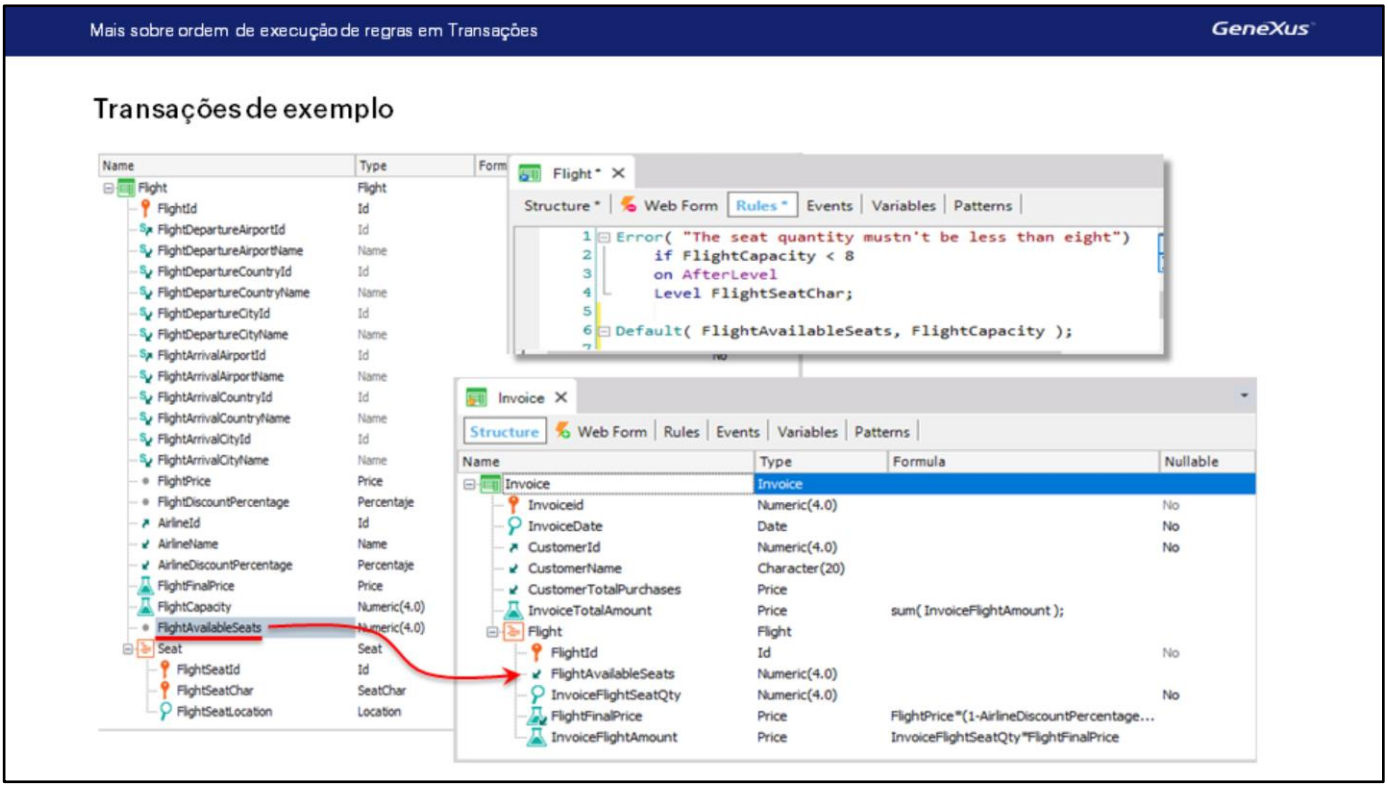
Mais sobre ordem de execução de regras em Transações

GeneXus 16



Daqui para frente abordaremos mais detalhadamente os momentos disponíveis para condicionar o disparo de regras, particularmente em transações de mais de um nível.

Para entender o tema, iremos nos basear na transação de Faturas (Invoice) que possui um segundo nível (Flight), representando os vôos incluidos na fatura.

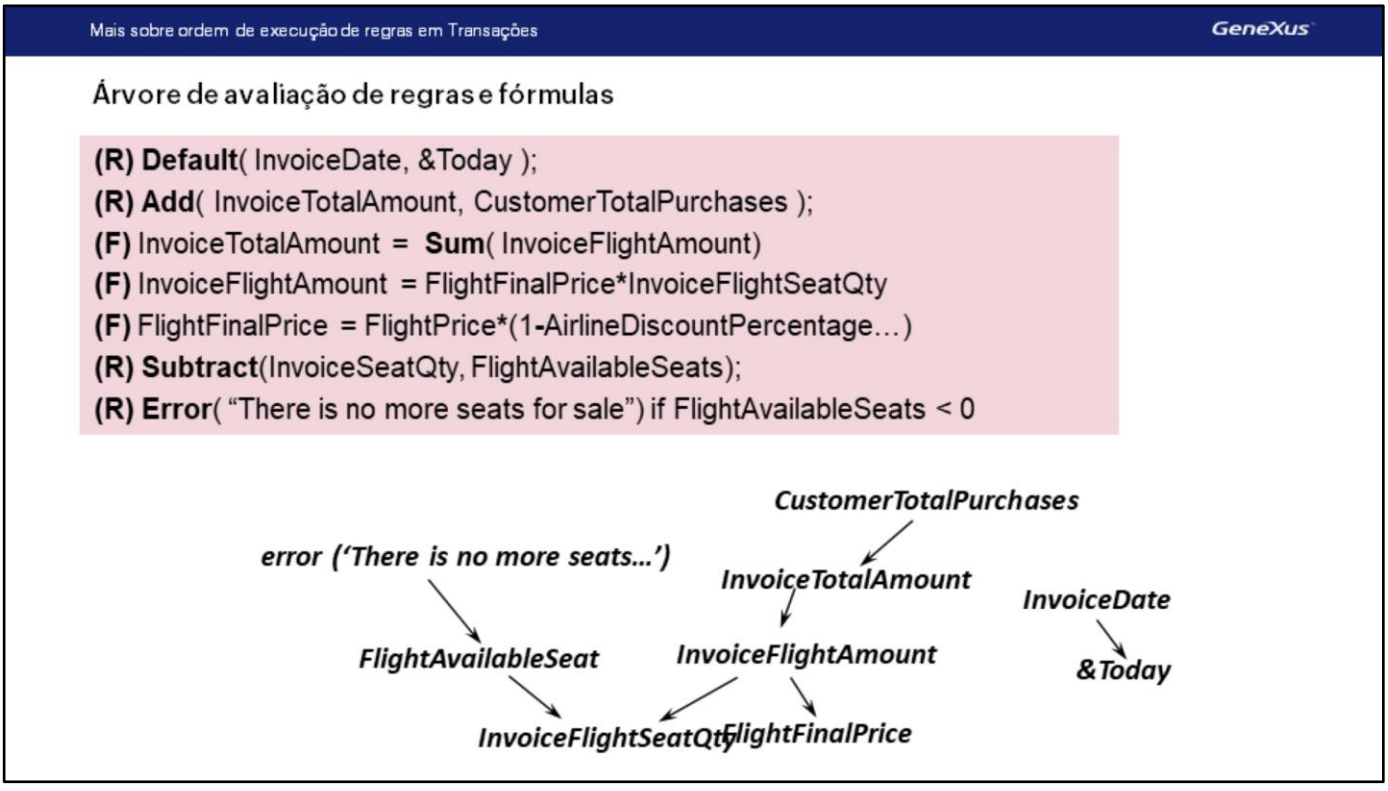
Observe que adicionamos o atributo **FlightAvailableSeats** na transação Flight, que será usado para registrar os assentos disponíveis de cada vôo, que irão diminuindo cada vez que é realizada uma fatura para um cliente que compra uma quantidade de lugares (assentos) em um vôo.

Nota: Também adicionamos o atributo CustomerTotalPurchases na transação Customer para registrar o total comprado pelo cliente em compras de passagens de vôos.

Observe que os atributos InvoiceTotalAmount, InvoiceFlightPrice e InvoiceFlightAmount da estrutura de Invoice são fórmulas. Em seguida, vamos ver as regras que definimos para Invoice, para especificar seu comportamento.

Regras da transação

Invoice	Invoice	
Invoiceid	Numeric(4.0)	
InvoiceDate	Date	
CustomerId	Numeric(4.0)	
CustomerName	Character(20)	
CustomerTotalPurchases	Price	
InvoiceTotalAmount	Price	sum(InvoiceFlightAmount);
Flight	Flight	
FlightId	Id	
FlightAvailableSeats	Numeric(4.0)	
InvoiceFlightSeatQty	Numeric(4.0)	
FlightFinalPrice	Price	FlightPrice*(1-AirlineDiscountPercentage...
InvoiceFlightAmount	Price	InvoiceFlightSeatQty*FlightFinalPrice



Para entender a ordem que GeneXus escolhe para disparar as regras e fórmulas, vamos considerar as definições de regras e fórmulas apresentadas na tela.

No momento de gerar o programa associado à transação Invoice, GeneXus irá extrair as dependências existentes entre as regras e fórmulas definidas e construirá, logicamente, uma árvore de dependências (ou árvore de avaliação) que determinará a sequência de disparo.

Podemos imaginar que o árvore será executada de baixo para cima, quer dizer que cada vez que o valor de um atributo é alterado, são executadas todas as regras e fórmulas que dependem desse atributo (e que, na árvore, se encontram para cima).

Por exemplo, se alterar a quantidade de assentos em uma linha de uma fatura (InvoiceFlightSeatQty), como este atributo interfere na fórmula que calcula o montante do vôo (InvoiceFlightAmount), essa fórmula será disparada. O mesmo aconteceria se alterassemos o preço final do vôo FlightFinalPrice, que também interfere na fórmula.

Ao alterar o montante de um vôo, deverá disparar também a fórmula correspondente ao total da fatura (InvoiceTotal) porque depende do valor de cada vôo da fatura. Por último, ao alterar o total, também será disparada a regra Add(InvoiceTotal, CustomerTotalPurchases) porque o total de compras do cliente deverá ser atualizado.

Além de serem disparadas todas as fórmulas e regras envolvidas no lado direito da árvore a partir do atributo InvoiceFlightSeatQty, também são disparadas as fórmulas e regras envolvidas no lado esquerdo. Quer dizer que, ao alterar o valor do atributo InvoiceFlightSeatQty, é disparada também a regra Subtract(InvoiceFlightSeatQty, FlightAvailableSeats) que atualiza a quantidade de assentos disponíveis no vôo (FlightAvailableSeats).

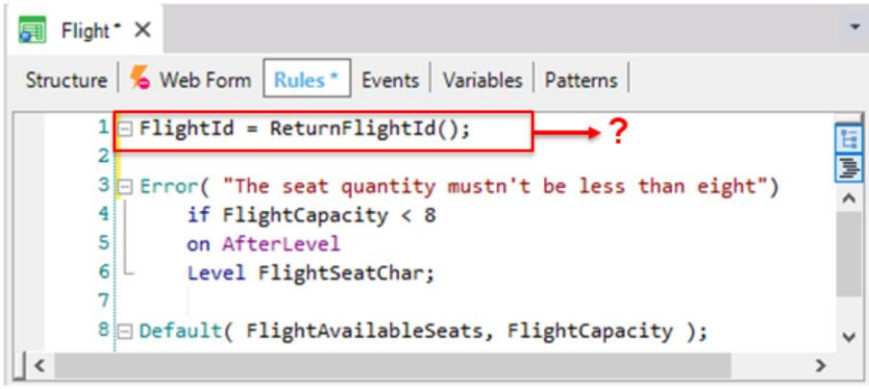
E consequentemente, ao modificar esta regra, o valor do atributo FlightAvailableSeats, será avaliado para ver se a regra Error('There is no more seats...') if FlightAvailableSeats< 0 será disparada ou não;

Concluindo, as regras e fórmulas definidas em uma transação geralmente estão interrelacionadas e GeneXus determina as dependências entre elas assim como sua ordem de disparo.

Revison eventos de disparo

- Geralmente as regras que definimos são executadas no momento que queremos.
- No entanto, em alguns casos, existe a necessidade de modificar o momento de disparo de uma regra.

Exemplo:



Na transação Flight, o identificador de vôo FlightId foi definido como auto-numérico.

Sabendo que, na realidade, o identificador do vôo é formado por letras que identificam a empresa e números, o atributo FlightId não pode ser numérico, nem auto-numérico.

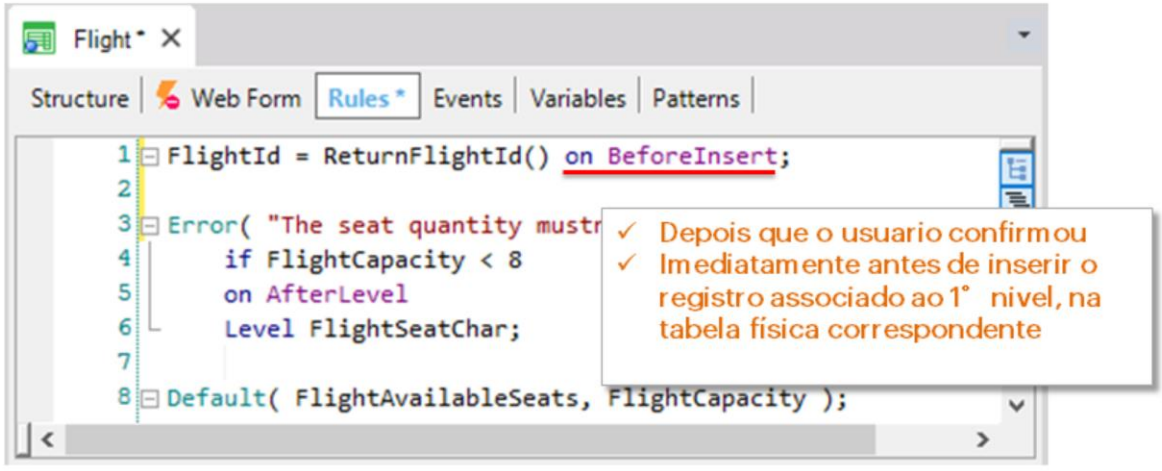
Vamos supôr que temos um procedimento, ReturnFlightId, responsável por retornar o identificador a ser atribuido para um novo vôo. (Na verdade, este procedimento deveria escolher o número de vôo de acordo com a companhia aérea, a origem e o destino, etc, porém, para simplificar não faremos isso).

Se escrevermos a regra que vemos acima, em que momento ela será disparada? No cabeçalho de Flight, para qualquer ação que estiver sendo realizada. Quer dizer que, se se modificarmos algo do vôo, o procedimento ReturnFlightId é chamado novamente para atribuir um novo identificador para o vôo, quando, na verdade, queremos que isso aconteça somente se estivermos inserindo um vôo novo.

Além disso, mesmo quando estivermos inserindo um vôo, deveríamos buscar um novo FlightId somente se confirmarmos a transação, já que pode acontecer de nós cancelarmos a inserção ou ainda, dar um erro que acarreta um cancelamento automático. Nestes casos, estaríamos "gastando" um novo número para o identificador do vôo, que não chegaria a ser usado.

Revison eventos de disparo

- Acrescentando um evento de disparo, modificamos o momento de execução por defecto da regra.

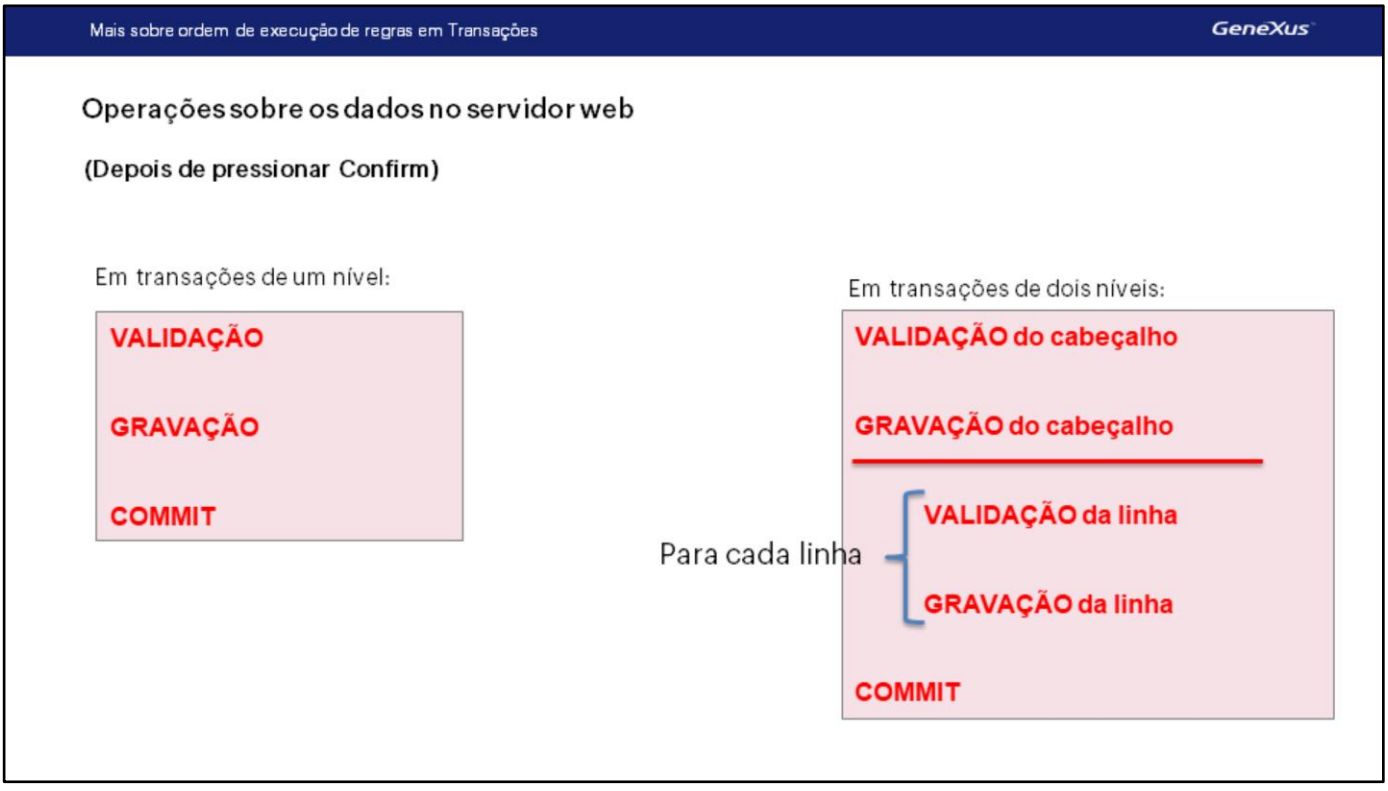


Para garantirmos, depois de confirmarmos a transação, que seja usado somente um número de id para o vôo e, além disso, que ele seja gerado somente quando estivermos inserindo um vôo, acrescentamos o momento de disparo on BeforeInsert na regra que alimenta o FlightId.

Este momento de disparo será executado depois que confirmarmos a transação, no servidor.

Além disso, before insert significa que será disparado depois que os dados do cabeçalho do vôo forem validados e imediatamente antes dos dados do cabeçalho serem gravados no banco de dados.

Veremos a seguir que existem alguns momentos chaves na operação do servidor, com relação ao processamento dos dados e o banco de dados.

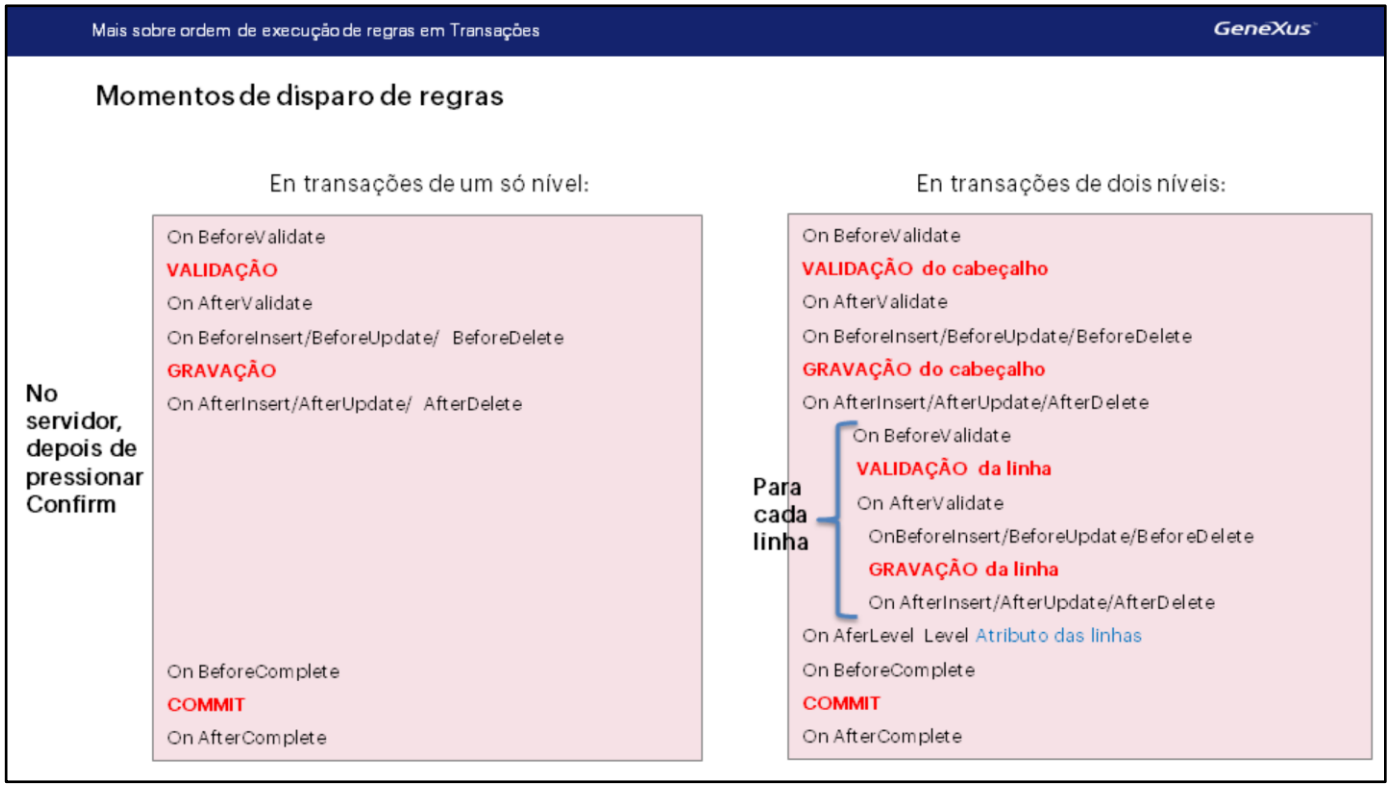


Depois de que o Confirm é pressionado, a informação viaja do cliente web (browser) até o servidor web.

No servidor, existem operações que são realizadas sobre os dados, que são:
- a Validação dos dados,
- a Gravação dos dados no banco de dados e
- o Commit no banco de dados.

Se a transação é de dois níveis, então depois da gravação do cabeçalho, será executado, para cada linha:
- a Validação e depois
- a Gravação da linha.

Por último, o Commit será efetuado, que irá consolidar os dados do cabeçalho e de todas as linhas da transação no banco de dados.



Uma vez que a informação da transação chega ao servidor web, as regras e fórmulas são disparadas novamente no server.

Nesta execução, temos disponíveis varios momentos relacionados às operações de Validação, Gravação e Commit dos dados no servidor. Estes momentos específicos podem ser incorporados nas regras, permitindo controlar quando estas devem disparar.

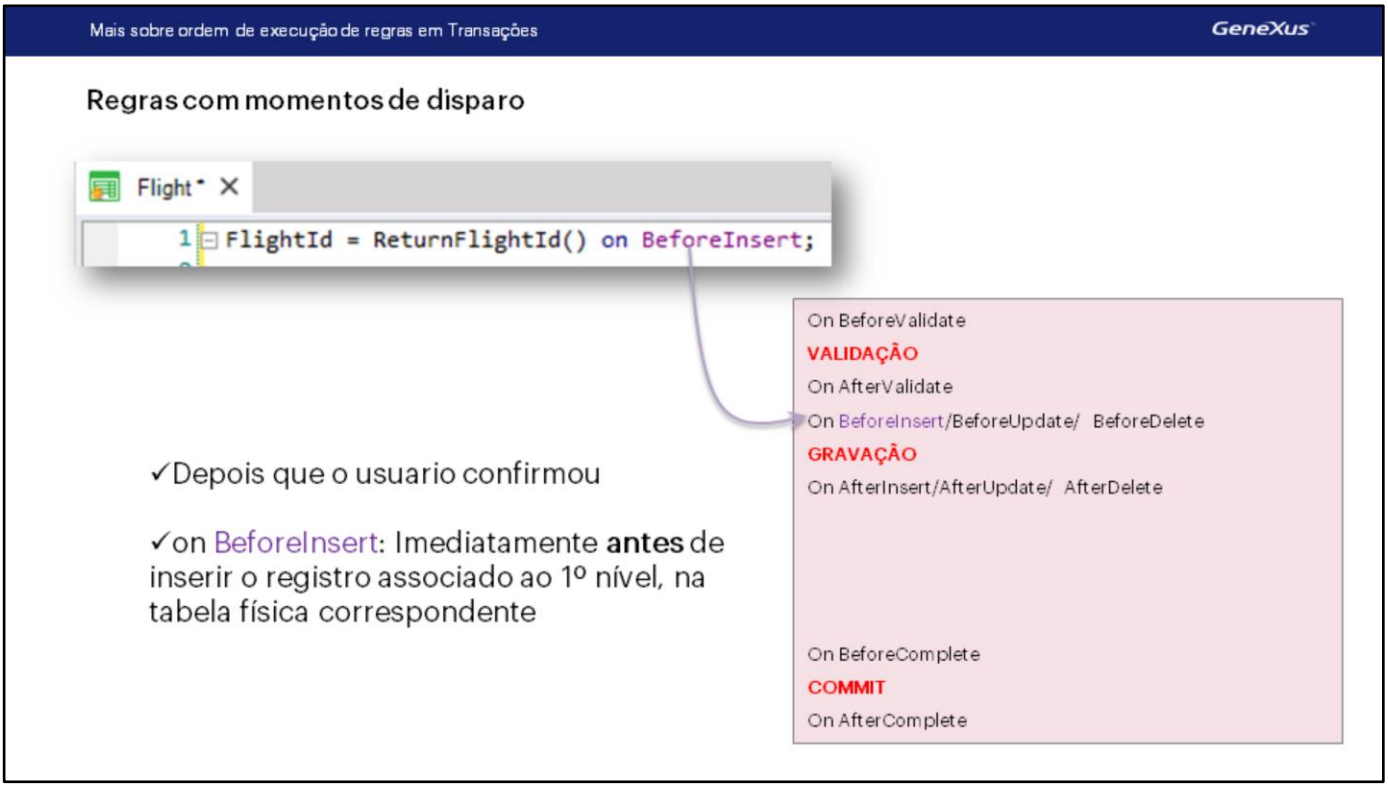
Isso nos permite personalizar o momento de disparo de uma regra, para que a mesma não seja disparada no momento escolhido pelo GeneXus, segundo sua árvore de avaliação, mas sim quando nós queremos.

Vamos começar pelos momentos relacionados à Validação: BeforeValidate e AfterValidate.

Evento de disparo: **BeforeValidate**
Este evento de disparo ocorre imediatamente antes em que a informação que está sendo instanciada (cabeçalho ou linha x) seja validada. Quer dizer, ocorrerá imediatamente antes da ação de "validação do cabeçalho" ou "validação da linha", conforme o caso. Observar que aqui também serão disparadas todas as regras que não estão condicionadas a nenhum evento de disparo.

Evento de disparo: **AfterValidate**
O evento de disparo AfterValidate permite especificar que uma regra seja executada imediatamente antes de gravar fisicamente cada instância do nível ao qual a regra está associada, na tabela física correspondente, e depois dos dados dessa instância terem sido validados.

Em outras palavras, se o evento de disparo AfterValidate for adicionado a uma regra, essa será executada para cada instância do nível ao qual está associada, imediatamente antes da instância ser gravada fisicamente (independente de estar inserindo, alterando ou excluindo) como um registro físico na tabela associada ao nível.



Às vezes não contamos com a possibilidade de utilizar a propriedade Autonumber para serializar os atributos que são chave primária simples. Tal funcionalidade é oferecida pelos gerenciadores de banco de dados (DBMSs). GeneXus aproveita isso e permite usá-la; porém, quando não trabalhamos com um DBMS, essa opção não está disponível.

No exemplo criamos, o identificador de cada vôo debe ser gerado de uma forma específica e o auto-number não irá servir. Por isso é que foi codificado um procedimento para gerar essa numeração.

Como foi explicado, precisamos que a regra seja executada imediatamente depois que o usuario confirmar e somente se estiver inserindo um vôo novo.

Estas duas definições são corretas para definir o que precisamos:

FlightId = ReturnFlightId() if Insert on AfterValidate;

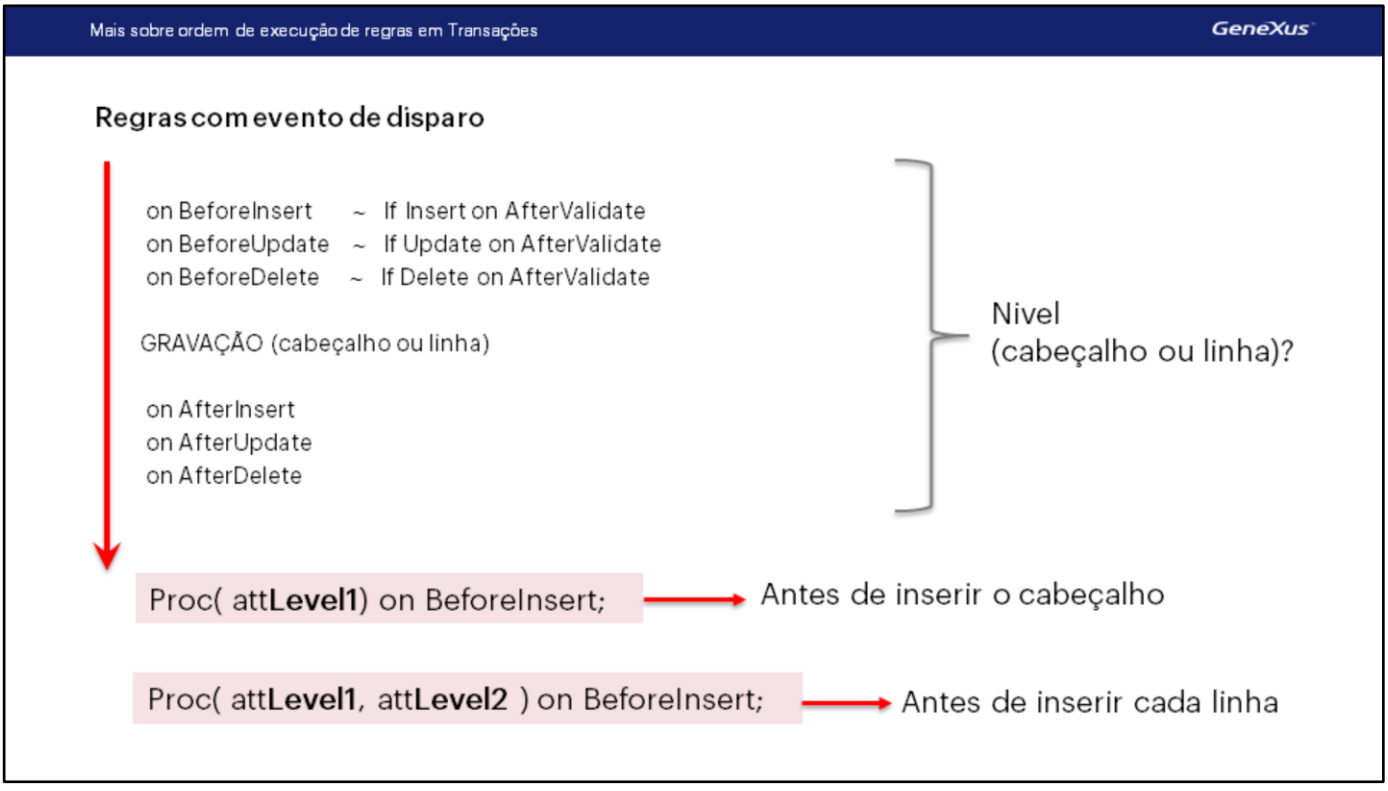
ou também

FlightId = ReturnFlightId() on BeforeInsert;

Na primeira definição estamos definindo que o procedimento seja executado somente qunado tratar-se de uma inserção (daí a condição de disparo: if Insert), imediatamente depois de validar os dados do primeiro nível (porque tem somente um atributo envolvido na regra, que pertence ao primeiro nível da transação) e imediatamente antes de gravar o registro fisicamente (pelo evento de disparo: on AfterValidate).

Existem três eventos de disparo que ocorrem no mesmo momento que o evento AfterValidate, mas estes já contém o modo explicitamente. Eles são: BeforeInsert, BeforeUpdate e BeforeDelete.

Por este motivo são equivalentes às regras apresentadas acima. Na 2ª proposta, seria redundante condicionar a regra a "If Insert", já que o BeforeInsert trata-se de uma inserção.



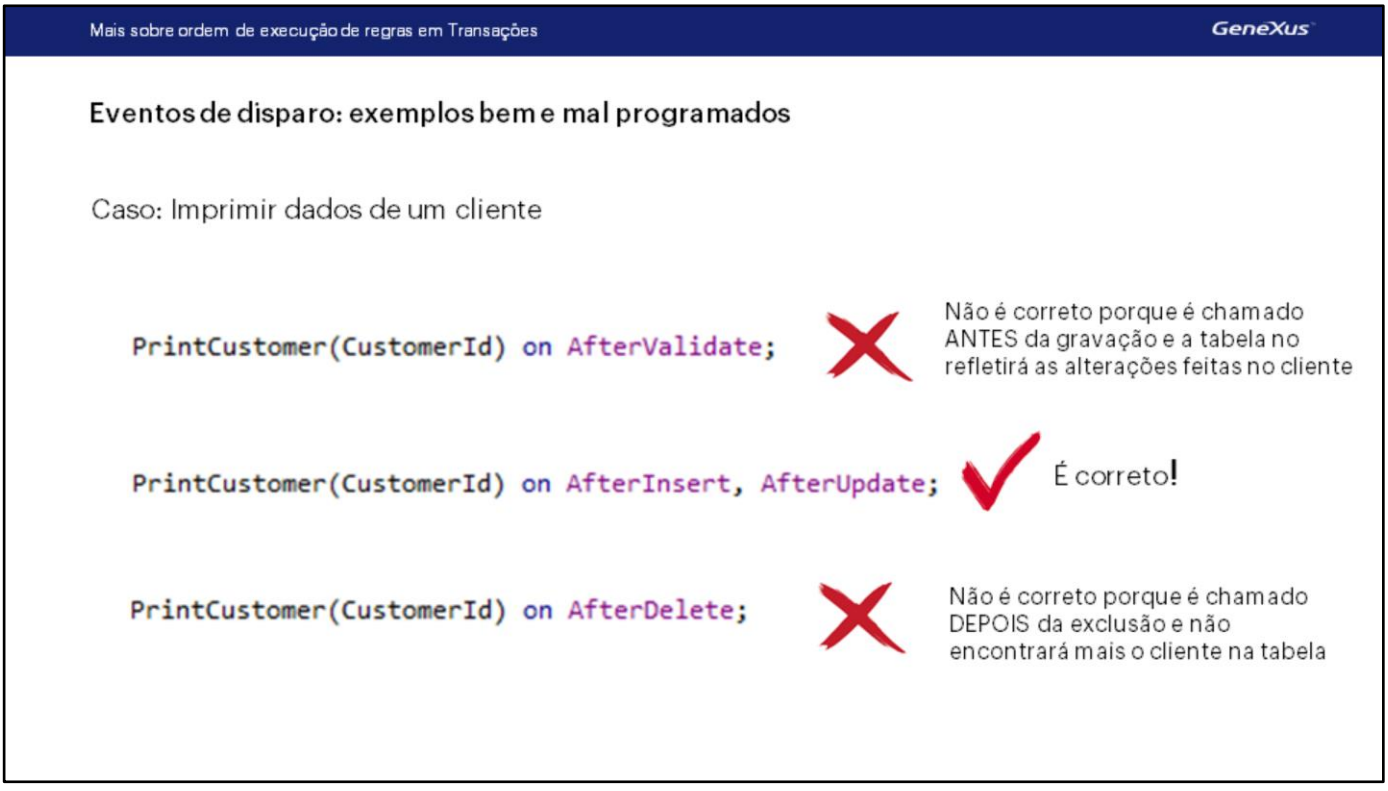
De modo que valem as seguintes equivalências:

on BeforeInsert ~ If Insert on AfterValidate
on BeforeUpdate ~ If Update on AfterValidate
on BeforeDelete ~ If Delete on AfterValidate

quando definimos uma regra na qual incluimos também o evento de disparo on AfterValidate, ou on BeforeInsert, BeforeDelete, BeforeUdate. Porém há uma diferença entre os dois exemplos. Se referenciamos na regra ao menos um atributo do segundo nível da transação, a mesma estará associada ao segundo nível. Portanto, a regra será executada imediatamente antes de gravar fisicamente cada instância correspondente ao segundo nível da transação.

Eventos de disparo: **AfterInsert**, **AfterUpdate**, **AfterDelete**
Assim como existe um evento de disparo que permite definir que determinadas regras sejam executadas imediatamente antes de acontecer a gravação física de cada instância de um nível (AfterValidate, BeforeInsert, BeforeUpdate, BeforeDelete), também existem eventos de disparo para definir que certas regras sejam executadas **imediatamente depois** de serem inseridos, alterados ou excluidos físicamente os dados. Estes eventos são AfterInsert, AfterUpdate e AfterDelete.

O evento de disparo AfterInsert permite definir que uma regra seja executada imediatamente depois de cada instância do nível ao qual está associada a regra seja inserida fisicamente; o AfterUdate depois da instância ser atualizada fisicamente, e o AfterDelete depois depois de excluir.



Vamos supôr que, na transação Customer, queremos chamar um procedimento que imprima dos dados de cada cliente com o qual estivermos trabalhando (inserindo ou alterando) através da transação.

Em que momento devemos realizar a chamada do relatório, na transação?

Proposta 1: PrintCustomer(CustomerId) on AfterValidate;

Não é adequado acrescentar este evento de disparo na regra porque o procedimento seria chamado imediatamente antes da gravação física de cada cliente. Consequentemente, não encontraria os dados do cliente na tabela CUSTOMER (se estivermos inserindo um cliente através da transação), ou estaria com seus dados desatualizados (se estivermos modificando um cliente através da transação). Ainda, se estivermos excluindo um cliente, os dados do cliente ainda se encontrariam na tabela CUSTOMER e seriam impressos justamente antes da exclusão física.

Se a ideia for emitir um relatório com os dados de cada cliente que for excluido, seria adequado definir a seguinte regra:

PrintCustomer(CustomerId) on BeforeDelete;

ou sua equivalente:

PrintCustomer(CustomerId) if Delete on AfterValidate;

para restringir o disparo da regra unicamente quando se está excluindo um cliente, porque é o único caso onde não seria correto utilizar o evento de disparo AfterValidate.

Proposta 2: PrintCustomer(CustomerId) on AfterInsert, AfterUpdate;

O momento de disparo AfterInsert ocorre imediatamente depois de inserir fisicamente cada instância associada a determinado nível da transação (neste caso, como o único atributo envolvido na regra é CustomerId, trata-se de uma regra associada ao primeiro e único nível da transação Customer).

Como é indicado claramente em seu nome, o evento de disparo AfterInsert só ocorre depois de inserir uma nova instância (precisamente depois de ser inserida como registro físico). É por isso que, quando adicionamos este evento de disparo a uma regra, não é necessário acrescentar a condição de disparo if insert.

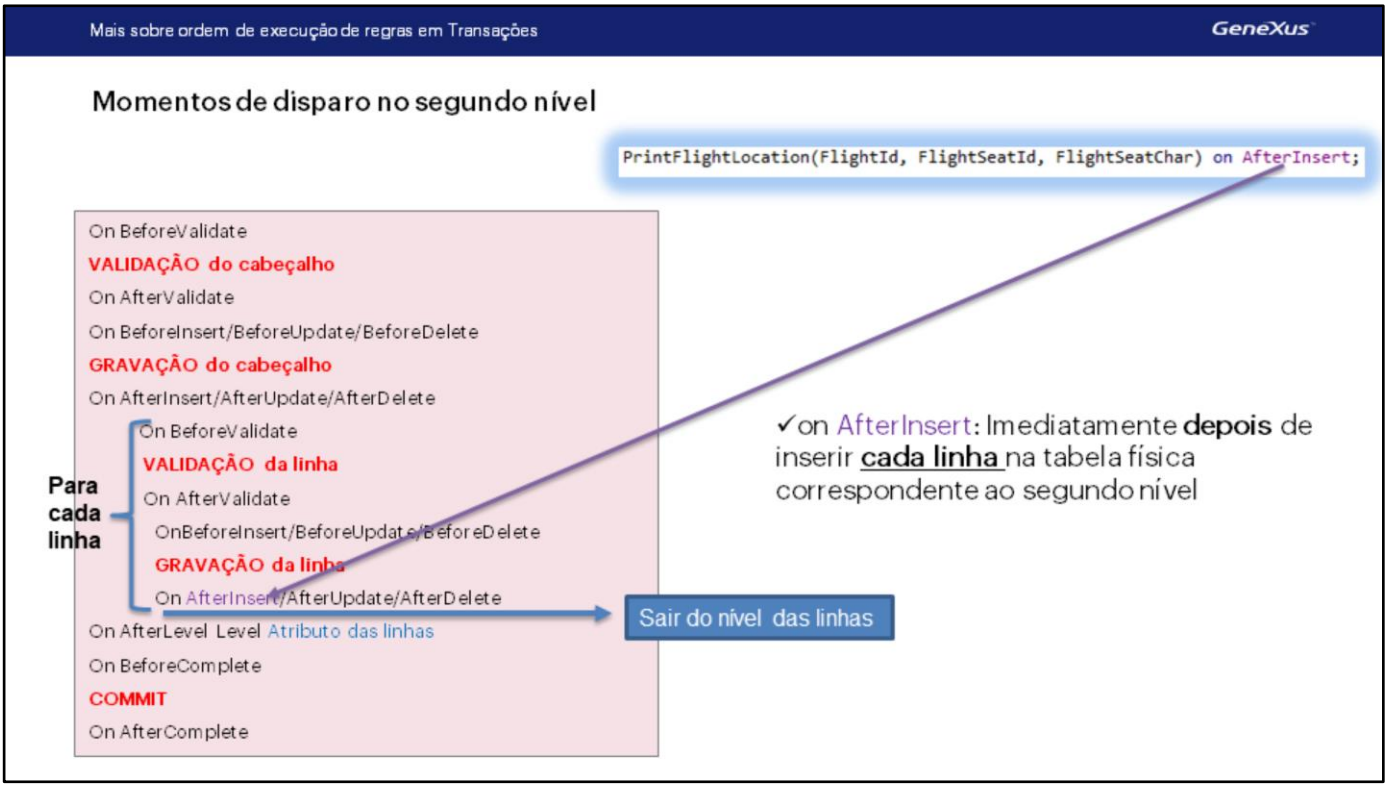
O momento de disparo AfterUpdate ocorre imediatamente depois de modificada fisicamente cada instância associada a determinado nível da transação (neste caso, trata-se de uma regra associada ao primeiro e único nível da transação Customer). O evento de disparo AfterUpdate só ocorre ao modificar um registro. É por isso que não é necessário acrescentar a condição de disparo if update.

A proposta é correta, já que acrescentar estes momentos de disparo na regra de chamada do procedimento fará com que ela seja executada imediatamente depois de inser ou alterar cada cliente. Assim, o procedimento encontrará o cliente com seus dados corretos na tabela CUSTOMER e os imprimirá. Agora, devemos ter claro que o cliente ainda não estará commitado!

Proposta 3: PrintCustomer(CustomerId) on AfterDelete;

O evento de disparo AfterDelete ocorre imediatamente depois de excluir fisicamente cada instância associada a determinado nível da transação (neste caso, como o único atributo envolvido na regra é CustomerId, trata-se de uma regra associada ao primeiro e único nível da transação Customer).

Não é correto adicionar este evento de disparo na regra de chamada do procedimento, porque este seria chamado imediatamente depois da eliminação física de cada cliente. Consequentemente, o procedimento não encontrará mais o cliente com seus dados na tabela CUSTOMER.



Se definimos uma regra onde incluimos o evento de disparo on AfterInsert, porém diferentemente dos exemplos vistos anteriormente, é referenciada na regra ao menos um atributo do segundo nível da transação, a mesma estará associada ao segundo nível. Portanto, a regra será executada **imediatamente depois** de **inserir** fisicamente cada instância correspondente ao segundo nível da transação.

Igualmente são os casos de on AfterUpdate e on AfterDelete.

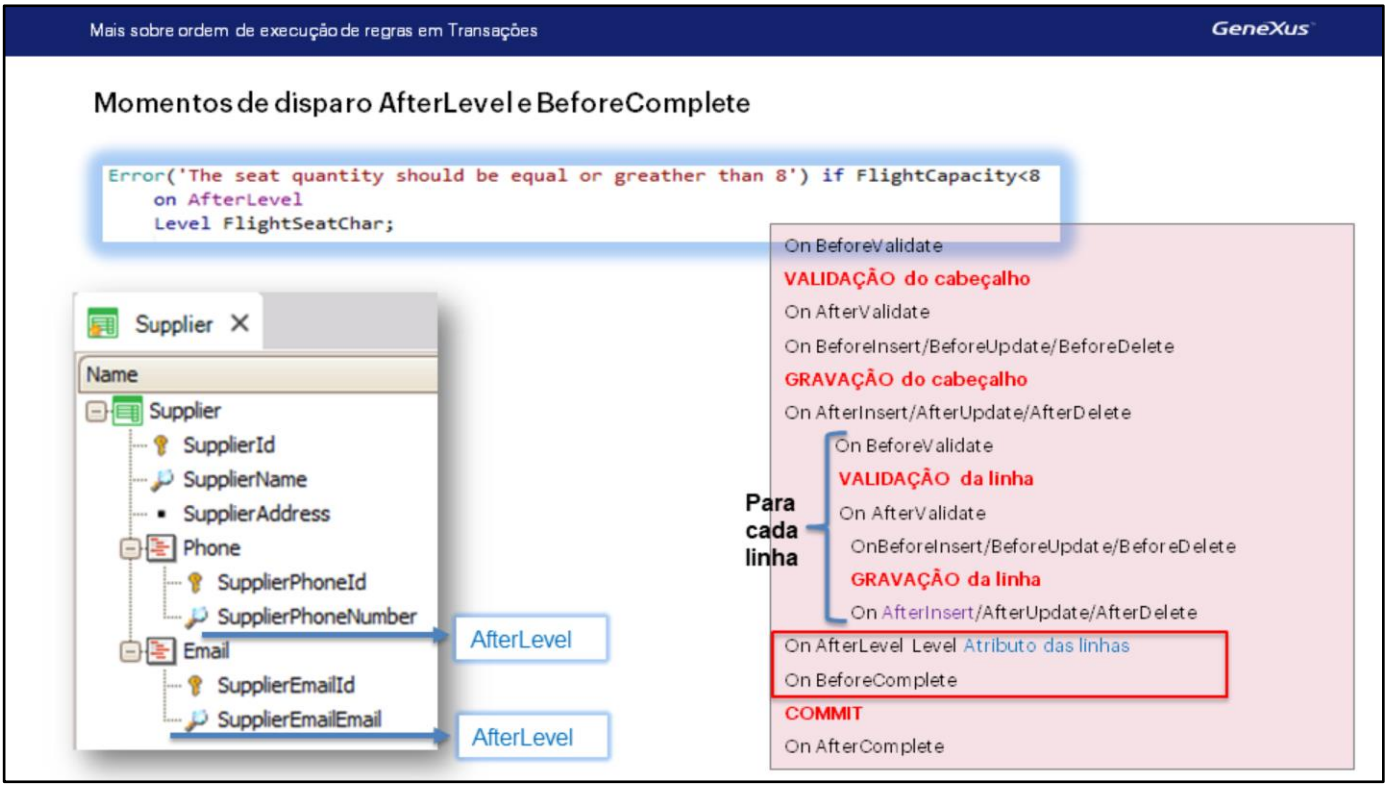
Ampliamos o esquema que havíamos visto antes, das ações que envolvem a os eventos de disparo vistos até agora:

VALIDAÇÃO DOS DATOS
AfterValidate – BeforeInsert – BeforeUpdate – BeforeDelete
GRAVAÇÃO DO REGISTRO (insert, update, delete)
AfterInsert – AfterUpdate – AfterDelete

Este esquema se repete para cada instância do nível. Por exemplo, vamos pensar no cadastro dos assentos de um vôo. Para cada linha com um assento ocorrerá este esquema. Podemos pensar nisso como um loop que é repetido até que a última linha do grid seja gravada como registro físico.

A ação que acontece logo após a gravação da última linha corresponde à saida desse nível (neste caso, o nível dos assentos dos vôos) E depois dessa ação, a menos que exista outro nível que voltaria a executar o mesmo loop, ocorrerá a última ação na execução, que é o commit.

Entre a ação de sair do nível, e o commit temos um evento (BeforeComplete) e outro para depois do commit (AfterComplete).



O evento de disparo **AfterLevel** permite definir que uma regra seja executada **imediatamente depois** de sair de determinado nível.

SINTAXE: *regra* [if *condição de disparo*] [**on AfterLevel Level** *atributo*];

ONDE:
regra: é uma regra qualquer permitida em transações
condição de disparo: é uma expressão booleana que permite envolver atributos, variáveis, constantes e funções, assim como os operadores Or, And, Not.
atributo: é um atributo pertencente ao nível no qual deseja-se que a regra seja executada.

Se o atributo especificado depois do evento de disparo **AfterLevel** pertence ao segundo nível da transação, a regra será executada quando terminar de processar todas as linhas do segundo nível.

E se o atributo especificado depois do evento de disparo **AfterLevel** pertence ao primeiro nível —seguindo o mesmo conceito— a regra será executada quando terminar de processar todos os cabeçalhos. Observar que esto se da ao final de todo, quer dizer, uma vez que se hayan ingresado todos os cabeçalhos e sus linhas e se cierre a transação (em ese momento se habrán iterado todos os cabeçalhos). Portanto, se o atributo especificado pertence ao primeiro nível, a regra será disparada uma somente vez antes do Evento Exit (é um evento que executado uma única vez quando fecha uma transação em tempo de execução, como veremos).

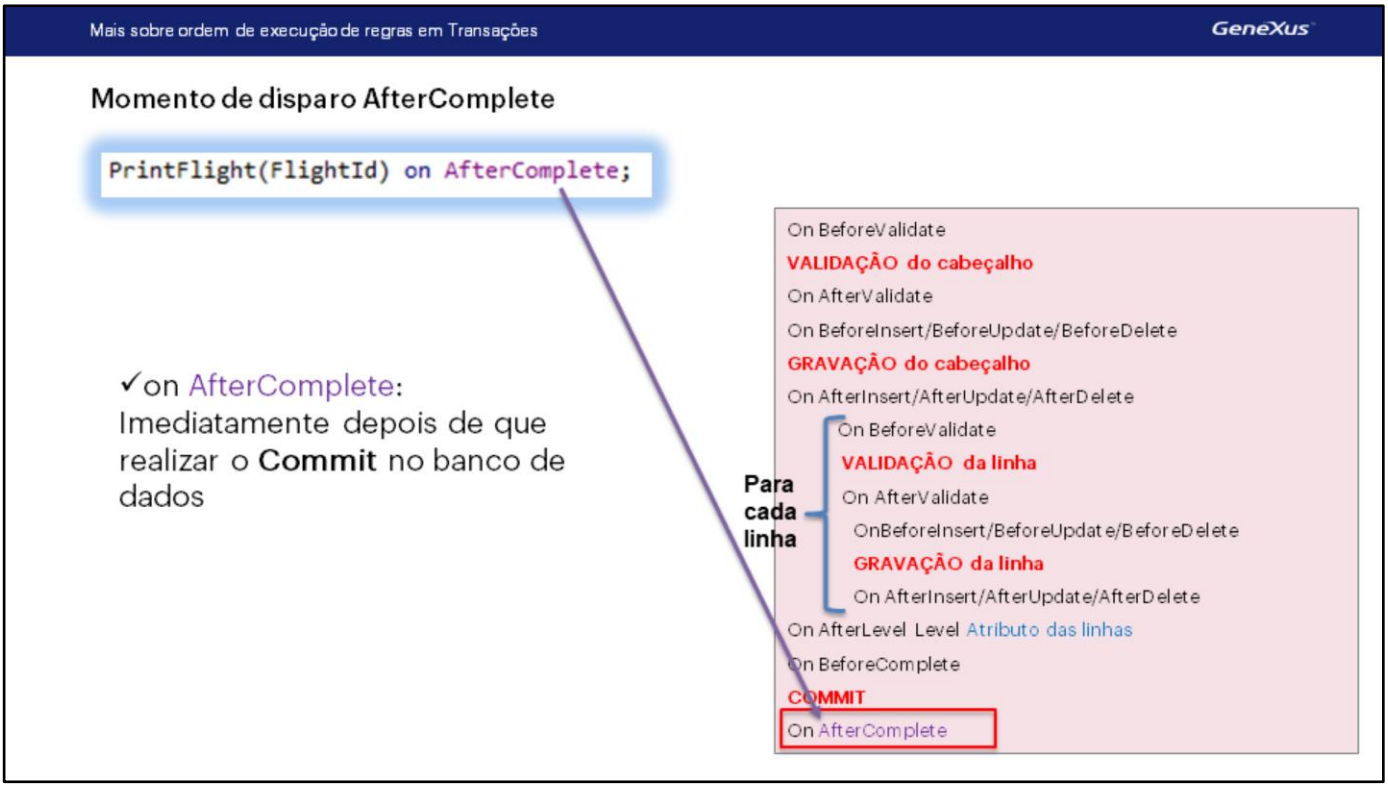
Utilizaremos este momento de disparo na regra que definimos para validar o cadastro de 8 assentos ou mais para cada vôo que vemos em tela.

O evento **BeforeComplete**, neste caso, coincide com o **AfterLevel**. Se observarmos o esquema apresentado na página anterior, podemos ver que o instante de tempo que existe entre a saída do último nível e a realização do commit é mesmo instante em que ocorrem estes eventos. Ambos os momentos de disparo são disparados no mesmo instante, sempre quando acontece a saída do último nível.

Para entender melhor isso, vamos supôr que temos uma transação com dos níveis paralelos, por exemplo, uma transação Supplier onde temos um nível para os números de telefone e outro para os endereços de mail.

O momento que deverá ser disparada uma regra condicionada ao **on AfterLevel Level *SupplierPhoneNumber*** NÃO IRÁ COINCIDIR com o de uma regra condicionada ao on BeforeComplete, No caso das regras condicionadas ao **on AfterLevel Level *SupplierEMailEMail,*** por ser o último nível subordinado da transação, esas sim, concidirão com on BeforeComplete.

Enquanto que a primeira é disparada quando acontece a saida do nível dos telefones, e antes de entrar na validação de todos os emails, a segunda é disparada depois que acontece a saida deste último nível.
Neste caso, o evento **BeforeComplete** coincidirá com o **AfterLevel Level *SupplierEMailEMail***.

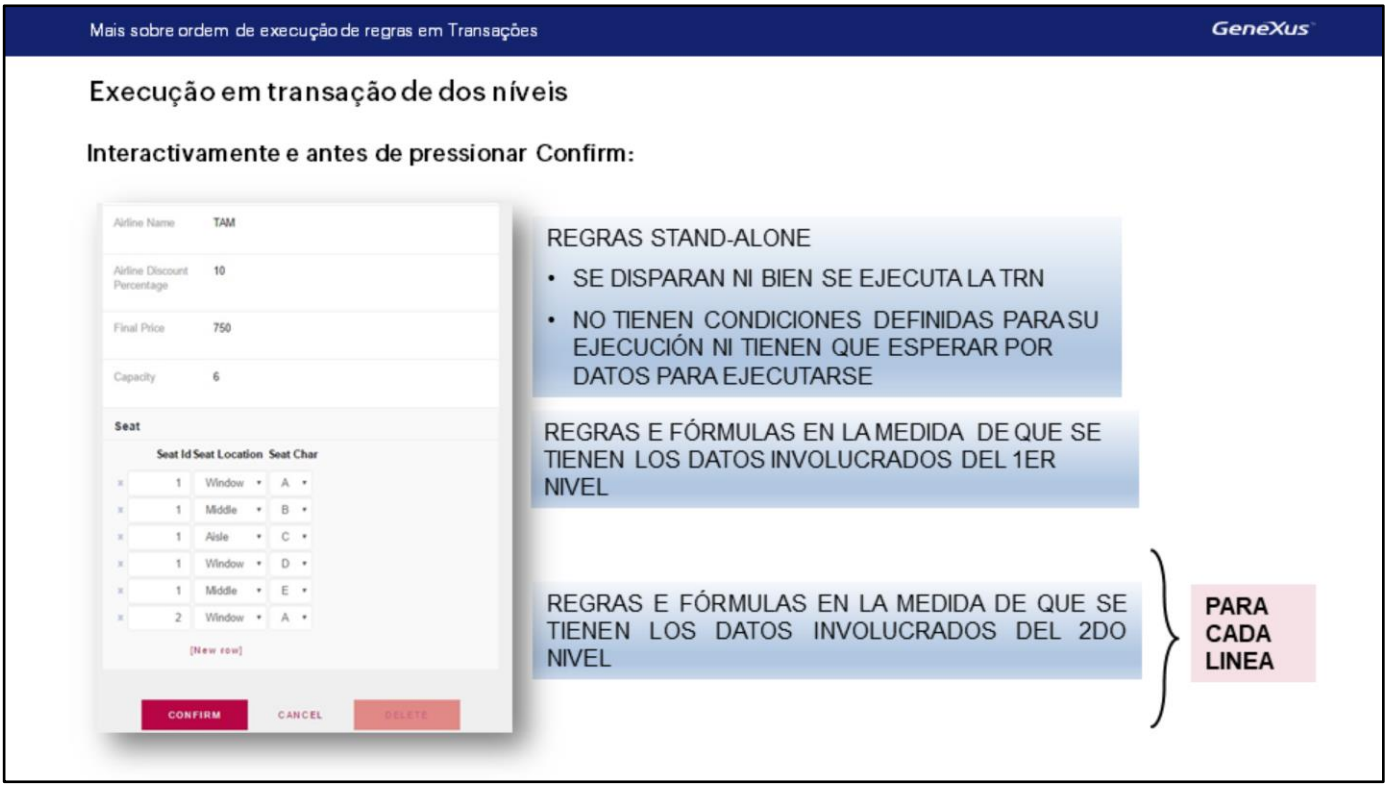


Este evento corresponde ao instante de tempo imediatamente posterior ao commit.

Se, na transação Flight, forem cadastrados 3 vôos (informação do 1º nível + seus respectivos assentos), terão ocorrido 3 commits e, depois de cada commit, terão sido executadas as regras com evento de disparo **on AfterComplete**.

Os atributos do cabeçalho ainda se encontram com valor em memoria quando as regras com evento de disparo AfterComplete são executadas. Depois que foi executado o commit, é muito comum chamar um procedimento que imprima a informação gravada e commitada, pasando ao objeto chamado, a chave primária como um parâmetro.

Falaremos mais deste evento quando estudarmos a **integridade transacional**.



À medida que o usuário vai adicionando dados na transação e antes de pressionar Confirm, serão disparadas as regras e fórmulas de acordo com a árvore de avaliação, na medida em que os diferentes atributos forem sendo envolvidos. As primeiras regras disparadas serão as regras stand-alone:

1. Podem ser executadas com a informação recebida pelos parâmetros.
2. Não dependem de nada para executar.

Exemplos de regras stand-alone (executam com a informação recebida pelos parâmetros):

```
&A = parámetro2;
Msg( '...' ) if parámetro1 = 7;
```

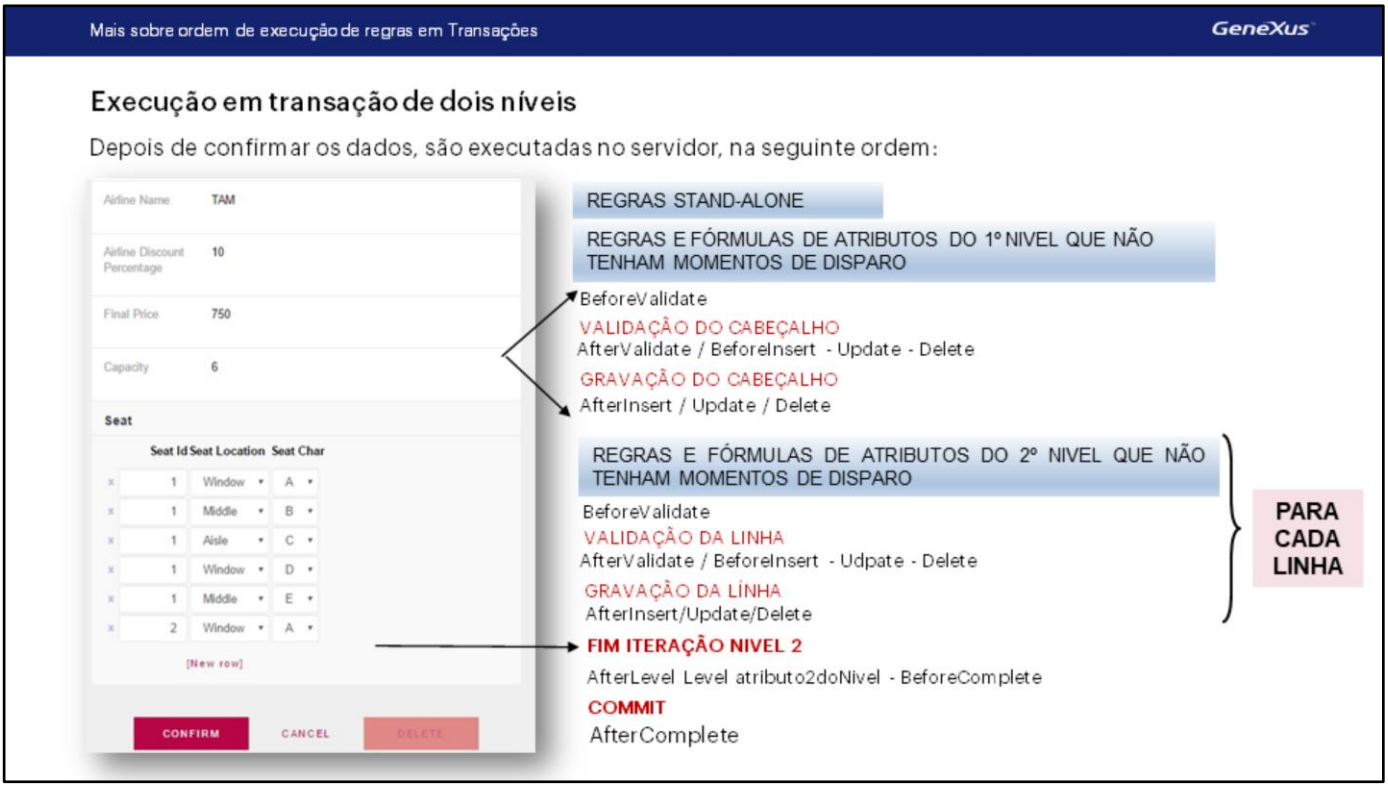
Exemplos de regras stand alone (não dependem de nada para executar):

```
msg( 'Eou are in the flight transaction');
&A = 7;
```

Portanto, são as primeiras regras que executam.

Depois da execução das regras stand alone, são executadas as regras e fórmulas associadas ao primeiro nível da transação que não tenham evento de disparo definido (quer dizer que não tenham especificado **on** ...) na medida em que validam os valores envolvidos na sua execução.

E, ao trabalhar no 2º nível (grid), para cada linha, são executadas as regras e fórmulas associadas ao 2º nível da transação que não tenham evento de disparo definido (quer dizer que não tenham especificado **on**...) na medida em que validam os valores envolvidos na sua execução.



Depois de que o usuario pressiona Confirm, os dados viajam até o servidor web. O servidor torna a percorrer o form como se fosse um usuário e as regras e fórmulas tornam a ser disparadas, mas, agora se houver alguma regra condicionada a um momento de disparo no servidor, será disparada em um momento específico e não conforma a árvore de avaliação.

O ordem de execução será a seguinte:

São executadas as regras stand-alone.

São executadas todas as regras e fórmulas associadas ao 1º nível que não tem evento de disparo definido.

São executadas as regras associadas ao 1º nível (cabeçalho) com eventos de disparo **BeforeValidate**.

É feita a VALIDAÇÃO dos dados cadastrados no 1º nível (cabeçalho).

São executadas as regras associadas ao 1º nível (cabeçalho) com eventos de disparo **AfterValidate**.

Depois da execução das regras com algum destes eventos de disparo associadas ao primeiro nível, acontece a ação de **gravação**; ou seja, *será gravada fisicamente a instância correspondente ao primeiro nível da transação, como um registro físico na tabela correspondente (neste exemplo, na tabela FLIGHT).*

Imediatamente depois de ter gravado essa instância:

- se a gravação correspondeu a uma inserção: serão executadas as regras associadas ao primeiro nível da transação com evento de disparo **AfterInsert**.
- se a gravação correspondeu a uma alteração: serão executadas as regras associadas ao primeiro nível da transação com evento de disparo **AfterUpdate**.
- se a gravação correspondeu a uma exclusão: serão executadas as regras associadas ao primeiro nível da transação com evento de disparo **AfterDelete**.

Depois de executar todas as operações explicadas até o momento, acontecerá um COMMIT e, em seguida, será executadas as regras com evento de disparo AfterComplete.

É muito importante ficar claro que todas as operações explicadas serão executadas na ordem descrita, para cada vôo manipulado através da transação Flight (inserindo, alterando ou excluindo).

É importante memorizar o ordem em que as regras são executadas em uma transação, quais são os eventos de disparo disponíveis para defini-las, quando são disparadas exatamente e quais ações ocorrem antes e depois de cada evento de disparo, já que, somente se conhecermos bem poderemos programar o comportamento das transações adequadamente.

Exercicios

Quando são disparadas as seguintes regras?

- *Something(FlightId) on BeforeInsert;*

  Um instante antes de inserir o registro correspondente ao 1º nível.

- *Something(FlightId, FlightSeatId, FlightSeatChar) on BeforeInsert;*

  Um instante antes de inserir cada registro correspondente ao 2º nível.

- *Something(FlightId) on BeforeInsert Level FlightSeatChar;*

  Ídem ao anterior. Observar que **Level FlightSeatChar** especifica que está tratando-se do BeforeInsert das linhas e não do cabeçalho.

Exercicios

Algumas regras estão mal programadas Quais?

- *FlightPrice = 2000 on AfterInsert;*

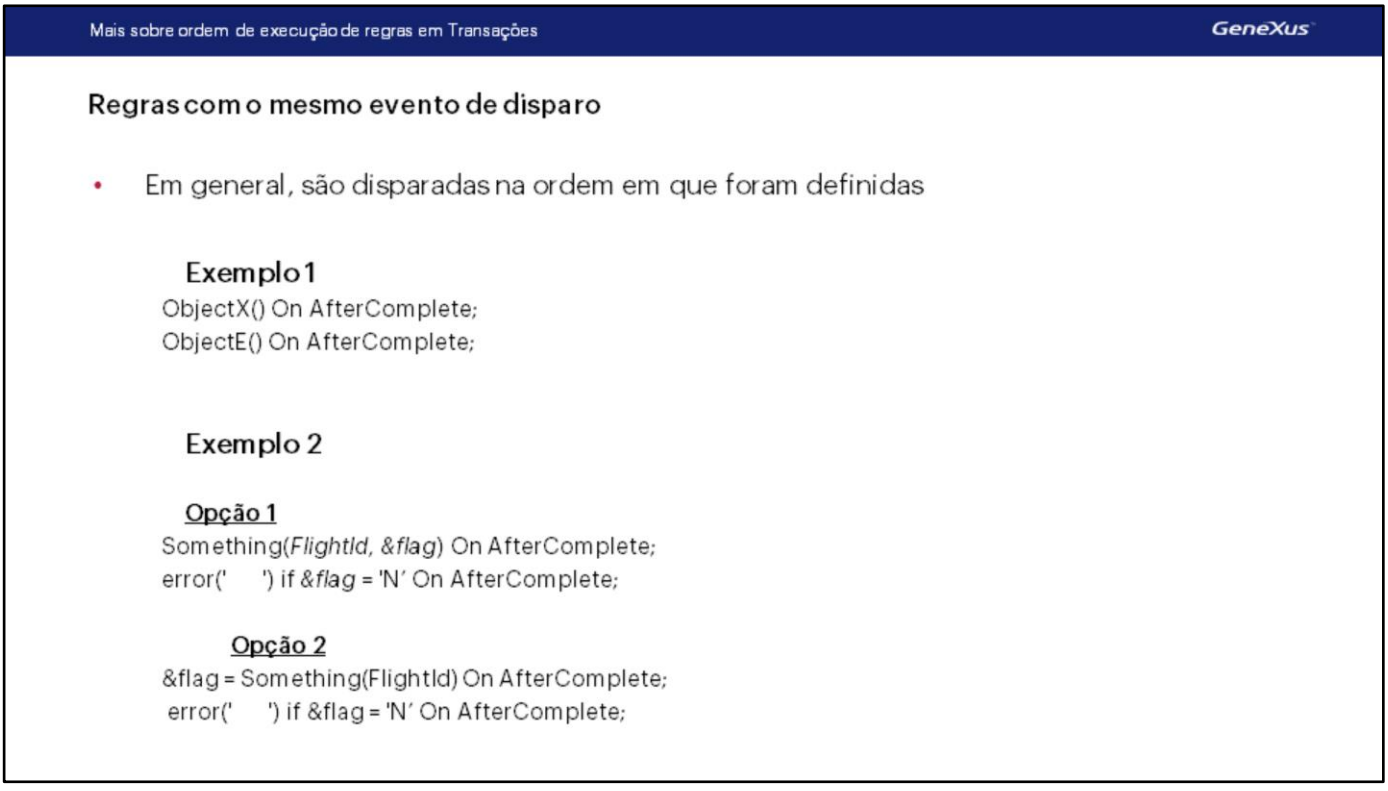
  Incorreto: O último momento para atribuir valor a um atributo do cabeçalho é imediatamente antes de sua gravação (BeforeInsert).

- *Something(FlightPrice) on AfterInsert;*

  Correto: aqui está passando o valor de um atributo do cabeçalho. Esse valor ainda está disponível em memória. Último momento possível para utilizar é AfterComplete.

- *Something(FlightId,FlightSeatId,FlightSeatChar) on AfterLevel Level FlightSeatChar;*

  Incorreto: a regra sem o evento de disparo está associada ao 2º nível, quer dizer, será disparada para cada linha. Porém, o evento de disparo condiciona a executa-la ao sair das linhas... e já não temos valores disponíveis de atributos do 2º nível.



Numa transação, quando são definadas duas ou mais regras com o mesmo evento de disparo e não existe nenhuma dependência entre elas, as mesmas serão executadas respeitando a ordem em que foram escritas.

Exemplos:

Exemplo 1: Como as duas regras definidas estão condicionadas com o mesmo evento de disparo e não existe nenhuma dependência entre elas, as mesmas serão executadas na ordem em que foram escritas.

Exemplo 2: Numa transação, é preciso chamar a um procedimento que realiza determinada validação e retorna um valor 'S' o 'N'; se o valor devolvido é 'N', deve-se emitir uma mensagem de erro.

Para resolver isso, analisaremos as duas opções apresentadas acima.

Na primeira alternativa foi definida uma regra que chama um objeto e uma regra error. Ambas as regras tem o mesmo evento de disparo, e aparentemente deveria existir uma dependência entre elas, já que a regra error está condicionada ao valor da variable &flag, e a variable &flag é recebida por parâmetro na chamada do objeto.

Porém, mesmo que a dependência possa parecer evidente para nós (porque no procedimento programaremos a variable &flag como uma saida), na seção de regras da transação (que é onde se encontram as regras que estamos vendo), o especificador do GeneXus não consegue saber se os parâmetros passados em uma chamada como um programa (CALL) são de entrada, de saida, ou de entrada-saida; consequentemente, o especificador não encontrará interdependência entre as regras call e error, já que a variable &flag poderia ser passada como variável de entrada para o procedimento e nesse caso, por exemplo, não haveria uma dependência onde primeiro se deveria executar a chamada do procedimento e depois a regra error.

Assim, concluimos que não haverá dependências entre as regras da opção 1, porque as mesmas serão

disparadas na ordem em que estão escritas. É importante ver que, se as regras estiverem escritas na ordem inversa (quer dizer, primeiro a regra error e depois a chamada do procedimento), o comportamento não será o esperado em muitos casos.

Já na segunda alternativa, observemos que a mesma consiste em uma regra que chama o objeto *Something* e uma regra error. Ambas as regras tem o mesmo evento de disparo e, neste caso sim, existe dependência entre elas, porque a regra error está condicionada ao valor da variable &flag. Como a chamada do procedimento é realizada como função (UDP), para o GeneXus fica claro que a variable &flag será modificada pelo procedimento; portanto GeneXus entende que, primeiro deve disparar a chamada do procedimento como função e depois a regra error, porque a variable &flag será carregada mediante a chamada do procedimento e depois que essa variável tiver valor é que avaliará se irá ou não disparar a regra error.

No caso da Opção 2 então, independente da ordem de definição de ambas as regras, a chamada ao procedimento como função será disparada primeiro, e depois disso, será disparada a regra error (caso se cumpra a condição de disparo, obviamente).

Por esta razão, é recomendável que, sempre que quiser definir validações desse tipo, chamar como função (UDP) em vez de programa (CALL).

GeneXus™

The power of doing.

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications