Base Language - Bug #5743

finally block: finalizable processing

10/19/2021 01:19 PM - Constantin Asofiei

Status:	New	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:		**% Done:**	0%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Base Language - Bug #5651: conditions raised from a finally block	**WIP**

History

#2 - 10/19/2021 01:35 PM - Constantin Asofiei

- Start date deleted (10/19/2021)

In OE, it seems that a FINALLY block doesn't inherit the TRANSACTION state of the associated block. But this test proves that a lock is not being released until the associated block terminates, even if it states that there is not an open TRANSACTION at the FINALLY block:

- first program:

```
repeat transaction:
   find first book exclusive-lock.
   book.book-id = 12345678.
   leave.
   finally:
      message "transaction" transaction. // this shows 'no'
      pause. // start the second program
   end.
end.
```

- second program:

```
find first book exclusive-lock where book.book-id = 12345678. // this blocks for the lock to be released
message book.book-id.
```

The above test shows also the 'index leak dirty share' which requires the dirty database in FWD (just make sure to change the '12345678' to something else each time you run the program).

In #5700, we've changed certain finalizables (like parameter processing) to execute 'on top-level block return'. But there is a change in RecordBuffer.openScope:7757 which is incorrect:

```
      // Register this buffer as a Finalizable with the TM, so that its
      // finished method is called when this buffer scope is about to close.
      ProcedureManager.executeOnReturn(this::finished, this.weight());
```

as it forces to execute on return, regardless of the buffer's scope block.

But what's interesting is that a FINALLY does have a transaction if it e.g. sets EXCLUSIVE-LOCK:

```
repeat:
```

```
   leave.
   finally:
      find first book exclusive-lock.
      message "transaction" transaction.
   end.
end.
```

#3 - 10/19/2021 01:46 PM - Constantin Asofiei

I've tried changing that code to this:

```
      if (isTopLevelBlock())
      {
         ProcedureManager.executeOnReturn(this::finished, this.weight());
      }
      else
      {
         txHelper.registerFinalizable(this, false);
      }
```

but it is not enough. I think some other registerFinalizable calls need to be moved to 'executeOnReturn'.

Or do more investigations and find which (if any) of the FWD finalizables need to be processed **before** the FINALLY block.

#4 - 10/19/2021 02:20 PM - Ovidiu Maxiniuc

Actually, I think it depends on the location of the 'finalizable' is configured. If it is configured for a do block, it must not executeOnReturn. However, if the scope of a similar 'finalizable' is opened at procedure level, the executeOnReturn must be used.

Two notes:
1. Today, I noticed something else. There are some resources for whom the finalization is constructed when the code is generated. See register_finish. My searches through code failed to detect these registerFinalizable initially because the method name is split in AST template definition.

2. I think we will need a new parameter global for executeOnReturn for global resources to be finalized when the main top-level procedure returns. Currently, the global resources are finalized with the main scope of that procedure. But if it has a finally: block where these global resources are used, they might be already finalized. When this parameter is true, the method should search the first procedure which was stacked and add the global resource at that level instead of using peek and storing the clean-up code at current procedure level.

Please let me know if you think otherwise.

#5 - 10/19/2021 03:08 PM - Ovidiu Maxiniuc

I tried to see the problem from another POV. Taking into account these new issues, I think the proper way is to move the executeOnReturn() to BlockManager instead of ProcedureManager.

There are 7 methods in which the wa.tm stack is changed. These are all the cases where a finally: block can be defined. I think moving the data structure from the ProcedureManager to TransactionManager.TransactionHelper or doing a parallel management with wa.tm.pushScope() / wa.tm.popScope() will do the trick. The only problem remains to see:

- which resources must be finalized (are reinitialized in a loop block, for example) so they must be registered with the TransactionManager. They do not survive from one iteration to another and their scope does not include the associated finally: block, if one exist; and
- which just need a single time clean-up code to be executedOnExit() of the block.

#6 - 10/20/2021 01:19 PM - Constantin Asofiei

Ovidiu, the complexity from this arises from the fact that a i.e. RecordBuffer.finished can't be moved by its own to 'executeOnExit', even for buffers scoped to the top-level-block: there is an entire 'finalizable' mechanism in the p2j.persist package which is related to a buffer. And from the looks of it, even if the documentation at https://docs.progress.com/bundle/openedge-abl-error-handling-117/page/Transaction-scope-in-a-FINALLY-block.html states that the tx is completed (success or failure) before FINALLY is execute, the example above shows that the locking is not released.

So yes, we need tests (especially in the persist area) to determine which resources need to be finalized before the FINALLY and which after it.

An idea might be to take The transaction of the associated block is either complete (success) or undone (failure) literally and just complete the tx, and not 'finalize' the resources... this would be done after the FINALIZABLE.

#8 - 11/11/2021 11:22 AM - Greg Shah

- Related to Bug #5651: conditions raised from a finally block added