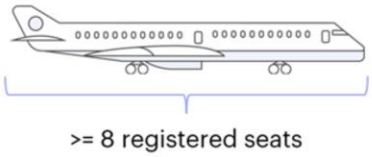
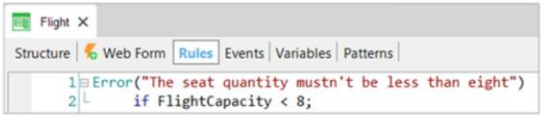
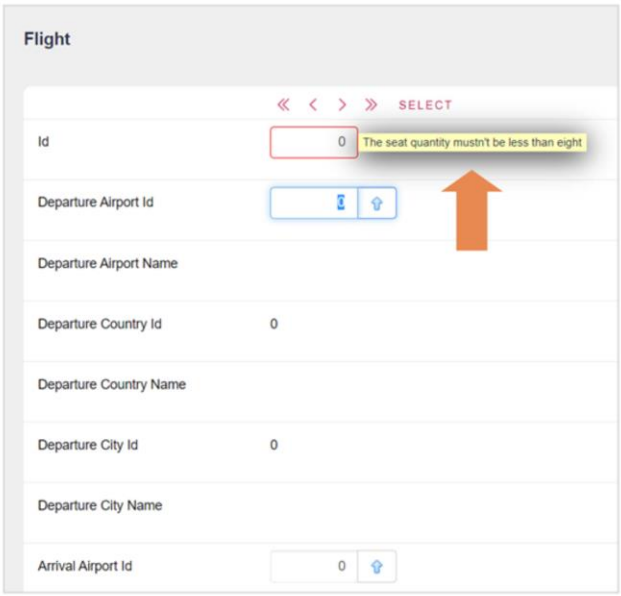
Eventos de disparo de regras em transações

Continuação

GeneXus

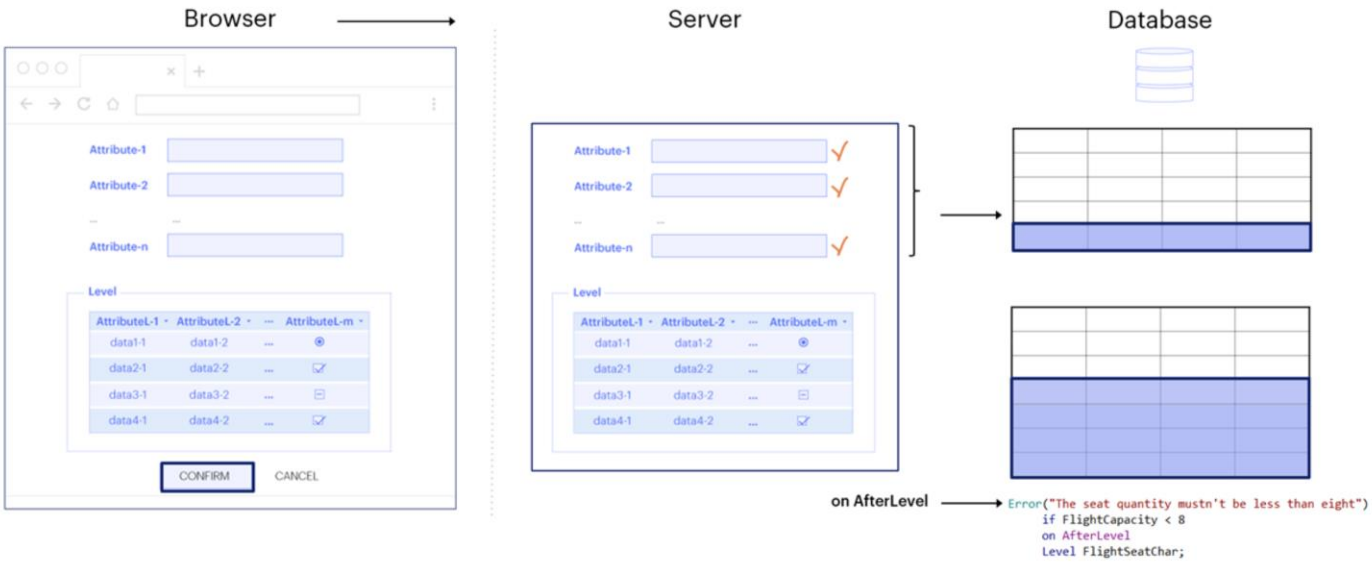






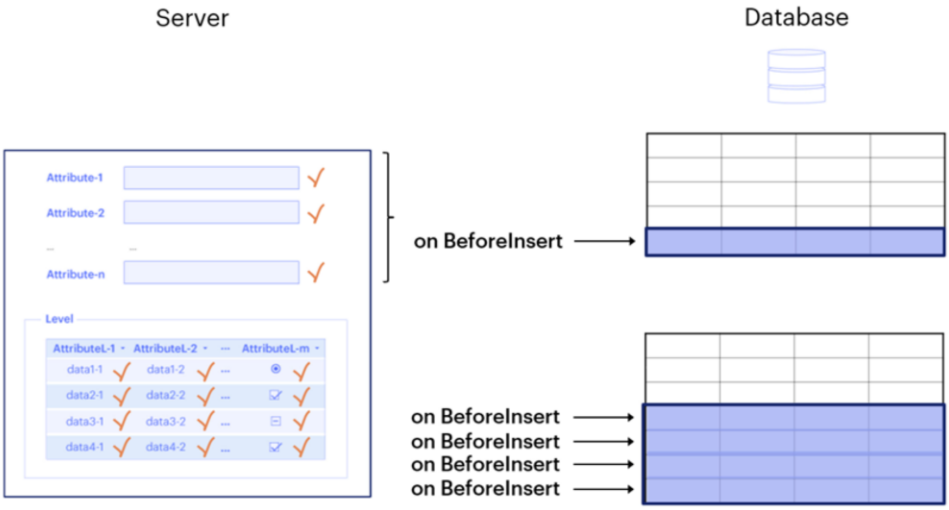
No vídeo anterior vimos que há casos em que o momento escolhido por GeneXus para executar uma regra não é o que necessitamos, por isso devemos poder indicar à regra qual é o momento adequado.
Estudamos o caso em que necessitávamos controlar que cada voo tivesse ao menos 8 assentos registrados; como GeneXus executava a regra antes de dar tempo ao usuário para que registrasse os assentos,

AfterLevel event



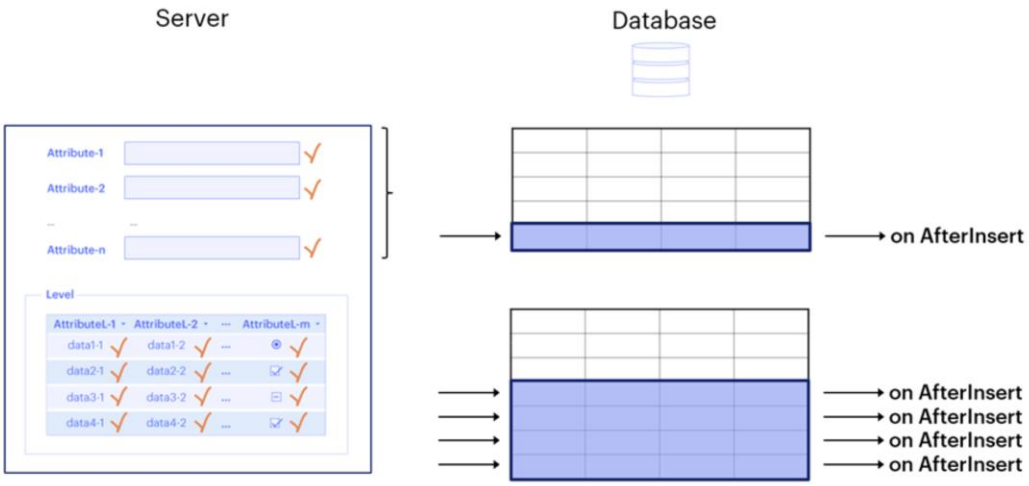
devemos atrasar o momento que tinha escolhido inicialmente para dispará-la, utilizando o momento de disparo ***on AfterLevel*** com um atributo das linhas (*FlightSeatChar*).
Com isto conseguimos que a regra se disparasse depois de percorrer o nível associado aos assentos.

BeforeInsert event



Mencionamos também que dispomos de outros momentos de disparo, como "**on BeforeInsert**" se quisermos fazer ou avaliar algo **imediatamente antes** de que os dados do cabeçalho ou de cada linha sejam inseridos na base de dados,

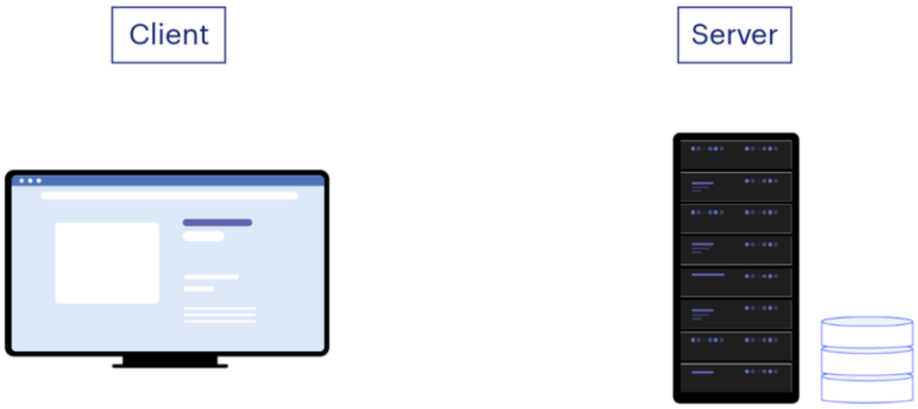
AfterInsert event



o momento "**on AfterInsert**" para indicar que a regra seja disparada **imediatamente após** a inserção de cada cabeçalho ou linha,

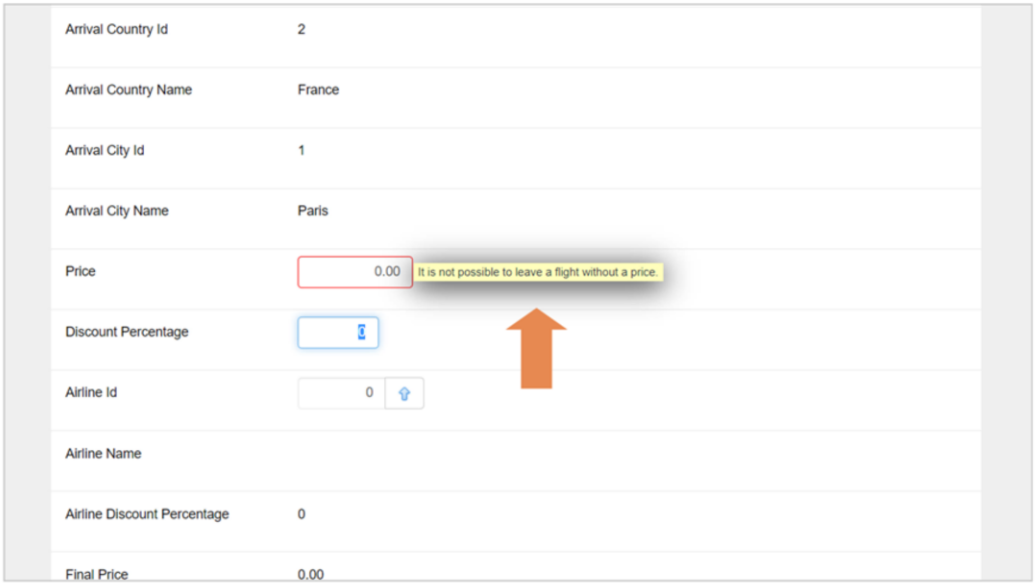
AfterComplete event

e o momento “on AfterComplete”, que corresponde ao instante de tempo imediatamente posterior ao Commit, comando cujo objetivo é confirmar os dados inseridos, modificados ou apagados.

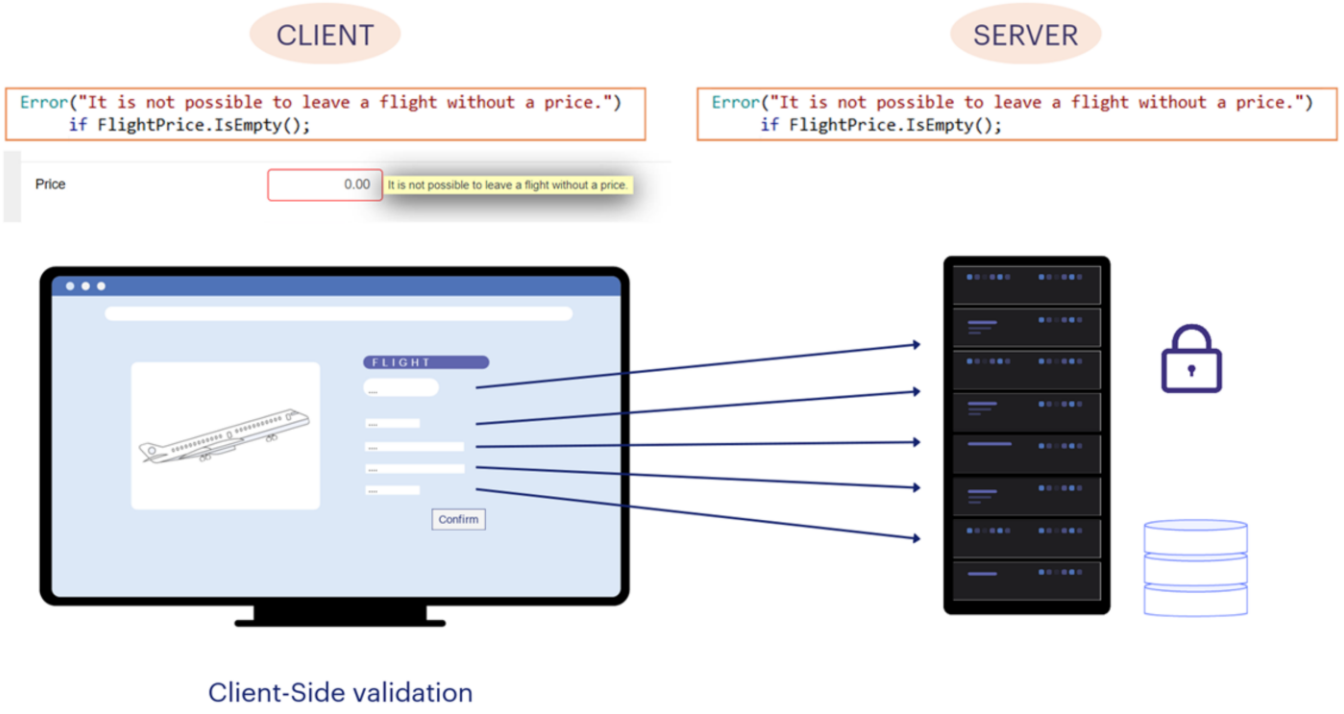


Neste vídeo, analisaremos com mais profundidade estes momentos de disparo. Mas, antes, lembremos que há regras que são validadas tanto no cliente (browser) como no servidor, e outras que o são exclusivamente no servidor, porque têm a ver com a base de dados.

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



Por exemplo: se tivéssemos uma regra Error que impedisse de inserir ou modificar um voo sem preço, logo que saíssemos desse campo apareceria a mensagem de erro.



Client-Side validation

Isto ocorre porque o cliente realiza a validação para proporcionar uma boa experiência ao usuário, dando-lhe respostas de forma ágil para que sinta que sua interação com o sistema é fluida. Esta validação realizada do lado do cliente, chama-se Client-Side Validation.

Mas esta regra voltará a ser executada posteriormente no servidor, já que é ele quem se encarrega de que não haja violações de segurança, e é o único que tem permissão de operar sobre a base de dados, devendo assegurar-se de que toda a lógica seja consistente. Executará todas as regras novamente, tendo toda a informação. Isso acontece depois que o usuário pressione o botão Confirm.
Ali os dados viajam do cliente web (browser) ao servidor web, e este volta a percorrer o form de cima a baixo, como se fosse um usuário, por isso todas as regras e fórmulas voltarão a ser disparadas na ordem que corresponde aos atributos do form aos quais estão associadas.

Mas, além disso, todas as regras que tenham um evento de disparo associado (isto é, que contem com o prefixo ***on*** ao final de sua declaração), serão executadas no momento correspondente a esse evento. Esses eventos só ocorrem no servidor e capturam o momento anterior ou posterior a um marco, que em geral tem a ver com a base de dados.

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:

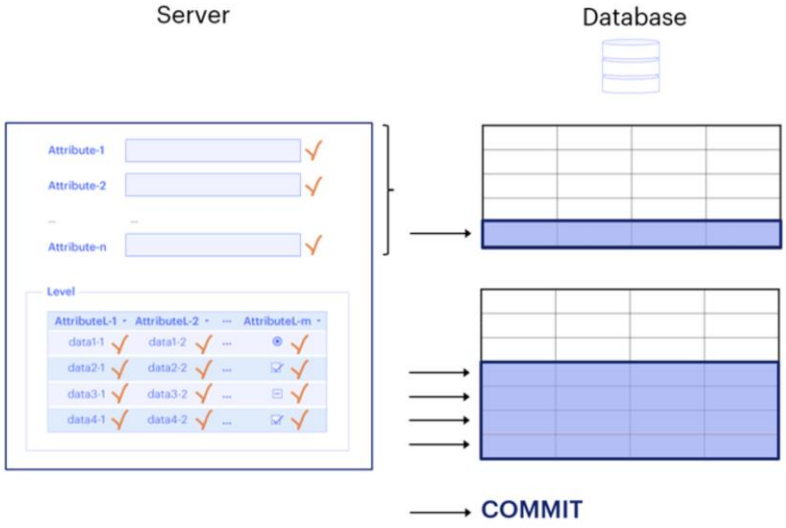
- Validate the line record to be inserted
- Insert the line record in the table

The next milestone is when:

- You finish working with the level

And the next one is:

- The Commit action



Quais são esses marcos?
Pensemos, para simplificar, que estamos inserindo um voo. Em ordem:

- Considerar válido o registro do cabeçalho que será inserido (assegura que os controles de integridade referencial e de duplicados não falharam; ocorre depois de ter passado por cada campo do cabeçalho e de ter disparado cada regra associada, ao final de tudo isso)
- Inserir o registro do cabeçalho na tabela correspondente

E depois o mesmo para cada linha:

- Considerar válido o registro da linha que será inserida (assegura que os controles de integridade referencial e de duplicados não falharam)
- Inserir o registro da linha na tabela correspondente

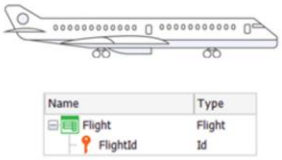
Ao finalizar este processo, onde tanto o registro correspondente ao cabeçalho como os correspondentes a todas as linhas foram inseridos na base de dados, o marco seguinte é quando:

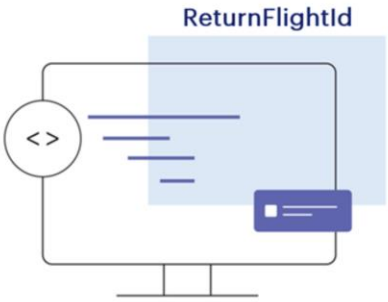
- Termina-se de trabalhar com o nível

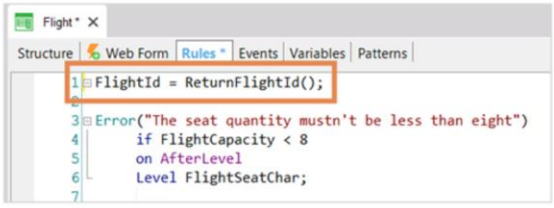
E o seguinte é:

- A ação de Commit, que considera corretas todas essas operações na base de dados, as deixa permanentes.

Então, temos eventos que capturam o momento anterior ou posterior a estes marcos, de forma a poder executar regras nesses momentos específicos.





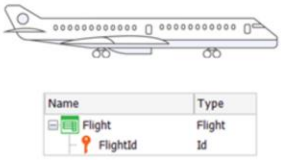


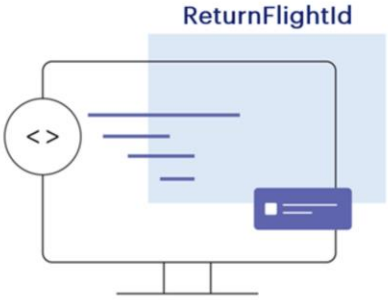
Atualmente, na transação Flight, definimos o identificador de voo (FlightId) como autonumerado.
Se, na realidade, como costuma ser o caso, o identificador de voo é composto por letras e números em função da companhia aérea, da origem e do destino, não nos serve o atributo FlightId numérico, nem autonumerado.

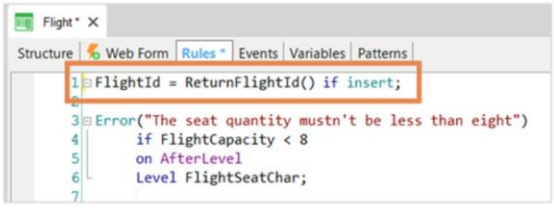
Suponhamos, então, que temos um procedimento chamado ReturnFlightId, ao qual podemos invocar para que se execute e nos devolva o identificador a ser atribuído a um novo voo.

(Estudaremos o objeto procedimento mais adiante, por isso você não poderá, por enquanto, reproduzir o que mostraremos aqui).

Se escrevermos a regra que vemos acima, onde estamos chamando-o e atribuindo seu resultado ao atributo FlightId, em que momento será disparado?
Assim que a transação for aberta ou se deseje editar um flight preexistente. Não condicionamos o disparo da regra ao modo, pelo que será executada quer o usuário esteja acessando à transação Flight para inserir, ou para modificar, ou para eliminar um voo, quando na realidade o que queremos é que o procedimento seja invocado somente se estivermos inserindo um voo novo.

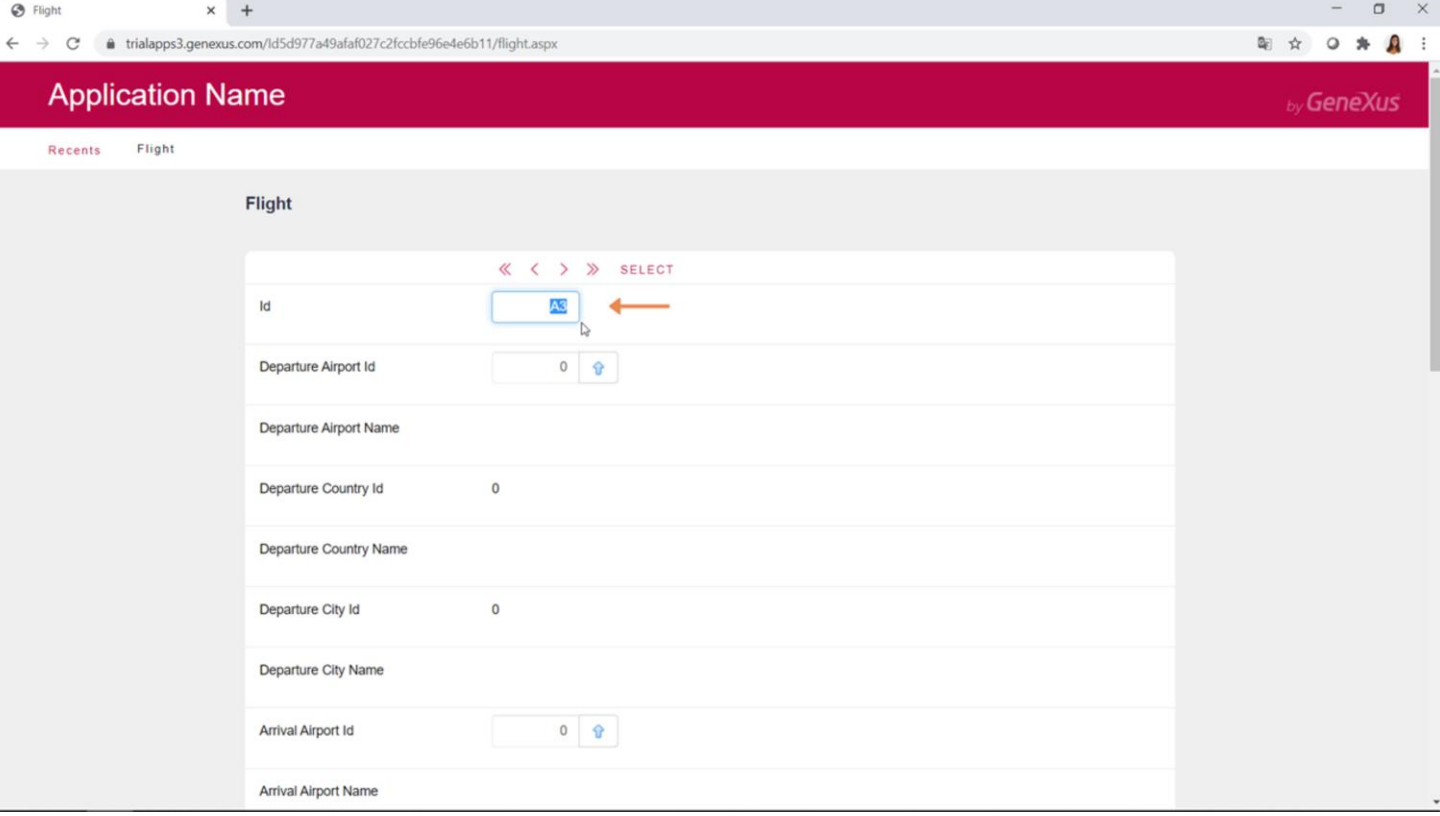






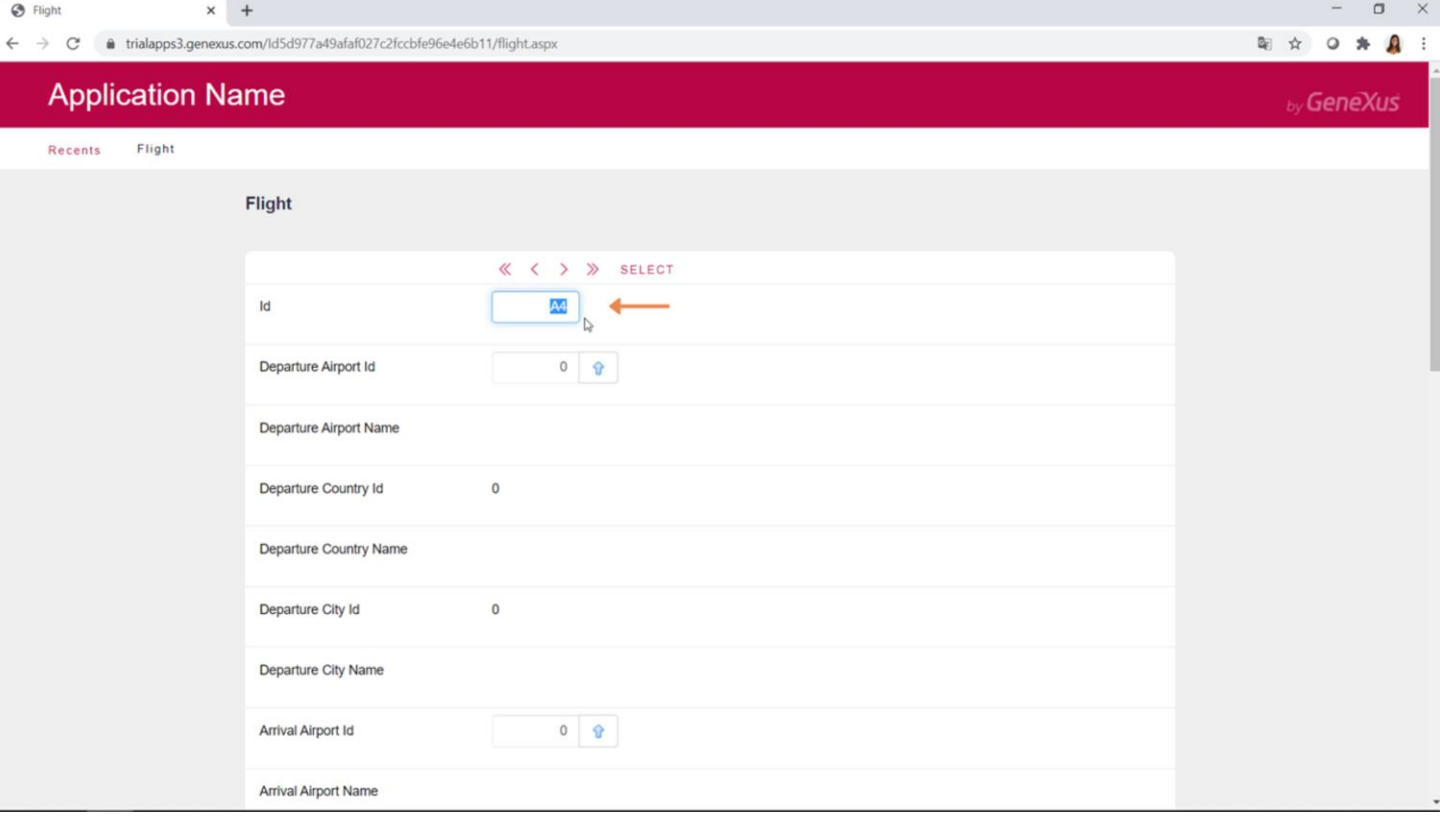
Além disso, convém esclarecer que se estivéssemos na transação Flight em modo Update (ou seja, se o voo já existisse e estivéssemos apenas modificando-o), não poderíamos modificar o valor do identificador de voo, já que as chaves primárias não podem ser modificadas. Se precisamos modificar uma chave primária, não teremos escolha a não ser criar um novo registro com o novo valor de chave, e apagar o anterior.

Assim, condicionamos a regra para que se execute somente quando se está querendo inserir.

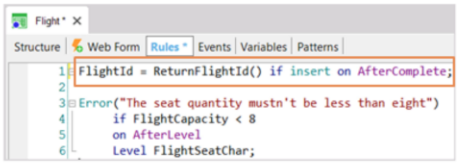


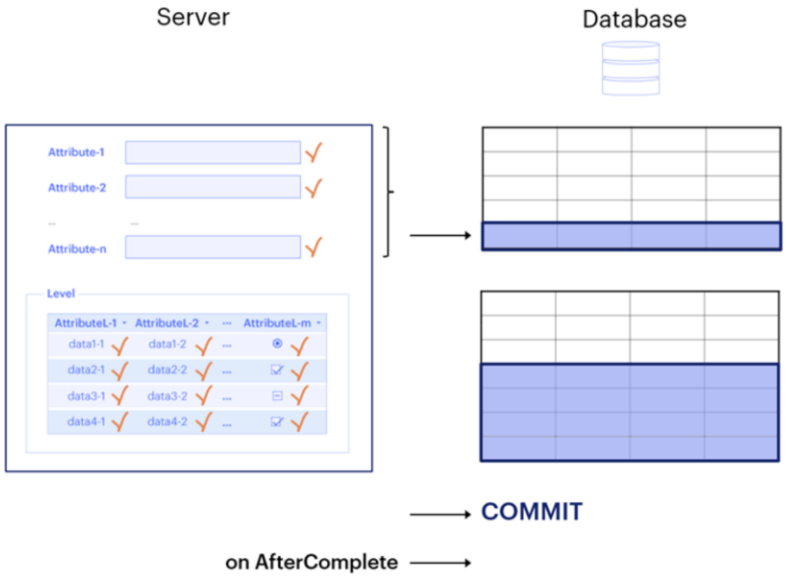
Vemos que como a transação se abre nesse modo, já está sendo disparada a regra que está atribuindo valor ao atributo, antes de ter feito qualquer coisa.

Mas poderia acontecer que tendo completado os dados do cabeçalho e alguma linha, tivéssemos que cancelar a inserção por alguma razão, para fazê-la mais tarde.

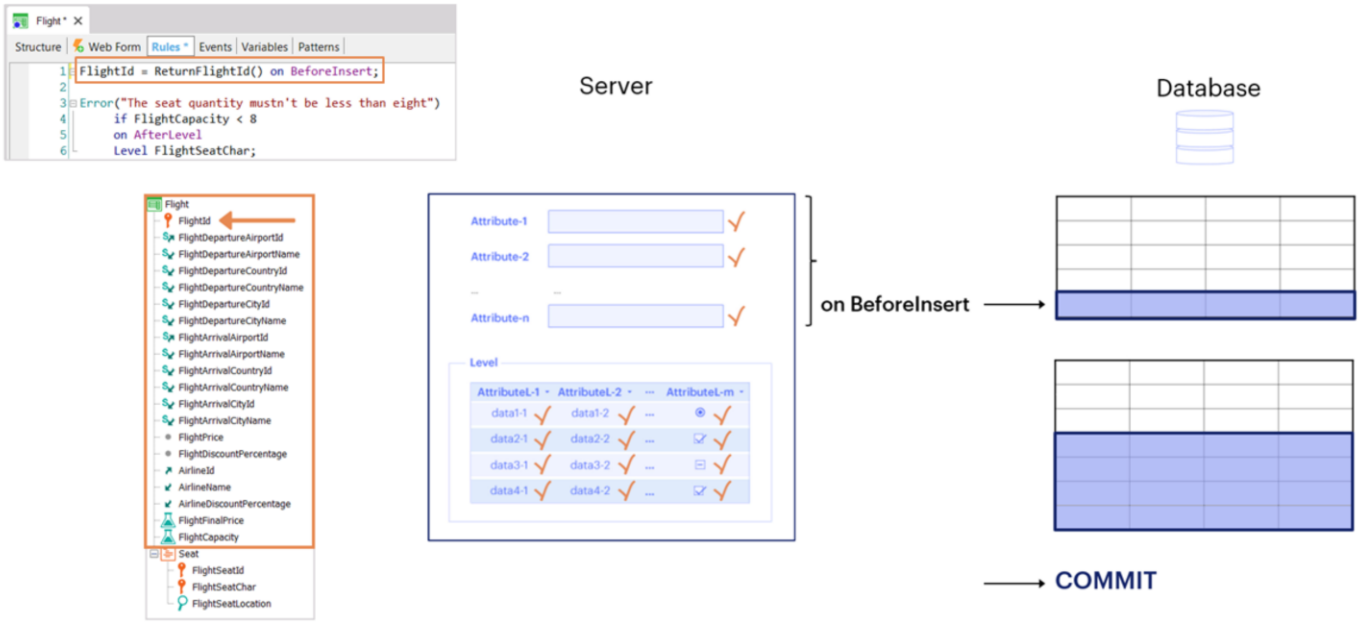


Quando voltamos a tentar, vemos que nos dará outro número de voo. Ocorre que tínhamos pedido um número ao procedimento que no final não usamos, e esse número ficou perdido.

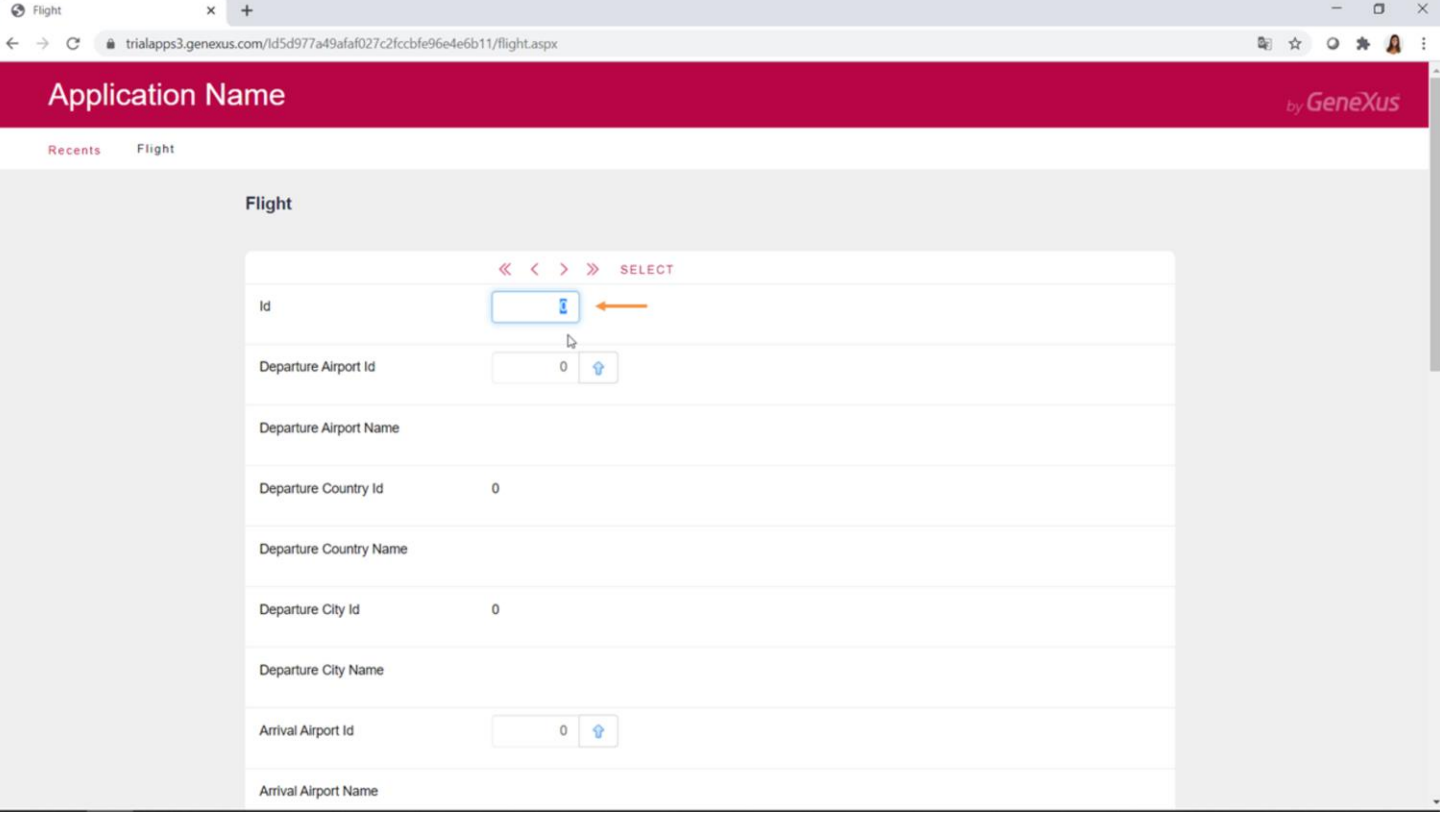




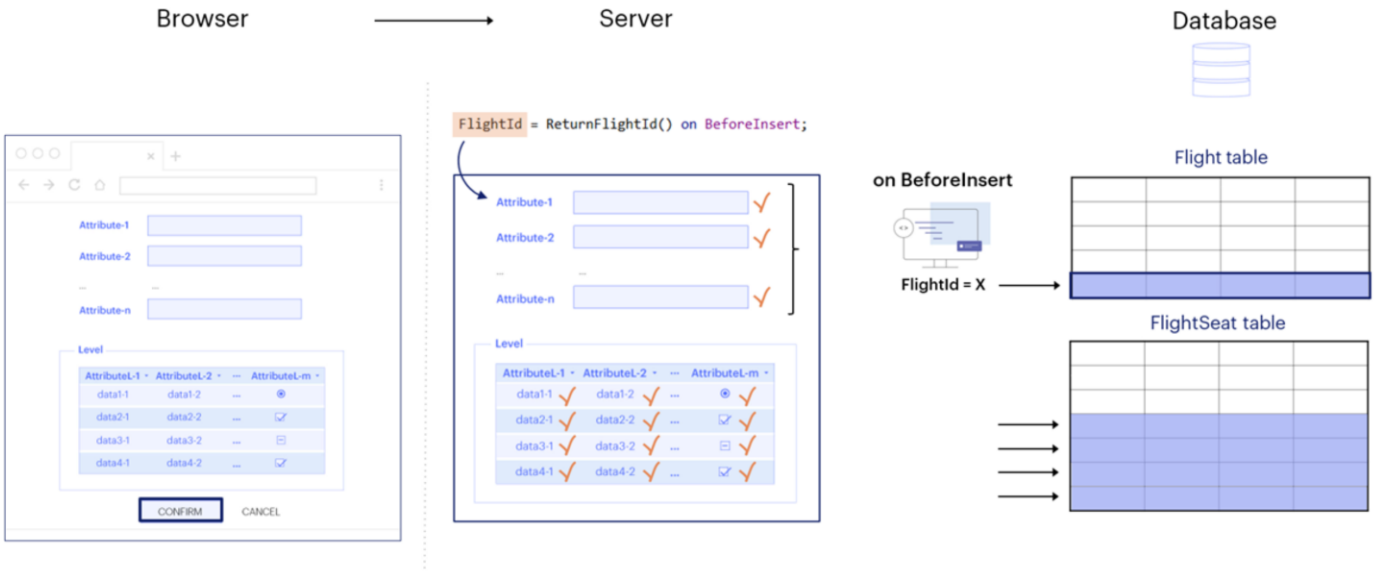
Para garantir de não gastar um número que não usaremos, que tal se chamamos o procedimento que nos dá o número depois do Commit, porque ali estamos completamente seguros de que tudo ficará na base de dados?



Hmmm, não, é muito tarde! Por quê? Porque FlightId é um atributo de cabeçalho. O último momento para lhe dar valor é o imediatamente anterior ao qual se insere o registro na base de dados, e esse momento é o BeforeInsert. Já não precisamos condicionar a If Insert, pois já está contemplado no evento de disparo, que só ocorrerá se estiver querendo inserir:



Se executamos, vemos que já não atribui de início o número ao Id. Nem sequer o faz quando passamos a inserir linhas.

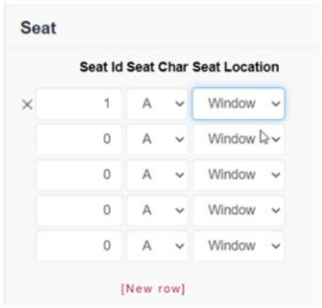


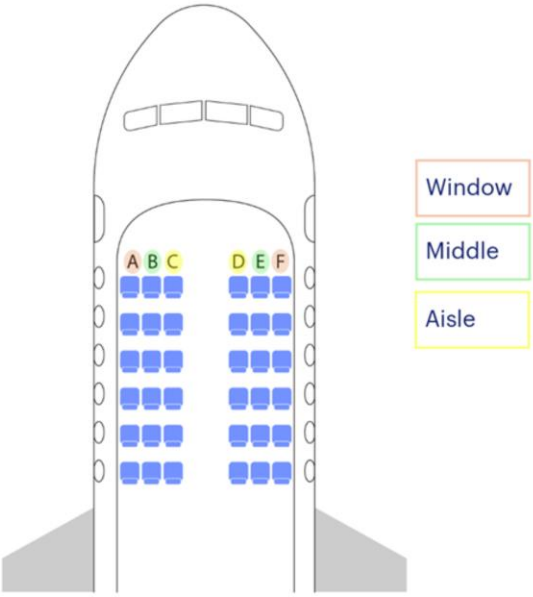
Apenas quando confirmamos, todas estas informações viajam para o servidor que começa a executar todos os controles de integridade referencial e regras à medida que vai validando cada campo e quando termina com os do cabeçalho, depois de ter validado tudo, ocorre o evento BeforeInsert. Ali atribui o número e, imediatamente depois, insere o registro na tabela Flight. Depois passa a validar cada campo da linha 1, e depois a insere; os da linha 2 e a insere e assim por diante.

Observe que não será disparada a regra antes de gravar cada linha. Esta regra somente será disparada no BeforeInsert do cabeçalho! Por quê? Porque na regra que definimos está participando um atributo pertencente ao cabeçalho.

Se definimos uma regra à qual lhe incluímos também o evento de disparo on BeforeInsert, mas diferente do exemplo que acabamos de ver, é referenciado na regra pelo menos um atributo do segundo nível da transação, a mesma estará associada ao segundo nível. Portanto, será executada imediatamente antes de ser gravada fisicamente cada instância correspondente ao segundo nível da transação.

Então, poderíamos dizer que, embora o nome do evento, BeforeInsert, é o mesmo, na verdade, são dois distintos: ou é o BeforeInsert do cabeçalho ou é o que aplica às linhas.

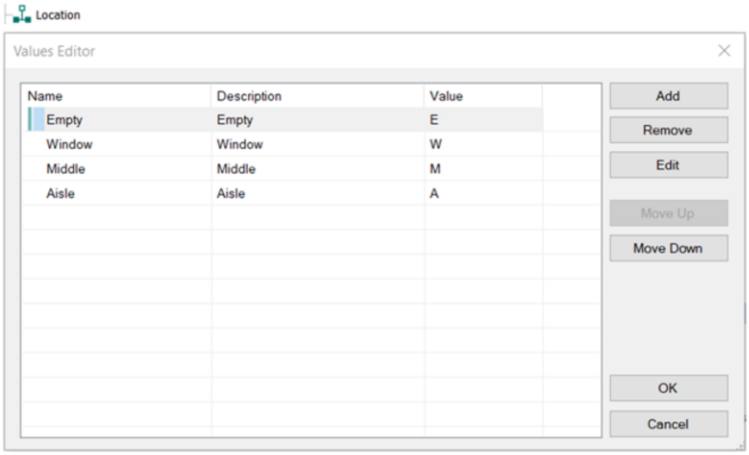




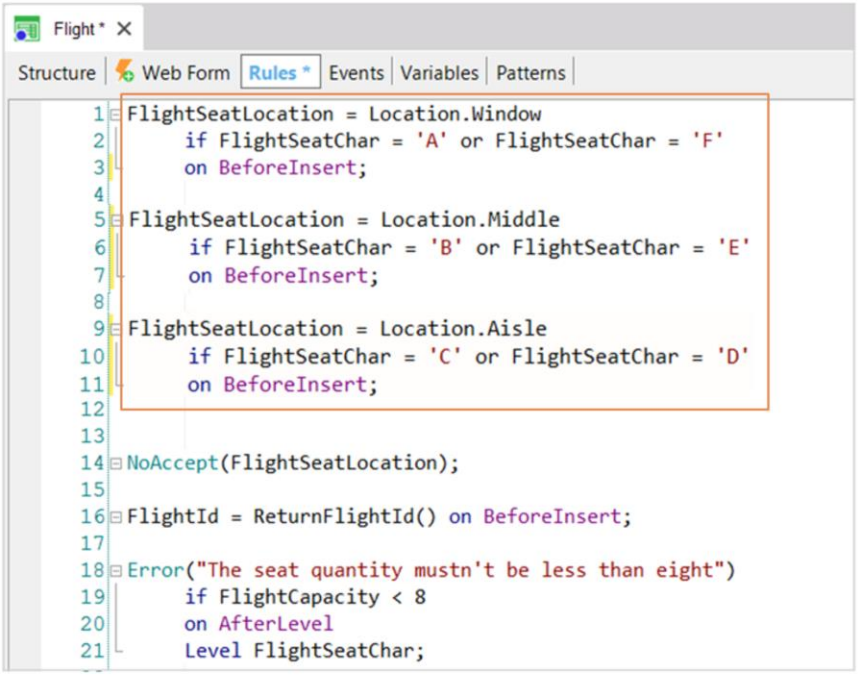
Vejamos um exemplo de BeforeInsert das linhas:

Suponhamos que queremos que o valor para o atributo FlightSeatLocation, em vez de ser escolhido pelo usuário que está trabalhando com a transação, seja atribuído por uma regra, coloque o valor "Janela" quando o valor do atributo FlightSeatChar é A ou F; "Meio" quando o valor de FlightSeatChar é B ou E; e "Corredor" quando o seu valor é C ou D.
Esclareçamos que isto poderíamos resolver sem necessidade de utilizar eventos de disparo, e até seria melhor, porque o usuário veria imediatamente na tela o valor do SeatLocation, já que a regra se executaria imediatamente no cliente, mas, só para compreender o momento do disparo, imaginemos que nos interessasse fazer esta atribuição apenas se tivermos a certeza de que o assento será inserido no voo, ou seja, imediatamente antes de inserir a linha.

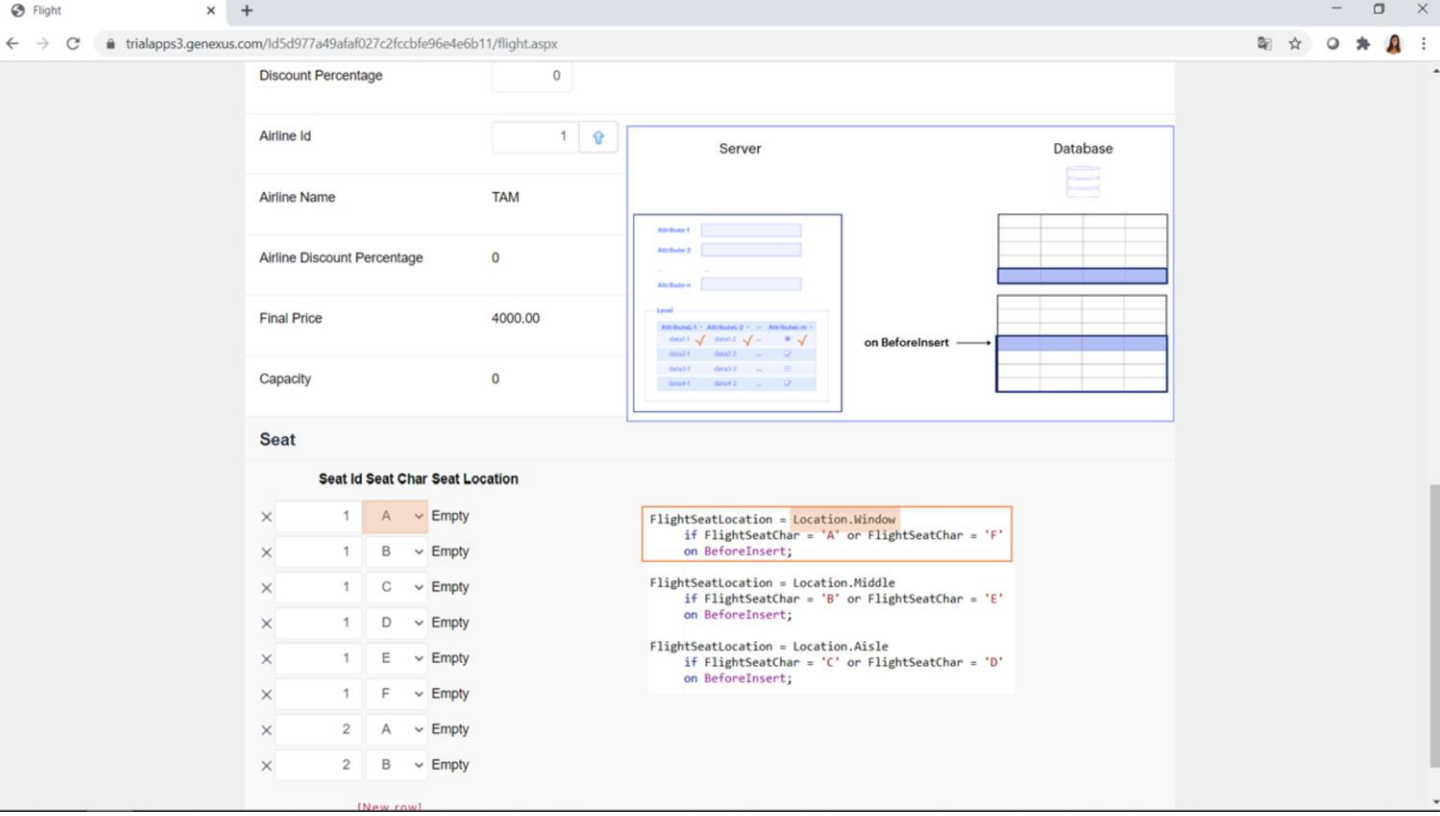

```
NoAccept(FlightSeatLocation);
```



Usaremos a regra NoAccept para impedir que o usuário escolha a localização, adicionaremos o valor "Empty" ao domínio Location,



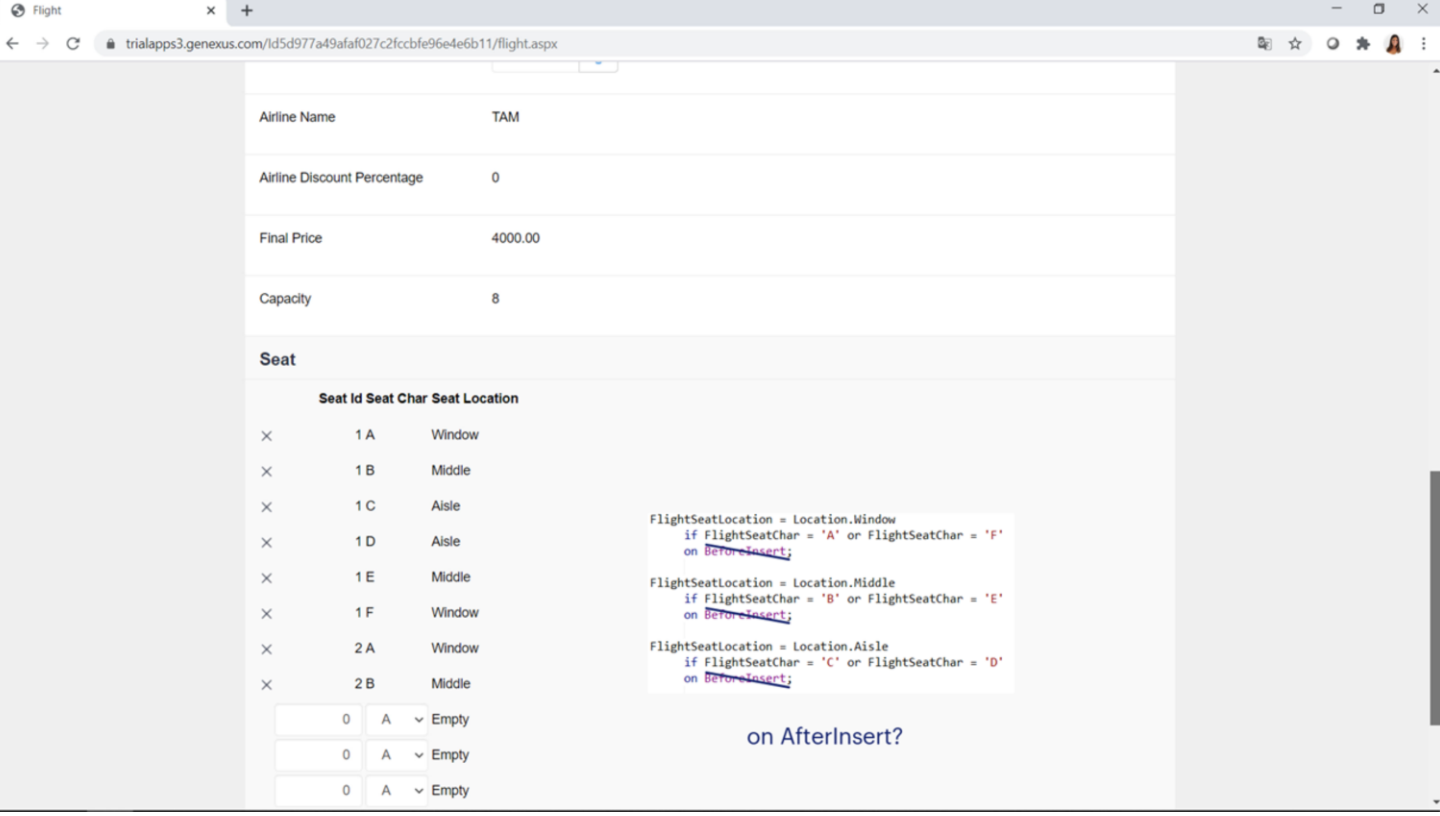
e criaremos as seguintes regras.



Vamos inserir um novo voo a partir do aeroporto de Guarulhos até o aeroporto Charles de Gaulle, custando 4000, sem desconto e companhia aérea TAM. Passemos agora a registrar seus assentos:
Completaremos a primeira fila, e mais dois assentos da segunda.

Antes da inserção física de cada linha, será executada a regra correspondente, segundo o FlightSeatChar que escolhemos.

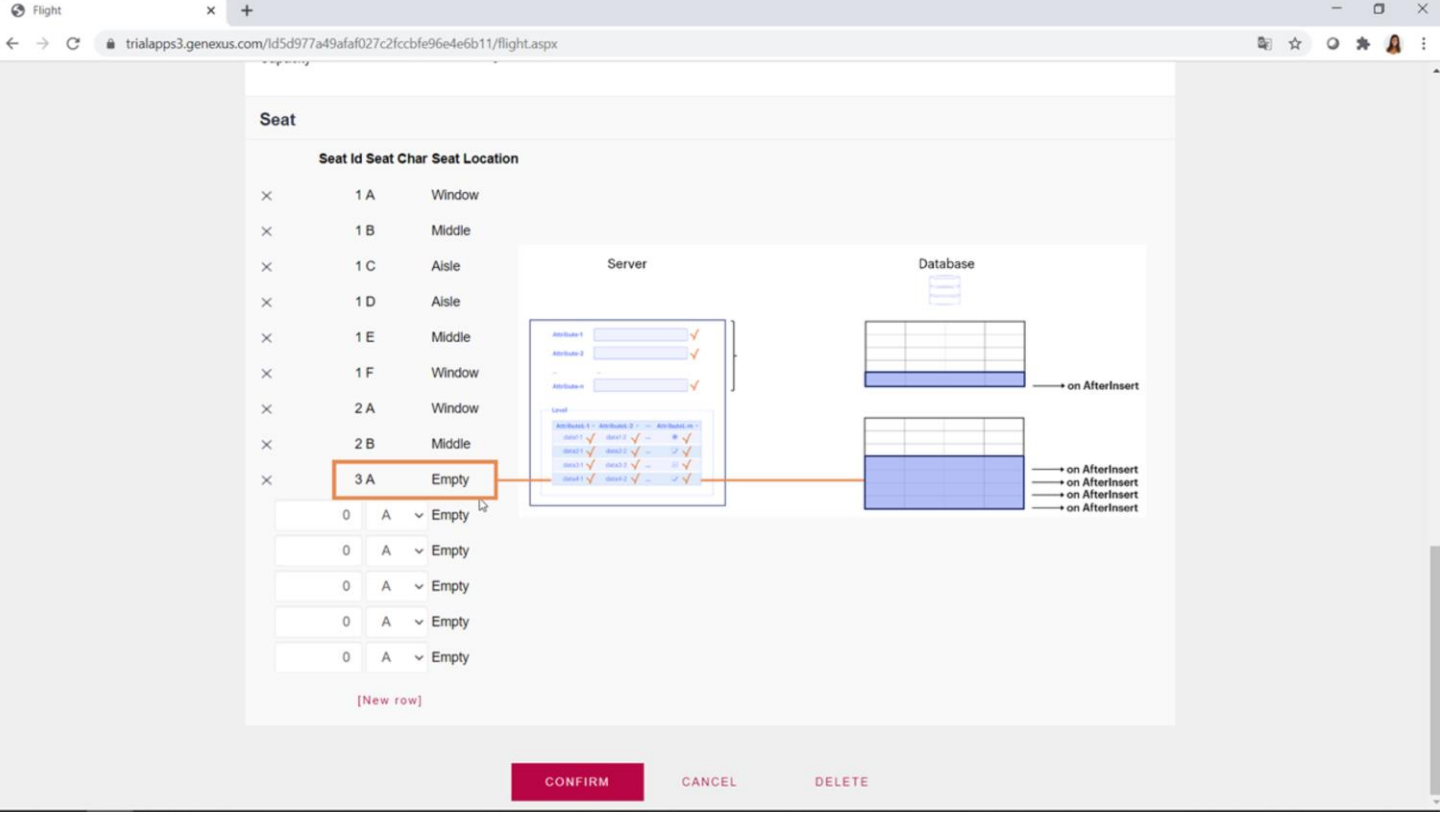
Assim, a transação no servidor irá validar os dados da primeira linha, onde SeatLocation ficará empty e, em seguida, executará a primeira regra, pois SeatChar é A, e então mudará o SeatLocation para Window, e imediatamente gravará a linha na tabela.



Em seguida, passará a fazer o mesmo com a segunda linha: validará todos os campos, SeatChar estará com valor Empty, e então executará as regras BeforeInsert que correspondam, que neste caso será a segunda, mudando o valor de FlightSeatLoaction para Middle, e imediatamente gravará a segunda linha na tabela FlightSeat. E assim seguirá com as demais linhas.

Vemos que a localização foi atribuída corretamente.

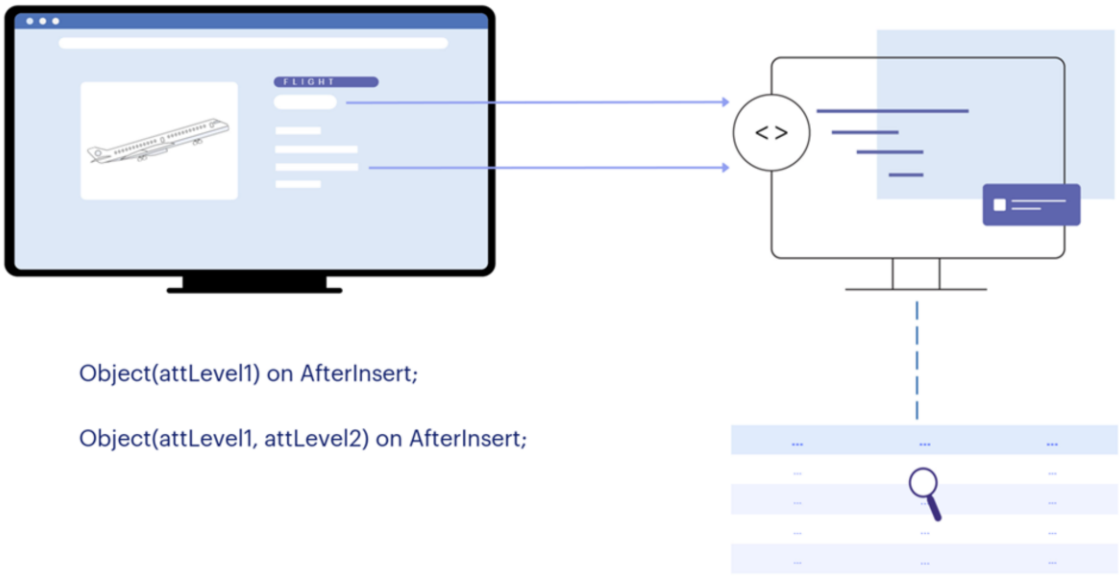
Mas... Poderíamos ter condicionado estas regras ao momento on AfterInsert?
Vamos fazer a mudança em GeneXus e vejamos o que acontece nesse caso:



Vamos modificar o voo que acabamos de criar, e adicionar um novo assento:
3 A.

Vejamos agora como ficou o voo: o assento que acabamos de registrar ficou com localização vazia! Por quê?

Porque o evento de disparo on AfterInsert é executado depois que o registro foi gravado na tabela correspondente, por isso já é tarde demais para atribuir um valor a um atributo.
Como já mencionamos, o último momento disponível para atribuir valores é o BeforeInsert, antes que o cabeçalho ou cada linha tenha sido gravada.

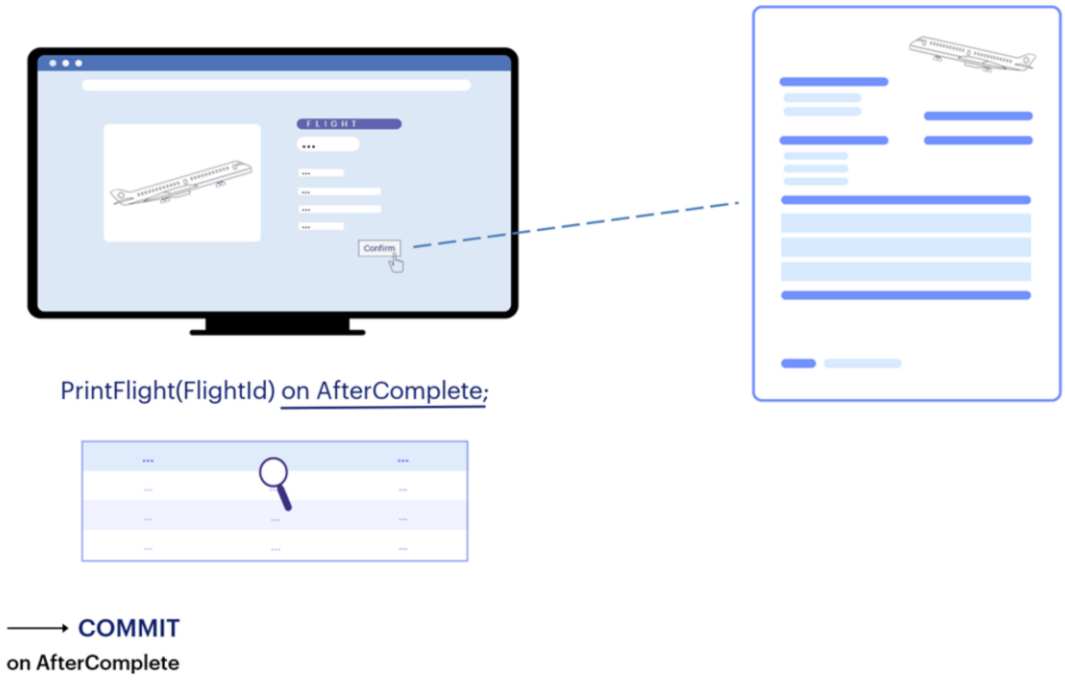


Portanto, poderíamos usar On AfterInsert se necessitássemos, por exemplo, chamar outro objeto da KB passando-lhe só o identificador do cabeçalho ou da linha para que esse objeto vá à tabela correspondente (a do cabeçalho ou a da linha) e ali o encontre (o cabeçalho ou a linha) e extraia dali toda a outra informação que necessite do registro, para fazer o que quer que seja que tenha que fazer com isso. Para isso, devemos ter certeza de chamar esse objeto depois de ter inserido o cabeçalho ou linha, conforme corresponda, portanto, on AfterInsert.

Então, se quiséssemos imprimir uma lista com os dados de um voo cada vez que se insere um novo, usaríamos on AfterInsert?

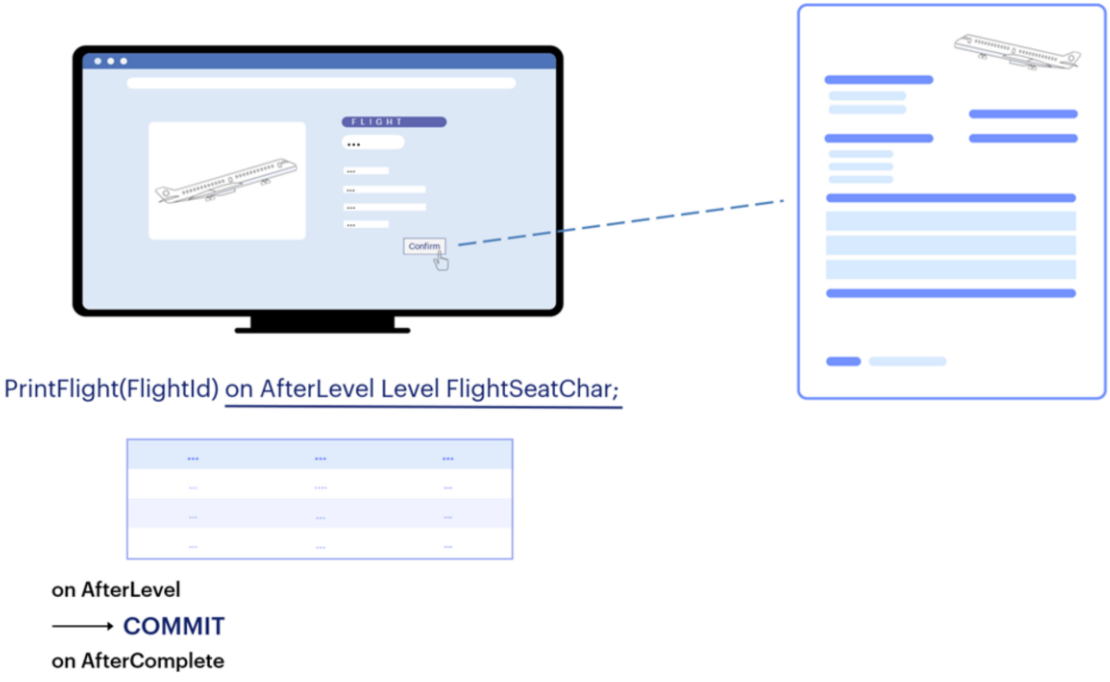
Poderíamos, mas aqui o procedimento só poderá imprimir a informação do cabeçalho, porque as linhas não foram manipuladas em absoluto e, muito menos, foram inseridas. Não existem ainda na tabela de assentos. Se isso não nos importasse, porque o procedimento PrintFlight só vai imprimir dados do cabeçalho do voo, também seria uma má ideia invocá-lo on AfterInsert, já que devemos levar em conta que nesse momento os dados ainda não estão seguros na tabela, porque não foi executado o Commit.

Isto significa que, se houvesse um corte de energia, ou um erro na validação de algum dos campos que seguem, ou seja, os das linhas, a gravação seria desfeita, então a lista teria sido gerada com informação que na realidade já não existe.



Para garantir que estamos trabalhando com informações que já estão seguras na base de dados, temos o evento on AfterComplete, que acontece depois do Commit.

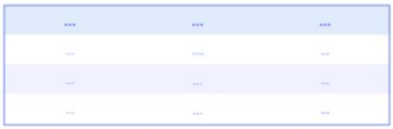
Neste caso, irá buscar na tabela Flight o registro correspondente ao FlightId enviado, e na tabela FlightSeat todos os registros desse FlightId, que são todos seus assentos.



O que aconteceria se invocássemos on AfterLevel de um atributo das linhas?

Isso será imediatamente antes do Commit, por isso os registos estarão gravados nas tabelas mas ainda não terão sido confirmados, então se quando acaba de imprimir a informação do voo há um corte de energia, e não chegou a realizar o Commit, e cabeçalho e linhas serão removidos da base de dados.

PrintFlight(FlightId, FlightSeatId, FlightSeatChar) on AfterComplete;



→ COMMIT
on AfterComplete

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

Agora, suponhamos que foi declarada a seguinte regra:

Que valores de FlightSeatId e de FlightSeatChar enviaria GeneXus, se no segundo nível há N assentos? Esta regra não tem sentido, é funcionalmente incorreta.

No momento AfterComplete, os atributos do cabeçalho ainda se encontram com seu valor em memória, ao contrário dos atributos do segundo nível, cujo valor se perdeu porque já saímos dele.

Rule Triggering Events

Neste vídeo mostramos exemplos dos momentos BeforeInsert e AfterInsert, que só serão disparados se estamos inserindo registros, mas também dispomos de BeforeUpdate e AfterUpdate para o caso em que estamos modificando-os, e BeforeDelete e AfterDelete, para o caso em que estamos eliminando-os.

Existem outros momentos de disparo que não estudaremos neste curso. Se está interessado, pode se aprofundar neste tema no curso do nível seguinte.

GeneXus™

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications