Manejo del Commit y Niveles de aislamiento

GeneXus™

Los objetos GeneXus de tipo Transacción y Procedimiento ofrecen la propiedad Commit on exit que puede tomar el valor Yes o No. De esta manera los programas generados ejecutarán un Commit automático o no.

Propiedad Commit on Exit

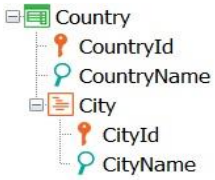
Commit on Exit = Yes

Procedimientos

```
For each Country
     Where CountryId = 2
     CountryName = "New country name"
endfor
```

Se incluye un Commit automático al final del source en los programas generados.

Transacciones



Se incluye un Commit automático en los programas generados, luego de realizar una modificación en la base de datos.

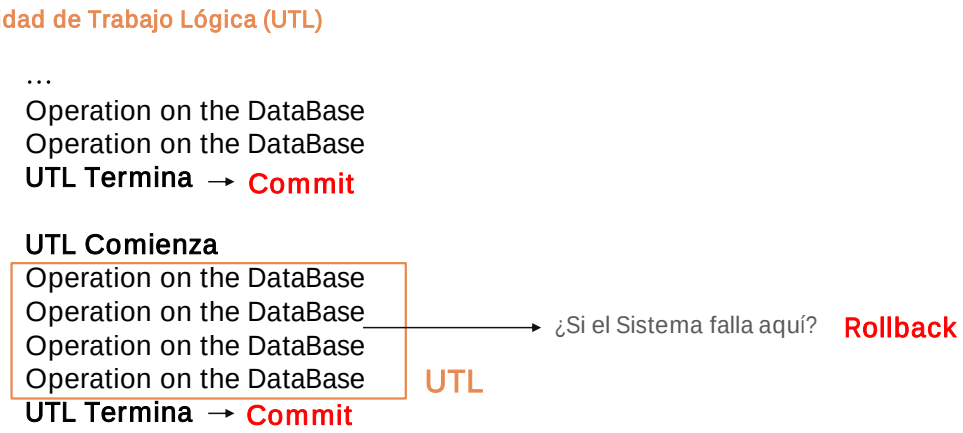
Por defecto, GeneXus incluye una sentencia Commit en los programas generados asociados a los objetos de tipo Transacción y Procedimiento.

- En los Procedimientos, se incluye un Commit automático al final del Source en los programas generados.
- En las Transacciones se incluye un Commit automático en los programas generados, luego de realizar una modificación en la base de datos. O sea, luego de insertar, modificar o eliminar datos, e inmediatamente **antes** de ejecutar el código asociado a las reglas condicionadas al momento de disparo AfterComplete y el código asociado al evento AfterTlrn.

Por ejemplo, si se tiene la transacción Country, con City como segundo nivel, y se insertan 5 países a través de su form, el Commit se ejecutará 5 veces, luego de grabada la información de cada país y sus ciudades, pero antes de la ejecución de las reglas condicionadas al momento AfterComplete.

Propiedad Commit on Exit

¿Qué motivos pueden existir para no ejecutar un Commit en una Transacción o en un Procedimiento?

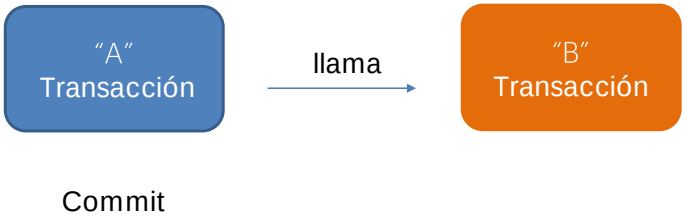


¿Qué motivos pueden existir para no ejecutar un Commit en una transacción o en un procedimiento?

Para personalizar una Unidad de Trabajo Lógica (LUW – Logical Unit of Work)

Esto significa que sea necesario expandir una UTL, de manera tal que, por ejemplo, varios procedimientos, o una transacción con uno o varios procedimientos, conformen una unidad de trabajo lógica, y sea necesario que un Commit abarque a todos ellos

Restricciones

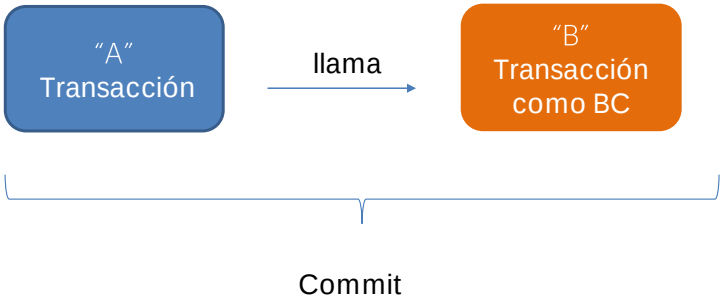


Veamos ahora algunas restricciones importantes en relación a este tema

En ambientes web, cada transacción puede commitear solamente su propio conjunto de operaciones realizadas sobre la base de datos, y de los procedimientos invocados por ella, pero no las operaciones realizadas por otra transacción.

O sea, si una transacción llama a otra transacción, el Commit ejecutado por una transacción no aplica a los registros insertados, modificados o eliminados por la otra. Por lo tanto, dos transacciones diferentes no pueden ser incluidas en una misma UTL

Restricciones



La propiedad Commit on Exit será ignorada en transacciones utilizadas como Business Component.

En caso de necesitar ejecutar operaciones a través de dos transacciones diferentes, y se desea conformar entre ellas una sola UTL, la solución es ejecutarlas utilizando el concepto de Business Component, incluyendo, entonces, el comando Commit luego de ejecutar las operaciones asociadas a ambas transacciones.

Vale destacar que la propiedad Commit on exit será ignorada en transacciones utilizadas como Business Component.

Esto significa que, a pesar de que la transacción tenga la propiedad Commit on exit con el valor Yes, si la misma se utiliza como Business Component, el commit no se ejecutará en forma automática, y será necesario declarar el comando commit en forma explícita.

El motivo de este comportamiento es permitir especificar unidades de trabajo lógica entre múltiples transacciones, incluyendo el comando commit donde sea necesario.

Propiedad Commit on Exit - Ejemplos

1

Commit on Exit = Yes


```
For each Country
     Where CountryId = 2
     CountryName = "New country name"
endfor
Commit
```

La propiedad Commit on Exit será considerada y GeneXus agregará el Commit en forma automática.

Vamos a entonces a plantear cuatro ejemplos muy concretos, y analizaremos el comportamiento:

Veamos entonces el primer ejemplo:

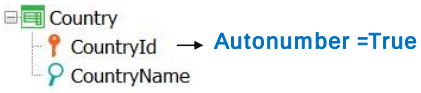
Supongamos la transacción Country y el source del procedimiento que se muestra. El procedimiento tiene la propiedad Commit on Exit con el valor Yes.

Como el For each realiza una actualización a la base de datos, GeneXus agregará el commit en forma automática y el país con valor CountryId = 2 quedará actualizado con su nuevo nombre.

Propiedad Commit on Exit - Ejemplos

2

Business Component = Yes



Commit on Exit = Yes

```
&Country.CountryName = "Brazil"
&Country.Insert()
```

La propiedad Commit on Exit será ignorada y el país no será commiteado.

Veamos el siguiente:

Supongamos la misma Country declarada como Business Component, y el source del procedimiento que se muestra:

El atributo Countryld es autonumerado, y el procedimiento tiene la propiedad Commit on Exit con el valor Yes.

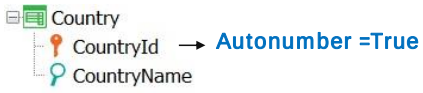
¿Qué comportamiento se tendrá?

Como el procedimiento solamente realiza operaciones con el Business Compoent, aunque el procedimiento tenga la propiedad Commit on Exit con el valor Yes, será ignorada y el país no se commiteará. Para lograrlo es necesario agregar el comando Commit en forma explícita.

Commit on Exit property - Ejemplos

3

Business Component = Yes



Commit on Exit = Yes

```
&Country.CountryName = "Brazil"
&Country.Insert()

For each Country
     Where CountryId = 2
     CountryName = "New country name"
endfor
```

La propiedad Commit on Exit será considerada y tanto las operaciones del BC como las del For each serán commiteadas.

Pasemos al siguiente ejemplo:

Supongamos otra vez la misma transacción Country declarada como Business Component, y el souce del procedimiento que se muestra:

El atributo CountryId es autonumerado, y el procedimiento tiene la propiedad Commit on Exit con el valor Yes

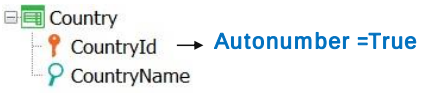
En este ejemplo, al tener operaciones con un Business Component en el source de un procedimiento, y tener además un For each que realiza actualizaciones a la base de datos, entonces sí se considera el valor Yes de la propiedad Commit on Exit y se commitean tanto las operaciones con el Business Component como las operaciones del For each.

Por lo tanto, se insertará el país de nombre Brazil y además el país con valor CountryId = 2 cambiará su nombre.

Commit on Exit property - Ejemplos

4

Business Component = Yes



Commit on Exit = Yes

```
For each Country
     Where CountryId < 3
     &Country.Load(CountryId)
     &Country.CountryName = "New country name"
     &Country.Update()
endfor
```

La propiedad Commit on Exit será ignorada.

Veamos el ultimo ejemplo:

Nuevamente, consideramos la misma transacción Country declarada como Business Component, y el souce del procedimiento que se muestra:

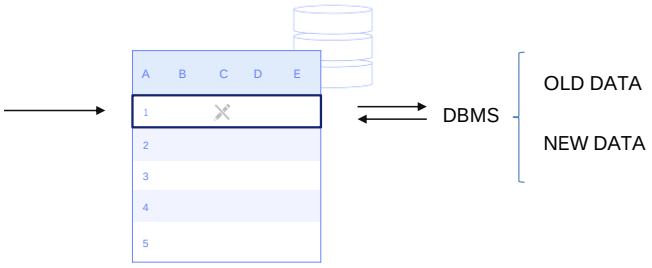
El atributo CountryId es autonumerado, y el procedimiento tiene la propiedad Commit on Exit con el valor Yes.

En este caso el valor Yes de la propiedad Commit on Exit también será ignorado, porque la actualización que se intenta hacer es a través del Business component y no directamente por el For each. Si este Business Component no estuviera, el For each por sí solo no realiza ninguna acción.
Por lo tanto, será necesario agregar el comando Commit en forma explícita.

Entonces, después de haber visto estos ejemplos, como regla general podemos decir que un objeto con la propiedad Commit on Exit en Yes hará Commit al finalizar, solamente si dicho objeto realiza alguna actualización a la base de datos que no sea solamente a través de un Business Component.

Niveles de aislamiento

Read Only



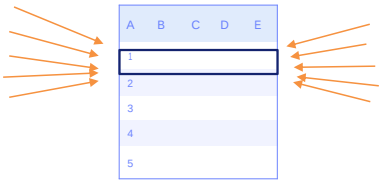
Bien. Veamos algo más:

Si bien no nos vamos a concentrar ahora en estudiar el control de concurrencia, vale mencionar que cuando varios usuarios realizan operaciones sobre la base de datos, si la información a leer está bloqueada por otro programa de escritura, los valores que se mostrarán dependerán del DBMS. Es el DBMS entonces quien decidirá si mostrar el valor antiguo o el nuevo de los datos que se consultan.

Especificar el **nivel de asilamiento** afecta la lectura y el control de concurrencia

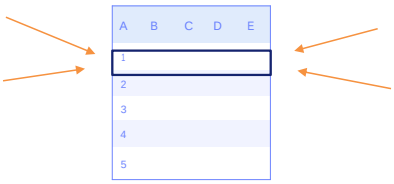
Niveles de aislamiento

Low isolation level



Increases simultaneous access effects

High isolation level



Reduces simultaneous access effects

En los sistemas de bases de datos, el aislamiento determina la forma en que la integridad de las transacciones a la base de datos es visible para otros usuarios y sistemas.

- Un nivel de aislamiento más bajo aumenta la capacidad de muchos usuarios para acceder a los mismos datos al mismo tiempo, pero aumenta también la cantidad de efectos de simultaneidad, como por ejemplo las lecturas sucias o actualizaciones perdidas, que los usuarios pueden encontrar. Aumenta también la posibilidad de interbloqueo, lo que requiere técnicas muy cuidadosas de análisis y programación para evitarlo.

- Por el contrario, un nivel de aislamiento más alto reduce los tipos de efectos de simultaneidad que pueden encontrar los usuarios, pero requiere más recursos del sistema y aumenta las posibilidades de que una transacción de base de datos bloquee a otra. El DBMS generalmente adquiere bloqueos en los datos que pueden resultar en una pérdida de concurrencia o implementa un control de concurrencia de múltiples versiones. Esto requiere agregar lógica para que la aplicación funcione correctamente.

Niveles de aislamiento

Los niveles de aislamiento son los siguientes:

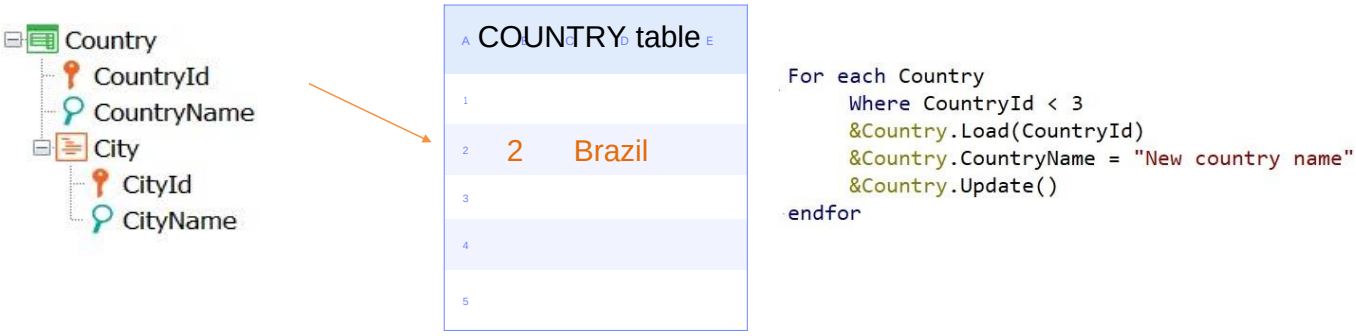
- **Serializable**: Es el nivel de aislamiento más alto. Requiere que los bloqueos de lectura y escritura (adquiridos en datos seleccionados) se liberen al final de la transacción. Cuando se utiliza el control de concurrencia no basado en bloqueos, si el sistema detecta una colisión de escritura entre varias transacciones simultáneas, solo una de ellas puede comprometerse.

- **Lecturas repetidas**: En este nivel de aislamiento, una implementación DBMS de control de concurrencia basada en bloqueos, mantiene los bloqueos de lectura y escritura hasta el final de la transacción. Sin embargo, los bloqueos de rango no se administran, por lo que pueden ocurrir lecturas fantasmas.

- **Leer confirmados:** Es un nivel de aislamiento que garantiza que cualquier lectura de datos se confirme en el momento en que se lea. Simplemente impide que el lector vea cualquier lectura intermedia, no comprometida y 'sucia'.

- **Leer no confirmados:** Es el nivel de aislamiento más bajo. Se permiten lecturas sucias, por lo que una transacción de base de datos puede ver cambios realizados por otras transacciones que aún no se han confirmado.

Niveles de aislamiento - Ejemplo



If Isolation level is Read Committed – CountryName = **"Brazil"**

If Isolation level is Read Uncommitted – CountryName = **"New country name"** DIRTY READ

Veamos un ejemplo para graficar estos conceptos:

Consideremos la tabla COUNTRY.
Un usuario accede al registro 2 y obtiene Brazil como valor del atributo CountryName
Luego ejecuta un código como el que estamos viendo.

¿Qué debemos tener en cuenta? !ue al tratarse de un Business Component y no haberse escrito explícitamente el commit en la base de datos, físicamente se mantiene el nombre "Brazil" hasta que efectivamente se ejecute un Commit.

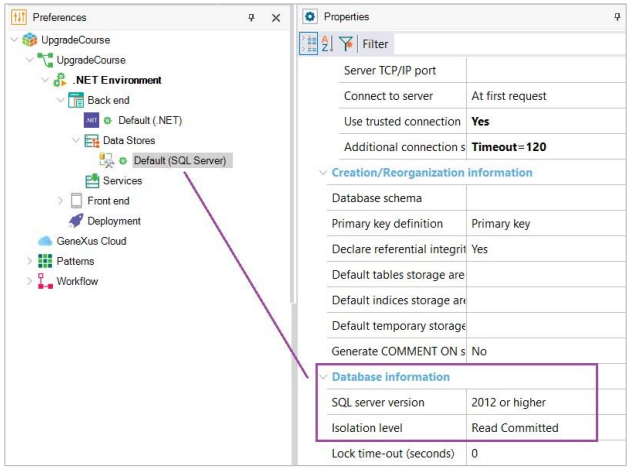
Ahora bien, antes de que se ejecute un commit entra otro usuario a leer el registro 2 y es entonces que dependiendo del nivel de aislamiento obtendrá diferentes valores

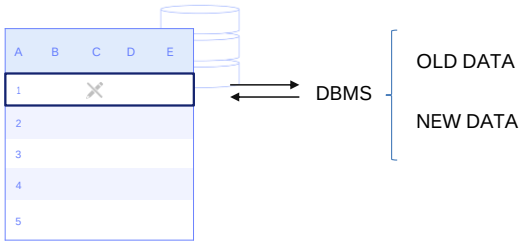
- Si el nivel de aislamiento es Read Commited, entonces obtiene el valor ″Brazil″ para el atributo CountryName
- Si el nivel de aislamiento es Read Uncommitted, entonces obtiene ″New Country Name″ como valor del atributo CountryName

Entonces:
Read Uncommited es un nivel bajo de aislamiento, mientras que Read Committed es un nivel alto.
Asociado al nivel Read Uncommitted se encuentra el concepto ″Dirty read″, ″lectura sucia″, que precisamente hace referencia a la posibilidad de leer datos que aún no han sido confirmados.

En cuanto a los niveles Serializable y Repeat Reading son aún más restrictivos ya que agregan un lock, o sea que el nuevo usuario ni siquiera podrá acceder al registro 2 porque estará lockeado

GeneXus – Isolation Level property





En GeneXus, la propiedad Isolation Level ubicada a nivel del Data Store, permite especificar el nivel de aislamiento de los cambios realizados por los programas.

Los posibles valores son los siguientes:

- **Read committed:** Los programas no ven los cambios realizados por otros usuarios hasta que se ejecute el commit. Este es el valor por defecto que toma esta propiedad.
- **Read Uncommitted:** En este caso los programas ven los cambios realizados por otros usuarios aún cuando todavía no se ha ejecutado el Commit.

Como hemos visto, especificar el nivel de asilamiento afecta la lectura de los datos y el control de concurrencia.

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