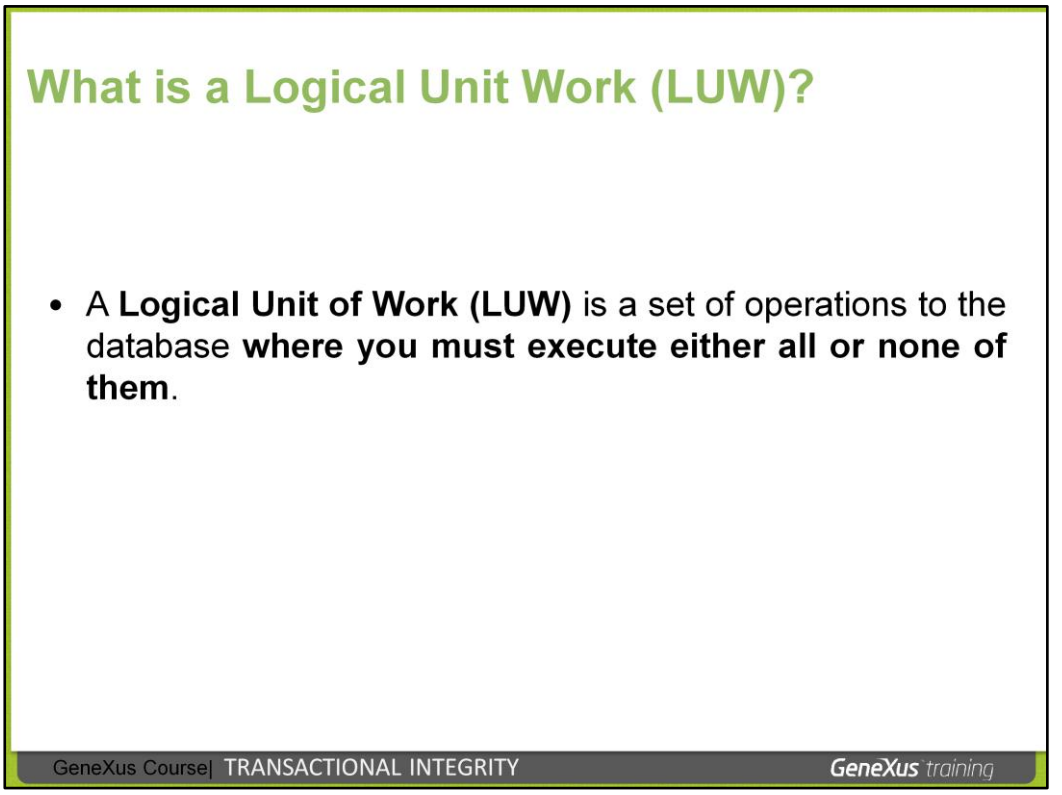
TRANSACTIONAL INTEGRITY

Database consisting in spite of "abnormal" endings

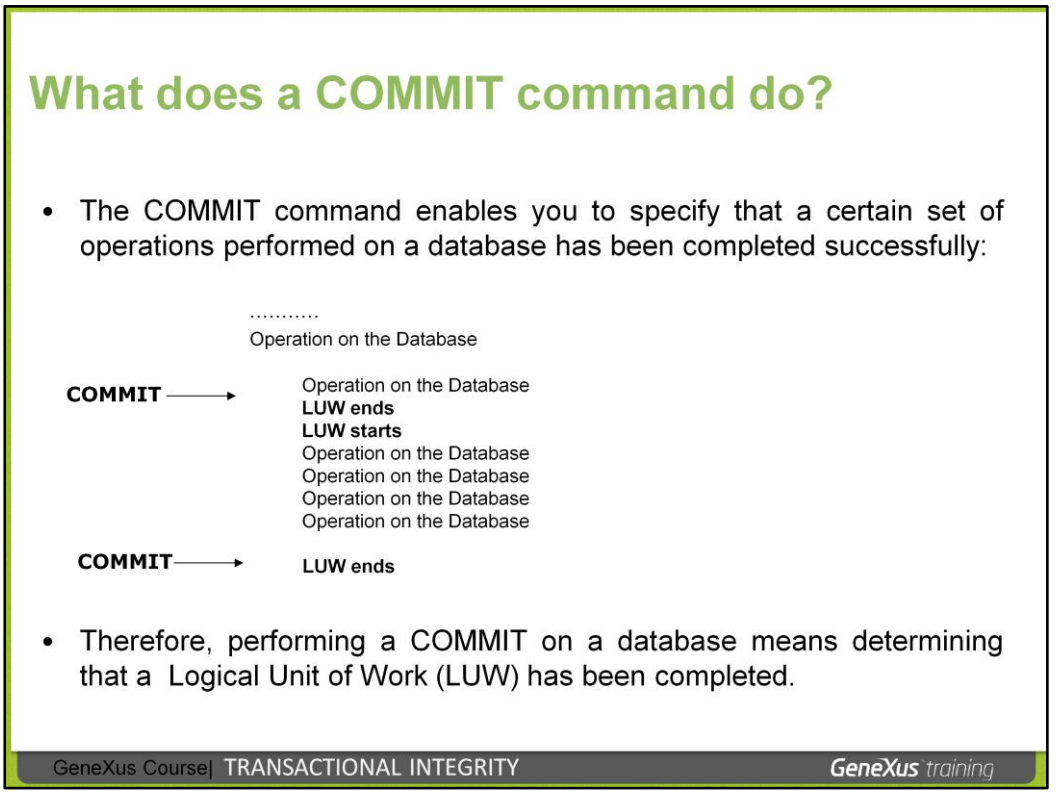
What does the term Transactional Integrity (TI) means?

- A set of updates to the database has **transactional integrity** when in case of "abnormal" ending, the database remains in a consistent state.

Many database managers (DBMSs) have failure recovery systems that ensure the database is in a consistent state when unforeseen events such as power cuts or system failures occur.



Database managers (DBMS) that offer transactional integrity allow establishing Logical Unit of Work (LUW) that, in fact, correspond to the database transaction concept.

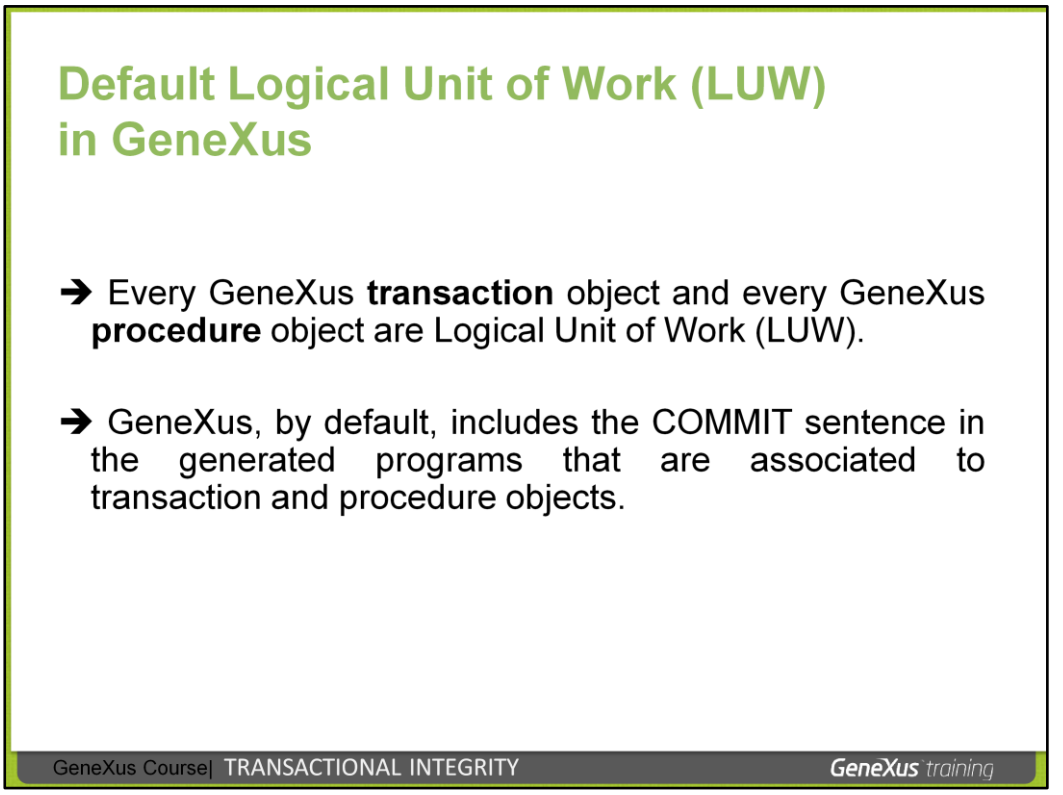


We can see that a Logical Unit of Work (LUW) is defined by the set of operations performed between two Commit commands.

What does a ROLLBACK command do?

- A **ROLLBACK** command enables you to undo all those operations performed on the database and which were not confirmed with COMMIT.
- This is done by undoing all the operations that came after the last COMMIT.

THE SAME CONCEPTS IN GENEXUS..



GeneXus, by default, includes the COMMIT sentence in the generated programs that are associated to transaction and procedure objects.

→ **In procedure object:** authomatic COMMIT at the end of the source program.

→ **In transaction object:** authomatic COMMIT at the end of each instance immediately before the rules with an AfterComplete.

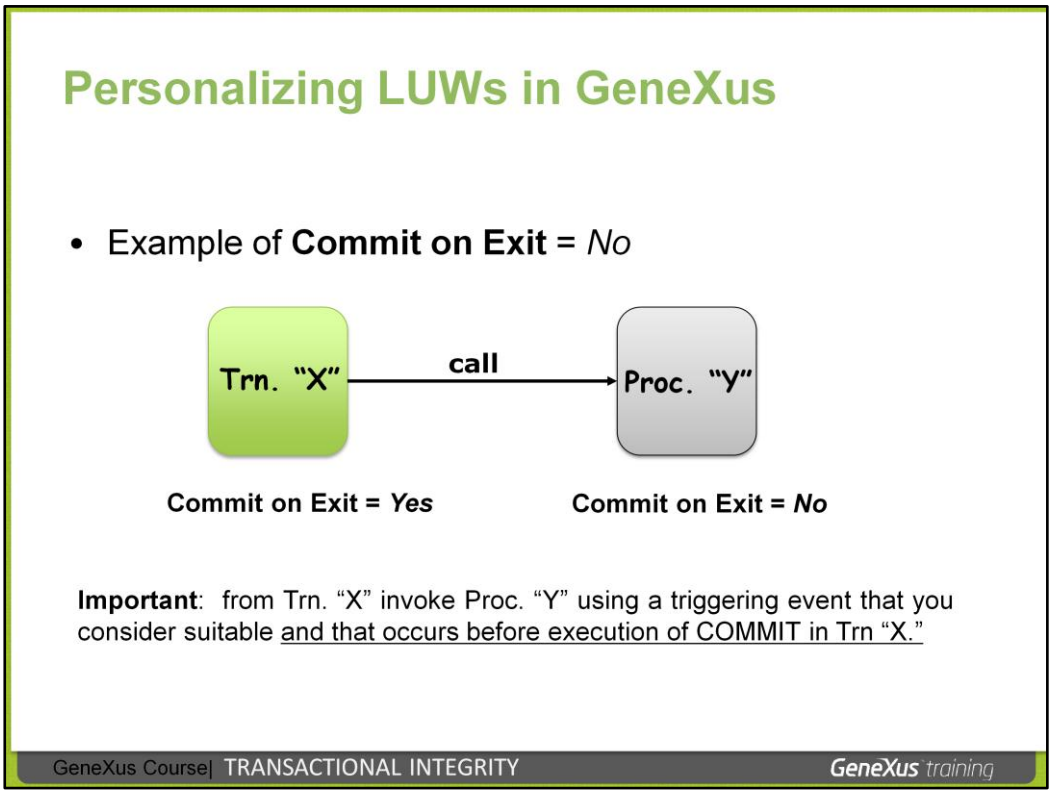
Personalizing LUWs in GeneXus

- **Commit on Exit** transaction and procedure property:

  Values:
  - ***Yes (Default):*** COMMIT is executed.
  - ***No*:** COMMIT is not executed.

GeneXus offers a property for each transaction and for each procedure in order to define if you wish the generated program perform COMMIT or not.

The name of the property is **Commit on Exit** and its default value is Yes (this is why all transactions and procedures perform COMMIT by default).



What reasons could there be for not performing a COMMIT in a transaction or procedure?

To personalize a Logical Unit of Work (LUW). That is, we may need to expand a Logical Unit of Work (LUW) so that several transactions[1] and/or procedures form a single Logical Unit of Work (LUW).

Example (shown above):

Transaction "X" invokes procedure "Y," and we want both objects to form a single LWU. The transaction updates certain records, and the procedure updates other records, and we want that whole set of operations to form a single LUW (to guarantee that, if a failure occurs, either the whole set of updates is performed on the database, or no update at all).

To achieve this, we can remove the COMMIT from the procedure and allow it to be performed in the transaction (when returning from the procedure to the transaction, so that it is executed at the end of all the operations); therefore, we would set the value of the procedure's **Commit on Exit** property to **No** and we would leave the transaction's **Commit on Exit** property set to **Yes**, the default value. But, in addition, it is vital that the procedure is invoked before the COMMIT is performed in the transaction (since both objects should form a single LUW and, in order for this to be possible, the COMMIT must be performed in the transaction when returning to the procedure); therefore, the invocation to the procedure must be defined in the transaction, with a triggering event occurring before the COMMIT (depending on the requirements and on whether it is a single-level or multi-level transaction; either AfterInsert, for instance, AfterUpdate, AfterLevel Level 2nd Level Attribute or BeforeComplete would do, but not AfterComplete).

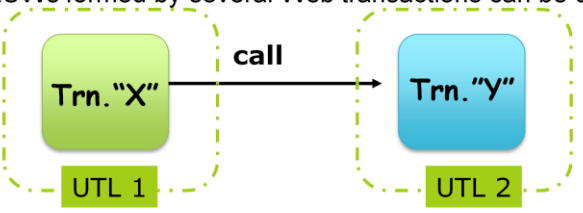
There is no single way of personalizing a LWU. What is important here is analyzing which object can perform a COMMIT (there could be more than one possibility) and, once the object has been chosen, determining what we need to invoke and what the appropriate times to do that are, depending on whether the COMMIT has been performed or not.

---

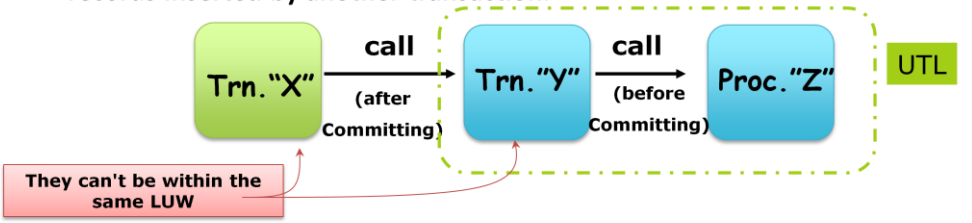
[1] [1] In Web environments, this has an important restriction: if a transaction is invoked from another transaction, the Commit performed by one of them does not apply to the records inserted/modified/deleted by the other one. That is, the Commit of each transaction is only "visible" on the records performed by that transaction, and not on the records performed by the other one; therefore, two different transactions cannot be included in the same LUW. In this case, no personalization is possible.

Personalizing LUWs in GeneXus

- No LUWs formed by several Web transactions can be defined.



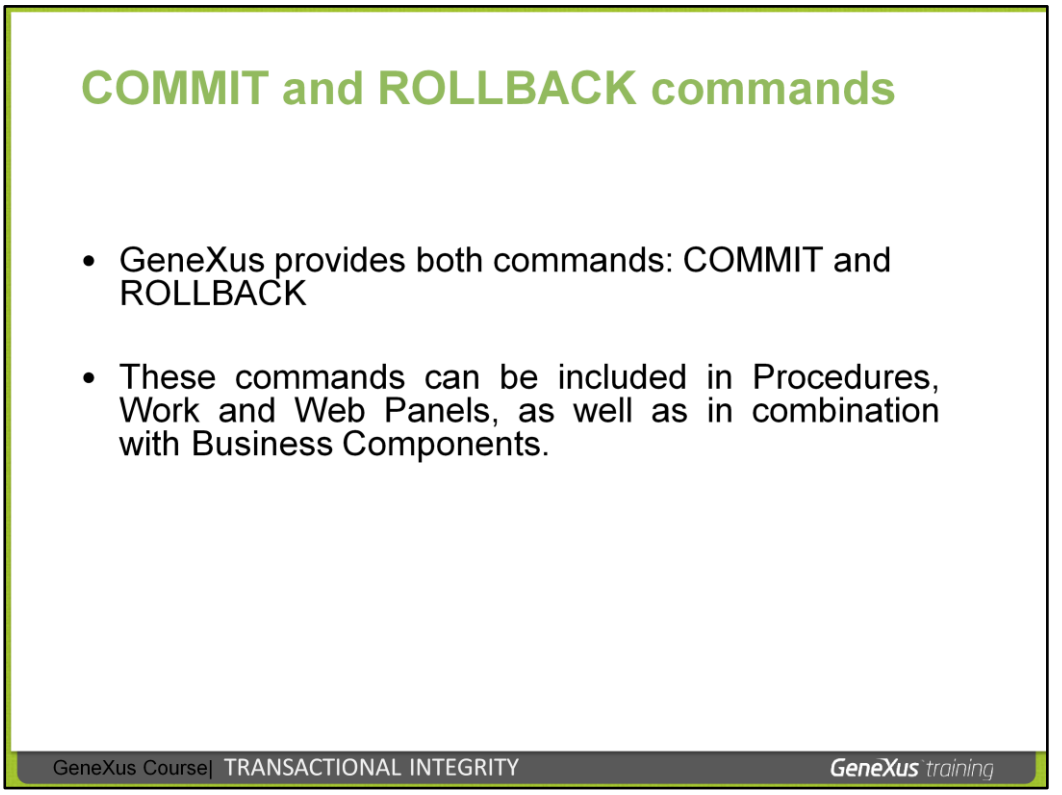
- The only records a Web transaction can Commit are those it itself inserted, or those inserted through procedures in an invocation chain, but **it cannot Commit records inserted by another transaction**.



Personalizing LUWs in GeneXus

- If we want insertions made through two different transactions to form a single LUW:

The solution we have is to use Business Components and the Commit command after inserting the records associated to both transactions through the Business Component variables (discussed later on).



Example (end user decides whether to execute Commit or Rollback):

Suppose we invoke from a web panel (in a certain event) several consecutive procedures. The **Commit on exit = No** property is configured in all of them.

The next sentence after invoking the last procedure, asks the user if he confirm; depending on the answer the user gives, either the COMMIT command or the ROLLBACK command will be executed.