

Out in Left Field

Volume IV, Issue IV

IBM i for Business Database Performance

today's
business
thought:



Top 10 reasons why you should create your own

SQL indexes

for JD Edwards

By Tom Davidson

Over the years I have come to discover that database administrators and application people seem to have an aversion to adding indexes to their systems. This appears to come from 'the old days', when machines were slow, cycles were expensive, disk was expensive, and the people were cheap.

This observation is not limited to people in the 'i' world. It crosses over into the Microsoft and Oracle worlds as well.

These observations lead me to try, with my apologies to David Letterman, to come up with a 'Top Ten reasons' why you should make your own SQL indexes in the EnterpriseOne world.

1) QBE

All shops use QBE. Even if you have QBE limited to existing indexes only, you probably have users complaining about how they want to get the data. JDE didn't know in advance how you wanted to get at the data, so they shipped a set of indexes that helped performance for their way of using it. Users are VERY creative creatures. They do not really enjoy restrictions. But the thing they hate more than restrictions is wasting time trying to do something and then having to quit because it takes too long. New indexes solve both these problems. If you restrict QBE, you can give them more options by adding indexes. If you do not restrict QBE, then, by studying how the users use it, you can add the proper indexes to make their most painful selections much faster.

2) Row Security

JDE has offered you a very versatile method of controlling users access to your data. Unfortunately when JDE decided what to ship to you, they had no way of knowing how you would want to set up security. This lack of foresight can cause the system to have to look at a large amount of data to decide how to fill a single screen. Row security is implemented by using a series of SQL 'BETWEEN' functions. If the item you are securing is not in the index, the system gets to read the index, then read the record. In effect you get 2 reads, for the price of, well, 2 reads. If you would have had an index you would have effectively gotten 2 reads for the price of one.

3) Dashboards

CEO's and other managers love dashboards. They also know exactly what they want to see, and they want to see it NOW. Unfortunately as we all know, how they want to see it doesn't seem to conform to any known way that others want to see it. You are left designing how to get to the data yourself. If you try to do this without creating new indexes, even if they LOVE the results, they probably won't love the performance.

4) Custom Reports

Everyone that does not have at least one custom report for their JDE system raise your hand...Come on, anyone? I thought not. If you have custom reports, by definition you are looking at something JDE either did not anticipate, or decided was not worth it for them to do themselves. In either case why would you think that JDE would have given you an efficient way to get at the data?

5) Modifications

Many of the shops I have been at have implemented at least some modifications. Most of the time these are invoices, purchase orders, or work orders. There

Inside this issue:

Top 10 reasons why you should create your own SQL indexes for JDE	1
Clustered data for improved performance	2
CTO Memo	3
Performance tuning QSQRVR Jobs	5
SQL Union Tip	6
Getting rid of unused indexes and keyed LFs	8

(Continued on page 7)



Clustered Data FOR IMPROVED PERFORMANCE

Those of you familiar with Microsoft SQL Server or Oracle's database management systems know that one of the most powerful performance tuning methods is a technique generically called "clustering".

The basic idea behind clustering is to put a table's rows in a predetermined order. The sorted nature of the rows allows the query engine to dramatically cut down on random disk I/O when multiple rows are processed by a query. Because synchronous disk operations are the slowest part of a computer system, they are the largest contributor to poor response times for most applications. Therefore, reducing the number of disk operations can significantly improve an application's performance.

Our website has a short video that visually demonstrates how disk I/O works with a "traditional" and a "clustered" index. In this example, a select statement like the following would process the index sequentially: **SELECT column FROM table WHERE column >= 'A' AND column <= 'I'**. In the video the blue rectangles represent index pages and the small

squares represent rows of data. You'll note that the clustered index does fewer disk I/Os to process the same rows because a block of records are brought into memory instead of being done one at a time. The video can be found at the following link:

<http://www.centerfieldtechnology.com/files/clustered1/clustered1.htm>

On Microsoft SQL Server, implementation of this concept is done with a special index type – called, coincidentally a "clustered index." In the case of SQL Server, on disk the data in the row is shared with the leaf page in the index. As a result, when the clustered index is read sequentially (like when a query asks for all values BETWEEN x and y) the index can be used to select the appropriate rows while at the same time doing blocked I/O at the table level. Oracle's approach is similar in concept but the implementation approach is different...but the bottom line is that both database management systems support a clustering approach as one tuning method.

(Continued on page 3)

As a JDE consultant, I'm always looking for ways to deliver more value to my customers.

**CTI's HomeRun
delivers the
performance
necessary
for today's
business.**

Centerfield
TECHNOLOGY 





There is an air of excitement, anticipation, and nervousness in the air. Yes...school has started.

As we send our children off to get an education that will (hopefully) prepare them for a happy, successful life, it is a good time to reflect on our own technical vitality. As readers of this newsletter, you are probably in that group that enjoys learning and acquiring new skills. In this edition we've provided a lot of information I hope you find useful in your job managing the iSeries, your DB2 database and SQL-based applications.

First we have an article I'm sure will be of interest to those of you with large production databases and a hunch that you have too many logical files that aren't being used. Second, there is an article on how to use the concept of data clustering to improve the performance of RPG batch jobs as well as SQL queries. Then there's some news about SQL unions and how to capture performance information for QSQRV jobs with our latest HomeRun release, version 6.2.

Many thanks to Tom Davidson for providing information for our JDE audience in his article "Top 10 reasons why you should

create your own SQL indexes for JD Edwards". I can't tell you how many times we've been able to help E1 customers with their performance problems, especially right after they go live.

Finally I'd like to welcome Dave Anderson to our esteemed group of expert consultants available to CTI customers. As you know, Dave has been called upon extensively by Oracle customers worldwide for performance situations.

Tom and Dave are available through Centerfield for your next Oracle project.

Best Regards,

Mark L Holm
Chief Technology Officer
Centerfield Technology, Inc.

(Clustered data for improved performance—Continued from page 2)

While DB2 on the System i (IBM i) does not have a way to define a clustering column or columns, the idea of ordering data based on a heavily used index still makes sense. Historically there have been two common ways to sort data. The first is with the format data (FMTDTA) command and the second is with the reorganize physical file member (RGZPFM) support. Format data works well with temporary work files while RGZPFM is normally used on production data – to remove wasted space taken up by deleted rows rather than to reorder the data. Until the advent of i5/OS V5R3, neither method worked particularly well in environments that required 7x24 availability. In V5R3 IBM introduced a way to reorganize files and tables while applications were active thus removing the key barrier to using RGZPFM. [See our previous newsletter articles about concurrent reorganize referenced by links at the end of this article.]

Centerfield, recognizing the opportunity to leverage concurrent reorganize to reduce disk I/O, introduced "clustered index" support in HomeRun Version 6.0. Our Autonomic Database Administrator (AutoDBA) analyzes tables and their associated indexes. If there is a single index that is very heavily used (as compared to all of the other indexes over the table), AutoDBA will recommend that SQL index (or logical file) be used on a RGZPFM command to re-order the rows in the table. Once the reorganize completes, sequential access of the popular index will result in fewer I/Os and improved performance. AutoDBA will also provide the exact command to implement the change assuming the use of "concurrent reorganize" rather than a "standalone reorganize" (functionally either would work).

If you'd like more information on clustering or AutoDBA please contact us at support@centerfieldtechnology.com

Further information and reading

Previous newsletter articles

<http://www.centerfieldtechnology.com/publications/archive/December%202006.pdf>

<http://www.centerfieldtechnology.com/publications/archive/Aug%202006A.pdf>

IBM documentation on Reorganization

<http://publib.boulder.ibm.com/infocenter/iseries/v5r3/topic/dbp/rbaforrqpf.htm#rbaforrqpf>

<http://publib.boulder.ibm.com/infocenter/iseries/v5r3/topic/dbp/rbaforeorgtypes.htm#rbaforeorgtypes>

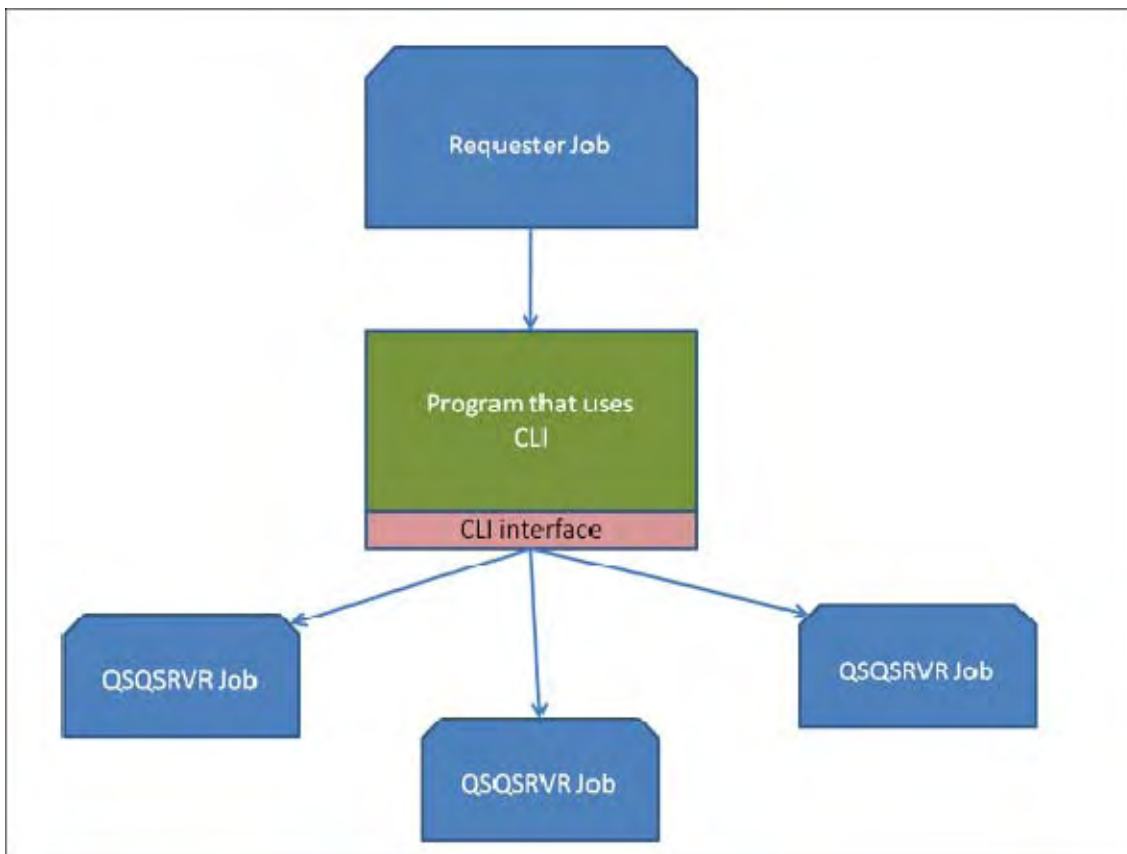
Microsoft SQL Server

http://www.sql-server-performance.com/articles/per/index_data_structures_p1.aspx

[http://en.wikipedia.org/wiki/Index_\(database\)](http://en.wikipedia.org/wiki/Index_(database))

Performance Tuning QSQSRVR Jobs

Centerfield recently hosted a webinar on the topic of database server jobs and the challenges they present (links below). One flavor, QSQSRVR jobs, are created and used on behalf of IBM's Call Level Interface (CLI). Specifically, these jobs execute SQL requests for programs that use CLI in what is called server mode. Essentially server mode delegates the execution of SQL statements to a QSQSRVR job that returns the results to the requester job. Visually it looks like this:



More and more applications that run on the iSeries are using CLI as a mechanism to execute SQL. One of the most popular is Oracle's EnterpriseOne and in particular UBE (Universal Batch Engine) jobs. IBM's own servers are now using CLI extensively including the DB2 Web Query product. Java-based applications that use IBM's native JDBC driver also use server mode.

One of the issues with programs that use the CLI interface in server mode is that the job that "uses" SQL to access DB2 is not the job that "executes" the SQL. For example, I may have a batch job called DAILYSALES that generates several reports. Unfortunately, this job is running slowly and I need to determine why and fix the problem. How can I do that when I don't know which QSQSRVR job will run the SQL statements?

One solution is to collect database monitor information for every QSQSRVR job on the system and try to figure out which statements are causing the problem even if I don't know if they were executed by the DAILYSALES job. A way to solve this issue is to run DAILYSALES when the system is "quiet" so that the QSQSRVR data is likely generated by the batch job. Unfortunately, neither of these options is very realistic or satisfying.

Another option is to collect database monitor information for a specific user profile. Since the QSQSRVR jobs will run under the profile used by DAILYSALES, the monitor information will be just for that user. The biggest roadblock to this solution is the use of general (or generic) user profiles. If the profile used to run DAILYSALES is used by a large number of jobs then specifying the user profile as a database monitor filter will be at best a partial solution.

SQL UNION TIP

Far too often I encounter developers not aware of the functional and performance differences between the two flavors of an SQL UNION clause. In fact, sometimes they're not aware that two flavors exist at all. This tip will try to illustrate the functional and performance differences between UNION **DISTINCT** and UNION **ALL** clauses.

UNION DISTINCT

Most developers using SQL's UNION clause do not specify optional qualifiers of **DISTINCT** or **ALL**. They may or may not be aware the default qualifier is the **DISTINCT** attribute.

Now that you're aware that **DISTINCT** is the default qualifier, some of the implications should be obvious. **DISTINCT** attribute instructs DB2 engine to weed out all of the duplicates from the final result set. This is a very nice feature if you want to see unique rows in the unionized (merged) result set only. However, work performed to weed out of duplicates out of the final result set is by no means trivial. If the final result set is large, **DISTINCT** processing can take significant time and resources. Translation - performance suffers.

UNION ALL

Based on my practical experiences, the **ALL** qualifier will work just fine in over 90% of situations where UNION is being used. There are two reasons behind that. One is that most of the time, the developer either really wants to display all of the rows from the unionized result set or doesn't mind that duplicates are included. The other reason **ALL** qualifier works in most situations is that quite often the two **SELECT** statements are bringing in result sets that are by default **DISTINCT**, or are explicitly made **DISTINCT** by their selection criteria (**WHERE** clause).

Since the performance characteristics of the **ALL** qualifier are orders of magnitude superior to its **DISTINCT** companion, I strongly recommend using it as default instead of simply specifying UNION and taking the easy default of **DISTINCT**.

Functional differences

Since I've made the recommendation to make every effort to utilize the **ALL** qualifier on the UNION clause for performance reasons, I'd be remiss not to explain what functional implications that has on the unionized result set. With that in mind, I'll present couple of scenarios and illustrate the difference in output between **DISTINCT** and **ALL** union variations.

Let's say we have two single-column tables, A & B. We want to merge their result sets into a single result set. Assuming following data in the two table's rows:

A: 1,2,2,3,5
B: 4,5

UNION [**DISTINCT**] output: 1,2,3,5,4
UNION **ALL** output: 1,2,2,3,5,4,5

Now, if I append **ORDER BY 1** to the **SELECT** statement, I'd get a result set in the cardinal order and outputs would look like:

UNION [**DISTINCT**] output: 1,2,3,4,5
UNION **ALL** output: 1,2,2,3,4,5,5

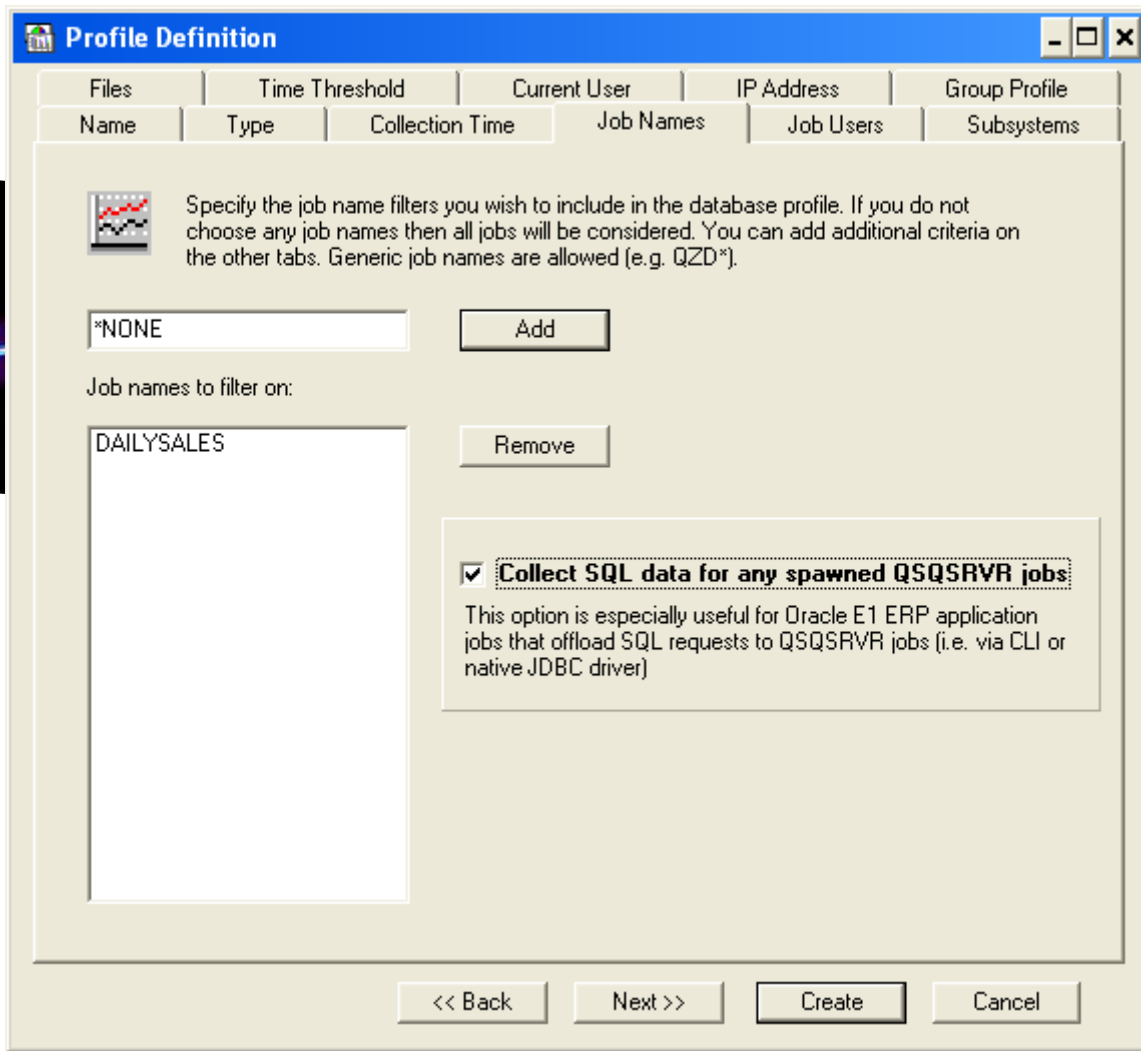
Conclusion

Don't succumb to the easy default when using the UNION clause (not specifying any qualifiers). Instead, develop a habit and specify UNION **ALL** whenever functional requirements allow it. The performance advantage it carries over the **DISTINCT** makes it a no-brainer.

Elvis

(Performance Tuning QSQRV Jobs—Continued from page 4)

The good news for Centerfield customers is that we have a solution to this issue in HomeRun Version 6.2. As can be seen in the following screen shot, there is a new option to specify the name of a job and capture the SQL in QSQRV jobs it uses.



In this example, when the DAILYSALES job starts, any QSQRV job that it uses via CLI will have its SQL captured as part of the database monitor information. This allows the analysis to be focused on the work done by the job of interest without having to tie the database server work back to the requester job.

QZDASOINIT & QSQRV jobs: Monitoring and auditing techniques

The presentation:
[www.centerfieldtechnology.com/files/CTI WEBINAR 12AUG08.pdf](http://www.centerfieldtechnology.com/files/CTI_WEBINAR_12AUG08.pdf)

The event recording:
[http://www.centerfieldtechnology.com/files/QZDASOINIT webinar.wmf](http://www.centerfieldtechnology.com/files/QZDASOINIT_webinar.wmf)

And the questions and answers for each session:
www.centerfieldtechnology.com/files/qq_qa.doc

The support and training from Centerfield is like having a DB2 expert on staff. They help us maximize our iSeries business investment.

Centerfield 
TECHNOLOGY





always seems to be a need for 'just that one extra piece of information'. The problem is that often the information that they want, doesn't seem to be in the files that the UBE or application has on hand. This means that you have to go get it yourself. In my experience, in at least 30% of these cases there is no efficient way to get that piece of information. The result is that the process that has been modified ends up dramatically slower.

6) 3rd party Add-ons

There are a myriad of 3rd party add-on's to E1. From CreateForms to Q-Software that provide extensions to the EnterpriseOne functionality. Each of these products gets at the data in ways that Edwards could not have possibly anticipated when they shipped your system. In order to make these add-ons fly you need to add indexes based on how you use them.

7) Ad-hoc query tools

For those of you who raised your hand in #4, you probably don't have custom reports because you use 3rd party tools like Cognos or Crystal. These tools are great; they give you even more flexibility to get the data you want. The problem is that they give you even more ways to get at the data you want. This comes at the cost of forcing your system to go through loads more data. Just try writing a really cool report tying orders to your accounting (F0901, F0911, F4201, F4211), or even better against order history, with no new indexes. Give me a call next month, or next year when it finishes.

8) What's a purge?

OK, admit it. You keep telling your users that they need to purge that F0911 file. It's up to 10 million records. Their response, 'But I need 10 years of history...'. In my experience, most data more than a month old, is never accessed again, except once at year end. Imagine if you could at least from an operational point of view ignore the 99% of the data that is 'old.' Indexes give you this ability, without having to purge.



9) Write once, update twice, read 100 (or 1000)

OK, I haven't convinced you yet why you need to create your own indexes. Here is a technical reason to do so. The primary reason I hear that people don't want to add indexes is a fear of degrading performance.

Let's look at what actually happens in the life of an order line. First the order is taken. The line gets written (the order is taken), 1 index maintenance. The order line is then picked, packed, shipped, and invoiced (The index is maintained each

time, total 5 index maintenances). You have maintained the index 6 times, and if you purge (see prior item), you get one more for 7. Also remember that indexes that have no fields changed don't get maintained.



Balance this against the times the record is read, you send an order confirmation (1), print a pick list(2), print the packing list (3), print the BOL (4), print the invoice (5), backorder report (6), product shipped report (7), daily sales report(8), weekly sales report(9), and the month end process (10). You now have 10 times you have read the record, on purpose. Now suppose you didn't have an index over the line number and status. Every time you need to update ANY lines status for this order you will get to read this line again just to eliminate it. These are what I call 'hidden' reads. They happen but you don't necessarily know they happened.

In reality I see records are actually updated a small fraction of the times they are accessed. It makes sense to bias your performance concerns towards making the reading more efficient instead of the write/update/delete more efficient.

10) Money

As they say, time is money. The cost of disk and cycles (CPU) is going down, and shows no signs of stopping anytime soon. The cost of the people, is going up, and also shows no sign of slowing anytime soon. Thank Goodness. Well defined indexes reduce the time it takes the system to serve data up to your user community, reducing the costs of your business. You will also reduce the cost of the system itself to your business by reducing the amount of CPU needed to process your query, the amount of disk resources to access your data, and the amount of memory needed to store data while it is being evaluated.



11) Your happiness

OK, I know this is more than ten, but it is impossible to quantify the increase in your quality of life when your users are not calling all the time telling you how slow the system is, and how much the software stinks. I know this from experience. I leave the proof as an exercise for the user.



Tom Davidson

Tom Davidson is an iSeries consultant with RS Infocon. Previously Tom was a global systems architect at JohnsonDiversey, located in Racine, Wisconsin. He has 25 years of experience on the IBM i, 17 years of performance tuning and a background in application development. Tom may be reached at tdavidson@rsinfocon.com or 262-898-7456.



Getting rid of unused indexes and keyed LFs

This recent SystemiNetwork article on unused index management: <http://systeminetwork.com/article/are-unused-database-indexes-killing-your-system-performance> served as the inspiration for this article. The author, Dan Riehl, offers IBM's Navigator 'Show Indexes' support as a method of addressing the unnecessary database overhead incurred by immediately maintained keyed access paths (aka indexes or keyed LFs or primary keys or unique/foreign constraints). The 'Show Indexes' option offers the necessary metrics and as long as you're willing to dedicate your time to look at the keyed access paths over each one of your database tables, you will indeed be able to remedy said overhead.

However, I just couldn't leave it at that, knowing that one of our tools possesses an autonomic feature that addresses this overhead without any administrator action whatsoever. In this era when human time is probably the most expensive part of any business' operating expense, a tool such as we offer simply cannot be overlooked.

Enter AutoDBA

As some of you are aware, our flagship pair of products, insure/INDEX & ANALYSIS, has been enhanced with the AutoDBA (Autonomic DBA) feature about a year and a half ago. Its intent was to make the DBA tasks as autonomic as possible (read automated, where no post-configuration administrator action is required). The first release of this new feature focused on the balanced index management - getting rid of the overhead of the existing keyed access paths as well as advising missing indexes that are needed for the existing queries.

One of the eight advice categories in AutoDBA was "Unused Index" advice. I will highlight some of the ways our 'Unused Index' advice differs from the 'Show Indexes' data available in the Navigator.

Unused Index advice highlights

- Unused Index advice is only made once an index (read any-keyed-access-path, including keyed LFs) has not been used longer than the configured threshold (40 days by default). That eliminates any already used indexes and freshly built indexes from consideration.
- Advice is immediately actionable via a right-click option in the GUI client. Multiple actions can be executed with a single right-click option (if administrator prefers manual action).
- Discovery (analysis) for the advice is only made on the schemas and tables administrator wishes, meaning the administrator can focus on high visibility, critical schemas (libraries).
- Advice execution can be automated. By default, no automatic action will take place. The administrator needs to both allow action via a set of permission options as well as define highly targeted object filters for the indexes action is allowed for (better safe than sorry).
- Overhead of the unused indexes is eliminated in stages to ensure no file access is overlooked by the administrator (i.e. end-of-the-year processing).
 - Stage one alters the immediate keyed access path maintenance to delayed maintenance (some overhead remains).
 - The second stage alters the delayed maintenance to rebuild maintenance (zero overhead remains).
 - Stage three is optional and involves removal of the unused file object (DLTF option) to eliminate clutter.
- Every unused index action, including deletion of the unused file object, is undoable (can be rolled back).
- Every action is automatically audited.

Conclusion

When combined with the automatic building of missing indexes, a case can be made that the administrator can truly just 'set it and forget it', and over a period of time achieve indexing nirvana. In this 'nirvana' scenario, all existing indexes would indeed be used by the query optimizer and/or native I/O programs, and there would be no missing indexes query optimizer is craving for, as high value indexes are automatically built by the AutoDBA.



Ready for V6R1 today.

It means ready for today's Power,
ready for today's business, ready
for today's workload.

Click here for your free full version evaluation:
[www.centerfieldtechnology.com/pdf/Evaluation & Product Information.pdf](http://www.centerfieldtechnology.com/pdf/Evaluation%20&%20Product%20Information.pdf)

Current customers may upgrade here:
[www.centerfieldtechnology.com/pdf/HomeRun 6.2 Upgrade.pdf](http://www.centerfieldtechnology.com/pdf/HomeRun%206.2%20Upgrade.pdf)

