ABL Performance Tips and Tricks

OpenEdge Developer’s Corner

Peter Judge
Principal Software Engineer

Wherefore art thou, performance?

“"I know it when I see it”
US Supreme Court Justice Potter Stewart

- As important as stability, features, quality
- Affected by
  - Changing user counts, data, activity, traffic
  - Application design & programming techniques
- Different functionality = different needs
- What determines good performance?
  - Using the available resources efficiently
Better performance improves user satisfaction

"It's not what the software does. It's what the user does."

@hugh

What can I do about it?

- Software upgrades vs. hardware upgrades
  - Hardware faster, cheaper over time
    Does faster hardware make you lazy?
- You can only change what you control
  - System configuration
- Design & code for performance
  - Don’t go overboard
  - Don’t make things worse
- Test & measure
Agenda

- Coding for performance
- Testing & measurement
- Deployment & runtime considerations

Coding design

- Good design is necessary
  - Can have performance cost
  - Classes generally faster than procedures

- Consider caching
- “Warehouse” into ProDataSets for reporting
  - Don’t try to do it all with one FOR EACH
Green coding

- The fastest code is no code at all
- Clean up after yourself
  - Manage widgets using WIDGET-POOL
  - Store references for deletion
- Define statically, access dynamically
  - Reuse what you can
- Only get the data you need
  - Filtered data for UI

Data across the network

- Network typically major bottleneck
  - Makes other performance problems worse
- Number of roundtrips
  - Making AppServer connection has cost
- Data volume per roundtrip

- Network topography has impact … … but usually out of our control
Reduce data volume

- Put your temp-tables on a diet
  - Use `MIN-SCHEMA-MARSHAL` or `NO-SCHEMA-MARSHAL`
    - Stay away from `RCODE-INFO`
  - Use transport temp-tables
  - Use enumerations to compress data

- Compression
  - `-mc` (message compression)

No deep copies

```plaintext
define temp-table ttData ...
run populateData (output table ttData).
run visualiseData (input table ttData).
run getChanges (output table ttData).
run saveChanges (input table ttData).

define temp-table ttData ...
h = buffer ttData:handle.
run populateData (output h).
run visualiseData (input h).
run getChanges (output h).
run saveChanges (input h).
```
No deep copies (2)

```plaintext
define temp-table ttData ...
run populateData (output table ttData).
run visualiseData (input table ttData).
run getChanges (output table ttData).
run saveChanges (input table ttData).
```

```plaintext
define temp-table ttData ...
run populateData (output table ttData by-reference).
run visualiseData (input table ttData by-reference).
run getChanges (output table ttData by-reference).
run saveChanges (input table ttData by-reference).
```

Understand how indexes work

- Understand index selection rules
  - Take care with **OR** in **WHERE** clause
  - Also **BY** clause

- Confirm actual indexes used
  - **COMPILE ... XREF**
  - **INDEX-INFORMATION** attribute

- Know your indexes
Sequences

```sql
find counters where
    counters.seq_name = "cust_num"
    exclusive-lock no-wait no-error.
/* check lock & wait status; act accordingly */
create customer.
assign customer.cust-num = counters.value
    counters.value = counters.value + 1
...
```

```sql
create customer.
assign customer.cust-num = next-value(seq_cust_num)
...
```

Unique values (when order doesn’t matter)

```sql
create customer.
assign customer.id = string(next-value(seq_table_id),
    "9999999")
...
```

```sql
create customer.
assign customer.id = guid()
...
```
Arrays faster than delimited lists

```c
// Delimiter string
char cDelim = ","; /* or CHR(3) or | or ... */

// List string
char cList = "item1" + cDelim + "item2" + cDelim
+ ... + "itemN";

// Do loop to extract entries
for (int i = 0; i < num-entries(cList, cDelim); i++)
    cEntry = entry(i, cList, cDelim);

// Array initialization
char cArray[1] = "item1";
char cArray[2] = "item2";
char cArray[n] = "itemN";

// Do loop to iterate over array
for (int i = 0; i < extent(cArray); i++)
    /* do stuff with */ cArray[i];
```

Group your ASSIGNments

```c
// String variables
char cVar = "abc";
int iVar = 123;
char dtVar = now;

// Tuple data
char ttData.indexField = "Pi";
char ttData.dataField = getPiVal();

// Assignments
assign
    cVar = "abc"
    iVar = 123
    dtVar = now
    ttData.indexField = "Pi"
    ttData.dataField = getPiVal();
```
Blocks

```
repeat i = 1 to 10:
  end.

if condition1 then
do:
cVar = cValue.
cTime = now.
end.
```

- **function**, **procedure**, **method** all blocks
  - Inline code may be faster
    - Use includes for code re-use

Error handling

```
do on error undo, return error 'oops':
o = new App.Module.Class().
o:Method1().
o:Method2().
end.
```

```
do on error undo, return error 'oops':
o = new App.Module.Class().
o:Method1().
o:Method2().
/* Some Other Stuff Happens */
catch e as Progress.Lang.Error:
  undo, throw new Progress.Lang.Error ('oops').
end catch.
end.
```
Error handling (2)

```pascal
run method1 in hdl () no-error.
if error-status: error then return error return-value.
run method2 in hdl () no-error.
if error-status: error then return error return-value
```

```pascal
run method1 in hdl().
run method2 in hdl().
catch e as Progress.Lang.Error:
    undo, throw e.
    /* alternatively, throw a new, different error */
    undo, throw new Progress.Lang.Error ('oops').
end catch.
```

Class properties

```pascal
class App.Module.ConferenceVenue:
def public property CityName as char no-undo
get. set.
def public property CityLocation as char no-undo
get ():
    /* gets lat/long of city as string
    from WebService */
    end get.
    set.
end class.
```

```pascal
o = new App.Module.Object().
o:CityName = "Paris".
```

```
Effectively a function invocation
```

```
Effectively an ASSIGN
```
Combining techniques

```plaintext
j = extent(cArray).
do i = 1 to j:
    cArray[i] = cArray2[i].
end.
```

Loop hoisting

```plaintext
j = extent(cArray).
n = 50. /* depends on data distribution */
do i = 1 to j by n:
    assign cArray[i] = cArray2[i]
    cArray[i+1] = cArray2[i+1]
    ...
    cArray[i+n] = cArray2[i+n].
end.
```

```plaintext
if condition1 then
    do i = 1 to n:
        if condition1 then
            do:
                /* stuff */
            end.
        else
            if condition2 then
                do:
                    /* more stuff */
                end.
            else
                do:
                    /* other stuff */
                end.
            end.
        end.
    end.
else
    if condition2 then
        do i = 1 to n:
            /* more stuff */
        end.
    else
        do i = 1 to n:
            /* other stuff */
        end.
end.
```
Testing for performance

- Measure performance
  - The only way to know for sure
  - Measure more than once
  - Measure against predefined goals
- Use regression test suite
  - Use constant, realistic environments
  - Compare against baselines
  - Automate as far as possible
- Ongoing, iterative process
  - Engineers’ technical awareness
  - User feedback
Finding performance problems

- Start with code that works
- Don’t assume you know what the problem is
- Change one thing at a time
- There’s no silver bullet
- You will make things worse (probably)
  - *Take those changes out*
- The goal posts are always moving
  - Stop when it’s good enough

Agenda

- Coding for performance
- Testing & measurement
- Deployment & runtime considerations
R-code & PROPATH

- Always use r-code in production
  - Use –q (Quick Request) parameter
- Reduce r-code size
  - Remove function prototypes
  - Limit use of GLOBAL variables
  - Compile with MIN-SIZE option
- Keep PROPATH as short as possible
  - Order entries by frequency of use
- Use procedure libraries
  - Memory-mapped when on same machine

In Summary

- Good performance needs upfront work
  - Design
  - Coding
- Measure and test
Q & A

Optimization matters *only* when it matters. When it matters, it matters a lot, but until you know that it matters, don't waste a lot of time doing it. Even if you know it matters, you need to know *where* it matters. Without performance data, you won't know what to optimize, and you'll probably optimize the wrong thing.

Joseph M. Newcomer
Optimization: Your Worst Enemy

Thank You
Optimize network roundtrips

```
loginWindow.w
run getLoginLanguages() <-> getLoginLanguages()
run getLoginCompanies() <-> getLoginCompanies()
run buildUI().
```

Optimize network roundtrips

```
loginWindow.w
run getUiData() <-> getUiData()
run getLoginLanguages()
run getLoginCompanies()
run buildUI().
```
Always use NO-UNDO

```plaintext
define variable cVar for char no-undo.
def public property AccessTime as datetime no-undo get.
set.
define input parameter piParam as int no-undo.
define temp-table ttData no-undo
  field indexField as character
  field dataField as decimal.
```

Evaluate functions as rarely as possible

```plaintext
do i = 1 to udfGetNumIter():
end.
do i = 1 to num-entries(cLst):
end.
do i = 1 to 10:
j = udfGetMaxValue().
  /* do something with j */
end.
j = udfGetNumIter().
do i = 1 to j: end.
j = num-entries(cLst).
do i = 1 to j: end.
j = udfGetMaxValue().
do i = 1 to 10:
  /* do something with j */
end.
```
Know your indexes

- Only maintain the indexes you need
  - Add more indexes
  - Get rid of unused indexes
- Multi-component indexes are good
- Unique indexes faster than non-unique
- **ROWID** access is very fast

Efficient data access

- Join from smallest bracket to largest
  - Not always parent to child
  - e.g. Date ranges

```sql
for each Customer where
    credit-limit >= 100,
    each Order of Customer where
        order-date = today:
        /* create report */
```

```sql
for each Order where
    order-date = today,
    first Customer of Order where
        credit-limit >= 100:
        /* create report */
```
Dealing with LOGICAL and status values

for each Employee where
gender = “Male” and /* or Female */
emp-status = “Perm” and /* or Contract */
currently-employed = true /* or false */

cStatus = “Male” + “|” + “Perm” + “|” + “CurEmp”.
for each Employee where
status contains cStatus:

for each Status where
status = “Perm” or
status = “Male” or
status = “CurEmp”,
each EmpStatus of Status,
each Employee of EmpStatus:

Avoid conditional WHERE clause processing

for each Customer where
(if iCustNum > 0 then cust-num = iCustNum
   else true).

define query qryCust for Customer.
if iCustNum > 0 then
   open query qryCust
   for each Customer where cust-num = iCustNum.
else
   open query qryCust for each Customer.
CASE faster than nested IF

```plaintext
If cAlpha = "A" then ...
  else if cAlpha = "B" then ...
    else if cAlpha = "C" then ...
    else ...

case cAlpha:
  when "A" then ...
  when "B" then ...
  when "C" then ...
  otherwise ...
end case.
```

For More Information, go to...

- **PSDN**
  - Profiler
  - LogRead 1.0 Tool Overview (English & Spanish)
  - Log Read Utility

- **Documentation:**
  - OpenEdge® Deployment: Managing ABL Applications
Relevant Exchange Sessions

- DEV-15: AppServer™ Mode Case Studies
- OPS-23: OpenEdge Performance Basics

Tools for measuring performance

- **PROFILER** system handle
  - Line-by-line execution timings
  - Tool available on PSDN
- **etime**
- Logging infrastructure
  - Useful in production environments
  - LogRead utility on PSDN
- **Wireshark** ([http://www.wireshark.org/](http://www.wireshark.org/))
  - Network analyzer
Combining techniques

```plaintext
j = extent(cArray).
do i = 1 to j:
    cArray[i] = cArray2[i].
end.
```

Loop hoisting

```plaintext
j = extent(cArray).
n = 50.    /* depends on data distribution */
do i = 1 to j by n:
    assign cArray[i] = cArray2[i]
    cArray[i+1] = cArray2[i+1]
    ...  
    cArray[i+n] = cArray2[i+n].
end.
```

```plaintext
if condition1 then
    do i = 1 to n:
        if condition1 then
            do:
                /* stuff */
            end.
        else
            if condition2 then
                do:
                    /* more stuff */
                end.
            else
                do:
                    /* other stuff */
                end.
            end.
        end.
    end.
else
    if condition2 then
        do i = 1 to n:
            /* more stuff */
        end.
    else
        do i = 1 to n:
            /* other stuff */
        end.
end.
```