Performance comparison of kbmMemTable Std/Pro and AnyDAC CDS

Something that make me trigger is when someone claims their stuff is faster than my stuff....
I dont know if it has something to do with the 'mine is bigger than yours' male ego thing, but nevertheless, I always trigger on those type statements :)

Anyway, the author behind the free (and seemingly nice) DB API abstraction framework, AnyDAC has now repeatedly claimed that their client dataset implementation is faster and more fullfeatured than kbmMemTable, although recognizing that kbmMemTable Pro may be faster than their product.

Now I decided to have a look at the performance side, to test if their claims really holds true.

We are using the old slightly modified benchmark program that we also used in our earlier battle against another vendor claiming superiority (and that was originally created by them).
The author behind AnyDAC also use a variant of that original application for their tests.

The following benchmarks has been run on a 2.8Ghz hyperthreading P4 with 2GB memory and lots of disks and diskspace.

Each vendors tests have been benchmarked seperately.
Hence we have stopped and started the application between each vendor test to try to ensure level playing grounds.

We have waited until the CPU usage has stabilized to 97-98% idle before actually running the benchmark by clicking the start button on the application.
When practically possible (which wasnt the case with the AnyDAC 100.000 records test due to lack of my patience), we have run the benchmark multiple times to allow for variations, and the following represents reasonably stable measurements. When single measurements have shown instability (by looking exceedingly high) we have rerun the test and compared the measurements over multiple tests fairly for all vendors/variations.

All values are ms.

All locates in the following are only repeated 1000 times locating different values in the datasets.

This is our first set of benchmarks. As the benchmark system can have contained different loads (specially memory wise) at time of run, we have decided to
add an additional set of benchmarks further down, documenting kbmMemTable vs AnyDAC also including AnyDAC's batch mode.

10.000 records

	kbmMT non batched	kbmMT batched	kbmMT Pro non batched	kbmMT Pro batched	AnyDAC cds non batched
Without indexes					
Insert:	3266	2563	2656	1953	3167
Edit:	2421	2484	1812	1969	3453
Locate by ID:	375	266	250	266	257
Locate by FInteger:	344	203	187	218	264
Locate by FString:	2188	2156	313	407	550
Delete:	218	250	93	93	1000
Append:	2531	2625	1937	1922	3860
Close:	16	16	15	16	15
With indexes					
Insert:	3921	3297	2516	2219	4734
Edit:	5016	3297	2844	2109	6422
Locate by ID:	109	78	47	47	281
Locate by FInteger:	63	62	47	47	313

Locate by FString:	188	109	78	62	625
Delete:	1156	328	422	78	2094
Append:	3844	3297	2584	2344	5265
Close:	15	16	16	15	31

100.000 records

	kbmMT non batched	kbmMT batched	kbmMT Pro non batched	kbmMT Pro batched	AnyDAC cds non batched
Without indexes					
Insert:	50017	26469	25594	20015	69031
Edit:	23985	26531	29125	22610	50422
Locate by ID:	234	301	300	265	313
Locate by FInteger:	328	406	251	235	344
Locate by FString:	1859	2547	499	457	688
Delete:	22329	27093	2172	1172	29547
Append:	25843	26391	28203	25062	48953
Close:	157	125	110	109	297
With indexes					
Insert:	107843	37797	34937	30563	133656
Edit:	169985	33000	40125	30546	326969
Locate by ID:	47	46	63	63	297
Locate by FInteger:	62	79	125	109	297
Locate by FString:	250	140	187	157	703
Delete:	67453	22453	7906	1406	175531
Append:	71187	34937	33656	30031	84469
Close:	172	157	110	109	312

This is our 2nd run of the benchmarks.
It was created due to we gained new knowledge about a couple of additional optimization parameters that could be issued on AnyDAC cds, namely a SilentMode property that we were told by the author, should be set to false, and that AnyDAC CDS also have a batch operation, that can be started with BeginBatch and ended with EndBatch. BeginBatch takes one optional argument that default is false. Setting it to false makes the tests fail for AnyDAC. Hence we set it to true.

We have remade all tests again to try to compare on a level ground.
Differences between this benchmark and the above indicates that the machine have had an undocumented load (probably memory or fragmentation of memory as we monitored CPU), temperature of CPU (newer ones may throttle speed if getting hot) etc, at the first time of the benchmark.
As values can vary quite alot between runs, we have decided to take the best run out of several for each product and document that. Thus the values are best possible values out of several runs each.
What can also be interpreted by such benchmarks is that its quite difficult to get 100% stable results on a machine that runs anything but the benchmark application. The best benchmark to make is probably to calculate the number of machine code instructions needed to perform the tasks, and according to manuals calculate the time taken for each and sum it all up. However thats not the point of this test.

10.000 records

	kbmMT non batched	kbmMT batched	kbmMT Pro non batched	kbmMT Pro batched	AnyDAC cds non batched	AnyDAC cds batched
Without indexes						
Insert:	1047	782	1343	922	2016	860
Edit:	1062	861	1219	1062	2265	906
Locate by ID:	172	109	140	156	157	125

Locate by FInteger:	172	78	110	125	125	94
Locate by FString:	922	704	203	219	344	250
Delete:	94	78	63	31	641	78
Append:	1172	922	1312	1172	2312	1094
Close:	0	0	0	0	15	16
With indexes						
Insert:	1750	1156	1593	1297	2937	1234
Edit:	2359	1156	1765	1265	3985	1188
Locate by ID:	32	15	16	15	188	125
Locate by FInteger:	62	15	47	16	125	109
Locate by FString:	94	47	47	31	406	282
Delete:	578	94	250	47	1234	109
Append:	1734	1250	1656	1312	3094	1250
Close:	0	0	16	0	15	16
CPU time	0:11	n/a	0:09	0:07	0:18	n/a

The two n/a CPU time measurements is due to I simply forgot to note them down.

100.000 records

	kbmMT non batched	kbmMT batched	kbmMT Pro non batched	kbmMT Pro batched	AnyDAC cds non batched	AnyDAC cds batched
Without indexes						
Insert:	24641	13766	13781	11703	41328	25063
Edit:	14266	14625	14687	13093	39187	18469
Locate by ID:	171	203	157	157	235	219
Locate by FInteger:	219	218	156	172	250	281
Locate by FString:	1188	1297	234	235	688	547
Delete:	14328	14203	860	593	23297	11359
Append:	18250	19407	16953	14735	43984	20860
Close:	125	47	78	62	281	235
With indexes						
Insert:	82844	27719	25141	19578	111906	41437
Edit:	151390	28765	32828	22094	294953	29672
Locate by ID:	47	46	31	31	297	296
Locate by FInteger:	62	63	63	47	297	313
Locate by FString:	204	109	187	140	656	579
Delete:	61921	19250	6797	981	155562	14187
Append:	62922	31672	31000	23375	76891	30625
Close:	172	94	109	94	297	219
CPU time (min:sec)	6:39	2:40	2:14	1:40	11:36	3:03

If we compare kbmMT Standard non batched with AnyDAC CDS non batched, we see kbmMT being faster regardless of dataset size, except for searching on strings.
However with 100.000 records we see that kbmMT is being faster by a large margin as it is running the complete test in approx. 54% the time taken for AnyDAC CDS.
The difference increases almost exponentially with the size of the datasets.

If we compare kbmMT Standard batched with AnyDAC CDS batched, we see kbmMT also being faster overall. On 10.000 records the difference is not big. But at
100.000 records the difference starts to show as kbmMT is running the test in about 88% the time taken for AnyDAC CDS. We can easily see that AnyDAC CDS performs
substantially better batched than unbatched.

If we then compare kbmMT Pro non batched, we see it being significantly faster than AnyDAC batched, running the

test in approx. 74% the time taken for AnyDAC CDS batched!
And finally comparing kbmMT Pro batched, we see it being very fast, running the test in approx. 54% of the time taken for AnyDAC CDS batched.

The difference is also indicating to grow with even larger datasets.

In addition I would like to comment, that its always 'easy' to cut functionality away. Essentially the fastest extremely simple, but quite non functional memory dataset would be to have a simple list where everything is just appended to. However a memory dataset should keep track of order of insertions etc too. This is for example something kbmMemTable always keeps track of, regardless if running in batch mode or not. The roworder index is always updated as that is the only way to guarantee that the order of record insertions/updates/deletes is well known.

Im not the one to give advices to potential competitors... but imo the author should definitely focus his adverticements on
his frameworks abilities to connect to different databases rather than on performance. His database abstraction API is what makes his framework
interesting imo. Specially because another free DB API abstraction framework, Zeos, seems to be stalling at the moment.

As I have now spend lots of precious time that could have been spend on developing our own products, I will not pursure this subject any longer.
kbmMemTable is (even by competitors) being recognized as the leader in performance as it often is being used as a reference in benchmarks for competing products. It is obviously a nice thing to be recognized. But its time to start focusing on other areas instead of the fruitless job of constant chasing kbmMemTable.

Benchmarks may be indicators, but as the reasons for the results are not always easily understood, the benchmarks may more damaging than useful to people making a choise! You may ultimately end up with something somewhat faster than kbmMemTable Pro for a specific task, but what is that worth if the overall usability of the component is destroyed because of the chase for extreme performance?

best regards
Kim Madsen