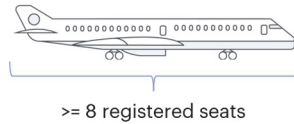


トランザクションにおける ルールのトリガーイベント

続き

*GeneXus*TM



```
Flight X
Structure | Web Form | Rules | Events | Variables | Patterns
1 Error("The seat quantity mustn't be less than eight")
2 if FlightCapacity < 8;
```

Flight

« < > » SELECT

Id 0 The seat quantity mustn't be less than eight

Departure Airport Id

Departure Airport Name

Departure Country Id 0

Departure Country Name

Departure City Id 0

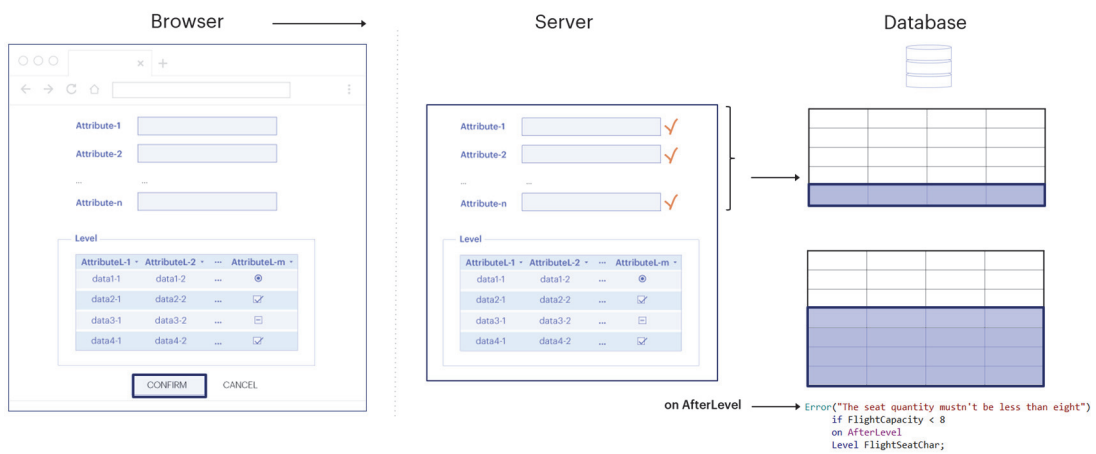
Departure City Name

Arrival Airport Id 0

前の章では、GeneXus によって選択されたルール実行のタイミングが必要なタイミングではなく、開発者が適切なタイミングを指定できないケースがあることを確認しました。

各フライトで少なくとも 8 座席の予約が必要である場合について考えました。ユーザーが座席を予約する前に、GeneXus がルールを実行したため、

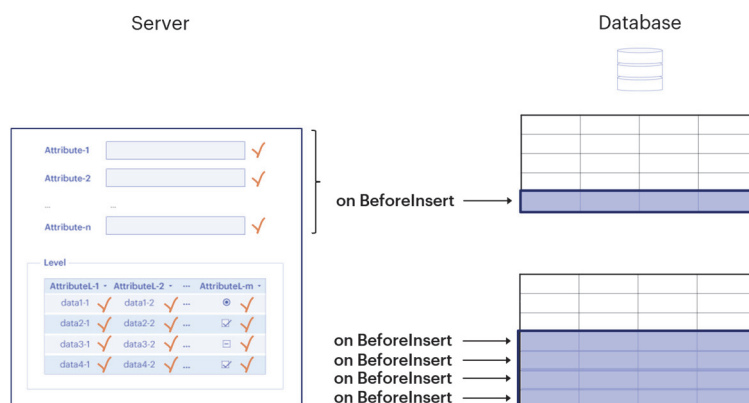
AfterLevel event



そのトリガーのタイミングを遅らせる必要がありました。その際にはトリガーのタイミング **on AfterLevel** を明細行の項目属性 (**FlightSeatChar**) とともに使用しました。

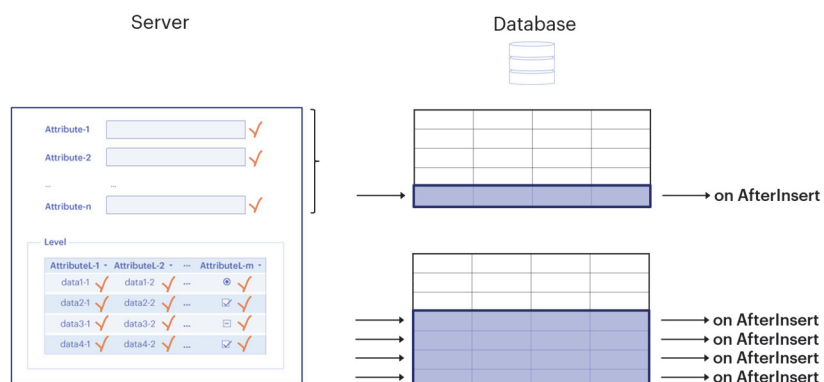
この方法で、座席に関連付けられているレベル全体を確認してからルールがトリガーされるようになりました。

BeforeInsert event



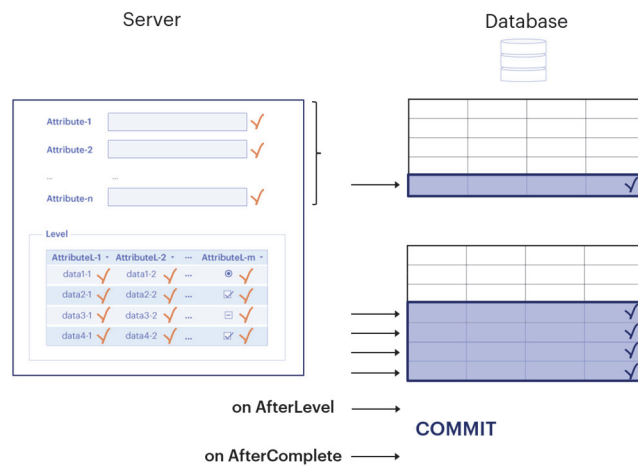
トリガーのタイミングはほかにもあることにも触れました。
 たとえば「**on BeforeInsert**」は、ヘッダーまたは各明細行のデータをデータベースに挿入する**直前**に何かを実施または評価する場合に使用します。

AfterInsert event



「on AfterInsert」では、ヘッダーまたは明細行の挿入直後にルールがトリガーされます。

AfterComplete event



そして「**on AfterComplete**」はコミット直後のタイミングに対応します。コミットは、挿入、変更、または削除されたデータの確定を目的とするコマンドです。

Execution of rules




ここでは、トリガーのタイミングについて詳しく見ていきます。その前に、ルールには、クライアント (ブラウザー) とサーバーの両方で検証されるものと、データベースと関係があるためサーバーでのみ検証されるものがあることを思い出してください。

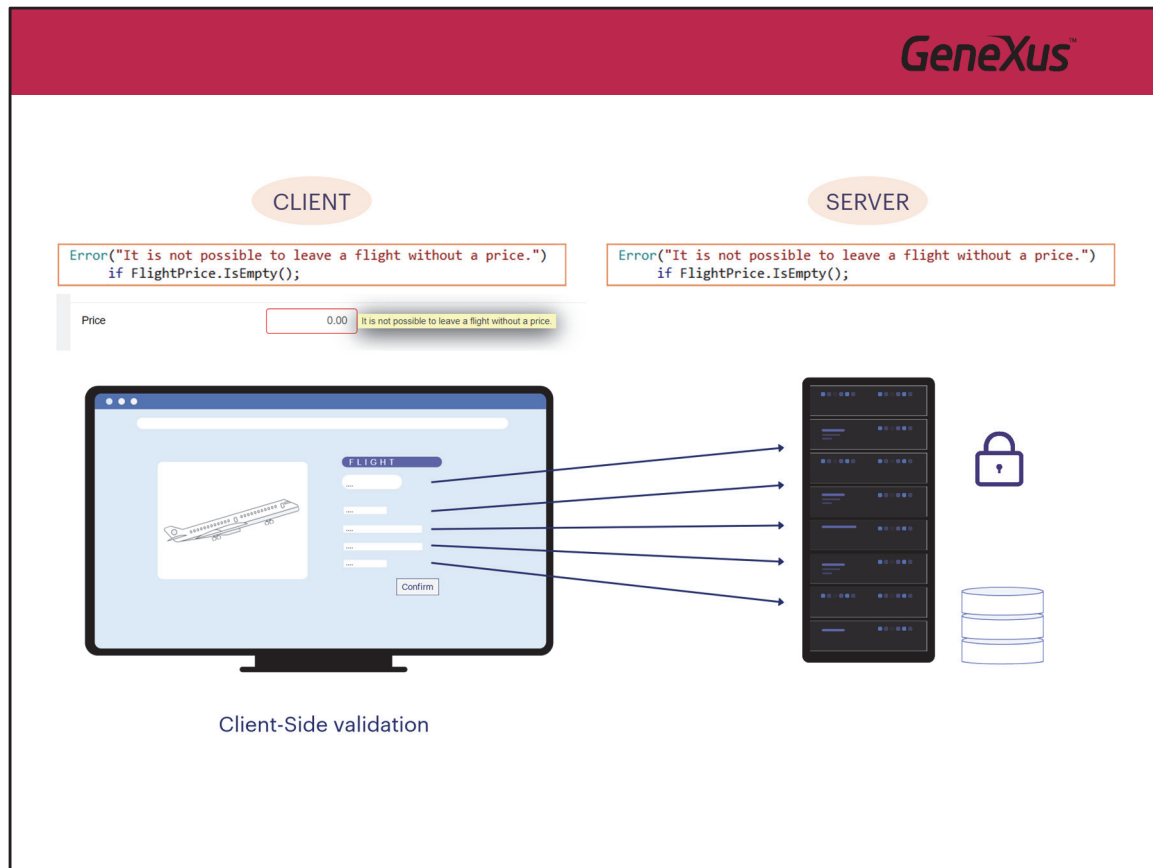
```
Error("It is not possible to leave a flight without a price.")  
if FlightPrice.IsEmpty();
```

Arrival Country Id	2
Arrival Country Name	France
Arrival City Id	1
Arrival City Name	Paris
Price	0.00
Discount Percentage	<input type="text" value="0"/>
Airline Id	0
Airline Name	
Airline Discount Percentage	0
Final Price	0.00

It is not possible to leave a flight without a price.



たとえば、料金なしでフライトを入力または変更することを防ぐ Error ルールがある場合、フィールドを離れるとすぐにエラーメッセージが表示されます。



これが可能なのは、ユーザーエクスペリエンス向上のためにクライアント側で検証が実行されるためです。迅速に応答があるため、システムの操作がシームレスになります。クライアント側で実行される検証をクライアント側検証といいます。

このルールは、その後サーバーでも実行されます。セキュリティ違反がないことを確認するのはサーバーの役目であり、またデータベースの操作が唯一許可されているのもサーバーであるため、サーバーがロジックの一貫性を確認する必要があります。情報がすべて揃ったときにすべてのルールが再び実行されます。これはユーザーが [実行] ボタンをクリックしたときに行われます。

このとき、データが Web クライアント (ブラウザ) から Web サーバーに送信されます。サーバーでは、ユーザー操作になぞらえ、フォームが最初から最後まで再び確認されます。このため、フォーム上の項目属性と同じ順序でルールや式が再びトリガーされます。

トリガーイベントが関連付けられている (ステートメント末尾に接頭語 **on** がある) ルールは、そのイベントに対応するタイミングで実行されます。これらのイベントはサーバー上でのみ実行され、通常はデータベースに関係があるマイルストーンの前か後にタイミングを捉えます。

Milestones when inserting a flight

- Validate the header record to be inserted
- Insert the header record in the table

Then the same for each line:

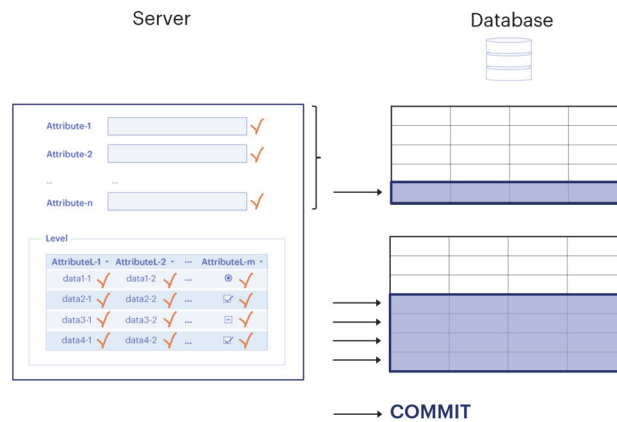
- Validate the line record to be inserted
- Insert the line record in the table

The next milestone is when:

- You finish working with the level

And the next one is:

- The Commit action



どのようなマイルストーンがあるでしょうか。

フライトを挿入する場合を例に考えます。これは次の順序で行います。

- 挿入するヘッダーレコードを検証します (参照整合性と一意制約に問題がないことを確認。ヘッダーの各フィールドを確認し、対応する各ルールをトリガーした後で、最後に発生)。
- ヘッダーレコードを対応するテーブルに挿入します。

各明細行について同様の手順を行います。

- 挿入する行のレコードを検証します (参照整合性と一意制約に問題がないことを確認)。
- 明細行レコードを対応するテーブルに挿入します。

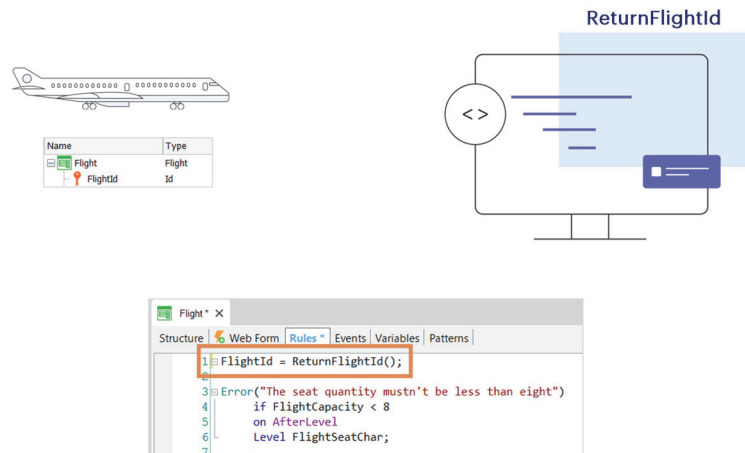
このプロセスの最後に、ヘッダーに対応するレコードとすべての明細行に対応するレコードがデータベースに挿入されたら、次のマイルストーンは、

- レベルの操作が完了したときになります。

その次は、

- データベースでこれらすべての操作を検証し、確定するコミットアクションになります。

これらのマイルストーンの前または後にタイミングを捉えるイベントにより、指定したタイミングでルールを実行できます。

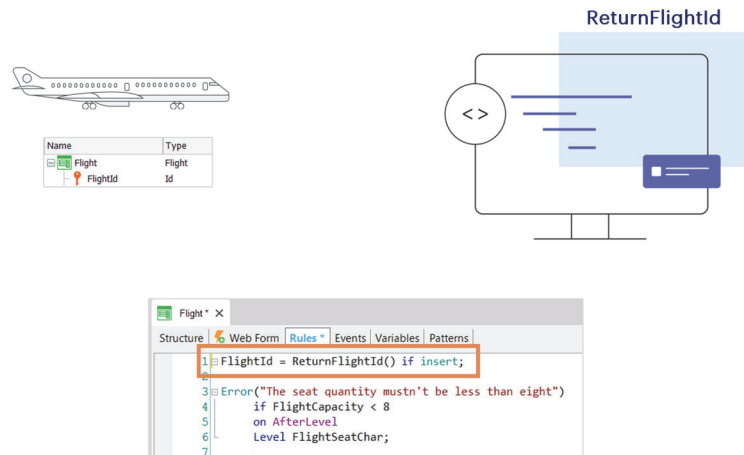


現時点では、Flight トランザクションでフライト識別子 (FlightId) を自動採番と定義しています。
 現実にはフライト識別子は航空会社、出発地と目的地に応じた文字と数字から構成され、自動採番による数字の FlightId 項目属性は実用的ではありません。

そこで、ReturnFlightId というプロシーチャーを考えます。呼び出して実行すると、新しいフライトに割り当てる識別子を返すプロシーチャーです。

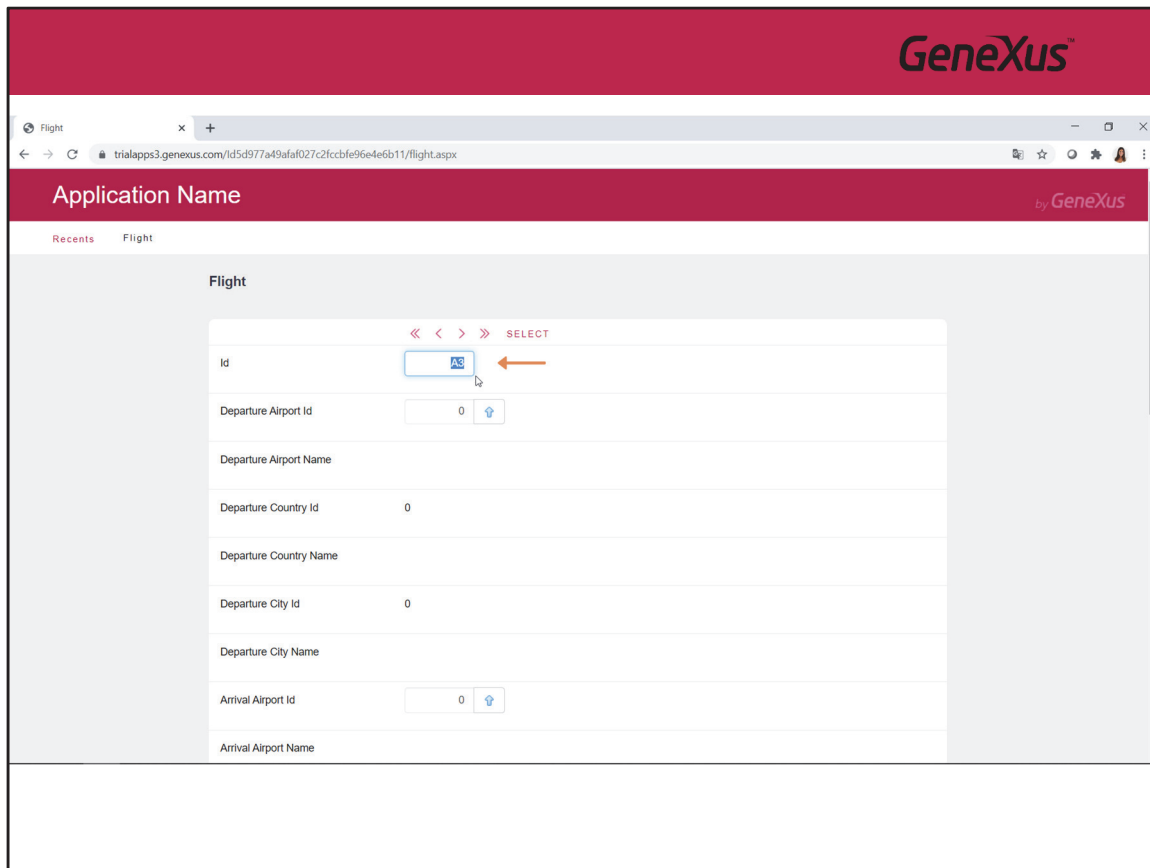
(Procedure オブジェクトについてはまだ学習していないため、ここで説明する内容は現時点では実行できません)。

前述のルールを記述し、プロシーチャーを呼び出して結果を FlightId 項目属性に割り当てた場合、これはいつトリガーされるでしょうか。
 それは、トランザクションを開くとき、または既存のフライトを編集するときです。
 ここでは、モードをルールのトリガー条件にしていなかったため、ユーザーが Flight トランザクションにアクセスする目的がフライトの挿入、変更、削除のいずれであってもルールは実行されます。しかし実際には、新しいフライトを挿入する場合にのみプロシーチャーを呼び出すことが求められます。

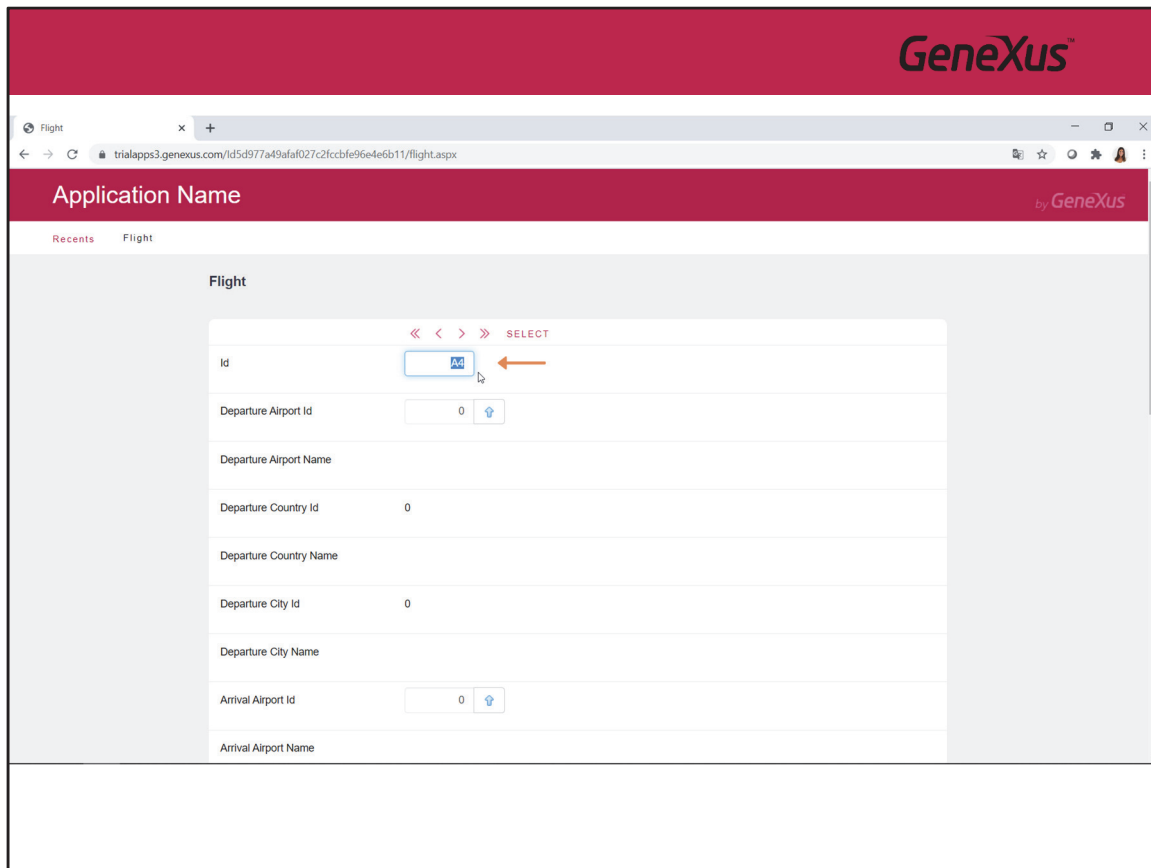


また、Flight トランザクションが更新モードの場合 (つまり既存するフライトを変更する場合)、フライト識別子の値は変更できません。主キーは変更できないからです。主キーを変更する必要がある場合、新しいキー値で新しいレコードを作成し、古いレコードを削除するしかありません。

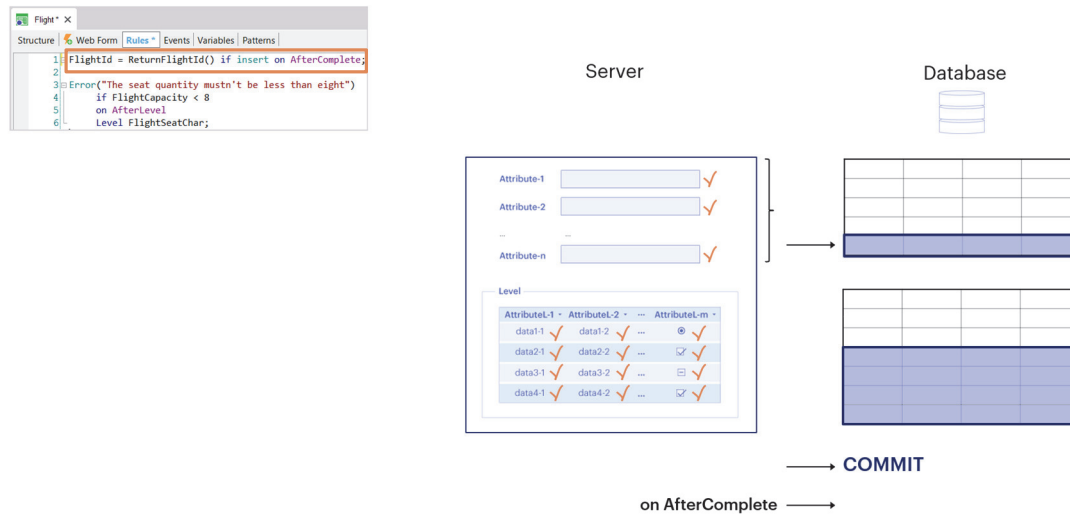
そこで、データを挿入する場合にのみルールが実行されるよう条件を設定します。



トランザクションが Insert モードで開くと、何か操作をする前に、項目属性に値を割り当てるルールが既にトリガーされています。
ここで、ヘッダーデータや行の完了後に何らかの理由で入力をキャンセルし、後でやり直す必要が生じた場合を考えます。

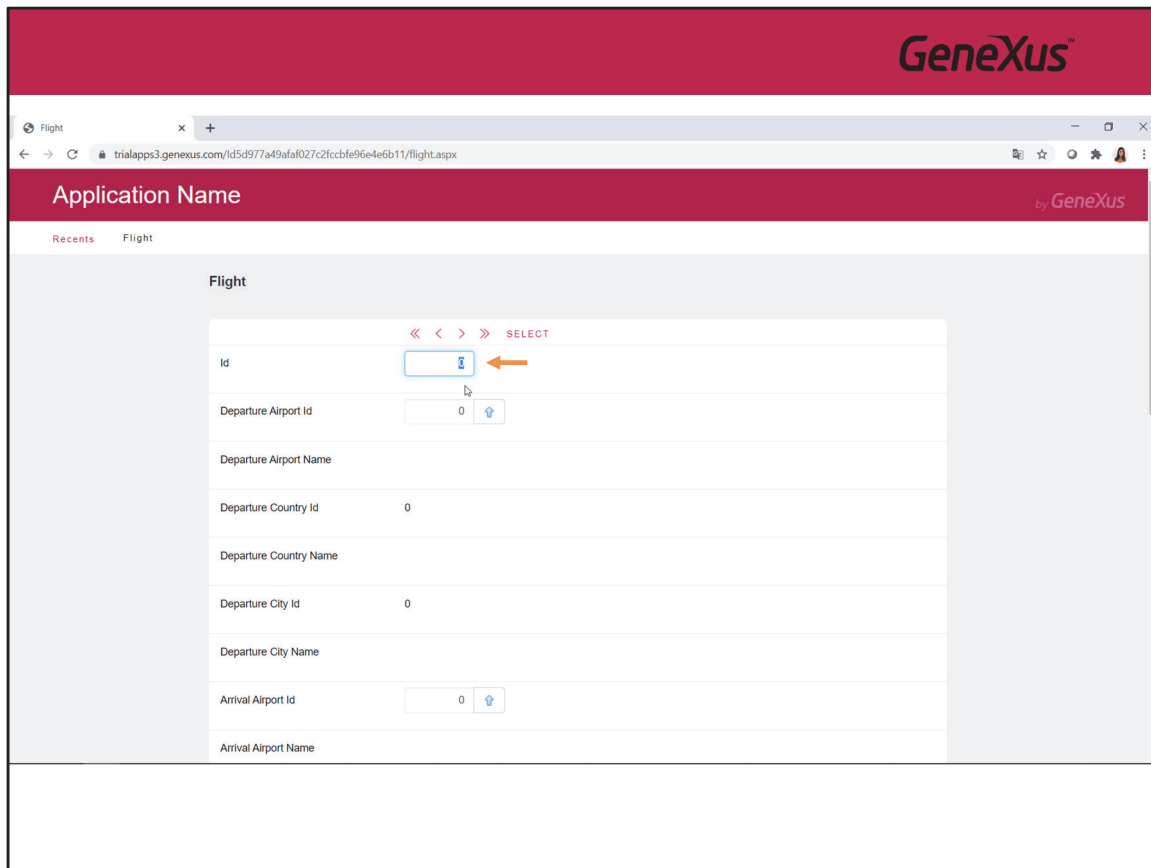


後で入力しようとする、別のフライト番号が割り当てられます。プロシージャーに求めた番号を最終的に使用しなかったため、その番号は失われてしまいました。

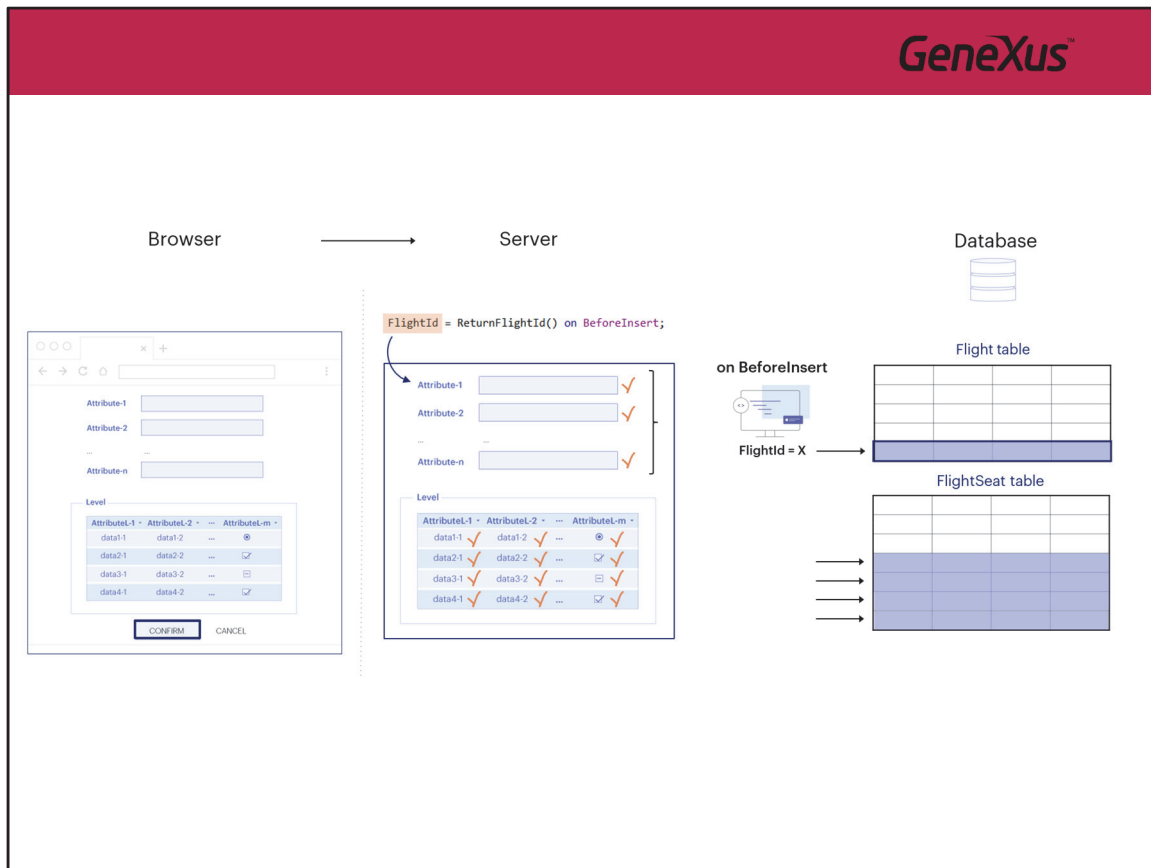


それでは、使わなかった番号が無駄にならないように、その時点で完全にすべて (新規に採番された ID も、挿入しようとしているレコードも) がデータベースに保持されているコミットの後、番号を振るプロシーチャーを呼び出したらどうでしょうか？

226



実行すると、ID にすぐに数字が割り当てられないことが分かります。明細行に情報を入力しても変わりません。

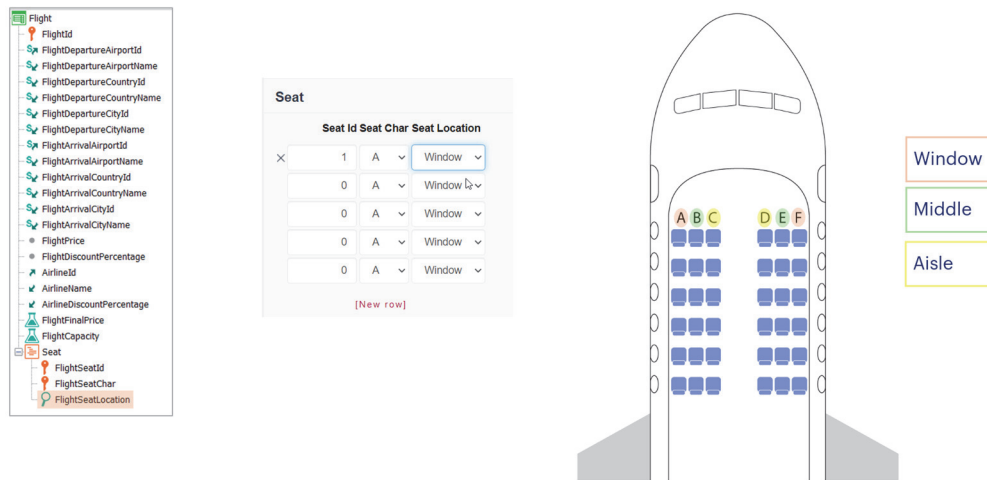


ユーザーが画面で確認したあと、情報がすべてサーバーに送信されてから、参照整合性のコントロールやルールが実行され、各フィールドが検証されます。ヘッダーが完了し、すべてが検証されたら、BeforeInsert イベントが発生します。ここで、番号が割り当てられ、その直後にレコードが Flight テーブルに挿入されます。その後、明細行 1 の各フィールドの検証、挿入、そして明細行 2 以降へと続きます。

ルールは、すべての行が記録される前にトリガーされるわけではありません。このルールはヘッダーの BeforeInsert にのみトリガーされます。なぜでしょうか。それはルール内で、ヘッダーに属する項目属性を定義しているからです。

別のルールを定義し、同様にトリガーイベント **on BeforeInsert** を追加するとします。ただし、前の例とは異なり、トランザクションの第 2 レベルの項目属性への参照がルールに 1 つ以上含まれる場合、このルールは第 2 レベルに関連付けられます。したがって、トランザクションの第 2 レベルに対応する各インスタンスが物理的に保存される直前に実行されます。

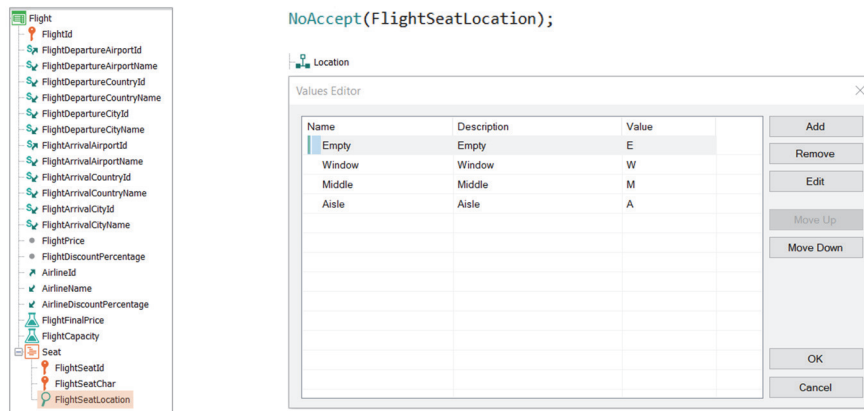
このように、イベント名 BeforeInsert は同じでも、ヘッダーと行のどちらに適用されるかによって、実際にはそれぞれ異なるものになります。



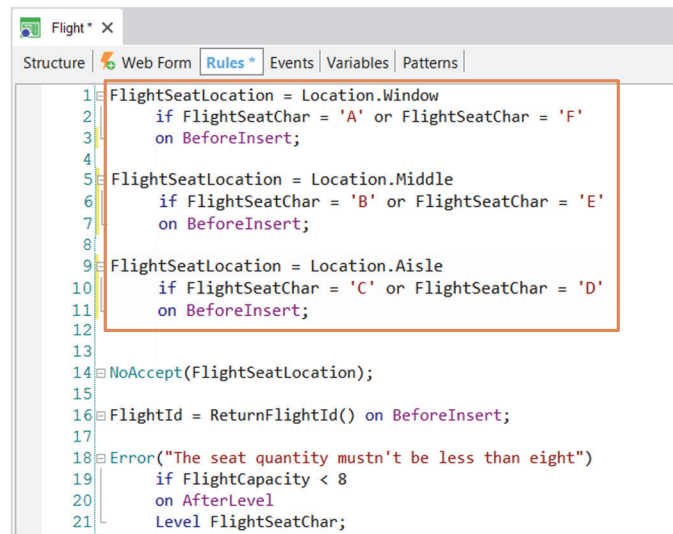
明細行に対する BeforeInsert の例を見てみます。

FlightSeatLocation 項目属性の値を、トランザクションを操作するユーザーが選択するのではなく、ルールで割り当てるとします。FlightSeatChar 項目属性の値が A または F の場合は「Window」、FlightSeatChar 項目属性が B または E のときは「Medium」、値が C または D のときは「Aisle」に設定します。

この例は実際にはトリガーイベントを使用せずに解決でき、そのほうが適切である可能性もあります。ルールはクライアント上ですぐに実行されるため、ユーザーは SeatLocation の値をすぐに画面上で確認できます。しかし、トリガーのタイミングを理解するため、ここではフライトで座席を入力されることが確実な場合にのみ、つまり行を挿入する直前に割り当てを行うと想定します。



NoAccept ルールを使用してユーザーが FlightSeatLocation 項目属性を選択できないようにし、Location ドメインに値「Empty」を追加します。



```
1 FlightSeatLocation = Location.Window
2   if FlightSeatChar = 'A' or FlightSeatChar = 'F'
3     on BeforeInsert;
4
5 FlightSeatLocation = Location.Middle
6   if FlightSeatChar = 'B' or FlightSeatChar = 'E'
7     on BeforeInsert;
8
9 FlightSeatLocation = Location.Aisle
10  if FlightSeatChar = 'C' or FlightSeatChar = 'D'
11    on BeforeInsert;
12
13
14 NoAccept(FlightSeatLocation);
15
16 FlightId = ReturnFlightId() on BeforeInsert;
17
18 Error("The seat quantity mustn't be less than eight")
19   if FlightCapacity < 8
20     on AfterLevel
21       Level FlightSeatChar;
```

そして次のルールを作成します。

GeneXus™

Flight
trialapps3.genexus.com/id5d977a49afa027c2fccbf9e4e6b11/flight.aspx

Discount Percentage:

Airline Id:

Airline Name: TAM

Airline Discount Percentage: 0

Final Price: 4000.00

Capacity: 0

Server

Attribute 1:

Attribute 2:

Attribute 4:

Level:

Attribute 1	Attribute 2	Attribute 4
seat1	seat2	seat4
seat1	seat2	seat4
seat1	seat2	seat4

on BeforeInsert →

Database

Seat

Seat Id	Seat Char	Seat Location
X 1	A	Empty
X 1	B	Empty
X 1	C	Empty
X 1	D	Empty
X 1	E	Empty
X 1	F	Empty
X 2	A	Empty
X 2	B	Empty

!New row!

```

FlightSeatLocation = Location.Window
if FlightSeatChar = 'A' or FlightSeatChar = 'F'
on BeforeInsert;

FlightSeatLocation = Location.Middle
if FlightSeatChar = 'B' or FlightSeatChar = 'E'
on BeforeInsert;

FlightSeatLocation = Location.Aisle
if FlightSeatChar = 'C' or FlightSeatChar = 'D'
on BeforeInsert;

```

グアルーリョス空港からシャルル・ド・ゴール空港までの新しいフライトを、料金 4000、割引なし、TAM 航空で入力します。次に座席を予約します。
1 列目を完了し、さらに 2 列目の 2 座席を加えます。

各明細行が物理的に挿入される前に、選択した FlightSeatChar に応じて、対応するルールが実行されます。

サーバー上のトランザクションで 1 行目のデータを検証すると、SeatLocation が Empty です。SeatChar が A であるため次のステップで最初のルールが実行されます。その後、SeatLocation が Window に変更されてから、行がすぐにテーブルに保存されます。

GeneXus™

Flight
+

trialapps3.genexus.com/Id5d977a49afaf027c2fccbf9e9e4e6b11/flight.aspx
🔍 ⭐ ⚙️ 👤

Airline Name	TAM
Airline Discount Percentage	0
Final Price	4000.00
Capacity	8

```

FlightSeatLocation = Location.Window
if FlightSeatChar = 'A' or FlightSeatChar = 'F'
on BeforeInsert;

FlightSeatLocation = Location.Middle
if FlightSeatChar = 'B' or FlightSeatChar = 'E'
on BeforeInsert;

FlightSeatLocation = Location.Aisle
if FlightSeatChar = 'C' or FlightSeatChar = 'D'
on BeforeInsert;

```

on AfterInsert?

Seat

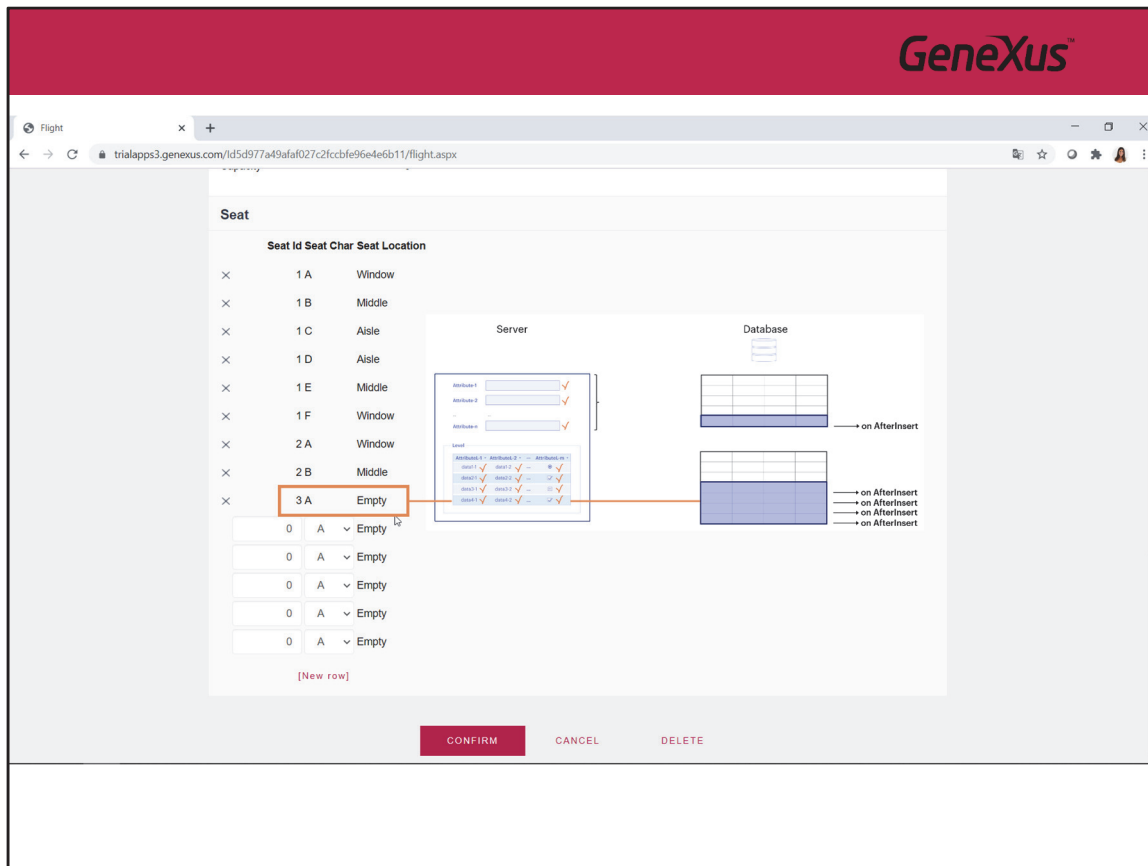
Seat Id	Seat Char	Seat Location
X	1 A	Window
X	1 B	Middle
X	1 C	Aisle
X	1 D	Aisle
X	1 E	Middle
X	1 F	Window
X	2 A	Window
X	2 B	Middle

0	A	▼	Empty
0	A	▼	Empty
0	A	▼	Empty

2 行目についても同様に処理されます。すべてのフィールドが検証され (SeatLocation の値は Empty)、対応する BeforeInsert ルールが実行されます。今度は 2 つ目のルールとなり、FlightSeatLocation の値が Middle に変更され、2 行目がすぐに FlightSeat テーブルに保存されます。ほかの行についても同様です。

位置が正しく割り当てられたことが分かります。

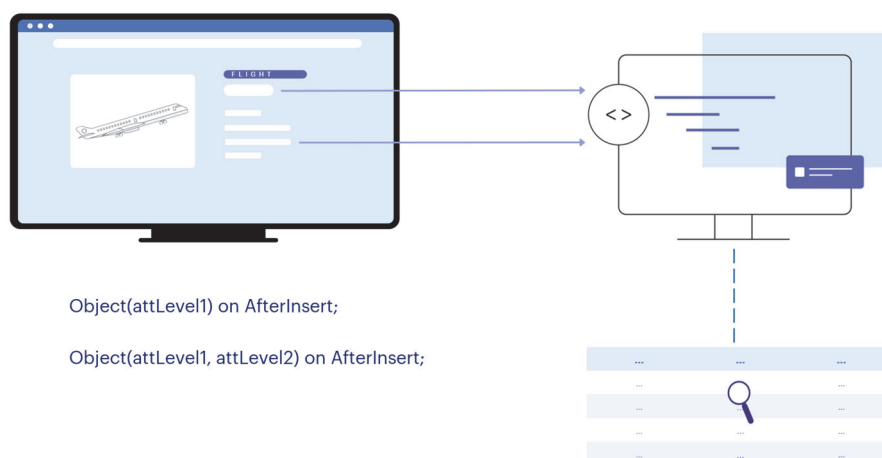
ここで、これらのルールの条件を on AfterInsert にできるかどうかを考えます。GeneXus で変更を行い、どうなるか見てみましょう。



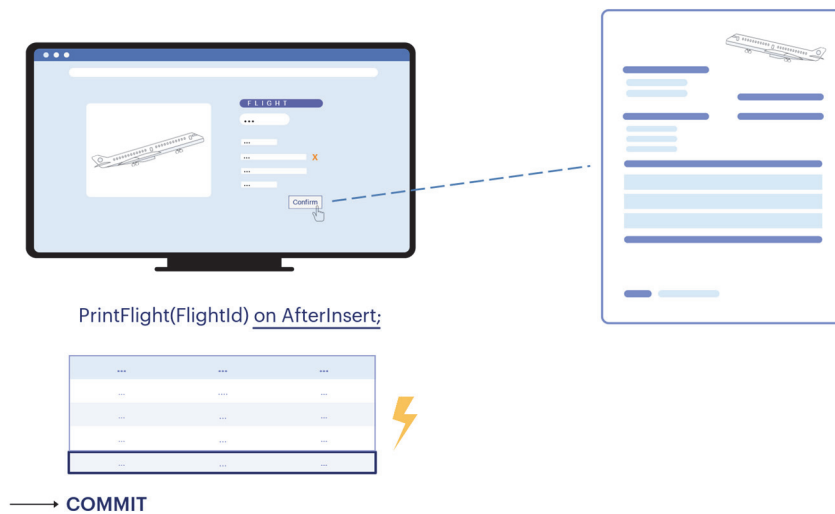
作成したフライトを変更し、新しい座席 3A を追加します。

フライトがどうなるか確認します。登録した座席は位置が Empty になっています。なぜでしょうか。

トリガーイベント on AfterInsert はレコードが対応するテーブルに保存された後に実行されるからです。これでは項目属性に値を割り当てるには遅すぎます。既に説明したように、値を割り当てることのできるタイミングは遅くとも BeforeInsert、つまりヘッダーまたは明細行が保存される前です。



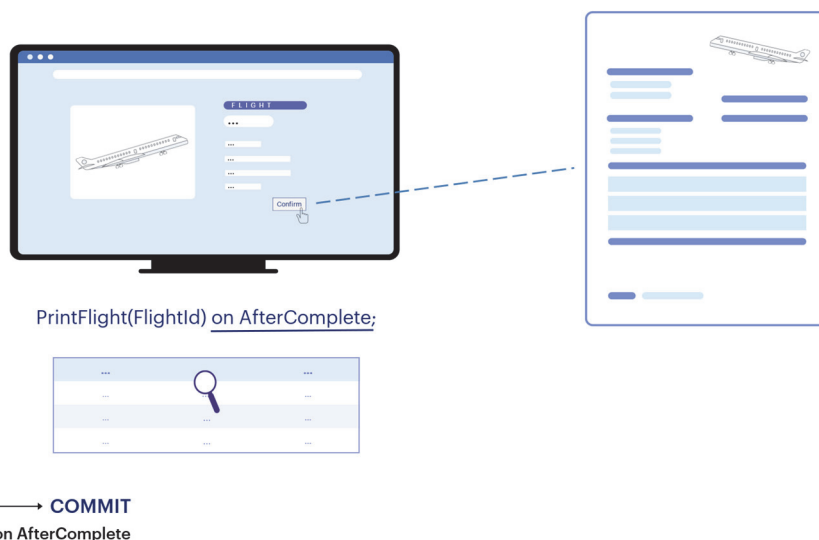
`on AfterInsert` の使用が必要な場合もあります。たとえば、ナレッジベースから別のオブジェクトを呼び出す場合、そのオブジェクトのヘッダーまたは行の識別子のみを送信して対応するテーブルにアクセスし、オブジェクトを見つけ、レコードから必要な情報をすべて抽出して必要な処理を実行できます。そのためには、ヘッダーまたは行の挿入後にオブジェクトを呼び出す必要があるため、`on AfterInsert` を使用します。



それでは、新しいフライトが挿入されるたびに、フライトの詳細情報を含むリストを出力する場合は on AfterInsert を使用すればいいでしょうか。

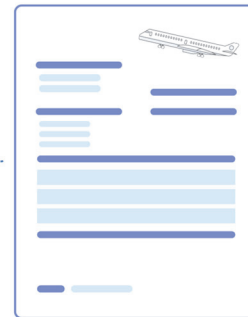
それも可能ですが、その場合、プロシージャはヘッダーデータのみを出力できます。明細行は変更も挿入もされていないからです。座席テーブルにはまだ存在しません。PrintFlight プロシージャがフライトのヘッダーのデータのみを出力するため、これが問題ではない場合でも、on AfterInsert での呼び出しは適切ではありません。その時点ではコミットの実行前であり、データはまだテーブルに確定されていないからです。

停電や、明細行の後続のフィールドで検証エラーが発生した場合は、記録が元に戻され、実際には存在しない情報でリストが生成されることになります。



データベースに確定済みの情報を確実に操作できるように、コミット後に実行される on AfterComplete イベントがあります。

この場合、送信される FlightId に対応するレコードが Flight テーブルで検索され、FlightId のすべてのレコード (つまりすべてのエントリー) が FlightSeat テーブルで検索されます。



PrintFlight(FlightId) on AfterLevel Level FlightSeatChar;

***	***	***
***	***	***
***	***	***
***	***	***

on AfterLevel

→ COMMIT

on AfterComplete

明細行の項目属性を AfterLevel で呼び出した場合はどうなるでしょうか。

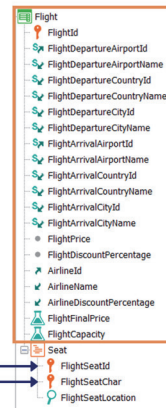
これはコミットの直前に発生するため、レコードはテーブルに保存されていますが、まだ確定されていません。このため、フライト情報の出力後に停電があり、コミットが実行されなかった場合、ヘッダーや行はデータベースから削除されます。



PrintFlight(FlightId, FlightSeatId, FlightSeatChar) on AfterComplete;

...
...
...
...

→ **COMMIT**
on AfterComplete



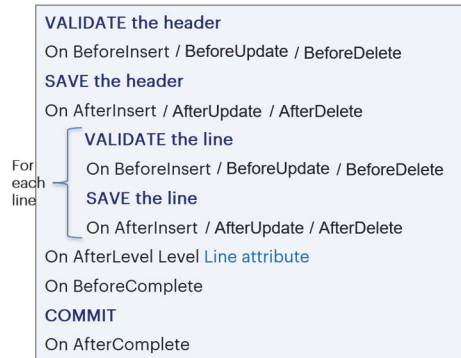
Only the header attributes are available in the on AfterComplete triggering event.

ここに示すルールが宣言されているとします。

第 2 レベルの座席数が N だった場合、GeneXus が送信する FlightSeatId と FlightSeatChar の値はどのようなのでしょうか。このルールは意味がなく、機能的に正しくありません。

AfterComplete のタイミングでは、ヘッダーの項目属性の値はまだメモリー内にあります。これに対して第 2 レベルの項目属性は、既に離れているため値が失われています。

Rule Triggering Events



この章では、BeforeInsert と AfterInsert のタイミングについて、例を用いて説明しました。これらはレコードを挿入する場合にのみトリガーされますが、レコードを更新する場合は BeforeUpdate と AfterUpdate、削除する場合は BeforeDelete と AfterDelete もあります。

ほかにもトリガーのタイミングはありますが、このコースでは取り上げません。関心がある場合は、次のレベルのコースでこのトピックについて学習できます。