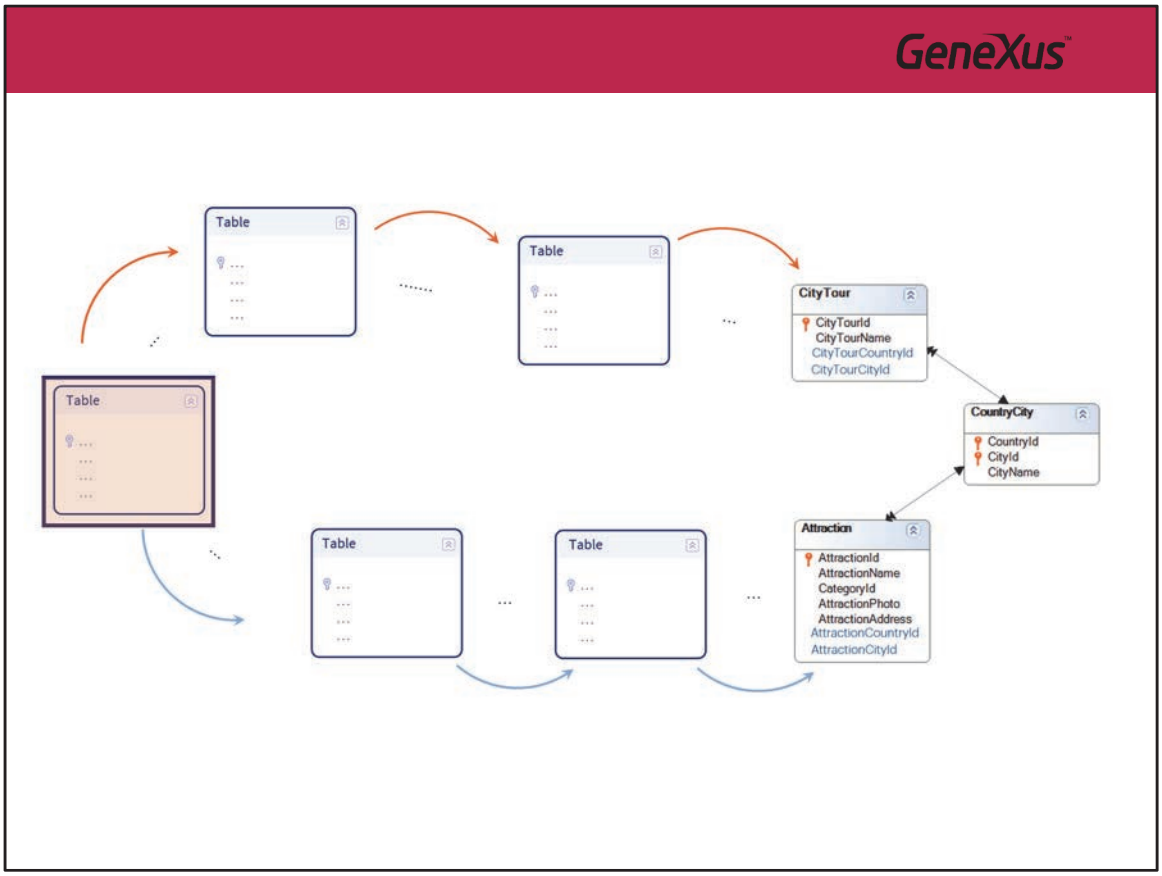
しかし、ナビゲーションで使用したい推論された情報すべてをトランザクション構造に毎回追加する必要があるのは、少し面倒かもしれません。



どのソリューションにも、そのソリューションを適用するユースケースによっては、メリットとデメリットがあります。

たとえば、すべてのトランザクションを一度に作成すれば、以前から古い項目属性を使用しているほかのオブジェクトについての問題はなくなります。ソリューション 1 および 2 は、その意味では問題がなくなります。しかし、それだけでは、サブタイプの使用を正当化することはできません。

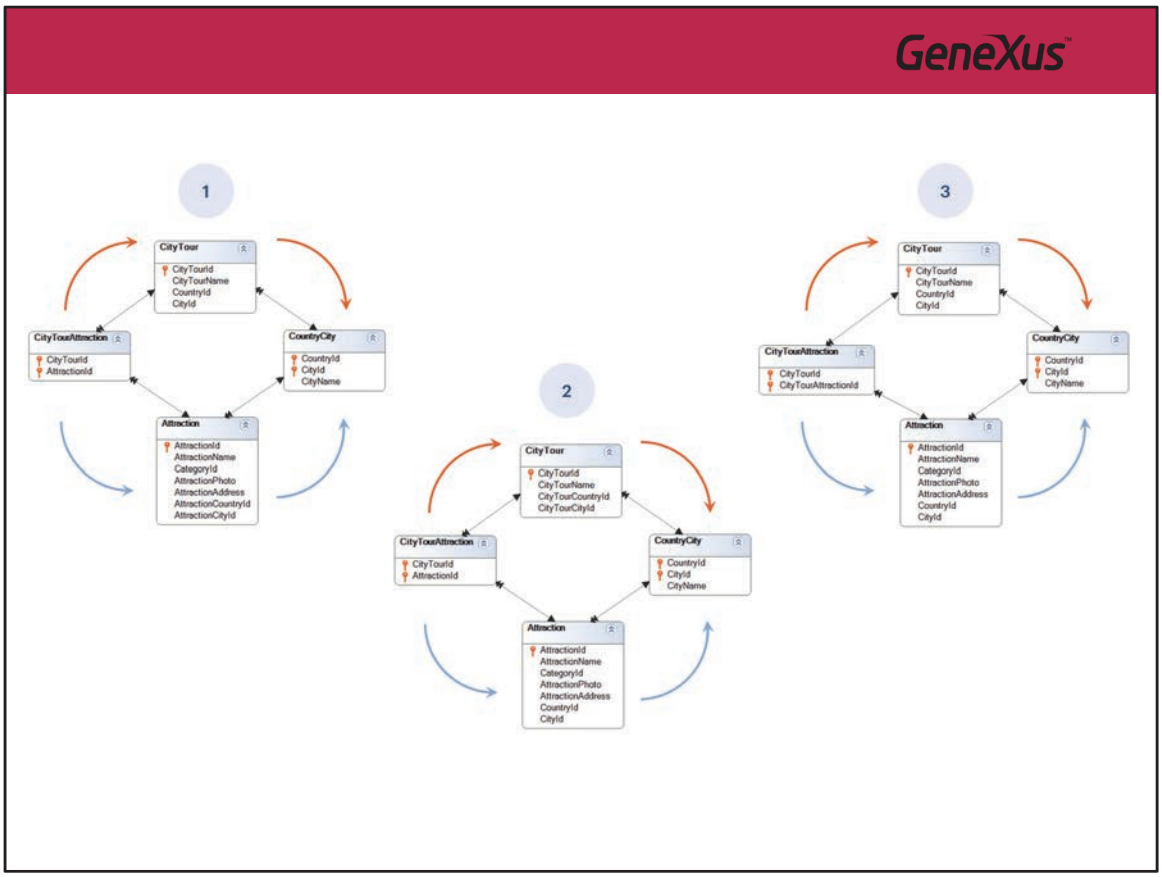
この場合があまり大きな問題にならないのは、ソリューション 1 の場合、Attraction から直接上位にあるテーブルを見るだけで理由がわかるからです。CityTour から見た場合、ソリューション 2 についても同じことが言えます。



しかし、ソリューション 1 を使用して Attraction で CountryId および CityId のサブタイプを定義した場合に、パスのあいまいさの原因となるテーブルが遠くにあると、より混乱を招きます。

または、ソリューション 2 で CityTour から見た場合も同じようになります。これらのサブタイプの働きを理解することが簡単ではなくなります。

一方で、あいまいさのあるテーブルでサブタイプが作成され、外部キーの 1 つがサブタイプとして定義された場合は、非常に明白になります。拡張テーブルを確認するだけで、複数のパスから到達可能なテーブルを見つけることができます。



これが 3 番目のソリューションです。このソリューションのデメリットは、外部キーサブタイプから取得する項目属性を使用する必要があるオブジェクトでは、当然ながら、サブタイプグループにそれらを追加する必要があることです。サブタイプグループでは推論された項目属性としてマークされますが、トランザクション構造に追加する必要もあります。

また、あいまいさが Country および City によって生じたものであることが、一見したところではわかりません。拡張テーブルを分析しないと、サブタイプグループのどの項目属性で定義されたかがわかりません。たとえば、CategoryId であったかもしれません。

ここに 3 つのソリューションが示されています。開発者は状況に応じた最適なソリューションを選択します。

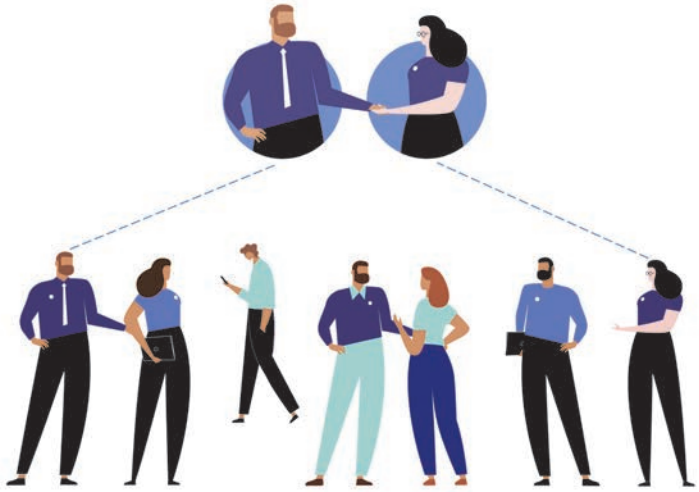
GeneXus

再帰的サブタイプ

次に、サブタイプの別のユースケースを見てみましょう。これは再帰的サブタイプと呼ばれるもので、エンティティが自己参照である必要があります。

GeneXus

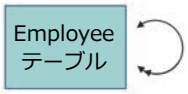
再帰的サブタイプ



このケースについて学習するために、旅行代理店の従業員の情報を表すことにしましょう。各従業員は、ほかの従業員 (1 人または複数の人) のマネージャである場合もあります。

GeneXus

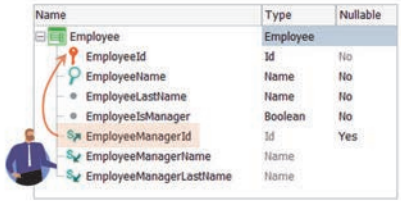
再帰的サブタイプ

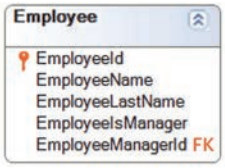


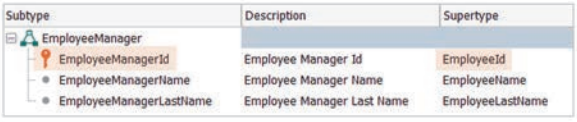
マネージャがいる従業員の場合は、誰がマネージャであるかを示す必要があります。このマネージャは従業員です。そのため、それ自体との関係性が従業員テーブルで確立されます。

GeneXus

再帰的サブタイプ



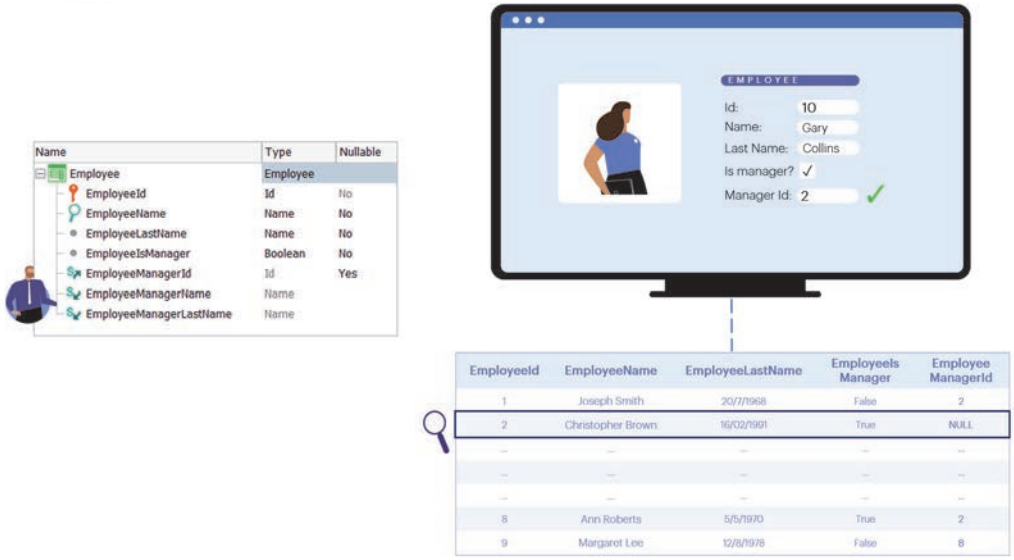




これを解決するには、その従業員のマネージャに関する情報を表すサブタイプグループを作成する必要があります。

EmployeeManagerId 項目属性は、EmployeeId として使用されるため、Employee テーブル自体に対する外部キーを形成します。

再帰的サブタイプ



そのため、従業員の情報をトランザクションを通じて入力する際に、ユーザーがEmployeeManagerId フィールドの値を選択すると、GeneXus で参照整合性がチェックされます。つまり、EmployeeId 項目属性にその値が設定されたレコードが従業員テーブル内に存在することが確認されます。

これまで説明したサブタイプのさまざまなユースケースを使用する必要がある実際の状況を考え、GeneXus で実装することをお勧めします。