

Re: Ada DB bindings and APQ

Source: <http://coding.derkeiler.com/Archive/Ada/comp.lang.ada/2004-12/0306.html>

From: Warren W. Gay VE3WWG (ve3wwg_at_NoSpam.cogeco.ca)

Date: 12/16/04

Date: Wed, 15 Dec 2004 21:53:11 -0500

Dmitry A. Kazakov wrote:

> On Tue, 14 Dec 2004 23:05:17 -0500, Warren W. Gay VE3WWG wrote:

>> Dmitry A. Kazakov wrote:

>>> Connection could be just a handle to some dynamic-scope connection object

>>> with reference counting. This will solve both the problem of above and

>>> still allow cloning connections.

>>

>> I didn't like the use of handles for database objects. I purposely

>> stuck to using controlled objects, so that they can go out of

>> scope gracefully, be finalized (perhaps committed or rolled back)

>> and then destructed. Handles, like open Ada File_Types, never

>> get that maintenance.

>

> Let's make them derived from Ada.Finalization.Controlled.

What's the point of that? Why not use the actual object? It seems like unnecessary indirection and complexity – what problem are you trying to solve with handles?

> if Root_Connection_Type were limited controlled, derived from

> Object.Entity, then I could create handles just by instantiating

> Object.Handle. Root_Query_Type can hold a handle to its connection. This

> will ensure that the connection will not go away until at least one query

> is alive.

But why? The objects themselves solve the problem already.

>>> BTW, one could get rid of query type altogether if there would be two

>>> different connection-cloning. One that returns a handle to a new

>>> "connection" (physically just new query) for the same underlying connection

>>> object, another that makes a new connection.

>>>

>>> I do not see much sense in all that levels of types exposed to the user. In

>>> ODBC it is even one more: environment > connection > statement. It is

>>> rather implementation details to me. (Everything is a file (:–))

>>

>> ODBC is not the only one. Sybase implements an object that

>> also fills this role. I believe that you want to keep the

comp.lang.ada: Re: Ada DB bindings and APQ

>>objects well separated by function, and I'll admit that I
>>compromised principle by combining environment & connection.
>>But I think OO design principles are such that you don't
>>want to roll everything into one massive object.
>
> Right, if the functionality differs. But so far there are little or no
> things which can be made with a connection. That is different to ODBC. Also
> ODBC has prepared statements. And that all just does ODBC too complex to
> use.

Here is food for thought ;-) — if Sybase and other databases implement their interface with 2–3 different classes of objects, why is the APQ design so wrong to do something similar? I really can't see why all the fuss over one object vs two from the user's perspective. Not everything is a hammer (read File_Type), and there is room for multi-object designs in modelling interfaces.

>>Now having said that, if you really insist on a one-object
>>approach, you might want to test drive the idea by
>>rolling your own "interface package" if you will, that
>>does just that using the APQ provided types. But I believe
>>that you'll quickly arrive at the unhappy conclusion that
>>it just gets very messy, and results in a more complicated
>>to understand object.
>
> I already did it for ODBC (it is presently under test.) Of course it is
> rather a specialized package to provide object persistence. But basically
> it has only one type: Persistent_Storage_Handle, which is a handle to
> Persistent_Storage_Object. I am considering to do the same with APQ. When
> do you plan to ship v 2.2, BTW?

The "code" has been done for months. The *NIX install part is mostly done, though I am not really happy with it.

The Windows install side is a nightmare. Here's why:

- gnat 3.14p/3.15p is one choice
- CYGWIN gcc with Ada support is another
- PostgreSQL can now be included for CYGWIN
- or the C client library libpq can be compiled independant of CYGWIN

So at this point, what I need to do is to get it up on SourceForge and let someone who has time and energy work this all out. I am just too busy to dedicate much time on this right now (I have become enchanted by another very fun, new Ada-involved project at the moment).

>>I prefer to have 2 simple objects
>>instead of one complicated to understand object. The

```
>>>code is much easier to read and understand this way.
>>>It also separates the finalization of the query from the
>>>finalization of the connection.
>
> I had in mind:
>
> declare
> DB : DB_Handle := Connect (Engine_MySQL, "localhost", "testdb",...);
> begin
> Prepare (DB, "SELECT ...");
> Execute (DB);
> Fetch (DB);
> ...
> end;
```

I would suggest you test drive APQ for a while. You might find that it works rather well the way it is. I am not saying that it cannot be improved (quite the contrary), but I think there are more important things to improve upon without munging all objects into one large massive one.

```
>>>Handle --> Query_Object --> Connection_Object
>>>
>>>1. Handle copy (+1 reference count of Query_Object)
>>>2. Light-weight copy: creates new query (+1 reference count of connection)
>>>3. Heavy-weight copy: creates a new query and a new connection
>>
>>When designing an API, I always try to look at the way
>>it is going to be used and why it must exist. I shy away
>>from unnecessary API, unless it is frequently needed (to
>>save or simplify code).
>>
>>So coming back to this, I have to ask, what is the
>>problem that this is intended to solve?
>
> You will get rid of parallel hierarchy of types:
>
> Root_Connection_Type <- MySQL.Connection_Type
> ||
> Root_Query_Type <- MySQL.Query_Type
>
> Ada is unable to provide any consistency here. You can mistakenly use
> PostgreSQL.Query_Type with MySQL.Connection_Type.
```

You can compile it, but you'll get an exception when you try to use Execute with a mismatched paramter. Don't forget that the connection gets "converted" to the MySQL.Connection_Type before it can be used, and of course this will fail if it is a PostgreSQL.Connection_Type.

So "Ada does provide consistency here", but not at compile time. There are a number of other things that are only checked at runtime when using databases, and I see this as a nit-pick. Better to use that sledgehammer on bigger flies!

I can tell you from experience, that I have written a number of test programs that copy from PostgreSQL database tables to MySQL, and then to Sybase, all in the same programs, and I never mixed them up.

> *You can finalize*
> *connection before a query that depends on it etc.*

That is a risk, but its generally pretty obvious at runtime when it happens! But if you structure your code normally, the Connection_Type is declared at the outermost scope and again, this tends not to be a problem.

> *But somebody should create that tables! It is quite normal that application*
> *initializes tables upon first start. Consider an e-mail program. It should*
> *store it address book into a DB of user choice. How to implement it in a DB*
> *independent way?*

I won't disagree with it for applications like this. "Normal" is a relative thing I guess, but normally in my book, installs usually run some SQL scripts to create and initialize databases for applications. An email application need not be different, though it can be, as you've suggested. The reason being is that these things are typically done once, and then never performed again (install scripts can be discarded once used).

>>*Doing table names in a portable way is problematic. Some databases*
>>*are not case sensitive, but they are case sensitive when they come*
>>*to TABLE names. Others can be configured to force lowercase table*
>>*names (MySQL IIRC), still others are always case sensitive (Sybase).*
>>
>>*This is why "case policy" was introduced into APQ 2.2. This is also*
>>*why it is necessary to distinguish between certain SQL building*
>>*APIs and other others like Append_Quoted.*
>>
>>*Believe me, if there was a unified way to do it all, I would have*
>>*welcomed it. But the reality is a bit more complicated than that ;-)*
>
> *The method that places a value into a query should simply dispatch on the*
> *query type. The implementation is then DB specific and will encode the*
> *value as appropriate for the DB.*

Heh heh, it all sounds very easy ;-)

comp.lang.ada: Re: Ada DB bindings and APQ

--

Warren W. Gay VE3WWG

<http://home.cogeco.ca/~ve3wwg>