トランザクションの整合性

GeneXus™

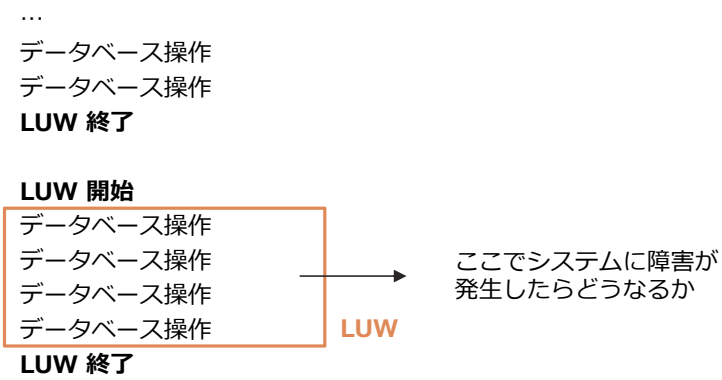
GeneXus

データベース管理システム

多くのデータベース管理システム (DBMS) は障害回復システムを備えており、停電やシステム障害など、予期しない事象が発生したときにデータベースの一貫性を保つことができます。

GeneXus

作業論理単位 (LUW)



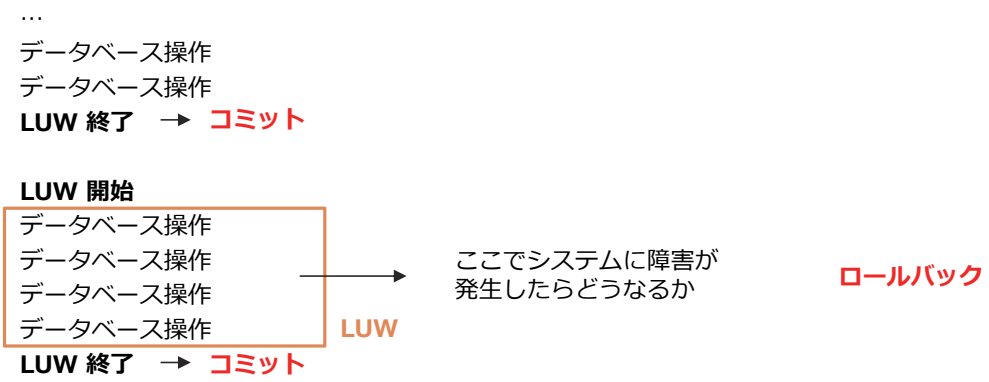
トランザクションの整合性を保つことができるデータベース管理システム (DBMS) では、作業論理単位 (LUW) を構成できます。
作業論理単位はデータベースの「トランザクション」の概念そのものに相当します。

基本的には LUW は一連のデータベース操作ですが、そのすべてが実行されるか、そのいずれも実行されません。つまり、LUW 内では一部の操作だけを実行することはできません。これによりデータベースの整合性が保証されます。

この例には、データベースに対する 4 つの操作からなる LUW があります。たとえば、挿入、更新または削除です。仮に、最初の 2 つの実行が成功したが、3 番目を実行する前にシステムに障害が発生したとします。その場合、LUW が完了していないため、完了した 2 つの操作を元に戻す必要があります。そうでない場合、作業論理単位は論理レベルでデータベースの整合性を定義するため、データベースの整合性が失われる可能性があります。

この場合、DBMS は回復のためにいわゆるロールバックを実行し、データベースで整合性があった最後の時点の状態を保持します。

作業論理単位 (LUW)



LUW の終了は Commit コマンドにより決定されます。
したがって、LUW は 2 つのコミットの間で実行される操作により定義されます。

図に示した場所でシステムに障害が発生した場合、最後のコミットの後で実行された 2 つの操作 (コミット保留中の操作) は、障害回復の際に DBMS で実行される自動ロールバックにより元に戻されます。

つまり、LUW に含まれるすべての操作が実行されたわけではないため、実行された一部の操作は元に戻されるか取り消されます。

GeneXus

GeneXus の LUW

トランザクション: 各インスタンスの最後、AfterComplete ルールの直前

プロシージャー: ソースの最後

ビジネスコンポーネント: GeneXus は Commit を記述しない

トランザクションおよびプロシージャーは、データベースの情報を更新するために作成される GeneXus オブジェクトです。このため、GeneXus は、定義されている言語でプログラムを生成するときに Commit コマンドを記述します。どこに記述するのでしょうか。

Transaction オブジェクトの場合は、各インスタンスの最後、AfterComplete トリガーイベントを持つルールの直前に記述します (つまり、ヘッダーおよび行が処理された後です)。
Procedure オブジェクトの場合は、ソースの最後に記述します。
トランザクションから作成されるビジネスコンポーネントの場合は、任意のオブジェクトで使用できるため Commit は含まれません。どこでコミットするか決めるのは開発者です。
これについては次で説明します。

GeneXus

トランザクションおよび自動コミット

ヘッダー

BeforeValidate

検証

AfterValidate / BeforeInsert – BeforeUpdate - BeforeDelete

記録

AfterInsert - AfterUpdate - AfterDelete

各行

検証

AfterValidate / BeforeInsert – BeforeUpdate - BeforeDelete

記録

AfterInsert - AfterUpdate - AfterDelete

繰り返し終了レベル 2

AfterLevel Level attLevel2 / BeforeComplete

コミット

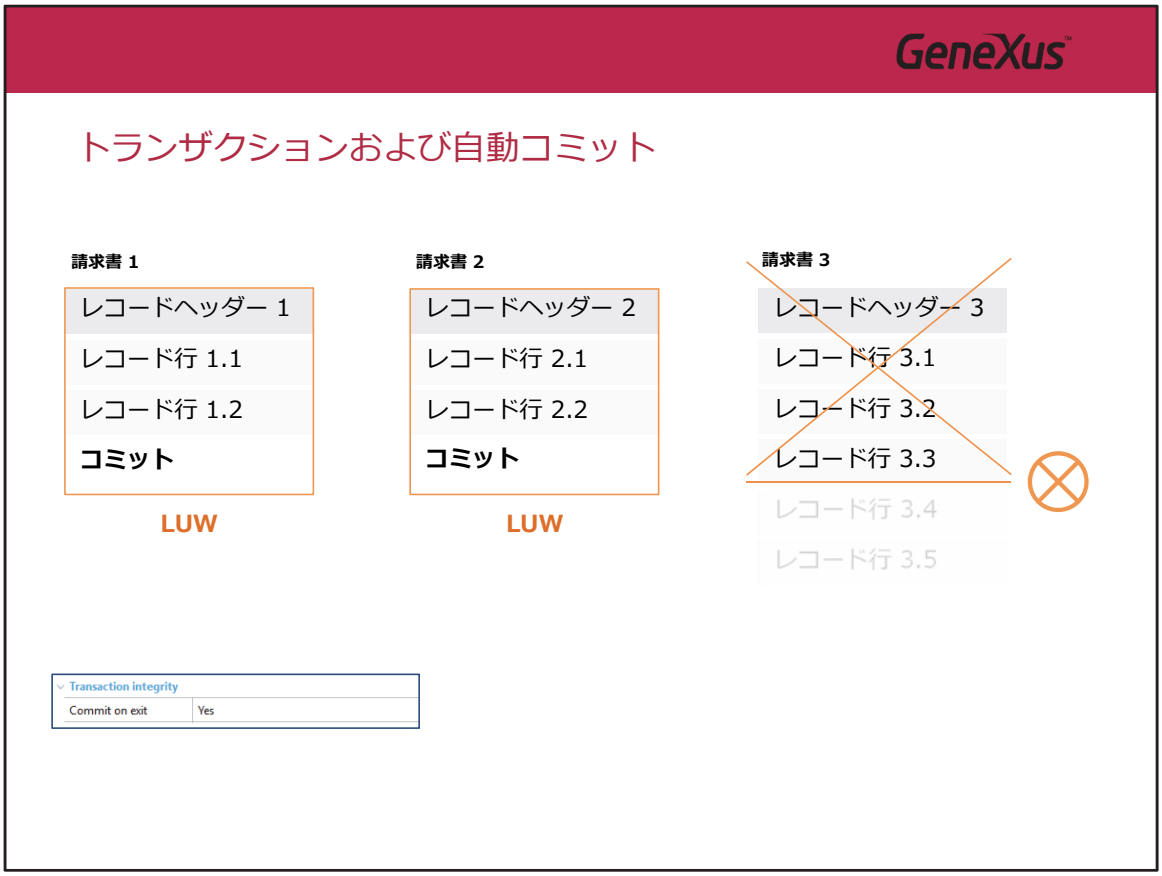
AfterComplete

ユーザーがヘッダーおよび行を調整して、[実行] をクリックします。サーバーでは第 1 レベルの評価ツリーに従ってルールおよび式が実行され、次に BeforeValidate イベントに条件付けられたルールがトリガーされます。ヘッダー情報が検証された後で、AfterValidate イベントに条件付けられたルールと、モードに応じて、BeforeInsert、BeforeUpdate または BeforeDelete に条件付けられたルールがトリガーされます。その後でヘッダーが保存され、モードに応じて、AfterInsert、AfterUpdate または AfterDelete に条件付けられたルールがトリガーされます。

次に、各行について次のように処理されます。

- 評価ツリーに従ってルールが実行されます。
- BeforeValidate トリガーイベントを持つルールは行レベルで実行されます。
- 行が検証されます (条件を満たしているとみなされます)。
- AfterValidate トリガーイベントを持つルール、またはモードに応じて BeforeInsert、BeforeUpdate あるいは BeforeDelete を持つルールが実行されます。
- データベースで、行に対応するレコードが挿入/変更/削除されます。
- 行のモードに応じて、AfterInsert、AfterUpdate または AfterDelete トリガーイベントを持つルールが実行されます。

最後の行の処理が完了した後で、トリガーイベントとして第 2 レベルの項目属性の AfterLevel を持つルールが実行されます。
別の並行レベルがある場合は、そのレベルについても同じ処理が繰り返されます。
最後のレベルが完了すると、BeforeComplete イベントに条件付けられたルールがトリガーされます。この後で、GeneXus が Commit コマンドを自動的に配置します。したがって Commit が実行されます。つまり、ヘッダーおよび行の情報がコミットされます。次に、AfterComplete イベントに条件付けられたルールがトリガーされます。



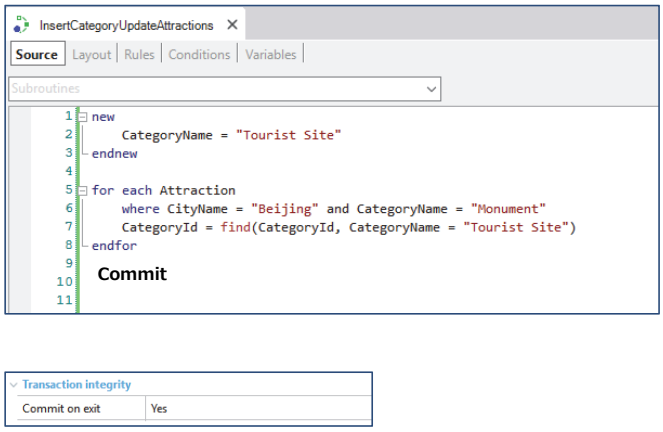
仮に、ユーザーが 3 つの請求書をシステムに入力したいとします。最初の請求書を入力し、次に 2 つ目を入力します。ユーザーが 3 つ目を確定する際、プログラムが 3 番目の行を処理しているときに、保存した後でシステムがクラッシュし、データベースを再起動しなければならなくなったとします。データベースはどのような状態になるでしょうか。

DBMS はロールバックを実行するため、コミットされていないすべての操作が元に戻されます。この場合、請求書 3 のヘッダーと 3 つの行に対応するレコードが削除されます。

ただし、Invoice トランザクションの自動コミットが無効になっていた場合 ([Commit on exit] プロパティが [No] の場合)、挿入したレコード (請求書 1、請求書 2、そしてもちろん請求書 3 のレコード) はいずれもデータベースには残りません。この場合、すべての操作で 1 つの LUW を構成しますが、[Commit on exit] プロパティの値が [Yes] の場合 (これが既定値) は、各ヘッダーおよびその行が個別の LUW を構成します。

トランザクションではこのプロパティを [Yes] に設定することをお勧めします。

プロシージャーおよび自動コミット



データベースにアクセスするすべてのプロシージャーで、([Commit on exit] プロパティを [No] に設定しない限り) GeneXus が Commit を自動的に追加します。

プロシージャーはデータベースの更新以外にも使用されます (たとえば、情報の一覧表示、計算の実行、データベースに対する単なるクエリなど)。そのため、GeneXus は、プロシージャーがデータベースを更新しようとしていると認識すると、生成するプログラムに Commit コマンドを自動的に挿入します。それ以外の場合は、[Commit on exit] プロパティの値に関係なく Commit は追加されません。

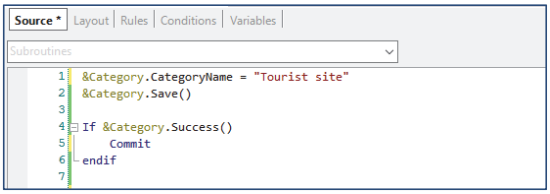
図の例では、New コマンドを使用してカテゴリを挿入し、Attraction トランザクションに関連付けられたテーブルの CategoryId 項目属性を For Each を使用して更新します。[Commit on exit] プロパティが [Yes] に設定されているため、GeneXus は生成するプログラムのコードの最後に Commit を自動的に記述します。つまり、CATEGORY テーブルに新しいカテゴリを挿入した後で、北京の 3 番目の遺跡を変更してカテゴリを新しいものにするときにシステムに障害が発生すると、以前の変更内容 (新しいカテゴリ、前の 2 つの遺跡の変更) はデータベースにはまったく残りません。このプロシージャーのすべての操作は同じ LUW に含まれます。この LUW はどこから開始するのでしょうか。

最後の Commit がどこで実行されたかによります。Commit が実行されたのがデータベースに対する最後の操作の後、かつこのプロシージャーを呼び出す前の場合、LUW はこのプロシージャーの New コマンドの時点で開始します。それ以外の場合は、このコード全体が前に開始した LUW に含まれます。それはどこかというと、前の Commit のすぐ後です。

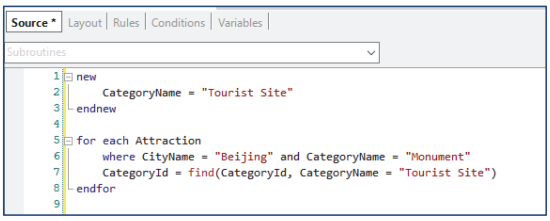
では、LUW はどこで終了するのでしょうか。[Commit on exit] プロパティが [Yes] に設定されている場合は、プロシージャーの最後に終了します。そうでない場合は、次の Commit が見つかったところで終了します (このプロシージャーの呼び出し元です。呼び出し後に何が起こるか確認する必要があります)。

プロシージャーおよび明示的コミット

明示的コミット



自動コミット



New

更新のための For Each

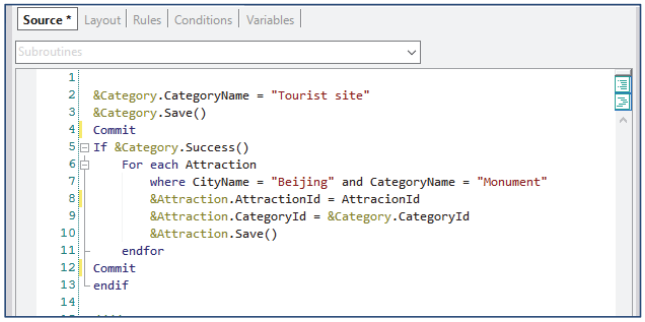
Delete

Load() または Delete()

New、更新のための For Each、Delete を実行するとき、GeneXus はプロシージャー内でデータベースにアクセスする必要があることを認識します。このような場合には暗示的なコミットが追加されます。それ以外の場合は、データベースアクセスが必要なことは認識されないため、プロシージャーで以下のように処理を行います。ビジネスコンポーネントの .Save()、またはその代わりに .Insert()、.Update()、.InsertOrUpdate() または .Delete() を実行する場合、Commit は追加されません。たとえ Insert メソッドを使用しても、GeneXus は、データベースに対して Insert を実行する必要があることを認識しないため、Commit コマンドを明示的に記述する必要があります。

先ほど述べたように、プロシージャーのコード内に New、更新のための For Each、Delete がある場合は、明示的に Commit を記述する必要はありません。プロシージャーに上記のコードがない場合は、Commit コマンドを明示的に追加する必要があります。

LUW のカスタマイズ

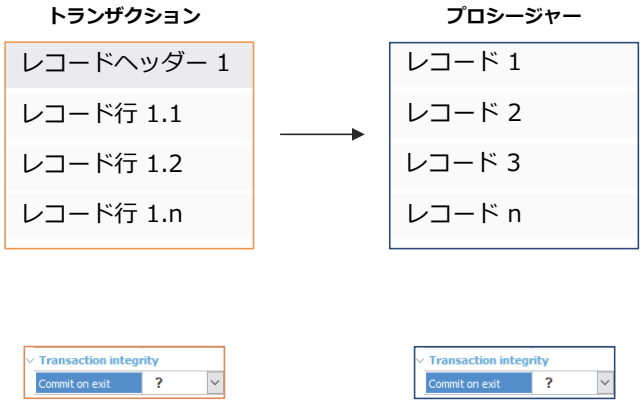


この例では GeneXus が自動的に Commit を最後に追加することを確認しました。では、カテゴリの記録を 1 つの LUW に含め、観光名所の記録を別の LUW に含めることで、観光名所の変更が終了する前にシステムに障害が発生したときに、カテゴリは入力済みだが、観光名所はそうでない状態にしたいとします。どうすればいいでしょうか。

Commit を使用します。カテゴリの挿入の後に Commit を記述し、すべての観光名所の挿入の後に 2 つ目の Commit を記述します。[Commit on exit] プロパティが有効な場合は、2 つ目の Commit は省略できます。いずれにしろ、プロシージャーのソースに後でさらに操作を追加する場合に備えて、Commit を明示的に記述するほうがいいでしょう。追加の操作は別の LUW に含めます。

GeneXus

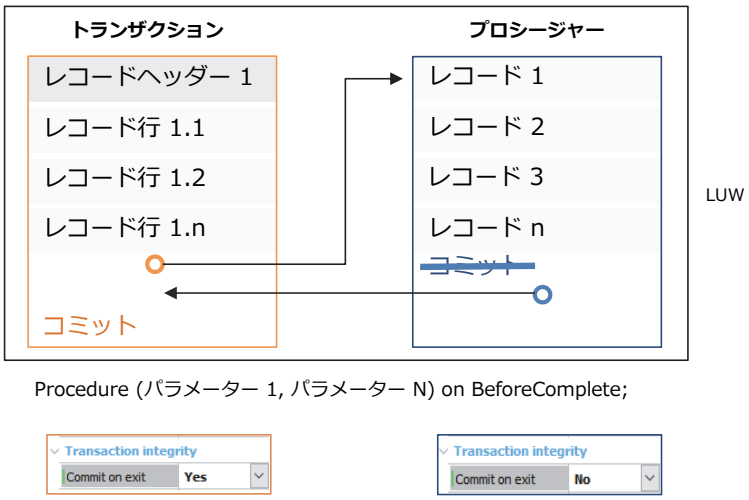
LUW のカスタマイズ



トランザクション「A」から、データベースに対する操作を実行するプロシージャー「B」を呼び出して、トランザクションのヘッダーレコードおよびすべての行、さらにプロシージャーのすべてのレコードの更新を単一の LUW に含める必要があるとします (したがって、すべてが完了する前にシステムがクラッシュしたら、すべての変更が元に戻されるようにします)。どうプログラムすればいいでしょうか。

これには、いくつかの方法があります。

LUW のカスタマイズ



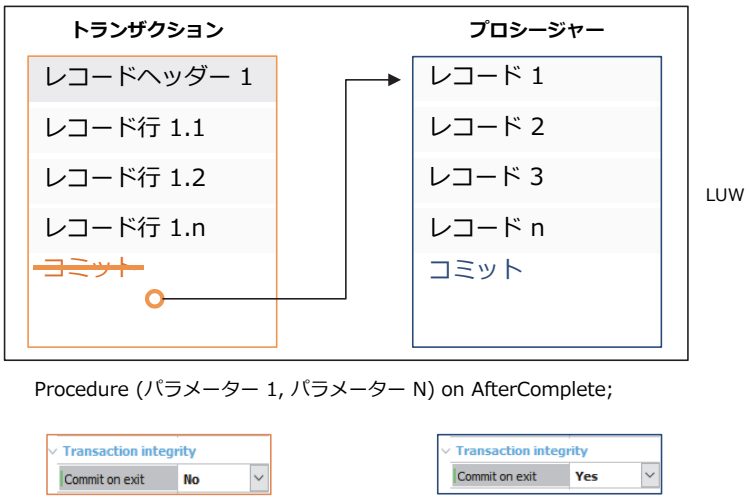
Procedure (パラメーター 1, パラメーター N) on BeforeComplete;

その 1 つは次のような方法です。

1. 自動コミットの前のある時点でプロシージャーを呼び出します。たとえば、「on BeforeComplete」です。
2. プロシージャーの自動コミットを無効にします。

これで、すべてのレコード (ヘッダーに対応するものと行に対応するもの) が保存された後でプロシージャーが呼び出されます。AfterLevel イベントに条件付けられたすべてのルールは既にトリガーされています。つまり、Commit の直前にプロシージャーが呼び出されます。プロシージャーによりすべてのデータベースレコードの更新が実行されます。自動コミットを無効に設定したため、開発者が Commit コマンドを明示的にコード内に含めない限り、LUW は終了しません。プロシージャーのコードの実行が終了すると、呼び出し元 (呼び出しの後のステートメント) に戻ります。ここに Commit があります。

LUW のカスタマイズ



もう 1 つは次のような方法です。

1. トランザクションのすべてのレコード (ヘッダーおよび行) が保存された後であれば、いつプロシージャーを呼び出してもかまいません。Commit の後であっても問題ありません。たとえば、「on AfterComplete」です。

2. トランザクションの自動コミットが無効で、プロシージャーの自動コミットが有効である限り問題はありません。

このポイントでプロシージャーを呼び出すことにより、すべてのヘッダーおよび行のレコードが確実に記録されます。そして、プロシージャーによりデータベースレコードの更新とコミットが実行され、すべてのレコード (プロシージャーのレコードとトランザクションのレコード) がコミットされます。

この 2 つ以外にも多くの方法があります。どれを選択するかは実装したいロジックによって異なります (通常は、プロシージャーを呼び出すタイミングを気にかけるでしょう)。

LUW のカスタマイズ

顧客および予約した旅行を記録する Customer トランザクションがあります。

ここでは、データベースの自動採番を使用せずに、データベースで独自の内部テーブルを使用して、各エンティティに与えられた最後の番号を記録し、識別子に番号を付けたいとします。

そのために Numbering トランザクションを作成します。NumberingCode 識別子項目属性で該当するエンティティの名前 (たとえば、Customer、Trip、Invoice、Category、Attraction など) を記録し、NumberingLastId 項目属性でそのエンティティに与えられた最後の番号を記録します。

次に、GetNextNumber プロシージャーをプログラムし、呼び出し元エンティティの識別子に割り当てる次の番号を取得します。

コードは後ほど確認します。

たとえば、ユーザーが Customer トランザクションから新しい顧客を入力したい場合、画面上の [CustomerId] フィールドを空のままにし、このフィールドから離れて次のフィールド [CustomerName] に移動すると、トランザクションは、ユーザーが新しいレコードを挿入しようとしていると認識します (「INS」、つまり挿入モードになります)。
CustomerId 項目属性は自動採番でないため、GetNextNumber プロシージャーを呼び出して CustomerId に割り当てる番号を取得する必要があります。
この場合、プロシージャーにパラメーターとして Customer を送り、プロシージャーが Numbering テーブルで Customer に割り当てた最後の番号を見つけられるようにする必要があります。

作成されたプロシージャーで何が実行されるか、詳しく見てみましょう。
まず、ルールで、入力パラメーターおよび出力パラメーターが 1 つずつ宣言されています。入力パラメーターを使用して番号を付ける対象 (顧客、旅行、請求書など) を指定します。この変数は Character タイプです。
出力として、Numeric タイプの nextNumber 変数を宣言します。これが必要な値を返します。

ソースに戻ります。Numbering トランザクションのテーブルが参照されます。
Where 節で、パラメーターで送った変数 (この場合は Customer) と NumberingCode が同じ値であるレコードを取得するように指定します。
見つかった場合は NumberingLastId 項目属性に 1 が追加され、この値が nextNumber 変数に割り当てられます。そして、NumberingLastId 項目属性に nextNumber の値が割り当てられます。

NumberingCode の値が送った変数と同じレコードが見つからない場合は、New コマンドを使用してデータベースにレコードを作成します。ここで NumberingCode はパラメーターで送った変数 (たとえば Invoice) の値を持ちます。次に、nextNumber 変数に値 1 を割り当て、NumberingLastId 項目属性に nextNumber の値を割り当てます。これはこのテーブルの新しいレコードであるため、1 から開始する必要があります。

次のようなトリガーイベントのない割り当てルールを使用してこのプロシージャーを呼び出したとします。
CustomerId = GetNextNumber(“Customer”) if Insert; 次のようなときにプロシージャーがトリガーされます。

1. 1 回目は、ユーザーが画面の [CustomerId] フィールドを空のままにしてフィールドから離れた直後。
2. 2 回目は、ユーザーが [実行] をクリックしたとき。サーバーでルールが順番に再度トリガーされます。

最後の顧客 ID が 5 だとしましょう。

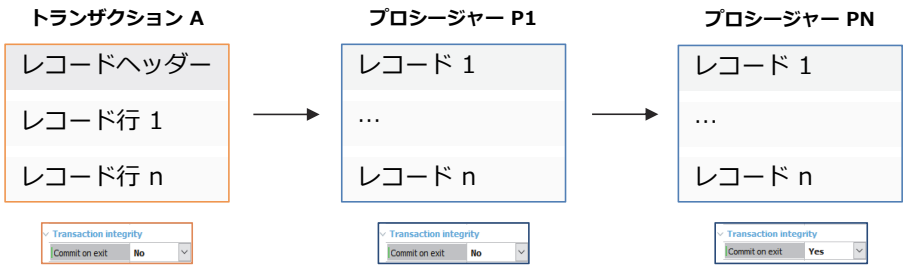
プロシージャーが実行されて、対応する Numbering テーブルのレコードが 6 に更新され、即座にユーザーに対して画面に 6 が表示されます。もしユーザーが考えを変えて、[実行] ではなく [キャンセル] をクリックしたらどうなるでしょうか。番号 6 は失われます。

たとえユーザーが実行する場合でも、プロシージャーが再実行され、顧客に割り当てられる番号が 7 になるため、番号 6 は失われます。

これをどう解決すればよいでしょうか。プロシージャーの呼び出しをトリガーイベントに条件付けます (こうすれば、ユーザーが画面で作業しているときに、クライアントで割り当てルールはトリガーされません)。適切なタイミングはいつでしょうか。もっとも遅いタイミングはヘッダーの BeforeInsert です。なぜでしょうか。このタイミングを過ぎると、レコードが既に保存されているため、項目属性に割り当てた値は機能しません。

GetNextNumber プロシージャーは Numbering テーブルのレコードを変更するか、レコードが存在しない場合はレコードを挿入します。既定では最後に Commit が自動的に配置されます。このように、トランザクションに戻った後で、顧客がテーブルに挿入され、3 番目の旅行の挿入中にシステムに障害が発生したとします。システムが復元されたときに、この顧客に関連付けられたレコードは記録されません。しかし、既にコミットされているため番号は失われます。トランザクションおよびプロシージャーを介して実行されるすべての操作を同じ LUW 内に残すには、この場合、プロシージャーで終了時のコミットを実行せず、トランザクションによりすべてのコミットを実行すれば十分です。

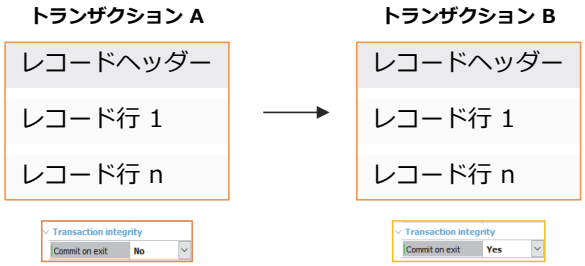
LUW のカスタマイズ



複数のトランザクションにおよぶ LUW を作成することはできません。

最後のものを除き、すべてのコミットを無効にして、別のプロシージャーを呼び出すプロシージャーをトランザクションから呼び出すとします (または、最後のもののコミットも無効にして、制御が返されたときにトランザクションでコミットすることもできます)。この場合、最後のトランザクションのレコードおよびすべてのプロシージャーのレコードがコミットされます。

LUW のカスタマイズ

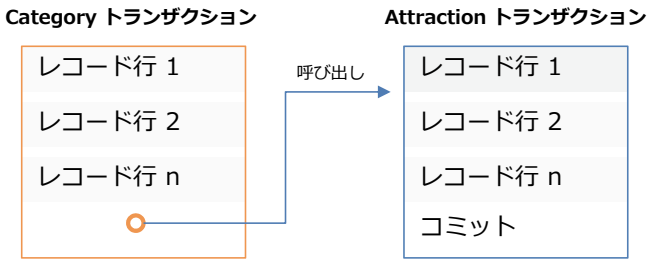


先ほどの説明のとおり、トランザクション (「A」) が別のトランザクション (「B」) を呼び出す場合は、2 番目のもの (「B」) のコミットでは、最初のもの (「A」) が処理するレコードはコミットされません。これらは独立したコミット操作です。したがって、「A」のコミットを無効にして「B」(コミットを実行する) を呼び出すと、「A」のレコードはまったくコミットされません。

それでは、トランザクション「A」のヘッダーおよび行が入力され、トランザクション「B」が呼び出されて何らかの関連情報が入力されるようにし、これらすべての操作を 1 つの LUW に含めるにはどうすればいいでしょうか。

GeneXus

LUW のカスタマイズ



Work With Categories で、[追加] ([Insert]) をクリックすると、ユーザーが新しいカテゴリを入力でき、直後にそのカテゴリの観光名所を入力できるようにしたいとします。これは、関連する観光名所なしでカテゴリが挿入されることを避けるためです。

そのためには、Category トランザクションから、AfterInsert のイベントで Attraction トランザクションを呼び出します (CategoryId はデータベースで自動採番された正しい値を持ちます)。そして、Attraction でパラメーターを受け取ることを宣言します。

これで、カテゴリを入力すると、Attraction トランザクションに移動し、観光名所を入力できます。パラメーターで送ったカテゴリが選択されます。

しかし、Attraction でコミットが実行された直後にシステムで障害が発生したらどうなるでしょうか。既に挿入されている Category レコードもこのコミットによりコミットされているでしょうか。既に説明したように、答えは「いいえ」です。

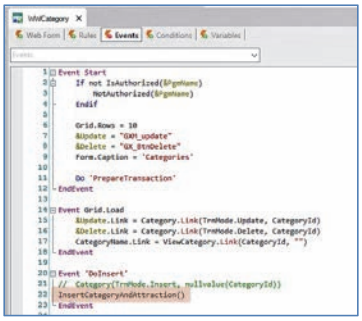
カテゴリの挿入およびその次の観光名所の挿入を同じ LUW に含めて、最後のコミットに両方のレコードを含める必要があります。どうすればいいでしょうか。

たくさんの方法がありますが、そのうち 2 つを見てみましょう。

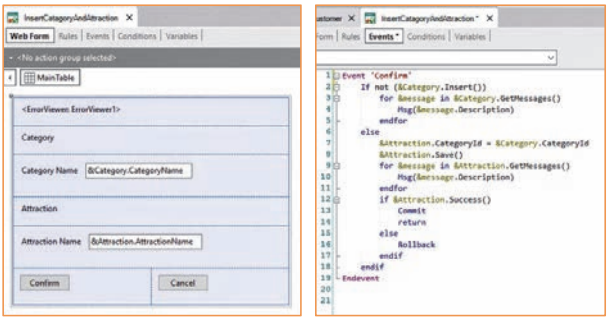
GeneXus

LUW のカスタマイズ

WorkWithCategory



InsertCategoryAndAttraction



1 つの解決法は、「Work With」の [追加] ([Insert]) ボタンで Web パネルを呼び出すことです。これはインターフェースを備えたオブジェクトで、開発者が動作を自由にプログラムできます。

ここで、&category および &attraction の 2 つの変数をフォームに挿入します。それぞれが Category と Attraction のビジネスコンポーネントになります。
ユーザーは、作成したいカテゴリの名前およびそのカテゴリの観光名所の名前を挿入するだけです。これを行って [実行] をクリックすると、関連付けられたイベントがトリガーされます。イベント画面のプログラムをご覧ください。

この例では、Country および Attraction トランザクションそれぞれの CountryId および AttractionId 項目属性の [Autonumber] プロパティが [True] に設定されています。

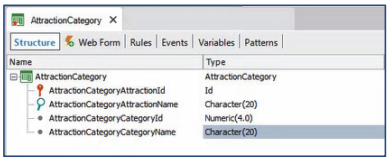
まず、Category のビジネスコンポーネントの挿入を試みます。失敗した場合 (たとえば、ユーザーが画面でカテゴリ名を空のままにし、トランザクションにはこれを避けるためのエラールールがある場合)、ビジネスコンポーネントにより生成されたメッセージを ErrorViewer コントロールに表示します。
この例では、エラーメッセージが 2 回トリガーされます。フィールドを空のままにしたときにクライアント側で 1 回、実行したときにサーバー側でもう 1 回トリガーされます。どちらの場合も結果は同じです。このようなタイプのルールをべき等といいます。

すべてのデータを正しく入力したら、Attraction のビジネスコンポーネント変数に、挿入したものをカテゴリとして割り当て保存を試みます。生成されたメッセージを表示し、操作が成功した場合はコミットします。これで両方のレコードがコミットされます。

操作が失敗した場合は、Rollback コマンドを使用して、挿入が成功した Category レコードの挿入を元に戻しています。

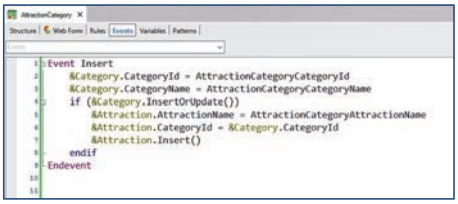
LUW のカスタマイズ

AttractionCategory トランザクション



データプロバイダー

Insert イベント



同じ課題を解決する第 2 の方法は、Web パネルではなくダイナミックトランザクションを使用する方法です。

ダイナミックトランザクションでは物理テーブルは生成されません。クエリされたデータは実行時に取得します。
これは次のようにトランザクションのプロパティを設定して定義します。[Data Provider] プロパティを [True] に、[Used To] プロパティを [Retrieve data] に設定します。
[Data Provider] プロパティを [True] に設定すると、Data Provider オブジェクトが自動的に作成されます。[Used To] プロパティを [Retrieve data] に設定すると、そのトランザクションに関連付けられたテーブルを作成する必要がないことを GeneXus が認識します。これは、データプロバイダーによりデータの取得元が宣言されるためです。

この例では、AttractionCategory ダイナミックトランザクションを作成します。各観光名所がカテゴリを持つことが認識され、Attraction テーブルから情報が取得されます。これは観光名所のビューのようになります。

ユーザーがこのトランザクションで観光名所を挿入したい場合、観光名所のデータおよびカテゴリのデータを入力することができます。Insert イベントでは、Category の &category ビジネスコンポーネントおよび Attraction の &attraction ビジネスコンポーネントを使用します。そして、そのビジネスコンポーネントのメンバーに、ユーザーがフォームを介してダイナミックトランザクションの項目属性で指定した値をコピーします。
カテゴリが存在しない場合は、観光名所を挿入する前に作成されます。それ以外の場合は、カテゴリの名前を更新できます。次に、観光名所がカテゴリとともに挿入されます。
トランザクションの [Commit on exit] プロパティを既定値の [Yes] のままにした場合、単一レベルのトランザクションであるため、Insert イベントを実行すると Commit が実行されます。これは、ビジネスコンポーネントを介して挿入された 2 つのレコードに影響します。