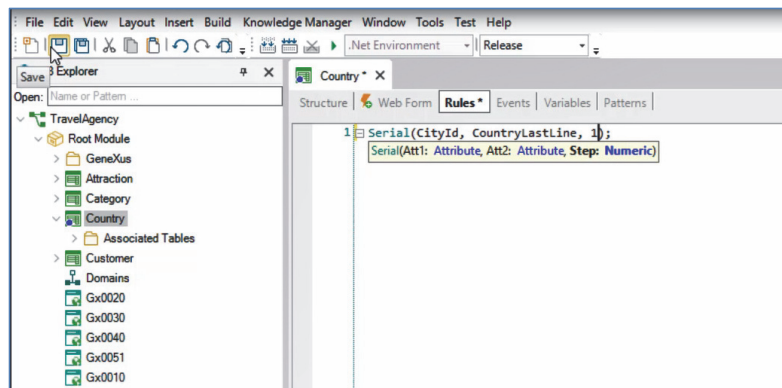


## 【補足】 Serialルール

## Serial ルール



Country トランザクションで生じる状況について、簡単に考えてみましょう。前の章で、このトランザクションに City という名前の第 2 レベルを作成したとき、GeneXus によってデータベース内に CountryCity テーブルが生成されました。既に見たように、このテーブルの主キーには、2 つの項目属性 CountryId および CityId があります。

CountryId と CityId のどちらの項目属性にも ID ドメインが割り当てられており、ID ドメインは自動採番されるように定義されています。

テーブルが作成されたとき、GeneXus によって警告がトリガーされ、CityId 項目属性の [Autonumber] プロパティが True に設定されていても無視されることが示されました。

なぜそうなるのでしょうか。

これは、[Autonumber] プロパティは単純な主キー、つまり項目属性が 1 つのものに適用されるからです。この場合は複合主キーであり、CountryId および CityId の 2 つの項目属性があるため、[Autonumber] プロパティは適用されません。第 2 レベルを自動採番するためには、Serial ルールを使用します。

Country トランザクションの [Rules] エlementに移動し、Serial ルールを追加します。

最初のパラメーターは自動採番される項目属性です。CityId と入力します。

2 つ目のパラメーターも項目属性です。この項目属性は、自動採番する項目属性 (この場合は CityId) に割り当てた最後の値を保存するために使われます。ここで新しい項目属性を作成します。この項目属性は、採番するものよりも上のレベルにある必要があります。

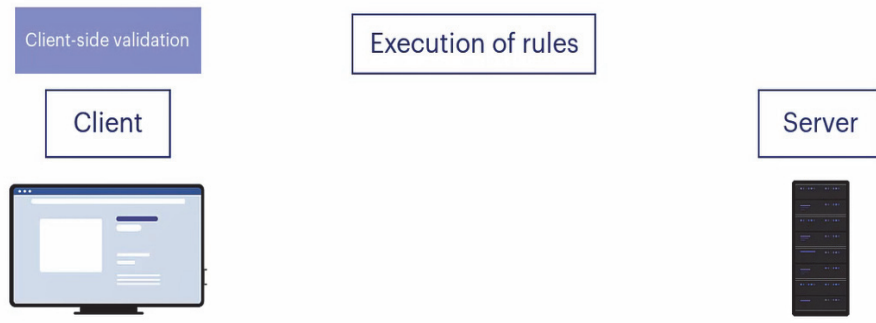
トランザクション構造に戻り、トランザクションの第 1 レベルに CountryLastLine という名前で項目属性を作成し、Numeric タイプにします。

変更を保存できるようにするために、[Rules] Elementに入力したコードの一部をコメントにします。コードは未完成なので、コメントにしないと保存できません。保存してからコメントを解除し、2 つ目のパラメーターとして CountryLastLine 項目属性を入力します。

ルールの最後となる 3 つ目のパラメーターは、自動採番の増分を示す数値です。「1」と入力します。こうすると、CityId の採番では数値が 1 ずつ増えます。変更を保存してアプリケーションを実行します。データベースを再編成して CountryLastLine 項目属性を Country テーブルに追加するよう求められます。再編成を実行します。Country トランザクションを開き、Germany にアクセスします。City の ID の値は入力せず、名前だけを入力します。この場合は Berlin と入力します。City の ID を選択したら、Tab キーを押してフィールドを移動します。これにより、この都市に CityId の値 1 が割り当てられます。項目属性 CountryLastLine にも同じ値が設定されます。前に述べたとおり、この項目属性は、CityId に割り当てた最後の値を保存します。

別の都市、Hamburg を入力します。先ほどと同様に、CityId フィールドに何も入力しないまま Tab キーを押して移動すると、ID に自動的に値が割り当てられます。今回は「2」が割り当てられ、CountryLastLine 項目属性にも同じ値が設定されます。もし 3 番目の都市を入力すると、作成した Serial ルールに基づいて、GeneXus は CountryLastLine 項目属性の値を取得し、それに増分値として定義した 1 を加えます。その結果の値が CityId および CountryLastLine に割り当てられます。4 番目以降の都市についても同様です。

【補足】 クライアントサイド検証と  
サーバサイド検証



これまで見てきたルールの動作を考えると、ルールはクライアントでのみ実行されるものだと思われるかもしれません。たとえば、顧客名の Error ルールの例では、フィールドを空にしたままユーザーがフィールドから離れると、直ちにメッセージがトリガーされました。しかし、ユーザーはほかのフィールドへの入力続けることができます。そして [実行] をクリックすると、トランザクションに関連するプログラムのサーバーで実行される部分が制御を担います。ルールが再度実行され、この場合は Error ルールであることから、レコードのデータベーステーブルへの保存が防止されます。

つまり、これまで見てきたルールは、クライアント (クライアント側の検証と呼ばれる) とサーバーの両方で検証されます。クライアントでのルールの実行を無効にすることはできますが、サーバーでのルールの実行を無効にすることはできません。データベースで何が行われるかを実際に制御するのはサーバーです。ブラウザーは過酷な環境であることに注意が必要です。ブラウザーは攻撃の対象となることがあり、改ざんされた情報をサーバーに送信する可能性があります。サーバーがデータについて最終的な制御を行える必要があります。

クライアントでのルールは、優れたユーザーエクスペリエンスを提供し、アプリケーションが常に動作しているとユーザーが感じることができるようにするために実行されます。

そのため、フロントエンド (クライアント上でサーバーに代わってステータスおよびビジネスロジックを維持する部分) で実行されるプログラムとバックエンド (ビジネスロジックの検証を行う部分) で実行されるプログラムの両方に、同じルールを含めることができます。

この点について理解することが、後ほど説明を進めて、ルールによって制御できる対象がより複雑になったときに重要になってきます。



動画	<a href="https://www.genexus.com/community-and-support-jp/training?ja">https://www.genexus.com/community-and-support-jp/training?ja</a>
ドキュメント	<a href="http://wiki.genexus.jp/">http://wiki.genexus.jp/</a>
認定資格	<a href="https://training.genexus.com/certifications">training.genexus.com/certifications</a>