Base Language - Bug #5035

Temp-Table index on P.L.O field

12/11/2020 03:29 AM - Marian Edu

Status:	Test	**Start date:**	
Priority:	Normal	**Due date:**	
Assignee:	Constantin Asofiei	**% Done:**	100%
Category:		**Estimated time:**	0.00 hour
Target version:			
billable:	No	**case_num:**	
vendor_id:	GCD	**version:**	

Description

Related issues:

Related to Base Language - Feature #4384: Builtin OO Implementation	**Closed**
Related to Runtime Infrastructure - Support #4086: review the RemoteObject ex...	**New**

History

#1 - 12/11/2020 03:29 AM - Marian Edu

Oddly enough in 4GL one can create an index on a P.L.O field, the ordering in this case is done on the 'resource identifier' - an int64 value. Did a sample test in table/index_plo.p.

There seems to be an issue with how FWD is assigning those resource id's - randomly instead of incremental as in 4GL where each and every object/handle created gets the next id (now I have to see what happens when they pass the long value limit). Apart from that, the behavior of index on P.L.O is escaping me - it's not in the order the records were added, doesn't seem to be ordered on P.L.O hash or 'toString', it's not even always the same on subsequent runs :)

The real use case is in Collections, AbstractTTCollection has a temp-table like that one an we can't get our items in order out of it :(

#2 - 12/11/2020 08:16 AM - Constantin Asofiei

Some notes here:

1. in the SQL temp-table, the progress.lang.object field is represented as a character (with the string representation of its hash), and not an integer. AFAIK this is how OE works.
2. in FWD, the index will sort these values lexicographically (as the field is character)
3. we use random integer values to represent the 'handle' value for an OO instance or any other handle. In 4GL, is just a coincidence that the handle values are incrementing - IIRC, at some point we did some tests and once the overflow value was reached (consider that the HANDLE function accepts a string representation of an integer), OE started to re-use smaller values.

So, I don't think this is a real issue. More, why would someone rely for iterating this collection by an object's ID?

#3 - 12/11/2020 08:26 AM - Marian Edu

Constantin Asofiei wrote:

> Some notes here:
>
> 1. in the SQL temp-table, the progress.lang.object field is represented as a character (with the string representation of its hash), and not an integer. AFAIK this is how OE works.

As far as I can see this is not how OE works, otherwise it will show the records as 'aaaa', 'ffff', 'xxxx' but it shows: 'xxxx', 'aaaa', 'ffff' which match only the order the objects were created :(

1. in FWD, the index will sort these values lexicographically (as the field is character)
2. we use random integer values to represent the 'handle' value for an OO instance or any other handle. In 4GL, is just a coincidence that the handle values are incrementing - IIRC, at some point we did some tests and once the overflow value was reached (consider that the HANDLE function accepts a string representation of an integer), OE started to re-use smaller values.

I expect this to happen indeed, most probably it won't reuse the ones still valid though.

> So, I don't think this is a real issue. More, why would someone rely for iterating this collection by an object's ID?

Well, I can't give you an answer to that one as I would never considered even define an index on a P.L.O field but it does look like a smart guy at PSC thought this could be a great idea :)

#4 - 12/11/2020 08:27 AM - Constantin Asofiei

Marian Edu wrote:

> Constantin Asofiei wrote:
>
> > Some notes here:
> >
> > 1. in the SQL temp-table, the progress.lang.object field is represented as a character (with the string representation of its hash), and not an integer. AFAIK this is how OE works.
>
> As far as I can see this is not how OE works, otherwise it will show the records as 'aaaa', 'ffff', 'xxxx' but it shows: 'xxxx', 'aaaa', 'ffff' which match only the order the objects were created :(

What exactly are you referring here by 'xxxxxx'? Only the numeric representation or the class name prefix from string(some-oo-var)?

#5 - 12/11/2020 08:30 AM - Marian Edu

Constantin Asofiei wrote:

> What exactly are you referring here by 'xxxxxx'? Only the numeric representation or the class name prefix from string(some-oo-var)?

We put three string objects in that table, 'xxxx', 'aaaa', 'ffff' are the values used in ctor for those so OE does not sort the records using ToString() otherwise it will sort them as 'aaaa', 'ffff', 'xxxx' but as the test shows it does not.

#6 - 12/11/2020 08:39 AM - Constantin Asofiei

Marian Edu wrote:

> Constantin Asofiei wrote:
>
> > What exactly are you referring here by 'xxxxxx'? Only the numeric representation or the class name prefix from string(some-oo-var)?
>
> We put three string objects in that table, 'xxxx', 'aaaa', 'ffff' are the values used in ctor for those so OE does not sort the records using ToString() otherwise it will sort them as 'aaaa', 'ffff', 'xxxx' but as the test shows it does not.

Why should OE sort them by the ToString()? The sort is done by their hash, which you can't 'see' if toString() is overridden (and OpenEdge.Core.String has an explicit ToString()).

In FWD, as this hashes are random, the sort will be different each time you run the program.

#7 - 12/11/2020 08:50 AM - Marian Edu

Constantin Asofiei wrote:

> Why should OE sort them by the ToString()? The sort is done by their hash, which you can't 'see' if toString() is overridden (and OpenEdge.Core.String has an explicit ToString()).

Now you lost me, this is what you've said before:
in the SQL temp-table, the progress.lang.object field is represented as a character (with the string representation of its hash), and not an integer.

AFAIK this is how OE works.

The hash you mention is still visible if you do int64(P.L.O), this will give you that resource id.

> In FWD, as this hashes are random, the sort will be different each time you run the program.

Yes, already seen that and that is giving different results compared with 4GL.

I'm not saying this should be fixed, simply that with a random resource id we can't sort the objects in collection - this is used when the toArray method is being called - so parts of our tests are falling. I'm expecting the iterator to also fail the tests because of the same indexing issue :(

#8 - 12/11/2020 08:54 AM - Constantin Asofiei

Marian Edu wrote:

> Now you lost me, this is what you've said before:
> *in the SQL temp-table, the progress.lang.object field is represented as a character (with the string representation of its hash), and not an integer.*
> ***AFAIK this is how OE works.***
>
> The hash you mention is still visible if you do int64(P.L.O), this will give you that resource id.

OK, I didn't now about int64(oo-instance) function. What I was referring is that the default Progress.Lang.Object.ToString() contains the hash, too, and I usually use that for my tests.

> I'm not saying this should be fixed, simply that with a random resource id we can't sort the objects in collection - this is used when the toArray method is being called - so parts of our tests are falling. I'm expecting the iterator to also fail the tests because of the same indexing issue :(

OK, then we should document this as an expected difference between OE and FWD; and just make sure the tests give the correct values, and do not rely on their order when using P.L.O values. Non-PLO values should work fine.

#9 - 01/27/2021 05:55 AM - Marian Edu

- Related to Feature #4384: Builtin OO Implementation added

#10 - 01/27/2021 06:02 AM - Marian Edu

Constantin Asofiei wrote:

> Marian Edu wrote:
>
> > I'm not saying this should be fixed, simply that with a random resource id we can't sort the objects in collection - this is used when the toArray method is being called - so parts of our tests are falling. I'm expecting the iterator to also fail the tests because of the same indexing issue :(
>
> OK, then we should document this as an expected difference between OE and FWD; and just make sure the tests give the correct values, and do not rely on their order when using P.L.O values. Non-PLO values should work fine.

Constantin, given the random nature of resource id in FWD it's impossible to write any tests for AbstractTTCollection (and others that extends it) with 'correct values' - if there are more than two objects in that collection then the order both in iterator and when using toArray can only match the 4GL order by pure chance and the test results are not idempotent either so this could make for a fug bug to debug (sic) sometime in the future :)

#11 - 01/27/2021 10:53 AM - Greg Shah

It seems this is a deviation that is not just a "quirk". It is really an issue, especially when you consider the collection implications.

Let's discuss potential solutions. Must our soution be compatible with the possible mechanisms to translate between int64, string, and object reference? Or can we implement our own indexing ordinal (incrementing based on order of creation and wrapping as needed) in the temp-table column?

#12 - 01/27/2021 01:18 PM - Constantin Asofiei

This would mean that these instances will need an 'always incrementing' ID, instead of our current random approach. And keep the random approach for other resources. This will complicate things, as currently the external programs emulated for these instances are kept in the same registry as other resources.

OTOH, an alternative would be to keep both the random and always-incrementing ID, and use this new ID only in certain cases, like:

1. toString
2. (de)serialization on a temp-table field as integer
3. int64(oo-instance) function
4. other cases?

Marian: is there a way to resolve an instance from its ID, an equivalent for handle function?

#13 - 02/18/2021 10:24 AM - Greg Shah

> is there a way to resolve an instance from its ID, an equivalent for handle function?

Marian: We need your thoughts on this question to move ahead with this task.

#14 - 02/18/2021 12:17 PM - Marian Edu

Greg Shah wrote:

> > is there a way to resolve an instance from its ID, an equivalent for handle function?
>
> Marian: We need your thoughts on this question to move ahead with this task.

The order in 4GL is like for an integer field, values are those of the `resource id` - int64(P.L.O). This is indeed incremented instead of random in 4GL but up to a point, when it reaches the max value for a long previous values not used anymore starts to be re-used so it more or less became sort of random - aka: instances created earlier will start to be ordered after newly created ones.

I would say a numeric index is better than one on a string field so I would stick to the same approach and maybe don't worry about the randomness of object's resource id, imho even allowing an index on a P.L.O is a bad design, the fact that they even used that when implementing the temp-table based collection only makes it worst :(

As far as I can tell, the is no ordering guaranteed when using this sort of collections so we can just take out any tests that check the order of elements in the collection as returned by the iterator.

This is not the biggest flow anyway, one can call remove on collection while looping through its elements using the iterator in 4G :)

#15 - 03/05/2021 05:18 AM - Constantin Asofiei

I have the changes to move the object ID to a long, incrementing value. This will be used by the toString and temp-table fields/indexes. Internal FWD representation still uses the random value, my changes just map this long resource ID to its internal resource ID (for the persistent program).

OTOH, the initial assumption that a temp-table handle field is mapped as a string is incorrect... adding an index on this field will sort the values as

numeric, not string.

So, Greg: should I leave it random for both handle and object fields, and just make the temp-table field a long?

#16 - 03/05/2021 08:42 AM - Greg Shah

> I have the changes to move the object ID to a long, incrementing value. This will be used by the toString and temp-table fields/indexes. Internal FWD representation still uses the random value, my changes just map this long resource ID to its internal resource ID (for the persistent program).

This seems reasonable. Do I understand correctly that the internal representation will never be seen by the converted 4GL code?

> should I leave it random for both handle and object fields, and just make the temp-table field a long?

We know that we need to do this "external ID representation" vs @internal ID representation" for objects. We need to decide if we need to also do that with handles.

Can the internal ID representation of a handle ever be seen by the converted 4GL code? If not, we can map internal random IDs of handles to external IDs. But at some point one must wonder if it makes sense.

Although it is much safer to use random IDs (security reasons), if all handle and object IDs are simple incrementing values, then it means that you can write code to access a handle or object just be trying different ID values starting at 1. And if this is valid in the 4GL then you know that someone somewhere is doing this ridiculous thing. We already support the HANDLE() function which does this. As soon as we map it to simple incrementing values then there is no longer any value in mapping to a random internal ID.

In regard to making the temp-table handle and temp-table P.L.O fields into long, I think we should do this.

#17 - 03/05/2021 09:43 AM - Constantin Asofiei

Greg Shah wrote:

> This seems reasonable. Do I understand correctly that the internal representation will never be seen by the converted 4GL code?

Yes.

> Can the internal ID representation of a handle ever be seen by the converted 4GL code? If not, we can map internal random IDs of handles to external IDs. But at some point one must wonder if it makes sense.

The major reason why the resource ID for the persistent procedures/handles/etc needs to be random is because you can resolve a resource via the handle(string) function. There is no such equivalent function for objects - only int64(object) which gets the resource ID.

> Although it is much safer to use random IDs (security reasons), if all handle and object IDs are simple incrementing values, then it means that you can write code to access a handle or object just be trying different ID values starting at 1. And if this is valid in the 4GL then you know that someone somewhere is doing this ridiculous thing. We already support the HANDLE() function which does this. As soon as we map it to simple incrementing values then there is no longer any value in mapping to a random internal ID.

Correct. I would keep the random resource IDs for non-object resources. I see no real reason to make these incrementing. The objects are safe to use incremental values because you there is no handle function equivalent for them.

> In regard to making the temp-table handle and temp-table P.L.O fields into long, I think we should do this.

OK.

#18 - 03/05/2021 09:59 AM - Greg Shah

> The major reason why the resource ID for the persistent procedures/handles/etc needs to be random is because you can resolve a resource via the handle(string) function.

I understand it is more secure but does it break any existing 4GL behavior? I don't recall the 4GL using random IDs for handles. If they do use random IDs, then there is no issue here. But if they increment, then it we might see incompatibilities in handle() and also in sorting on a handle column for temp-tables.

#19 - 03/05/2021 10:33 AM - Constantin Asofiei

Greg Shah wrote:

> I understand it is more secure but does it break any existing 4GL behavior?

Only in cases of weird code, like some application knows that building 10 resources will have their IDs incrementing by 1, and use handle function instead of the next-sibling or something else to walk them. 4GL allows this, but is very bad practice.

> I don't recall the 4GL using random IDs for handles. If they do use random IDs, then there is no issue here.

They always increment for both object and handle.

> But if they increment, then it we might see incompatibilities in handle() and also in sorting on a handle column for temp-tables.

Yes, with random IDs we can think that FWD is 'incompatible'. An application may be coded to have checks 'at this point in execution the handle ID must be X'. And maybe lots of more weird ways where some application code would depend on the incremental handle ID values, including sorting their IDs.

A reason in FWD why random IDs are better is because we expose via the embedded driver direct access to handle IDs (i.e. deleting a persistent program, running a program persistent, and maybe more). Having random IDs is another level of protection; FWD already allows deletion only for procedure handles which are obtained by the remote (i.e. embedded) application. But there is no protection for INVOKE and maybe other APIs. At the least, we should add more security here and limit handle IDs used by these APIs to 'known IDs' exposed to the embedded application.

#20 - 03/05/2021 11:07 AM - Marian Edu

I find this illusion of security kinda funny, random or incremental is exactly the same in a single threaded environment like Progress, its not like being able to access other process memory or anything.

Beside, anyone can walk through session first object/handle and use next-sibling to get access to everything... really no need to try to 'guess' a valid handle/object, the level of 'protection' added by FWD is something I fail to see here :)

Last but not least the way the random id is generated by keep on trying random till the generated id is not already used is potentially more costly that keeping it incremental, even when the higher limit is reached and it needs to look for ones that can be reused (ymmv).

> anyone can walk through session first object/handle and use next-sibling to get access to everything

Yes, this is a fair point.

> A reason in FWD why random IDs are better is because we expose via the embedded driver direct access to handle IDs (i.e. deleting a persistent program, running a program persistent, and maybe more).

This is a real issue. The embedded mode web client includes a kind of javascript appserver feature. This means that arbitrary javascript code can "up-call" from the browser into FWD to execute 4GL code. As with appserver, such usage requires persistent procedures and handles for anything that is a function or internal procedure. Since a handle is defined as an ID, it is easily passed down to javascript and back up from javascript, using a number. This makes it easy for javascript code to up-call and attempt to invoke functions/internal procedures on some handle, possibly via a brute force approach. A random 64-bit ID will greatly slow this down where a simple incrementing value makes it trivial.

Considering that the javascript code can be anything and might very well have lots of 3rd party code and unknown cross-site scripting problems, I think this is a real issue. Of course, it is opened up by the web capabilities of FWD and is not present in the 4GL itself.

> But there is no protection for INVOKE and maybe other APIs. At the least, we should add more security here and limit handle IDs used by these APIs to 'known IDs' exposed to the embedded application.

Yes, I think this is the right answer. I think it is best to shift to the "simple increment" approach for handles as well as objects, and secure this API so that a brute force attack is not possible.

We will have to keep thinking about any other exposure that exists. For example, I'm wondering whether REST, SOAP and appserver access also share this flaw. I think they might in FWD. And I suspect that the REST/SOAP access may have this flaw in the 4GL too. The direct appserver usage (from 4GL to 4GL) may be safe if the handle proxying is secured. I don't know enough about that to comment. But the other question is whether usage from an Open Client is safe in 4GL.

Marian: Is remote REST/SOAP/appserver access safe in OE? Or is it possible to use an arbitrary handle in a call/invocation as either a parameter or a reference upon which a function/internal procedure is executed?

#22 - 03/05/2021 03:13 PM - Constantin Asofiei

I've switched both handle and object fields to Long and incremental value. There are two more issues to check, unknown values in fields and this test:

```
def var h as handle.
def var plo as progress.lang.object.

plo = new progress.lang.object().
def var ch as char.
ch = string(plo).
ch = substring(ch, index(ch, "_") + 1).
message ch.

h = handle(ch) no-error.
message error-status:error error-status:get-message(1).
// yes Cannot pass an object reference to the WIDGET-HANDLE function. (13443)
```

As objects and handles share the same 'ID table' in 4GL, one can use handle function to try to access an object instance. In 4GL, the commented error appears in such cases.

I haven't touched com-handle field, I assume this is also mapped as Long in 4GL, but its weird to test in FWD. And in 4GL the ID is distinct from handle/object ID sequence (and kind of random).

#23 - 03/05/2021 04:05 PM - Greg Shah

> I haven't touched com-handle field, I assume this is also mapped as Long in 4GL, but its weird to test in FWD. And in 4GL the ID is distinct from handle/object ID sequence (and kind of random).

I think it is very possible that this is either:

- A DWORD data type (32-bit integer) WIN32 "handle"; OR
- A pointer to a memory location for the COM object.

#24 - 03/06/2021 07:29 AM - Constantin Asofiei

- Status changed from New to WIP

- % Done changed from 0 to 100

The changes are in 3821c rev 12088.

Ovidiu: please review.

#25 - 03/06/2021 10:41 AM - Greg Shah

Code Review Task Branch 3821c Revision 12088

The changes are OK for me.

Ovidiu: If you are OK with the persistence changes, then I think this task can be put into test mode.

Constantin: Is there anything else to do here?

#26 - 03/06/2021 01:29 PM - Constantin Asofiei

Greg Shah wrote:

> Constantin: Is there anything else to do here?

No.

#27 - 03/07/2021 09:13 AM - Greg Shah

- Status changed from WIP to Test

#28 - 03/08/2021 05:35 AM - Marian Edu

Greg Shah wrote:

> This is a real issue. The embedded mode web client includes a kind of javascript appserver feature. This means that arbitrary javascript code can "up-call" from the browser into FWD to execute 4GL code. As with appserver, such usage requires persistent procedures and handles for anything that is a function or internal procedure. Since a handle is defined as an ID, it is easily passed down to javascript and back up from javascript, using a number. This makes it easy for javascript code to up-call and attempt to invoke functions/internal procedures on some handle, possibly via a brute force approach. A random 64-bit ID will greatly slow this down where a simple incrementing value makes it trivial.

Don't know all the details here nor how exactly it is implemented in 4GL but a simple check on wether or not the handle id sent by the client correspond to a procedure that was actually instantiated by the same client should be enough. The 4GL client gives you a handle, open client a kind of procedure handle object so we don't send directly the ID but somewhere downstream it's clearly the ID that gets passed to the appsrv at some point just that the client can't specify a random value, it had to be instantiated by the client first.

> Marian: Is remote REST/SOAP/appserver access safe in OE? Or is it possible to use an arbitrary handle in a call/invocation as either a parameter or a reference upon which a function/internal procedure is executed?

In REST you can't start random persistent procedure nor call-in a specific procedure by using the ID - unless there is some application call that allows you to do this, say by using dynamic call. Oddly enough in SOAP one can instantiate a persistent procedure and then use it's endpoint to call-in into its internal entries but again you don't get the internal procedure ID but an UUID that is probably mapped to the actual procedure handle inside the appsrv, trying to guess a valid UUID is kinda hard in general and this has nothing to do to how the internal ID is assigned on the 4GL side.

As with any RPC the appsrv is a secure as the application it hosts, apart from DOS or buffer overflow attacks... so, imho, if this is an issue in FWD is not because the ID's are incremental or random but with the fact that the actual (internal) ID can be used by the client code.

#29 - 03/08/2021 06:52 AM - Ovidiu Maxiniuc

Greg Shah wrote:

> Code Review Task Branch 3821c Revision 12088
> The changes are OK for me.
> Ovidiu: If you are OK with the persistence changes, then I think this task can be put into test mode.

I do not see any issue with the changes from String to Long type of resource id. In fact it might increase performance as long/int8 operations are faster than String/varchar.

#30 - 03/08/2021 07:12 AM - Greg Shah

- Related to Support #4086: review the RemoteObject exports in StandardServer and other common runtime classes for security implications added

#31 - 03/08/2021 07:21 AM - Greg Shah

> if this is an issue in FWD is not because the ID's are incremental or random but with the fact that the actual (internal) ID can be used by the client code.

It certainly would need some "cooperation" from the application. My worry is in trying to minimize the exposure of the application for such things. That is why we implemented the random IDs in the first place. It just makes it harder for exploitations. Using UUIDs is basically the same idea. As you note, this is very hard to brute force. We always make choices to have improved security wherever doing so does not compromise compatibility.

Constantin: Please put protection into the embedded mode driver to ensure all handle usage is only with handles which have been explicitly returned to the client.

#32 - 03/09/2021 01:41 AM - Marian Edu

Greg Shah wrote:

> is there a way to resolve an instance from its ID, an equivalent for handle function?

Marian: We need your thoughts on this question to move ahead with this task.

I've missed that one... no, there is no built-in function to get an object instance from its ID. But, since all object instances are available in session object chain it's trivial to write a function to pass through the stack and see if there is a match for the given ID.