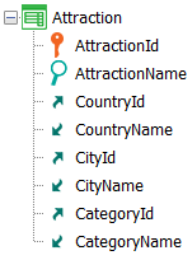
Atualização para a Base de dados com comandos específicos de procedimentos.

Como atualizar (update)

GeneXus

Previously: New Command

	Uniqueness check
New command	✓



AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

3

New

AttractionId = 3
AttractionName = "Eiffel Tower"
CountryId = 2
CityId = 1
CategoryId = 3

When duplicate

for each Attraction
CategoryId = 3
endfor

endnew

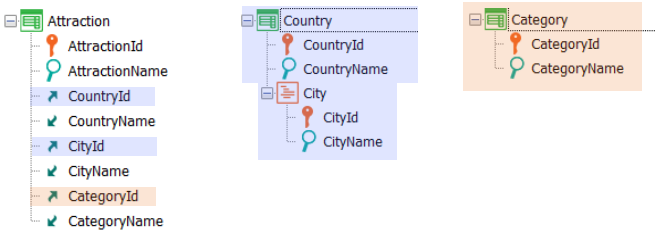
No vídeo em que estudamos o comando New para inserir um registro em uma tabela por procedimento, vimos que se o registro que queríamos inserir fosse duplicado por chave primária ou chave candidata, então poderíamos escolher modificá-lo, escrevendo um for each dentro da cláusula when duplicate.

Mas, e se já sabemos que o registro existe e o que queremos é justamente atualizá-lo?
No exemplo, mudar a categoria para a atração turística 3, Torre Eiffel, que sabemos que existe.

Update

Existe um comando especial Update em procedimentos?

For each Command



Insert, Update, Delete

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

3

For each Attraction
Where AttractionName = “Eiffel Tower”

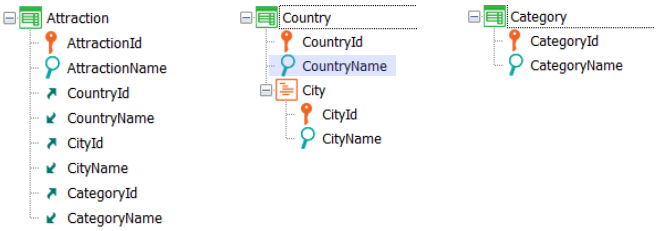
CategoryId = 3

endfor

A resposta é NÃO. Para atualizar se utiliza o conhecido comando For each. Com isto estamos dizendo que o For each não é utilizado apenas para consultar a base de dados, mas também para atualizá-la.

No exemplo, se queremos alterar a categoria do registro correspondente à Torre Eiffel, então será suficiente escrever este For each. Estamos percorrendo a tabela Attraction, filtrando por AttractionName “Eiffel Tower” e para o registro encontrado, atribuindo diretamente valor ao atributo que queremos modificar. Obviamente, poderíamos modificar o valor de quase todos os atributos do registro. E não só dele, mas também de todos os registros relacionados da tabela estendida. Aqui sim, podem ser atualizados muitos registros em uma única operação, ao contrário do que acontecia com o new.

For each Command



Insert, Update, Delete

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

CountryId	CountryName
1	Brazil
2	France
3	China

Francia

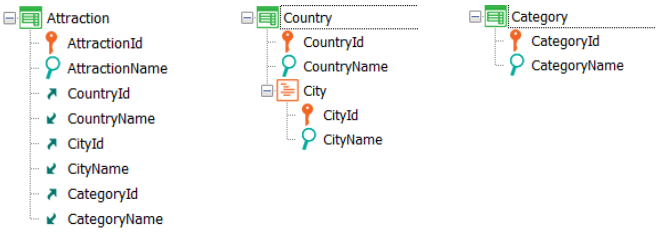
For each Attraction
Where AttractionName = “Eiffel Tower”

CategoryId = 3
CountryName = “Francia”

endfor

Por exemplo, poderíamos modificar o nome do país (escrevendo em espanhol –vemos que na tabela está em inglês-). Então, o for each se posiciona no registro de AttractionName “Eiffel Tower”, modifica seu atributo CategoryId, e acessa o registro do país relacionado, por tabela estendida, e nele modifica seu atributo CountryName, colocando “Francia”, em espanhol.

For each Command



Insert, Update, Delete

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	3
4	Forbidden city	3	1	2

CountryId	CountryName
1	Brazil
2	Francia
3	China

For each Attraction
Where AttractionName = "Eiffel Tower"

AttractionId = 5
CategoryId = 3
CountryName = "Francia

endfor

A próxima pergunta é se todos os atributos da tabela estendida podem ser atualizados ou existem restrições.

Por exemplo, podemos atualizar atributos da chave primária? Poderíamos mudar o Id de atração para 5? Não, podem ser atualizados todos os atributos do registro e dos relacionados por tabela estendida, exceto da chave primária. A lista de navegação indicará este erro.

Portanto, se precisarmos alterar a chave primária para um registro, não teremos alternativa a não ser criar um novo registro com a nova chave e excluir o anterior.

For each Command

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2
5	Eiffel Tower	2	1	3

Insert, Update, Delete

For each Attraction
Where AttractionName = "Eiffel Tower"

AttractionId = 5
CategoryId = 3

endfor

For each Attraction
Where AttractionName = "Eiffel Tower"

new
AttractionId = 5
CategoryId = 3
endnew

Delete

endfor

Assim, por exemplo, se o que queremos é mudar para a atração chamada "Eiffel Tower" o id para 5 e a categoria para 3, executamos um comando new, que queremos que tenha a mesma tabela base, Attraction. Ali especificamos o novo valor de chave primária e, para o restante dos atributos, queremos que assuma os mesmos valores que os do registro do for each em que estamos, com exceção de CategoryId, que queríamos aproveitar e alterar para 3.
A seguir, o que faremos será executar o comando Delete, que apaga o registro no qual estávamos posicionamos no for each, ou seja, no nosso caso o de Id 3.

	Uniqueness check	Referential Integrity check	Rules/Events execution
Assignment in For each	✓		

Unique Index

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	2

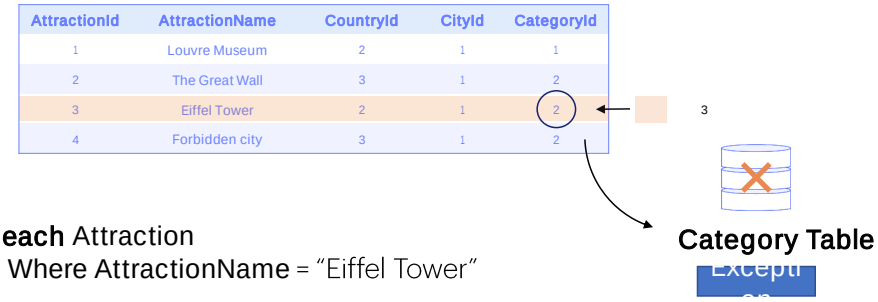
```
For each Attraction
    Where AttractionId = 3

    AttractionName = "Louvre Museum"

endfor
```

```
For each Attraction
    Where AttractionId = 3

    AttractionName = "Louvre Museum"

When duplicate
    ....
endfor
```

Como vimos no caso do comando new, a atualização por meio de For each realizará verificações de unicidade de registros. Neste caso, só se aplicará quando houver chaves candidatas, ou seja, quando exista um índice unique definido sobre algum atributo. Imaginemos que é o caso de AttractionName. E que então, estamos querendo modificar no for each o nome da atração 3, para que passe a ser "Louvre Museum".
Antes de tentar fazer essa atualização, o for each irá verificar, utilizando o índice unique, que já não exista um registro com esse valor. Como neste caso sim, existe, então não fará nada. O mesmo que acontecia com o new.

Mas também, como no new, não fará nada a menos que... tenhamos programado cláusula when duplicate. Nesse caso, será executado seu conteúdo. Voltaremos a isso mais tarde.

	Uniqueness check	Referential Integrity check	Rules/Events execution
Assignment in For each	✓	✗	



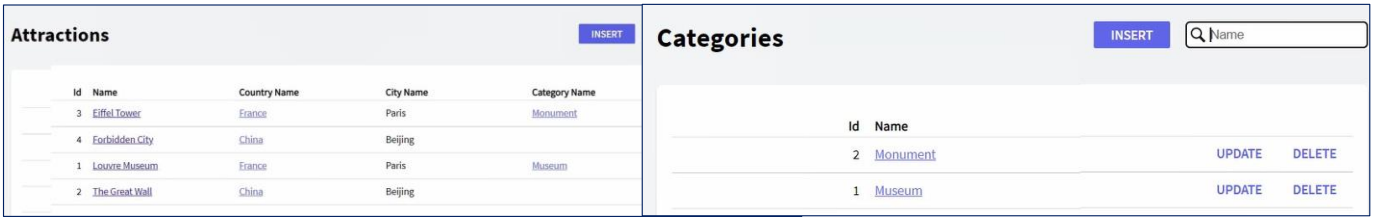
For each Attraction
Where AttractionName = "Eiffel Tower"

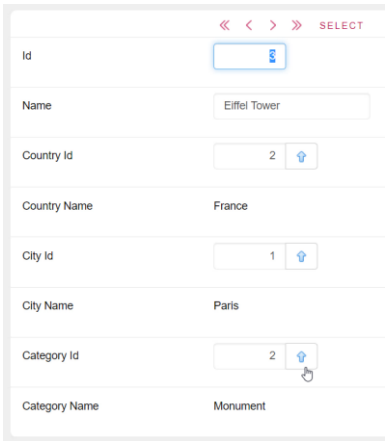
CategoryId = 3

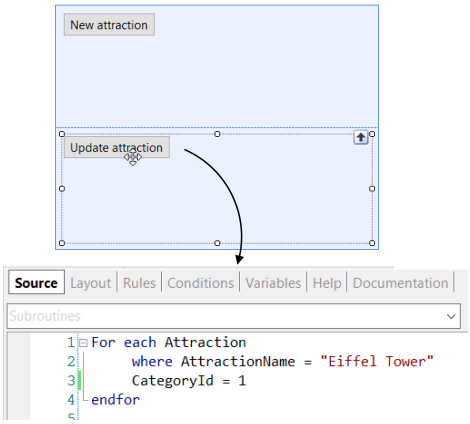
endfor

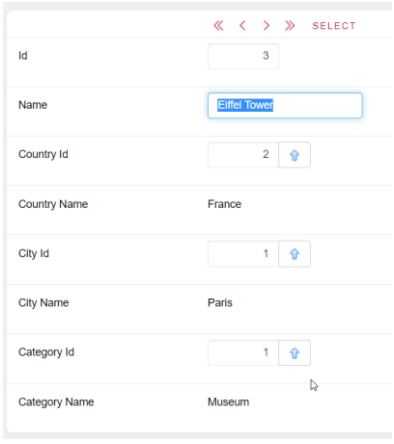
Por outro lado, como vimos no caso do comando new, a atualização através de For each não realizará verificações de integridade referencial, portanto, se a categoria 3 não existe na tabela Category, o programa não controlará. Novamente, se a base de dados tiver declarada a integridade referencial, ela sim a controlará, portanto, lançará uma exceção e o programa será cancelado.

Vejamos na prática.



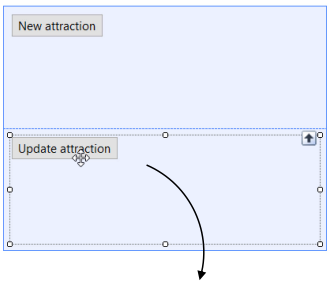


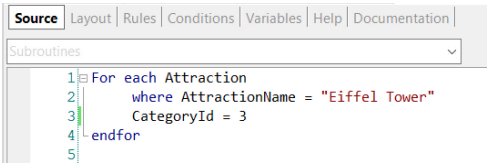


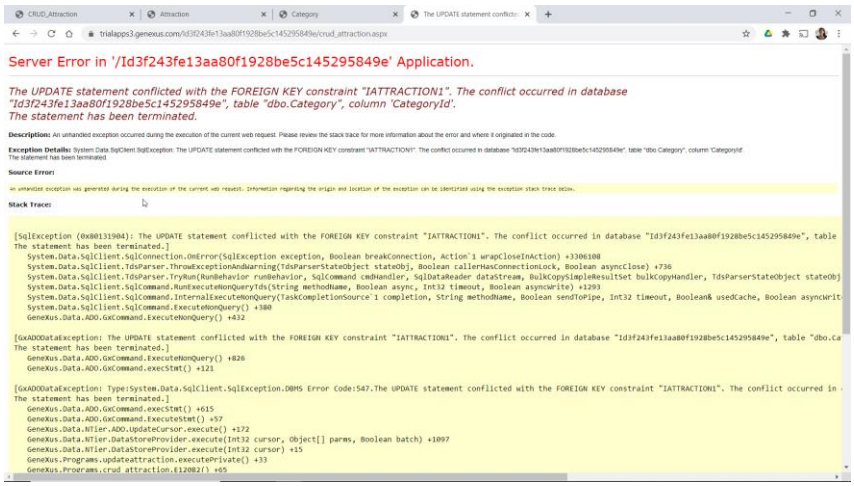


Temos as quatro atrações, a terceira é a Torre Eiffel, que tem como categoria, a 2.
E se vamos observar os dados de categorias, temos apenas 2 categorias: a 1 e a 2.

Bem, agora vamos ao GeneXus. E vemos que temos este web panel, no qual programamos, no evento associado ao botão, a invocação do procedimento UpdateAttraction, no qual temos este for each, que o que tenta fazer é mudar a categoria da Torre Eiffel para 1, a qual sabemos que existe.







Agora observemos o que acontece se em vez de atribuir a categoria 1 que sabemos que existe, atribuímos a ela a categoria 3, que não existe. Testemos.

O programa foi interrompido porque o for each não fez controle de integridade e tentou realizar a atualização, mas a base de dados realizou o controle. Portanto, lançou esta exceção que não foi capturada por nosso procedimento para fazer algo com ela. Isto pode ser feito e será nomeado em outro vídeo.

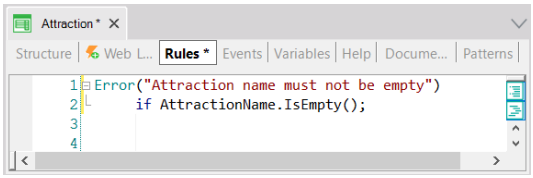
	Uniqueness check	Referential Integrity check	Rules/Events execution
Assignment in For each	✓	✗	✗

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	2
3		2	1	2
4	Forbidden city	3	1	2

For each Attraction
Where AttractionId = 3

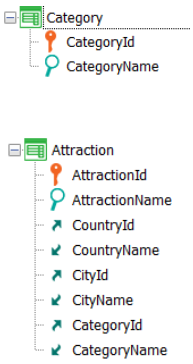
AttractionName = ""

endfor



Claro, assim como dissemos para o caso do new, a atualização por atribuição dentro do for each de um procedimento também não executará absolutamente nenhuma regra ou evento de transação. Ao contrário do que acontece quando se atualiza por meio de Business Component.

For each Command



CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist Site

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redemmer	1	2	2

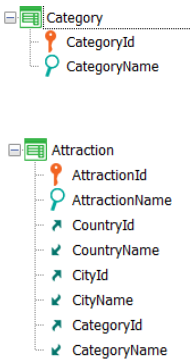
```
New
    CategoryName = "Tourist Site"
endnew

For each Attraction
    Where CountryName = "China" and CityName = "Beijing"
    Where CategoryName = "Monument"
    CategoryId = find( CategoryId, CategoryName = "Tourist
    COMMIT
endfor
```

Aqui vemos um exemplo onde inserimos uma nova categoria, Tourist Site, na tabela Category, e imediatamente percorremos com um for each as atrações turísticas do país China e cidade Beijing, de categoria "Monument", alterando essa categoria para a nova, "Tourist Site".

Outra vez, este código só é válido no Source de um procedimento. E o que acontece com o Commit? Quando são commitados o registro inserido em Category e os dois registros modificados em Attraction?

Se não escrevemos comando Commit de forma explícita no Source, então, se também não modificamos o valor padrão da propriedade Commit on Exit do procedimento, GeneXus adicionará um Commit ao final. Isto porque já terá descoberto que no Source se está querendo atualizar a base de dados. Em um procedimento onde GeneXus não encontra que a base de dados será atualizada, não o adiciona, mesmo que a propriedade Commit on Exit esteja configurada em Yes.

For each Command

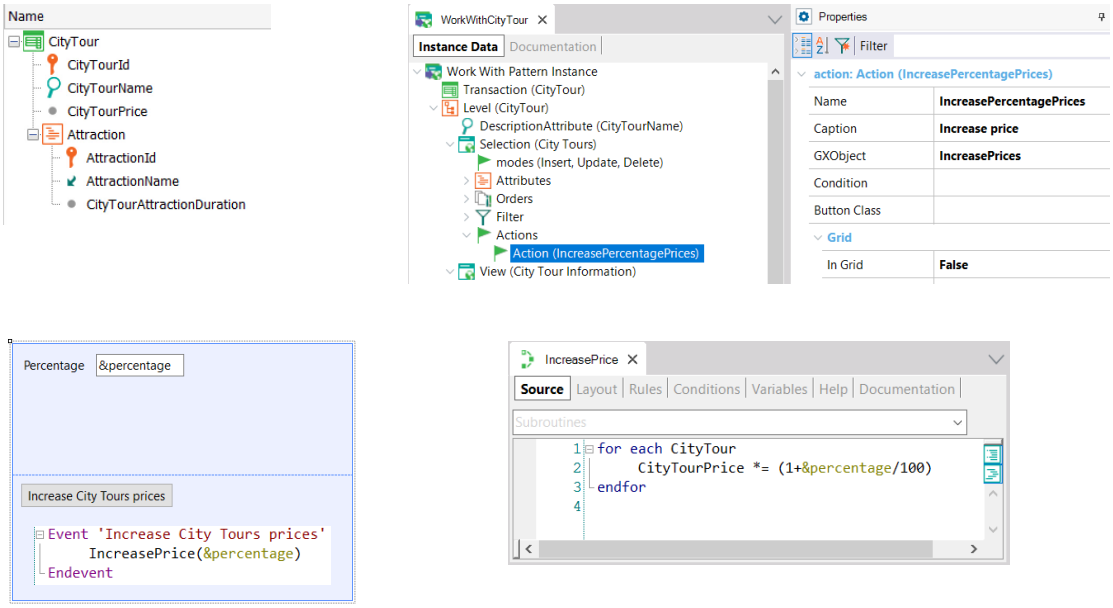


CategoryId	CategoryName
1	Museum
2	Monument
3	Tourist Site

AttractionId	AttractionName	CountryId	CityId	CategoryId
1	Louvre Museum	2	1	1
2	The Great Wall	3	1	3
3	Eiffel Tower	2	1	2
4	Forbidden city	3	1	3
5	Christ the Redemmer	1	2	2

```
New
    CategoryName = "Tourist Site"
Endnew
Commit

For each Attraction
    Where CountryName = "China" and CityName = "Beijing"
    Where CategoryName = "Monument"
    CategoryId = find( CategoryId, CategoryName = "Tourist Site" )
Endfor
Commit
```

Claro, se quiséssemos que fosse realizado um Commit após a inserção da categoria, e depois, após modificar as atrações, bastaria explicitá-lo.



Adicionamos uma transação CityTour para representar os tours por cidades oferecidos ao público. Cada city tour realizará um percurso por um conjunto de atrações turísticas, e terá um preço determinado. Aplicamos o pattern work with e adicionamos uma ação que chamará um web panel que criamos, que pede ao usuário uma porcentagem, e chama um procedimento que aumentará o preço de todos os city tours com base nessa porcentagem.

Para isso, o que fazemos é percorrer a tabela CityTour e ao atributo CityTourPrice atribuir como novo valor aquele que tinha, multiplicado por este fator daqui, que é o que aplica o aumento. Também poderíamos ter utilizado diretamente o operador "vezes igual" para não repetir o atributo.

Observemos que a lista de navegação nos informa qual atributo é o que está atualizando da tabela CityTour.

Se agora executamos... vemos que temos dois city tours inseridos: um de valor 300, outro de valor 200, e o que vamos fazer é executar então esse web panel, vamos dizer que queremos que aumente em 10%... E agora vemos como realmente o fez.

Summary

For each *BaseTransaction*
skip *expression*$_1$ **count** *expression*$_2$
order *att*$_{11}$, *att*$_{12}$, ... *att*$_{1n}$ **[when** *condition***]**
order *att*$_{21}$, *att*$_{22}$, ... *att*$_{2n}$ **[when** *condition* | **otherwise]**
using *DataSelector*(*parm*$_1$, ..., *parm*$_n$)
unique *att*$_1$, ..., *att*$_n$
where *condition* **[when** *condition***]**
where *condition* **[when** *condition***]**
where *att* **in** *DataSelector*(*parm*$_1$,..., *parm*$_n$)
blocking *NumericExpression*
...
Attribute$_1$ = *expression*$_1$
Attribute$_2$ = *expression*$_2$
...
Attribute$_N$ = *expression*$_N$
...
When duplicate
...
When none ...
endfor

	Uniqueness check	Referential Integrity check
Assignment	✓	✗

COMMIT

Transaction integrity

Commit on exit	Yes

Em suma, para atualizar registros especificamente por procedimento, temos o comando For each.

Em seu corpo, além de poder fazer outras coisas, podem ser atualizados tanto atributos da tabela base como da tabela estendida. A única restrição é que a chave primária da tabela base do for each não pode ser atualizada.

Por outro lado, havíamos visto que o único controle programático que se realiza é o controle de unicidade. Para o caso do for each, será controlado que nenhuma chave candidata se repita. Se o for each descobre que, se realiza a atribuição, isso repetiria essa chave, então não faz nada, a menos que seja programada a cláusula when duplicate.
No código desta cláusula, pode ser programado um new para inserir um novo registro, por exemplo.

Sabemos que o for each não faz controle de integridade referencial, por isso tentará realizar a atualização que se tenha especificado independentemente da existência ou não do registro referenciado. Isto é por motivos de desempenho. Mas as bases de dados em geral realizam a verificação, a menos que desativemos essa funcionalidade, portanto, se não for desativada e a integridade falhar, lançarão uma exceção.

Por último: para que o registro fique commitado na base de dados, devemos nos certificar de que o comando Commit seja executado. Em um procedimento, por padrão, é colocado um Commit implícito ao final

(desde que seja entendido que no Source está sendo acessada em algum lugar a base de dados para atualizá-la). Mas podemos escrever explicitamente Commits no Source, onde seja conveniente para nós.

Não veremos aqui, mas, opcionalmente, pode ser especificada uma cláusula Blocking, que o que faz é permitir fazer atualizações em bloco, em vez de registro a registro. Ou seja, processará os registros em blocos de N para reduzir os acessos e melhorar o desempenho.

Por último, algo que não dissemos até o momento é que as redundâncias não são mantidas automaticamente quando se fazem atualizações por procedimento. É responsabilidade do desenvolvedor fazer isso. Isto significa que se uma redundância depende de um atributo que está sendo atualizado, GeneXus não buscará ou calculará o novo valor para armazenar no atributo redundante. O desenvolvedor tem que fazer isso.

Sobre tudo isto podemos encontrar mais informações em nossa wiki.

GeneXus™

training.genexus.com
wiki.genexus.com