Database - Feature #6187

eliminate explicit buffer usage in can-find

03/17/2022 07:23 AM - Greg Shah

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

Description

History

#1 - 03/17/2022 07:24 AM - Greg Shah

I don't recall why we require this extra buffer that is not actually a buffer in the 4GL. I do recall how painful this makes the converted code. We are contorting ourselves to create fake buffers.

Why can't we just deal with this at runtime as needed and drop the conversion time buffer? I guess I don't understand why we need the buffer at runtime either but at least if it is really needed (and wanted) then we should implicitly handle this instead of making extra buffers in the converted code.

#2 - 03/18/2022 08:26 AM - Greg Shah

It seems in 2015, this may already have been resolved (from record_scoping_prep.rules):

```
** 022 ECF 20150317          Completely reworked CAN-FIND buffers. We no longer use "fake" buffers
**                           with the "*can-find-" unique name prefix. CAN-FIND buffers now use
**                           existing buffers where appropriate, but do not expand their scope.
**                           When there is no other reference to a buffer a CAN-FIND references,
**                           its references are converted from no-reference to free-reference
**                           (in canfind_scop_prep.rules), so a dedicated buffer with the proper
**                           scoping is created naturally.
```

Eric: Feel free to mark this as 100% Done and Rejected status if indeed there is nothing to do and nothing to discuss.

#3 - 03/25/2022 04:30 AM - Constantin Asofiei

This is still an issue. It can be see with this simple test:

```
.\a.p                                03/25/2022 01:26:03   PROGRESS(R) Page 1
```

```
{} Line Blk
-- ---- ---
      1     def temp-table tt1 field f1 as int.
      2
      3     if can-find(first tt1 where tt1.f1 = 0) then message "found".
      4     if can-find(first customer) then message "found".
.\a.p                                 03/25/2022 01:26:03   PROGRESS(R) Page 2

     File Name       Line Blk. Type   Tran            Blk. Label
-------------------- ---- ----------- ---- --------------------------------
.\a.p                   0 Procedure   No
```

No buffer scopes are opened in 4GL, while in FWD the program looks like:

```
@DatabaseReferences(aliases = 
{
   "p2j_test" 
})
public class Qryscope6
{
   Tt1_1_1.Buf tt1 = TemporaryBuffer.define(Tt1_1_1.Buf.class, "tt1", "tt1", false);

   Customer.Buf customer = RecordBuffer.define(Customer.Buf.class, "p2j_test", "customer", "customer");

   /**
    * External procedure (converted to Java from the 4GL source code
    * in qryscope6.p).
    */
   @LegacySignature(type = Type.MAIN, name = "qryscope6.p")
   public void execute()
   {
      externalProcedure(Qryscope6.this, new Block((Body) () -> 
      {
         RecordBuffer.openScope(tt1, customer);

         if ((new FindQuery(tt1, "tt1.f1 = 0", null, "tt1.recid asc", LockType.NONE).hasAny()).booleanValue())
         {
            message("found");
         }

         if ((new FindQuery(customer, (String) null, null, "customer.recid asc", LockType.NONE).hasAny()).bool
eanValue())
         {
            message("found");
         }
      }));
   }
}
```

Note that removing the implicit scope for the tt1 temp-table will cause runtime problems, as FWD assumes that if a buffer closes its last multiplex scope, it must empty the temp-table. So, if the implicit buffer is never used, and the program relies only on explicit buffers defined in an internal procedure, the temp-table will be emptied when that internal procedure ends.

#4 - 03/25/2022 06:23 AM - Greg Shah

We really should just implement a set of static canFind() methods which have the <dmo>.Buf.class as a parameter along with the WHERE clause and lock type. The rest can be handled inside the runtime. We can still use the FindQuery under the covers if that makes sense, though perhaps there are some optimization possibilities without the buffer usage. If we really need to keep using FindQuery then we can create a special buffer at runtime. This approach would eliminate the hard linkage to the business logic buffers.

There is still the implicit scope issue, I have no input on that.

#5 - 03/25/2022 06:27 AM - Greg Shah

In my opinion, the converted code will read better with canFind(...) than new FindQuery(...).hasAny().