OpenEdge Database Performance Tuning
Tom Bascom, White Star Software

Abstract: Users are complaining that your database is slow? Come to this session to find out what you can do about that! What are the critical performance tuning parameters and options? How do you know which ones to tweak and what values to use? Join this session to find out the answers to all these questions and more!
OpenEdge Database Performance Tuning

aka: How To Make It Go Fast!

Tom Bascom, White Star Software
tom@wss.com
A Few Words about the Speaker

• Tom Bascom: Progress user & roaming DBA since 1987
• Partner: White Star Software & DBAppraise, LLC
  – Expert consulting services related to all aspects of Progress and OpenEdge.
  – Remote database management service for OpenEdge.
  – Author of: protop
  – Simplifying the job of managing and monitoring the world’s best business applications.
  – tom@wss.com
Your kilometerage will vary.

These tips are in no special order. But sometimes order does matter.

In an ideal world you will change one thing at a time, test and remove changes that do not have the desired effect.

The real world isn’t very ideal.
The List!
The performance enhancement possible with a given improvement is limited by the fraction of the execution time that the improved feature is used.

-- Amdahl’s Law

$$S = \frac{T_s}{T} = \frac{1}{f_p/n + f_s}$$
#30 In other words:

- Trying to improve small things that nobody notices probably isn’t the road to fame and fortune.

- Big queries that return lots of data and which are frequently used by lots of users will be much more noticeable.
#29 Stay Current

• Up to date releases of Progress, your OS and your application are essential components of a well tuned system.

• You cannot take advantage of the best techniques, improved algorithms or new hardware without staying up to date.

• Failure to stay up to date may mean poor performance, increased costs and general uncompetitiveness.
#29 Stay Current

- TRX throughput in v8 -> v9 (CCLP)
- Major 4GL execution speed improvements in 9.1E and OE10
- 64 bit platform support and large memory.
- OE 10 – Type 2 Storage Areas
- Significant improvements in DB read performance
- 10.2b06 and oe11 added –lruskips and –prefetch*
- 11.4 adds table partitioning
- 11.7 brings CDC …
#28 Set *rangesize

• The default value of 50 is useless.

• Without the full set of tables enabled for monitoring many diagnostic techniques cannot be used.

• You need to restart the db to change these so do it proactively:
  – Not when you already have a problem.
  – Round up a bit to avoid needing to adjust with every schema change.
  – Keep an eye on your coverage! Don’t get out of sync.
#27 Type 2 Storage Areas

- Type 2 storage areas are the foundation for all advanced features of the OpenEdge database.
- Data blocks in Type 2 areas contain data from just one table!
- Type 2 areas have “cluster sizes” of 8, 64 or 512.

```
  d "Data":20,128;512 /db/dbname_20.d1
  d "Indexes":21,1;8 /db/dbname_21.d1
  d "LOBs":22,64;512 /db/dbname_22.d1
```

- Cluster sizes of 0 or 1 are Type 1 areas 😞
#27 Type 2 Storage Areas

- **Always** use type 2 areas...
- ... for areas that contain data, indexes or LOBS.
- The schema area is a type 1 area 😞
#27 Type 2 Storage Areas

- **Always** use type 2 areas...
- ... for areas that contain data, indexes or LOBS.
- The schema area is a type 1 area 😞

- **Thus NO data should ever be in the schema area!**
#27 Type 2 Storage Areas

- **Always** use type 2 areas...
- ... for areas that contain data, indexes or LOBS.
- The schema area is a type 1 area 😞

- If you think that you have a legitimate exception I expect to see a detailed talk about it next year.
#26 Rapid Readers

- Similar to a runaway – consumes a whole CPU
- But is actually doing db “logical” IO
- Usually caused by:
  - Table scans
  - Poor index selection.
  - Unmonitored batch processes and app-servers.
  - Really bad algorithm choices.
## #26 Rapid Readers

### Direct Auto Sampling JSON 15255 4576 0.227
```
Hit% 99.06 Commits: 0 New RM: 0 Oldest TRX: 00:00:00 Connections: 10
Log Reads: 194,472 Undos: 0 From RM: 0 Curr BIClstr: 0 Brokers: 1
OS Reads: 1,835 Lock Tbl HMM: 48 From Free: 0 Oldest BIClstr: 0 4gl Servers: 0
Rec Reads: 95,269 Curr # Locks: 0 Examined: 0 Checkpoints: 1,139 SQL Servers: 0
LogRead/RecRd: 2.04 Modified Bufs: -16 Front2Bk: 0 Curr AI Extent: Disabled 4gl Clients: 6
Log Writes: 0 IO Response: 3.13 Remove LA: 0 Curr AI Seg#: 0 SQL Clients: 0
OS Writes: 0 BogOMIFS: 0.00
Rec Creates: 0 BogOMIFS%: 0.00
Rec Deletes: 0
Rec Updates: 0
Rec Locks: 10 BIN/AIW Writes% 0 0 AFW Writes: 0 AFW%: 2
Rec Waits: 0 Writes to Log: 0 0 Bufs Scanned: 0 Local: 2
Idx Blk Spl: 0 BIN/AIW Writes: 0 0 AFW Scan Wtts: 0 Remote: 0
Recsc Waits: 0 Partial Buf Wr: 0 0 AFW Q Wtts: 0 Batch: 4
Latch Waits: 2 Busy Buf Waits: 0 0 Chkpt Q Wtts: 0 TRX: 0
Empty Buf Wtts: 0 0 Flushed Bufs: 0 Blocked: 0
```

### Tables Activity
```
<table>
<thead>
<tr>
<th>Tbl#</th>
<th>Area#</th>
<th>Table Name</th>
<th>#Records</th>
<th>Turns</th>
<th>Create</th>
<th>Read v</th>
<th>Update</th>
<th>Delete</th>
<th>OS Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 8</td>
<td>22</td>
<td>activity</td>
<td></td>
<td>0.00</td>
<td>0</td>
<td>57,351</td>
<td>0</td>
<td>0</td>
<td>1,731</td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>site</td>
<td></td>
<td>0.00</td>
<td>0</td>
<td>37,874</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

### User IO Activity
```
<table>
<thead>
<tr>
<th>Usr#</th>
<th>Name</th>
<th>PID</th>
<th>Flags</th>
<th>Blk Ac</th>
<th>OS Rd</th>
<th>OS Wr</th>
<th>Hit%</th>
<th>Rec Lck</th>
<th>Rec Wts</th>
<th>Line#</th>
<th>Program Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt; 16</td>
<td>tom</td>
<td>14505</td>
<td>S4B</td>
<td>194216</td>
<td>1826</td>
<td>0</td>
<td>99.06%</td>
<td>0</td>
<td>0</td>
<td>34</td>
<td>.test/churn.p</td>
</tr>
<tr>
<td>12</td>
<td>tom</td>
<td>22668</td>
<td>S4</td>
<td>62</td>
<td>0</td>
<td>0</td>
<td>100.00%</td>
<td>6</td>
<td>0</td>
<td>630</td>
<td>lib/vstlib.p</td>
</tr>
</tbody>
</table>
```
#25  -lruskips 100

- It is simple and effective.
- Eliminates a lot of pointless internal housekeeping.
- Big benefit for busy systems!
- No negative impact on quiet systems.
- Makes the impact of bad code slightly less awful.
- Go ahead and set –lruskips2 100 while you’re at it.
#24 Transactions

- Distinguish between a “business transaction” and a “database transaction”.
- Do not try to abuse a database transaction to enforce a business rule:
  - Huge Lock Tables are a sign of this.
  - You may need to create “reversing (business) transactions”.
  - Or restartable transactions.
- For large database operations “chunk” your transactions.
define variable i as integer no-undo.

outer: do for customer transaction while true:

inner: do while true:
    i = i + 1.
    find next customer exclusive-lock no-error.
    if not available customer then leave outer.
    discount = 0. /* the "work" part of things... */
    if i modulo 100 = 0 then next outer.
end.

end. /* outer */
Define Buffer Table for Table.

- Very useful in internal procedures.
- It prevents the accidental “borrowing” of a buffer.
- Limits the damage from unintentional free references.

procedure x:
define buffer customer for customer. /* ← comment me out! */
    find first customer no-lock.
    display cust-name.
end.
find last customer no-lock.
display cust-name.
run x.
display cust-name.
#21 Parallelize

- Many legacy processes were designed when the system was much smaller or when multiple CPUs were unusual.
- Step outside of the single-threaded box and consider what portions of your system could benefit from being parallelized:
  - MRP Runs
  - Nightly Processing
  - Reports
  - Data Extracts
  - Data Imports
#21 Parallelize

$ mbpro dbname –p exp.p –param “01|0,3000”
$ mbpro dbname –p exp.p –param “02|3000,6000”
$ mbpro dbname –p exp.p –param “03|6000,9999”

/* exp.p */

define variable startCust as integer no-undo.
define variable endCust as integer no-undo.

startCust = integer( entry( 1, entry( 2, session:parameter, “|” )))
endCust = integer( entry( 2, entry( 2, session:parameter, “|” ))).

output to value( “export.” + entry( 1, session:parameter, “|” )).
for each customer no-lock where custNum >= startCust and custNum < endCust:
    export customer.
end.
output close.
quit.

White Star Software
#21 Parallelize

• But don’t go crazy!
#20 Delete 1 record at a time

- Be careful with large scale purge processes.
- Do NOT “chunk” deletes.
- And do not parallelize them.
#19 Implement -bithold

- The bi file should not be allowed to grow beyond approximately $1/4$th the free space of the filesystem holding it.
- That is because crash recovery may require 2x to 3x the size of the bi file to complete and requesting extra disk space at that point in time is generally not possible.
- If you have never had a problem with excess bi growth, this is a way to ensure that you never do.

-bithold 500
#18 Otherargs

- The most important property in conmgr.properties
- Use for parameters that are not supported by exploder.
- Also allows you to point to a .pf file and maybe even avoid having to use OpenEdge Explorer at all!
#17 Dump & Load

- You *shouldn’t* need to *routinely* dump & load.
- If you’re on OE10+,
- Using type 2 areas,
- That have been well defined,
- And you have a well-behaved application.

- The rest of us benefit from an occasional D&L.
#17 Dump & Load

• Do not be afraid to use the proutil binary dump & load.
• “Binary” refers to the record contents – not the block structure.
• Binary dump is portable:
  – Upwards across Progress versions (i.e. v9 to oe11)
  – Between all Platforms (i.e. Solaris to Windows)
  – Between “bitness” (i.e. 32 bit Windows v9 to 64 bit Linux OE11)
• Most people can get by with some simple scripts.
#16 Larger db Blocks

- Larger blocks result in much more efficient IO.
- Fewer IO ops mean less contention for disk.
- Moving from 1k to 4k is huge. 4k to 8k is relatively less huge but still valuable.
- 8k works best in most cases. Especially read-heavy workloads.
- Better disk space efficiency (tighter packing, less overhead).
- Don’t forget to adjust –B and Rows Per Block!

*White Star Software*
#16 Larger db Blocks

- Large Blocks reduce IO, fewer operations are needed to move the same amount of data.
- More data can be packed into the same space because there is proportionally less overhead.
- Because a large block can contain more data it has improved odds of being a cache “hit”.
- Large blocks enable HW features to be leveraged. Especially SAN HW.
#15 Manage Temp File IO

- Temp-file IO can exceed db IO.
- Sometimes by 2:1, 3:1 or more!
- -T isolates temp file IO.
- -t helps you to crudely diagnose the source of IO.
- -y provides some detail regarding r-code swapping.
- -mmax buffers r-code, 4096 is a good start for ChUI, 16384 for GUI.
- Memory mapped procedure libraries cache r-code.
- Use –Bt & -tmpbsize to tune 4GL temp-tables.
#15 Manage Temp File IO

-rw-r--r-- 1 VEILLELA users 579312 Oct 19 15:16 srtrAyhEb
-rw-r--r-- 1 wrightb users 35697664 Oct 19 15:16 srtH6miqb
-rw-r--r-- 1 STEELEJL users 36772864 Oct 19 15:16 srtz37kyb
-rw-r--r-- 1 THERRIKS users 0 Oct 19 07:12 srt--Elab
-rw-r--r-- 1 root users 17649 Oct 19 15:16 lbiV6Qp7a
-rw-r--r-- 1 root users 34704 Oct 19 15:16 lbi-TymMa
-rw-r--r-- 1 wrightb users 811008 Oct 19 15:16 DBIHDmiqc
-rw-r--r-- 1 BECKERLM users 8192 Oct 19 11:06 DBI--Abac
-rw-r--r-- 1 CALUBACJ users 8192 Oct 19 09:16 DBI--Abyc

CLIENT.MON (-y)

Program access statistics: Times Bytes
Reads from temp file: 0 0
 Writes to temp file: 0 0
Loads of .r programs: 14 524594
Saves of compilation .r's: 0 0
Compilations of .p's: 0 0
Checks of files with stat: 165 0
#14 Update Statistics

- SQL-92 uses a cost based optimizer
- But it cannot optimize without knowledge of the cost! (data distribution).
- Monthly or quarterly “update statistics” is appropriate for most people.
- Or when 20% of your data has changed.
- This is a data intense process:
  - Run it during off hours if you can
  - You might want to only do a few tables/indexes at a time
#14 UPDATE STATISTICS

/* genUpdateSQL.p */

mpro dbName -p genUpdateSQL.p -param "updstats.sql"

sqlexp -user user -password passWord -db dbName -S port -infile updstats.sql -outfile updstats.log

output to value( session:parameter ).
for each _file no-lock where _hidden = no:
    put unformatted
        "UPDATE TABLE STATISTICS AND INDEX STATISTICS AND "
        "ALL COLUMN STATISTICS FOR PUB."
        "'_file._file-name '"'
    skip
    "commit work;"
skip.
end.
output close.
#13 Progress AppServer

- Used to reduce network traffic and latency.
- When properly implemented it will minimize the path length between business logic and data persistence layers.
- IOW, for best performance, the AppServer should live on the same server as the database and use a self-service connection.
- In a VM environment your app server might not be self-service but it should be on the same physical host.
#12 fork() & exec()

define variable i as integer no-undo.

define variable fSize as integer no-undo.

etime( yes ).
do i = 1 to 1000:
  input through value( "ls -ld .." ).
  import ^ ^ ^ ^ fSize.
  input close.
end.
display etime fSize.

3140ms, **at least** 1000 calls to each of open(), close(), fork(), exec(), read() complete with multiple context switches per invocation.

---

define variable i as integer no-undo.

define variable fSize as integer no-undo.

etime( yes ).
do i = 1 to 1000:
  file-info:file-name = "..".
  fSize = file-info:file-size.
end.
display etime fSize.

16ms, 1,000 stat() calls.
#11 -spin

- All new machines, even desktops, laptops, phones and watches are now multi-core.
- Do **NOT** use the old X * # of CPUs rule to set –spin. *It is bogus.*
- Bigger is not always better with –spin!
- Modest values (5,000 to 10,000) *generally* provide the best and most consistent results for the vast majority of people.
- Use `readprobe.p` to explore.
- Check out Rich Banville’s Superb Exchange 2008 Presentation!

White Star Software
The idea is to reduce the number and frequency of checkpoints giving APWs plenty of time to work.

Larger bi clusters permit spikes in the workload to take place without ambushing the APWs.

Easy to overlook when building new db via prostrct create...

512 is the default OE 10 bi cluster size.

8192 is reasonable for small systems.

16384 is “a good start” for larger systems.

Longer REDO phase on startup so don’t get crazy.

**NOT** a good idea for “Workgroup” database licenses.

– For WG small values (512 or 1024) are better.
#10 bi cluster size

```
$ grep '(4250)' dbname.lg
(4250)Before-Image Cluster Size: 524288.  (= 512k)
```

The value above is the oe10 default value of 512k so let’s make it larger:

```
$ proutil dbname -C truncate bi  -bi 16384
...
(1620) Before-image cluster size set to 16384 kb.
(1621)Before-Image Cluster Size: 16777216.  (= 16384k)
```

```
$ proutil dbname -C -bigrow 8
```

White Star Software
#9 APW, AIW, BIW & WDOG

- Always start a BIW
- Always start an AIW
- Start WDOG
- One or Two APWs are usually enough:
  - Do **NOT** follow the old “1 APW per disk” suggestion.
  - Too many is just a (small) waste of CPU cycles.
  - If you are consistently flushing buffers at checkpoints increase bi cluster size and add an APW (one at a time until buffer flushes stop).
#8 Be Wary of Virtualization

• Virtualization is not magic!
• The default assumption with virtualization is that you do not **really** need all of the resources available to you.
• Overcommitting resources **will** hurt performance.
• Virtualization can not create capacity from thin air.
• As a DBA you must size db servers for **peak** load. With shared resources you must size for **simultaneous peaks** in multiple VMs! (Sys-admins often assume simultaneous **valleys** for all resources...)
#8 Not magic...

- A Partner tells a customer “you can support 50 connections on that server”.
- The customer has 200 users...
- So they virtualize the server and bring up 4 VMs!
#8 Right Size Your Virtual Environment

- A database server is NOT the same as a file server
- Do NOT over allocate Resources
- Do NOT “thin provision”
- Ensure that you always have spare capacity
#7 Minimize Network Traffic

- Use FIELDS list in queries.
- Use –cache and –pls.
- NO-LOCK queries pack multiple records into a request.
- Watch out for client-side sorting and selection on queries.
- Remember that CAN-DO is evaluated on the CLIENT (yet another reason not to use it).
- Use -noautoresultlist/FORWARD-ONLY for dynamic queries.
#7 Minimize Network Traffic

- Use a secondary broker to isolate high activity clients (such as reports).
- Set –Mm to 8192 or larger.
- Use –Mn and –Ma to keep the number of clients per server low (3 or less).
- Use –Mi 1 to spread connections across servers.
#7 Minimize Network Traffic

- Jumbo Frames!
- -prefetchDelay
- -prefetchFactor 100
- -prefetchNumRecs 10000
## Impact of Message Size & Prefetch Options

### for each _index fields(_field-name) no-lock: end.

<table>
<thead>
<tr>
<th>totMsgs</th>
<th>qryRecv</th>
<th>recSent</th>
<th>recs/qry</th>
<th>etime</th>
<th>net time</th>
<th>Description...</th>
</tr>
</thead>
<tbody>
<tr>
<td>198</td>
<td>97</td>
<td>1758</td>
<td>18</td>
<td>10</td>
<td>208</td>
<td>-Mm 1024</td>
</tr>
<tr>
<td>208</td>
<td>102</td>
<td>1758</td>
<td>17</td>
<td>14</td>
<td>222</td>
<td>-Mm 4096</td>
</tr>
<tr>
<td>192</td>
<td>94</td>
<td>1758</td>
<td>19</td>
<td>9</td>
<td>201</td>
<td>-Mm 8192</td>
</tr>
<tr>
<td>180</td>
<td>88</td>
<td>1758</td>
<td>20</td>
<td>11</td>
<td>191</td>
<td>-Mm 16384</td>
</tr>
<tr>
<td>162</td>
<td>79</td>
<td>1758</td>
<td>22</td>
<td>14</td>
<td>176</td>
<td>-Mm 32600</td>
</tr>
<tr>
<td>152</td>
<td>74</td>
<td>1758</td>
<td>24</td>
<td>8</td>
<td>160</td>
<td>-prefetchDelay</td>
</tr>
<tr>
<td>154</td>
<td>75</td>
<td>1758</td>
<td>23</td>
<td>12</td>
<td>166</td>
<td>-prefetchDelay -prefetchFactor 100</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>22</td>
<td>30</td>
<td>-prefetchDelay -prefetchFactor 100 -prefetchNumRecs 10000</td>
</tr>
</tbody>
</table>
### Impact of Message Size & PrefetchNumRecs

(prefetchNumRecs dominates!)

<table>
<thead>
<tr>
<th>totMsgs</th>
<th>qryRecv</th>
<th>recSent</th>
<th>recs/qry</th>
<th>etime</th>
<th>net time</th>
<th>Description...</th>
</tr>
</thead>
<tbody>
<tr>
<td>146</td>
<td>71</td>
<td>1758</td>
<td>25</td>
<td>13</td>
<td>159</td>
<td>-prefetchDelay -prefetchFactor 100</td>
</tr>
<tr>
<td>102</td>
<td>49</td>
<td>1758</td>
<td>36</td>
<td>8</td>
<td>110</td>
<td>-Mm 1024 -prefetchNumRecs 10000</td>
</tr>
<tr>
<td>28</td>
<td>12</td>
<td>1758</td>
<td>147</td>
<td>9</td>
<td>37</td>
<td>-Mm 4096 -prefetchNumRecs 10000</td>
</tr>
<tr>
<td>16</td>
<td>6</td>
<td>1758</td>
<td>293</td>
<td>13</td>
<td>29</td>
<td>-Mm 8192 -prefetchNumRecs 10000</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>1758</td>
<td>586</td>
<td>9</td>
<td>19</td>
<td>-Mm 16384 -prefetchNumRecs 10000</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>23</td>
<td>31</td>
<td>-Mm 32600 -prefetchNumRecs 10000</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>22</td>
<td>30</td>
<td>-prefetchDelay -prefetchNumRecs 10000</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>22</td>
<td>30</td>
<td>-prefetchDelay -prefetchFactor 100 -prefetchNumRecs 10000</td>
</tr>
</tbody>
</table>

White Star Software
## Impact of Basic Coding Approaches

<table>
<thead>
<tr>
<th>totMsgs</th>
<th>qryRecv</th>
<th>recSent</th>
<th>recs/qry</th>
<th>etime</th>
<th>net time</th>
<th>Description...</th>
</tr>
</thead>
<tbody>
<tr>
<td>3519</td>
<td>0</td>
<td>1758</td>
<td>?</td>
<td>139</td>
<td>3658</td>
<td>do while … find no-lock</td>
</tr>
<tr>
<td>5276</td>
<td>0</td>
<td>1758</td>
<td>?</td>
<td>174</td>
<td>5450</td>
<td>do while … find share-lock</td>
</tr>
<tr>
<td>28</td>
<td>12</td>
<td>1758</td>
<td>147</td>
<td>7</td>
<td>35</td>
<td>for each no-lock</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>22</td>
<td>30</td>
<td>FENL fields()</td>
</tr>
<tr>
<td>5277</td>
<td>1758</td>
<td>1758</td>
<td>1</td>
<td>155</td>
<td>5432</td>
<td>FE share-lock</td>
</tr>
<tr>
<td>3520</td>
<td>1758</td>
<td>1758</td>
<td>1</td>
<td>134</td>
<td>3654</td>
<td>FE exclusive-lock</td>
</tr>
<tr>
<td>3519</td>
<td>1758</td>
<td>1758</td>
<td>1</td>
<td>87</td>
<td>3606</td>
<td>open query</td>
</tr>
<tr>
<td>3519</td>
<td>1758</td>
<td>1758</td>
<td>1</td>
<td>84</td>
<td>3603</td>
<td>open query fields()</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>29</td>
<td>36</td>
<td>open query fields() cache 50</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>30</td>
<td>37</td>
<td>open query fields() cache 5000</td>
</tr>
<tr>
<td>20</td>
<td>8</td>
<td>1758</td>
<td>220</td>
<td>113</td>
<td>133</td>
<td>sql89 select *</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1758</td>
<td>879</td>
<td>37</td>
<td>45</td>
<td>sql89 select _field-name</td>
</tr>
</tbody>
</table>

No, I am not endorsing SQL89
Nesting and Joins

/* simple nesting */

for each _file no-lock:
    for each _field no-lock of _file:
        end.
end.

/* ugly sort criteria */

for each _file no-lock,
    each _field no-lock of _file
    by _field-name:
end.

/* use a join instead of nesting */

for each _file no-lock,
    each _field no-lock of _file:
end.

<table>
<thead>
<tr>
<th>totMsgs</th>
<th>qryRecv</th>
<th>recSent</th>
<th>recs/qry</th>
<th>etime</th>
<th>net time</th>
<th>Description...</th>
</tr>
</thead>
<tbody>
<tr>
<td>587</td>
<td>195</td>
<td>1950</td>
<td>10</td>
<td>33</td>
<td>620</td>
<td>nested FE</td>
</tr>
<tr>
<td>587</td>
<td>195</td>
<td>1950</td>
<td>10</td>
<td>33</td>
<td>620</td>
<td>joined FE</td>
</tr>
<tr>
<td>6183</td>
<td>195</td>
<td>4748</td>
<td>24</td>
<td>207</td>
<td>6390</td>
<td>joined FE w/ sort on inner field</td>
</tr>
<tr>
<td>33</td>
<td>14</td>
<td>1951</td>
<td>139</td>
<td>150</td>
<td>183</td>
<td>TT option</td>
</tr>
</tbody>
</table>
#6 The Buffer Cache

• The cure for disk IO is RAM.
• Use RAM to buffer and cache IO ops.
• Efficiency of –B:
  – Is loosely measured by hit ratio.
  – Changes follow an inverse square law.
  – To make a noticeable change you must make a large change to –B.
• 100,000 is “a good start” (800MB @ 8k blocks)
• 1,000,000 is not unusual
• 8,000,000 is my current largest customer value in production
#6 The Buffer Cache

## In Big B You Should Trust!

<table>
<thead>
<tr>
<th>Layer</th>
<th>Time</th>
<th># of Recs</th>
<th># of Ops</th>
<th>Cost per Op</th>
<th>Relative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Progress to –B</td>
<td>0.96</td>
<td>100,000</td>
<td>203,473</td>
<td>0.000005</td>
<td>1</td>
</tr>
<tr>
<td>-B to FS Cache</td>
<td>10.24</td>
<td>100,000</td>
<td>26,711</td>
<td>0.000383</td>
<td>75</td>
</tr>
<tr>
<td>FS Cache to SAN</td>
<td>5.93</td>
<td>100,000</td>
<td>26,711</td>
<td>0.000222</td>
<td>45</td>
</tr>
<tr>
<td>-B to SAN Cache</td>
<td>11.17</td>
<td>100,000</td>
<td>26,711</td>
<td>0.000605</td>
<td>120</td>
</tr>
<tr>
<td>SAN Cache to Disk</td>
<td>200.35</td>
<td>100,000</td>
<td>26,711</td>
<td>0.007500</td>
<td>1500</td>
</tr>
<tr>
<td>-B to Disk</td>
<td>211.52</td>
<td>100,000</td>
<td>26,711</td>
<td>0.007919</td>
<td>1585</td>
</tr>
</tbody>
</table>

(Approximately 4 records per read op in non –B cases.)
#5 IO Subsystems

- Use more than one disk:
  - A fast disk can do 150 to 200 **random** IO Ops/sec.
  - Kbytes/sec is a measure of **sequential** IO.
  - OLTP is mostly **random**.
- Don’t waste time trying to “manually stripe”.
- Instead, use “hardware” striping and mirroring.
- Isolate AI extents for **safety**, not performance.
- Isolate temp-file, application, OS and “other” IO.
#5 The SAN Scam

- External, Shared storage is NOT a performance enhancer
- DB IO ops are random and seek time is critical
  - The better tuned your db, the more random the IO becomes
- Accessing data over a network connection is only “fast” in the fevered imagination of a delusional sales person
- Lightspeed = 1 foot per nanosecond
RAID #5

“Don’t be cheated first and surprised later!”

• Great performance when there is no load:
  – And when there are no disk failures.
  – And if your database is roughly the same size as the SAN cache.
  – Which nobody else is using.
  – All of the RAM that you can use – cleverly placed where it will do you the least amount of good.
  – All of the performance of a single disk with none of the cost savings.
#5 RAID 6

- How can we possibly make RAID 5 Worse?
- Add another parity disk!
#5 BAARF.com - Enough is Enough

- Battle Against Any Raid (Free, Four, Five or Fix)
  - http://www.miracleas.com/BAARF/RAID5_vs_RAID10.txt
  - http://www.facebook.com/pages/BAARF
  - http://gurucollege.net/rants/BAARF-or-why-raid5-isnt-safe/

- Don’t be cheated first and surprised later!
#4 Index Compact

- Compacts Indexes.
- Removes deleted record placeholders.
- Improves “utilization” = fewer levels & blocks and more index entries per read.
- Runs online or offline.
- Available since version 9.
#4 Index Compact

proutil dbname –C idxcompact table.index target%

- Do NOT set target % for 100!
- Consider compacting when utilization < 70%
- ... and blocks > 1,000.

INDEX BLOCK SUMMARY FOR AREA "APP_FLAGS_Idx" : 96

<table>
<thead>
<tr>
<th>Table</th>
<th>Index Fields</th>
<th>Levels</th>
<th>Blocks</th>
<th>Size</th>
<th>%Util</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUB.APP_FLAGS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AppNo</td>
<td>183</td>
<td>1</td>
<td>3</td>
<td>4764</td>
<td>37.1M</td>
<td>89.9</td>
</tr>
<tr>
<td>FaxDateTime</td>
<td>184</td>
<td>2</td>
<td>2</td>
<td>45</td>
<td>259.8K</td>
<td>72.4</td>
</tr>
<tr>
<td>FaxUserNotified</td>
<td>185</td>
<td>2</td>
<td>2</td>
<td>86</td>
<td>450.1K</td>
<td>65.6</td>
</tr>
</tbody>
</table>

INDEX BLOCK SUMMARY FOR AREA "Cust_Index" : 10

<table>
<thead>
<tr>
<th>Table</th>
<th>Index Fields</th>
<th>Levels</th>
<th>Blocks</th>
<th>Size</th>
<th>%Util</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUB.Customer</td>
<td>13</td>
<td>1</td>
<td>1</td>
<td>874.0B</td>
<td>21.5</td>
<td>1.0</td>
</tr>
<tr>
<td>Comments</td>
<td>14</td>
<td>2</td>
<td>2</td>
<td>9.0K</td>
<td>56.5</td>
<td>1.9</td>
</tr>
<tr>
<td>CountryPost</td>
<td>12</td>
<td>1</td>
<td>2</td>
<td>9.9K</td>
<td>62.2</td>
<td>1.8</td>
</tr>
<tr>
<td>CustNum</td>
<td>15</td>
<td>1</td>
<td>2</td>
<td>22.5K</td>
<td>62.9</td>
<td>1.7</td>
</tr>
<tr>
<td>Name</td>
<td>16</td>
<td>1</td>
<td>1</td>
<td>1.3K</td>
<td>33.6</td>
<td>1.0</td>
</tr>
</tbody>
</table>

White Star Software
#2 Solid State Disks

- Virtualization is not magic – but SSD is pretty darn close!
- Literally thousands of times faster than rotating rust.
- Perfectly safe and reliable.
- But -- disable Windows disk defrag programs.
- Will not cure all performance problems – but they sure do help with IO throughput.
- It is still necessary to optimize code properly and use lots of RAM for –B (see #6)
#1 Stupid 4GL Tricks

- Bad code will defeat any amount of heroic tuning and excellent hardware.
- Luckily bad code is often advertised by the perpetrator as having been developed to “improve performance”.
- Just because a feature of the 4GL can do something doesn’t mean that it should be used to do it.
/* SR#1234 – enhanced lookup to improve performance! */

update cName.

find first customer where cName matches customer.name
use-index custNum no-error.

-Or - /* different variations used in different bits of code... */

find first customer where can-do( cName, name )
use-index custNum no-error.
#1 Stupid 4GL Tricks

- There is a lot of crap code in the world.
- Stop dumpster-diving for code!
- Don't emulate it, eliminate it.
Questions?
Thank You!