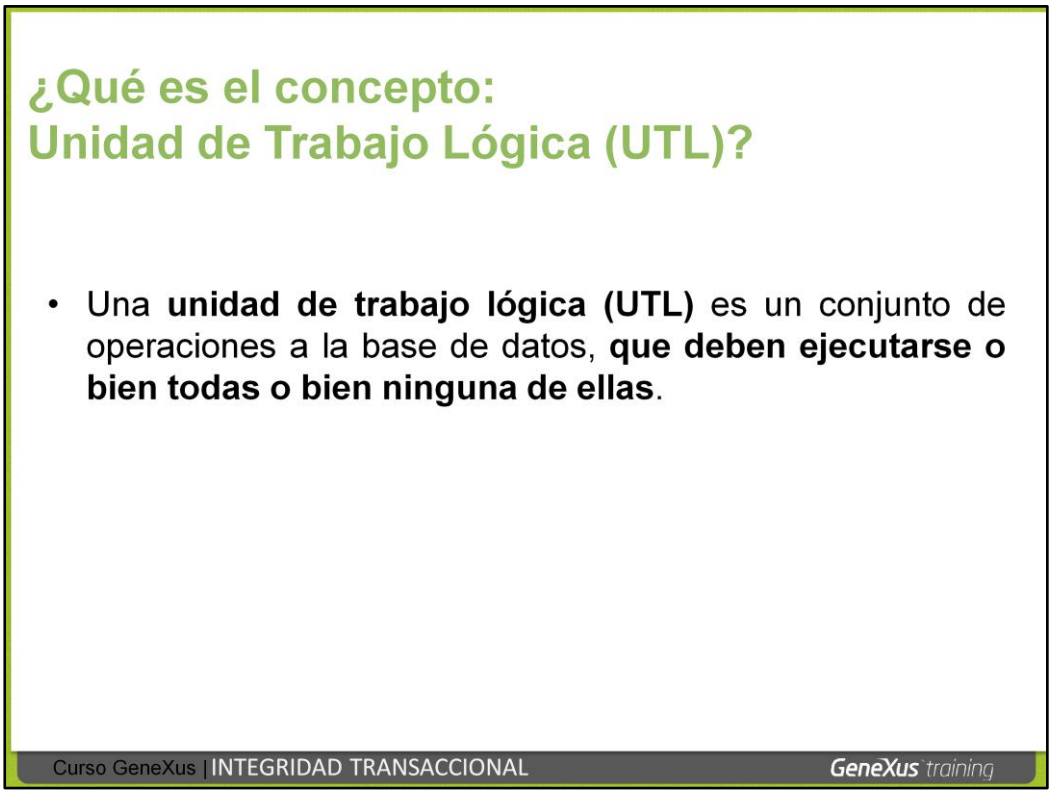
INTEGRIDAD TRANSACCIONAL
Base de datos consistente a pesar de caídas abruptas

CONCEPTOS

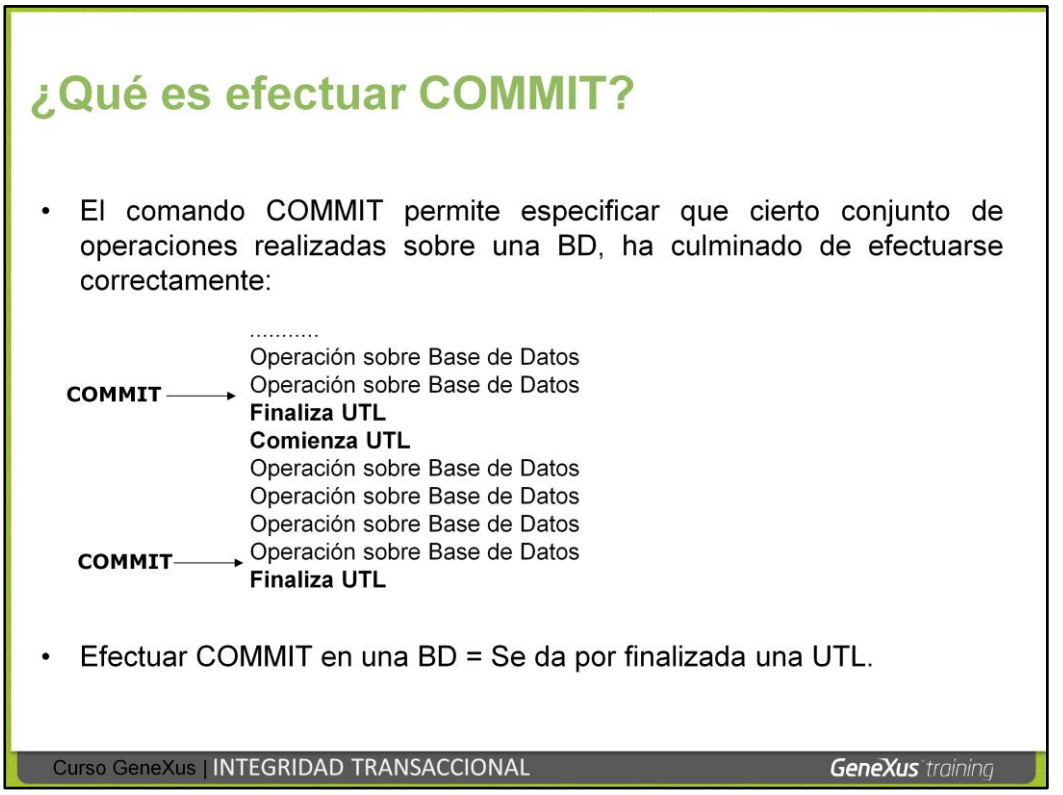
¿Qué es el concepto: Integridad Transaccional (IT)?

- Un conjunto de actualizaciones a la base de datos tiene **integridad transaccional** cuando en caso de una finalización “anormal”, la base de datos permanece en estado consistente.

Muchos manejadores de bases de datos (DBMSs) cuentan con sistemas de recuperación ante fallos, que permiten dejar la base de datos en estado consistente cuando ocurren imprevistos tales como apagones o caídas del sistema



Los manejadores de bases de datos (DBMSs) que ofrecen integridad transaccional permiten establecer unidades de trabajo lógicas (UTLs), que corresponden ni más ni menos que al concepto de transacciones de base de datos.

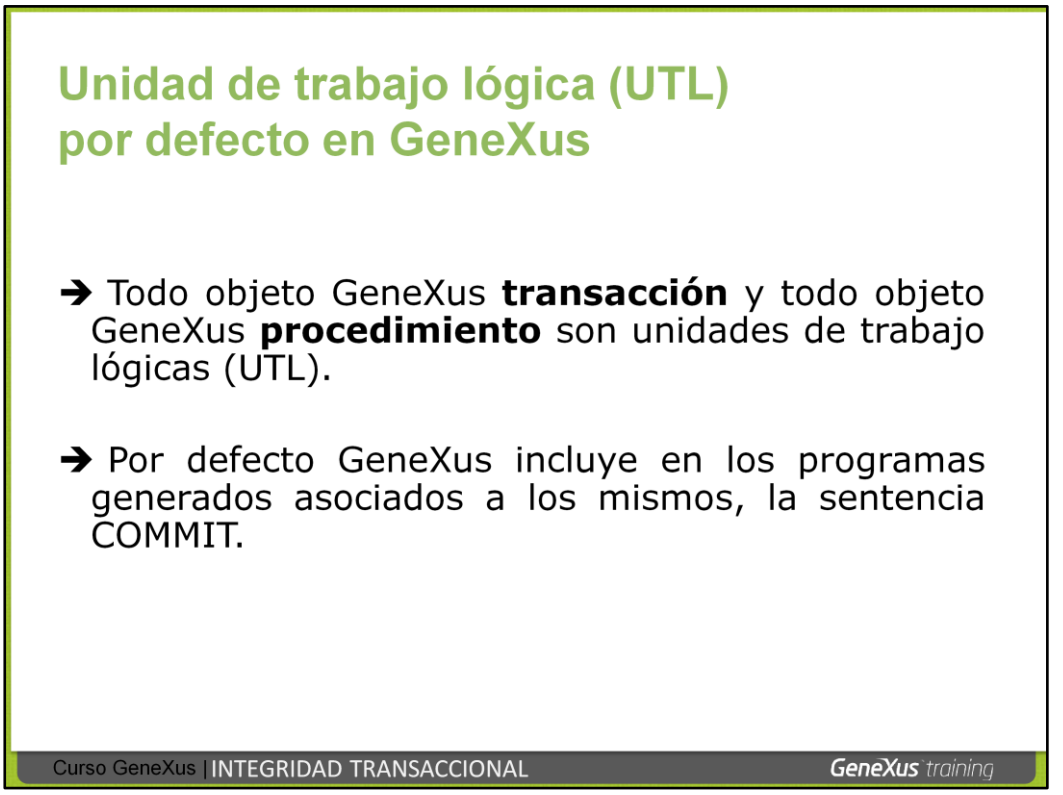


Podemos ver que una unidad de trabajo lógica (UTL) queda definida por el conjunto de operaciones entre un par de Commits.

¿Qué es efectuar ROLLBACK?

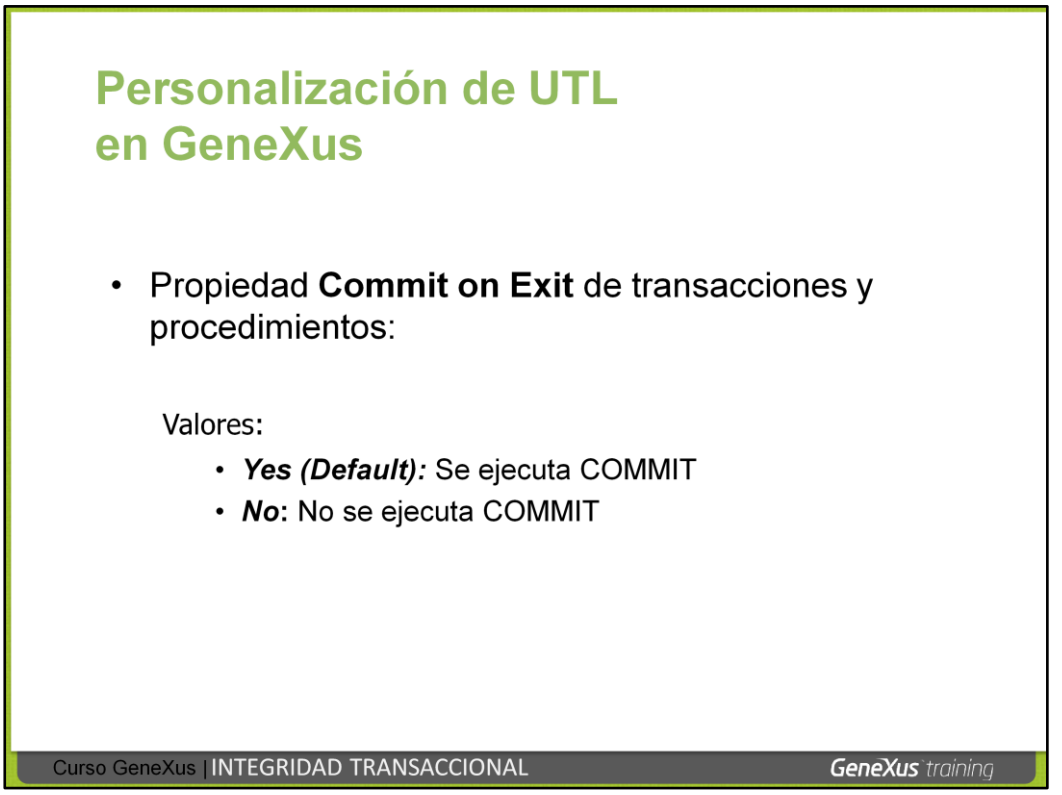
- Hacer **ROLLBACK (vuelta a atrás)** provoca que se deshagan todas las operaciones efectuadas en la base de datos que no hayan quedado con COMMIT.
- Esto se resuelve deshaciendo todas las operaciones posteriores al último COMMIT.

LOS MISMOS CONCEPTOS EN GENEXUS..



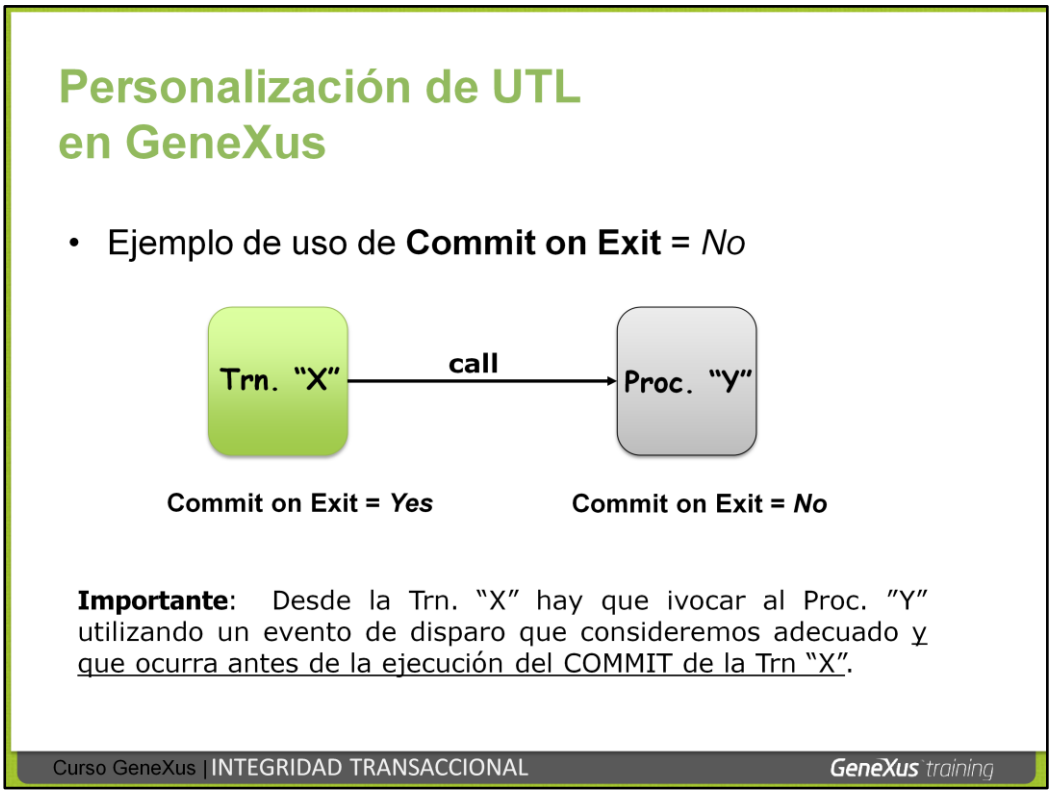
Por defecto GeneXus incluye en los programas generados asociados a los mismos, la sentencia COMMIT.

- ➔ En el objeto procedimiento: COMMIT automático **al final del Source.**
- ➔ En el objeto transacción: COMMIT automático **al final de cada instancia, inmediatamente antes de las reglas con evento de disparo AfterComplete.**



GeneXus ofrece una propiedad a nivel de cada objeto transacción y procedimiento, para definir si se quiere que el programa generado asociado al objeto efectúe COMMIT, o no.

El nombre de la propiedad es **Commit on Exit** y su valor por defecto es Yes (por eso, toda transacción y procedimiento por defecto efectúan COMMIT).



¿Por qué motivo se puede necesitar no efectuar COMMIT?
Para personalizar una unidad de trabajo lógica (UTL). Es decir, podemos necesitar ampliar una unidad de trabajo lógica (UTL) para que varios procedimientos o una transacción y algún procedimiento, juntos, conformen una única unidad de trabajo lógica (UTL) transacciones[1].

Ejemplo (mostrado arriba):

La transacción "X" invoca al procedimiento "Y" y se desea que ambos objetos conformen una única UTL. Es decir, la transacción actualiza ciertos registros y el procedimiento otros, y se desea que ese conjunto total de operaciones conforme una única UTL (para asegurarnos que si ocurre una falla, quede efectuado o bien el conjunto completo de actualizaciones a la base de datos, o nada).

Para lograrlo podemos eliminar el COMMIT del procedimiento y dejar que se realice en la transacción (al retornar del procedimiento a la transacción, para que se ejecute al final de todas las operaciones).

De modo que configuraríamos la propiedad **Commit on Exit** del procedimiento con valor: **No** y dejaríamos la propiedad **Commit on Exit** de la transacción con el valor por defecto: **Yes**. Pero además de esto, es fundamental que la invocación al procedimiento se realice antes de que se ejecute el COMMIT en la transacción (ya que la idea es que ambos objetos conformen una única UTL y para ello el COMMIT debe efectuarse en la transacción al retornar del procedimiento). Así que la invocación al procedimiento deberá definirse en la transacción, con un evento de disparo que ocurra antes de la ejecución del COMMIT (dependiendo de si la transacción es de un nivel o más, y de los requerimientos, podría servir AfterInsert por ejemplo, AfterUpdate, o AfterLevel Level *Atributo del 2do nivel*, o BeforeComplete, pero no AfterComplete).

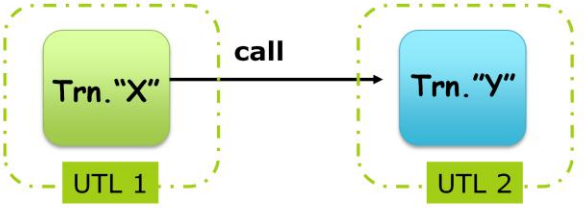
No existe una única solución para personalizar una UTL. Lo fundamental es analizar cuál objeto puede hacer COMMIT (pudiendo haber más de una posibilidad) y una vez que se decida cuál objeto efectuará COMMIT, las invocaciones que se requieran hacer, deberán efectuarse en momentos adecuados, considerando si ya se efectuó el COMMIT o no.

---

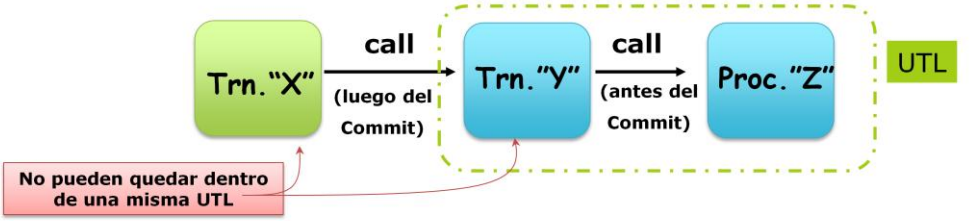
[1] En ambiente Web existe una importante restricción a este respecto: si desde una transacción se invoca a otra, el Commit que realice una no aplica sobre los registros ingresados/modificados/eliminados por la otra. Es decir, el Commit de cada transacción solo tiene "visibilidad" sobre los registros operados por esa transacción, y no por la otra, por lo que dos transacciones distintas no pueden quedar incluidas en una misma UTL. No puede realizarse personalización en este caso.

Personalización de UTL en GeneXus

- No puede definirse una UTL compuesta por varias transacciones Web.

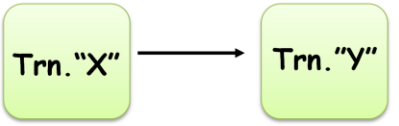


- Una transacción Web sólo puede Commitear los registros insertados por ella misma, o por procedimientos en una cadena de invocaciones, pero no puede Commitear los registros insertados por otra transacción..

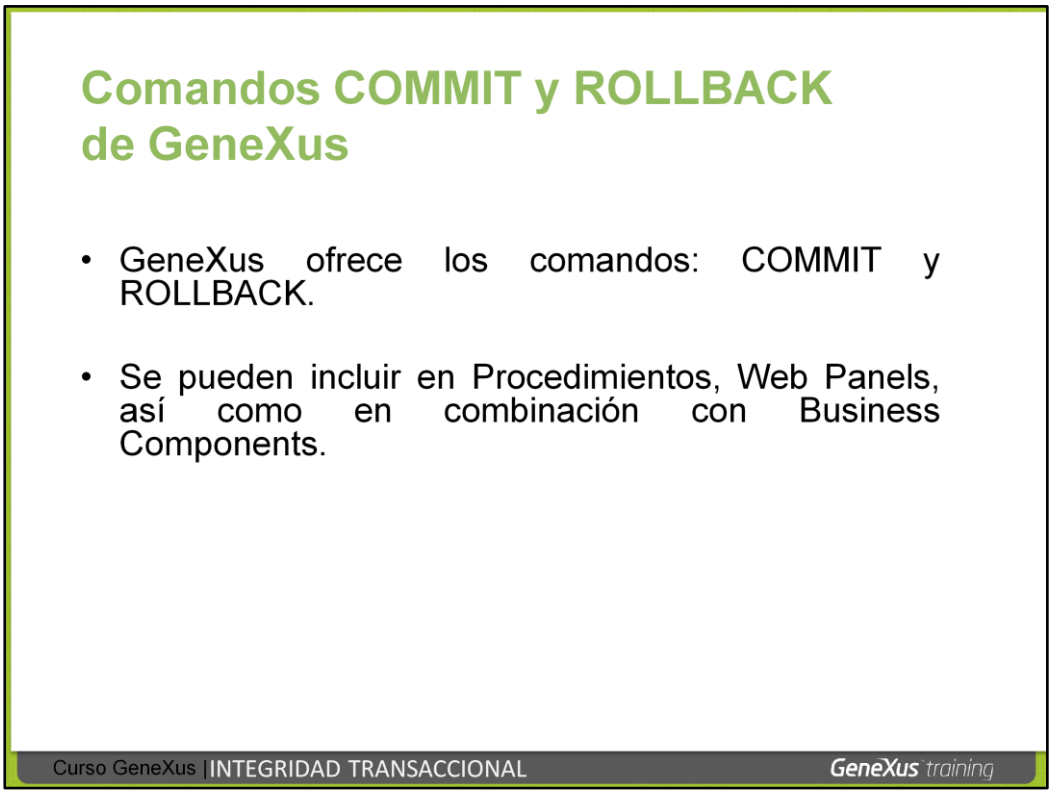


Personalización de UTL en GeneXus

- Si deseamos que las inserciones mediante dos transacciones distintas conformen una única UTL:



Tenemos una solución: Utilizar Business Components y el comando Commit al terminar de insertar los registros asociados a ambas transacciones.



Ejemplo: El usuario final decide si ejecutar Commit o Rollback

Se invoca desde un web panel (en determinado evento) a varios procedimientos consecutivos. Se les configura a todos ellos la propiedad **Commit on exit = No.**
La sentencia siguiente a la invocación al último procedimiento es una pregunta al usuario sobre si confirma todo lo ejecutado o no. Dependiendo de la respuesta del usuario, habrá que ejecutar el comando COMMIT o ROLLBACK.