

All About Oracle Database Fragmentation

Craig A. Shallahamer

OraPub, Inc.

1. Abstract

There are many types of Oracle database fragmentation. Some are harmful and some are not. An active Oracle database will naturally foster harmful database fragmentation unless the DBA takes proactive application design and space management steps. This paper addresses tablespace freespace, segment, data block, index leaf block, and row fragmentation. Each fragmentation type is described in detail along with how to detect the fragmentation, how to quantify its intensity, what its performance and administration affects are, and how to resolve only the harmful fragmentation. Various scripts are used throughout the paper and references are made to publicly available tools.

The most updated version of this paper is available for free at OraPub's web-site, www.orapub.com.

2. Introduction

Whether you like it or not, database fragmentation continues to be of interest to the Oracle community. Maybe it's because it's fun to share what is happening with your database, maybe it's challenging to overcome some of the harmful affects of database fragmentation, maybe it gives DBAs a sense of job security, maybe vendors instill fear to create a demand for their products, or maybe it's just plain fun to look and see how an application is chopping up your database. Whatever the case, the topic of database fragmentation continues to be of interest.

One would think with all the talk, technical papers, vendor products, and Oracle's slow but steady pace towards a "zero fragmented database," fragmentation would no longer be an issue. But the fact is, fragmentation does *not* need to be an issue if the DBA proactively and properly manages their database. Very few applications corner the DBA with regards to harmful fragmentation.

This leaves one with a basic choice; either proactively manage your database or purchase a product to do it for you. Managing fragmentation yourself is not always the best answer. It depends... It depends on how many excess CPU cycles, how much extra disk space, how much slower performance your users can endure, and possibly how much available database down time is available. If you have plenty of these things, then by all means, purchase a product or use the new on-line defragmentation features Oracle is slowly but steadily building into the kernel.

In all seriousness, you do have choice. And you should seriously consider whether it is worth your time to proactively manage harmful fragmentation or let a product/tool do it for you. As you might expect, I believe the answer lies somewhere in between.

With this said, one might be wondering why I'm taking the time to write this paper. Well the fact is, people are interested in database fragmentation and there are many, many sources of partial and misleading information. This is having a harmful impact on databases and wasting DBA's time.

There are many types of Oracle database fragmentation. And DBAs need to know that some are harmful and some are not. As you will come to understand, an active Oracle database will *naturally* foster harmful database fragmentation *unless* the DBA takes proactive application design and space management steps. That's just the way it is and there is no getting around it.

This paper addresses the most important types of database fragmentation. Tablespace freespace, segment, data block, index leaf block, and row fragmentation are covered. Each fragmentation type is described in detail along with how to detect the fragmentation, how to quantify its intensity, what the performance and administration affects are, and how to resolve only the harmful fragmentation.

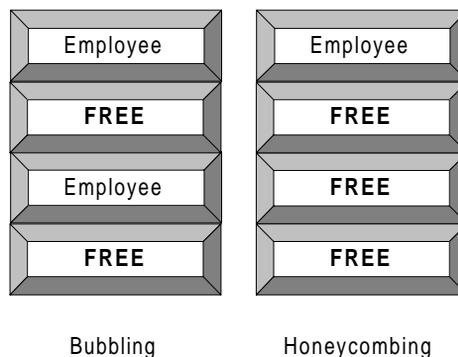
Understanding fragmentation definitions is not enough. One must understand how to easily detect the fragmentation and then quantify its intensity. Without the ability to quantify, one is in a weakened position to determine the appropriate next step. Fragmentation not only affects performance but can also affect administration. By understanding this, resolving problem fragmentation can be properly addressed.

Throughout this paper, I will reference some of my publicly available tools. The tools along with the related *OraPub Internet Video Seminar* are located on OraPub's web-site at <http://www.orapub.com>. The tools can be downloaded free.

3. Tablespace Freespace Fragmentation

When people think of fragmentation, they usually are thinking about *tablespace freespace fragmentation* (TFF). It is one of the easiest fragmentation types to detect and it can be used to create horribly looking reports, charts, and graphs¹. TFF refers to the existence of tablespace level free space *chunks*. These chunks are available for segment growth, which occurs by creating a new extent and placing it within a free space chunk.

There are two fundamental types of TFF; bubbles and honeycombs. *Bubbles* are stand-alone chunks of freespace, whereas *honeycombs* are multiple adjacent bubbles. Think of a beehive honeycomb and you will quickly understand the concept. The figure below is one of ways to visualize TFF.



Starting in Oracle7, the **smon** background process is responsible for coalescing a honeycomb, that is, squishing adjacent bubbles into a single bubble. Various bugs, the algorithm, the speed at which this is done, and how to control the coalescing has changed with various Oracle releases, so I will not take the time to cover this.

3.1 The Cause

Because TFF occurs at the extent level, only operations that reference free extents, that is the **sys.fet\$** table, will be affected by TFF. Think for a moment about which operations reference free extents. Deleting a row(s) affects an object's blocks. Updating a row(s) could possibly, however extremely unlikely, cause the need for a new extent. Inserting a row(s) could possibly cause the need for a new extent. However, if this frequently occurs, known as rapid dynamic extension, you may need to increase your segment's extent size. Selecting a row(s) certainly does not affect TFF because Oracle will only reference segment blocks, not free extents. So DML will typically not cause TFF.

If an object is *created*, the free extent and used extent tables are referenced and changed, but TFF is not the result. However, if an object is *dropped*, its extents become free. That is, a row(s) is deleted from the used extent table and a row(s) is inserted into the free extent table. If the resulting free extents reside next to each other, then a honeycomb will be created (which **smon** may quickly coalesce) or scattered bubbles will result.

In summary, only *dropping* an object will create TFF and usually this can be controlled with solid application design and properly managed storage parameters.

¹ Vendors and bored DBAs love this stuff.

3.2 Detection Methods

Detecting and viewing TFF is very simple and does not significantly affect performance. TFF is detected and observed by looking directly at the *free extent* data dictionary table, **sys.fet\$**. The free extent table contains a single row for each free extent or bubble. By querying **sys.fet\$** one can tell the severity of TFF. By carefully looking at the starting file and block number combined with the free extent's size, one can detect if TFF exists and if the TFF is a bubble or a honeycomb. A fantastic view of TFF can be achieved by combining the free extent table with the *used extent table*, **sys.uet\$**. An example of this is shown directly below.

```

Database: mfg21                                14-DEC-99 09:52am
Report:   tsmap.sql TOOLS                      OSM by OraPub, Inc.      Page      46
                Tablespace Block Map

  Tablespace   File   Block Id   Size           Segment
-----
TOOLS         2     20,211     20 OSMMON.O$PING
              2     20,231    4,936 GL.GL_BALANCES
              2     25,167      5 GL.GLCC
              2     25,172     10 GL.GLCC
              2     25,182      5 <free>
              2     25,187     15 GL.GLCC_S12
              2     25,202     20 GL.GLCC
              2     25,222     45 GL.GLCC
              2     25,267    1,870 <free>
              5         2         20 OSMMON.O$DBA_DATA_FILES
              5        22         20 <free>
              5        42         20 <free>
              5        62         20 <free>
              5        82         20 OSMMON.O$FILESTAT
              5       102         60 <free>
              5       162         20 OSMMON.O$LATCH_U1
              5       182         20 OSMMON.O$LATCH
    
```

Below is report that can be used to quickly identify which tablespace contains TFF. The report is based on the **sys.fet\$**, the **sys.uet\$**, and the **sys.ts\$** (tablespace details) tables.

```

SQL> @tss

Database: mfg21                                14-DEC-99 09:47am
Report:   tss.sql                            OSM by OraPub, Inc.      Page      1
                Oracle Database Tablespace Space Summary

                Total   Free   Used           Biggest           Biggest
                Space  Space  Space   Pct   Data   Num of   Free
Tablespace     Space  Space  Space  Used  Ext   Data   Ext   Num Free
              (MB)  (MB)  (MB)  (MB)  (MB)  Exts  (MB)  Exts
-----
    
```

RBS	2500	300	22	87	0	445	3	1
PAYROLL	1500	1500	0	0	0	0	15	1
SYSTEM	200	20	18	88	0	372	1	8
TEMP	700	700	0	0	0	0	7	1
TOOLS	20	20	0	0	0	0	2	51
USER	73	15	58	80	10	1,412	4	11
	-----	-----				-----	-----	-----
sum	4993	2555	97			2,229		73

3.3 Performance Impact

By understanding how TFF is created, one will quickly realize that only creating an object, an object extending, and dropping an object can possibly be negatively impacted by TFF. The likelihood of a segment rapidly *creating extents* can be resolved with proper storage parameter management. The *dropping issue* can also be reduced with the proper storage parameter management (i.e., creating fewer extents) and eliminated by simply *truncating* the table. The bottom line is, it is extremely rare TFF will negatively impact performance. If performance is being negatively impacted, adjusting the object's storage parameters or simple application adjustments will eliminate the problem.

3.4 Administration Impact

While TFF does not impact performance, it can impact the database from an operational or administrative perspective. For example, if two non-adjacent 100MB bubbles exist and a segment needs to extend and therefore asks for 200MB of contiguous space, it will fail. Remember, an extent requires a *contiguous* chunk of freespace. At the present time, Oracle can not and will not split an extent.

This fact continues to surprise many young DBAs who can not understand why their database has 500MB of freespace yet a 100MB extent can not be created. It is because there is not a single 100MB or larger freespace bubble.

3.5 Resolution

TFF can be resolved from an application perspective or from an administrative perspective. The *application* perspective focuses on setting object storage parameters (e.g., using standard extent sizes) and good code practices (e.g., using **truncate** instead of **drop/create**). The *administrative* perspective focuses on some type of "rebuild."

Rebuilding any part of an Oracle database is undesirable for a number of reasons. Typically, there is associated downtime or non-full application usage, which may make it impossible to rebuild. If storage parameters are set correctly or application coding practices take TFF into account, rebuilding will not be necessary. And don't forget that making a simple typographical mistake can be disastrous.

If a rebuild is necessary, get as granular as possible. That is, do not rebuild the entire database. Focus on just the object(s) or tablespace in need. This reduces downtime, reduces complexity, and reduces risk.

The only way to eliminate TFF is to physically rearrange Oracle segments. This can be done a number of ways, but the basic strategy is to move a segment(s) out from the fragmented tablespace, de-fragment the tablespace, and move the segment(s) back into the tablespace. Unless one is using a commercial tool, this typically requires application downtime.

It is important to give some serious thought into whether or not it is a good decision to rebuild. With the cost of disks continuing to drop in price, the downtime, the risk, and the complexity with rebuilding, it is usually not worth the effort to rebuild.

3.6 Case Study

Fred understands that TFF is very common in today's Oracle databases. However, he also understands that with properly application design and good DBA practices, TFF will be minimal. Fred began his analysis by running a simple query, **select tablespace_name, count(*) from dba_extents group by tablespace_name** to see were TFF

manifests. Fred quickly noticed the SYSTEM tablespace had a tremendous number of free extents. To gather some more clues, he looked closely into the SYSTEM tablespace. This was done by running OraPub's **tsmap.sql** script (sample output was shown in a previous subsection). Observing the report, it was very apparent the SYSTEM tablespace was suffering from severe TFF bubble fragmentation.

At this point, he knew of three objectives. The first was to determine the cause of the TFF, the second was to eliminate TFF from occurring in the future, and the third objective was to eliminate the current bubbles. Fred discovered many users were *sorting* in the SYSTEM tablespace. It turns out a junior DBA did not set the user's *temporary* tablespace and therefore, Oracle created the user's temporary objects to the SYSTEM tablespace. Because users were sorting, massive TFF resulted. Because **smon** coalesces honeycombs, the problem actually looked much better than it really was.

Fred's second objective was to eliminate TFF from occurring in the future. This was simple. He simply ran **alter user <user> temporary tablespace TEMP** for each user and the problem was solved. The third objective was to eliminate the existing TFF. Because the TFF resides in the SYSTEM tablespace, he could not simply rebuild an object or the entire tablespace. Therefore, he will simply let the bubbles remain and as new extents are created, they will slowly fill the bubbles. Fred also kept in mind that he was *not* dealing with a performance issue, rather he was dealing with an administrative issue.

4. Segment Fragmentation

Segment fragmentation (SF) is probably the most misunderstood Oracle fragmentation type. While the misunderstanding has been reduced over the years, I continue to hear that SF hurts performance and that SF is a valid cause for rebuilding an object or an entire database. In general, SF is *not* a performance issue. In fact, SF is a natural and desirable phenomenon in an Oracle database.

SF is the existence of multiple segment extents. When an Oracle segment has more than one extent, it is technically fragmented. If there is two extents or two hundred extents, the segment is still fragmented.

4.1 The Cause

Oracle objects naturally grow by extension. When the current extent fills and more space is needed, a server processes acquires a contiguous chunk of freespace and creates a new extent. On a production database, this process is rare. Think about it... How often do your key application segments extend? Hopefully, not very often or you will quickly run out of space².

4.2 Detection Methods

Detecting segment fragmentation is very simple because all one needs to do is count the number of object extents. The script below is an example of detecting segment fragmentation.

```
col owner format a10
col segment_name format a20
col segment_type format a15

select owner,segment_name,segment_type,count(*)
from dba_extents
group by owner, segment_name ,segment_type;
```

OWNER	SEGMENT_NAME	SEGMENT_TYPE	COUNT(*)
TRADER	ODO_N1234	INDEX	8

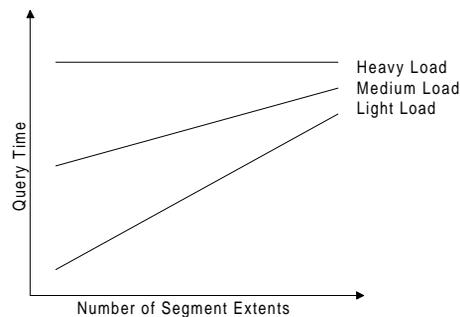
² That is, unless your extent size is something like 32KB, which I have seen before, or some other ridiculous size.

TRADER	ODO_N2	INDEX	6
TRADER	ODO_N3	INDEX	7
TRADER	ODO_N4	INDEX	7
TRADER	ORDERS	TABLE	1
TRADER	STOCK	TABLE	1

All but the ORDERS and the STOCK table have fragmented segments, that is, they have more than one extent.

4.3 Performance Impact

The segment fragmentation performance threat has been debated for years. There have been numerous articles, technical papers, and chapters written about the horrible affects of SF. Nearly all of this published information is *completely false*. Research and clear thinking about real-life production systems show segment fragmentation does not impact real-life production systems performance. *To summarize, as the system load increases, the performance impact of multiple segment extents decreases.*



The issues center around reading and writing for index scans and around full table scans. Writing is not an issue unless *rapid dynamic extension* is occurring. Rapid dynamic extension is a result of poor extent sizing and can be resolved by properly sizing segments and/or application (re)design. So the remaining issues are index reading and full table scan reading.

Index reads is also not an issue. Index reads are nearly always single block reads and therefore the concept of an extent is not relevant. If many contiguous index leaf blocks are read then the characteristic of the leaf block read approaches that of a full table scan. The remaining points address full table scanning and therefore will be relevant in this rare but relevant situation.

Because writing is not a performance issue and index reading and writing is not a performance issue, only the performance issues surrounding full table scanning an object with multiple extents remains. The arguments boil down to three basic points:

Number one. Oracle retrieves data by the block, not by the extent, and not by the segment. When a server process needs data, it asks for it by the file number and block number. A server process never says, "Give me extent number 5." It says, assuming the `db_file_multiblock_readcount` is 8, "Give me file number 6, block numbers 10, 11, 12, 13, 14, 15, 16, and 17."

Number two. Only production-like systems are the issue. Do not be fooled by arguments that are valid only with development and meaningless experimental databases. We are grappling with large OLTP, Web, and DW application objects. Let others argue about the performance issues on their laptop. The rest of us have real issues to deal with.

Number three. Oracle's reads (single or multi block) compete with other I/O requests for I/O. When an Oracle server process asks the operating system for I/O, the operating system places the I/O request in the queue along with all the other I/O requests. The operating system does not say, "Hey everybody, this guy's I/O requests are here, so everyone else get out of the way!"³ On laptops or a single user system, this may be the case, but not in the real world.

There are other issues. For example, recursive SQL is used while working with multiple extents and the dictionary cache may be larger because there is more extent information to cache. But these types of arguments are bogus when dealing with today's large production systems.

At this point, I suspect many of you reading this are saying to yourself, "I agree with what you are saying, but I know performance increased when I rebuilt an object into a single extent." Your observations are correct. However, the performance increase you observed was not the result of eliminating SF. It was the elimination of row fragmentation, the elimination of block fragmentation, and the resetting of the high-water mark. Both row and block fragmentation are discussed later in this paper.

In summary, on an active production system, segment fragmentation does not impact performance. One would have a very difficult time constructing a valid case to justify rebuilding an object, multiple objects, or an entire database because of segment fragmentation. It could be a career decision.

4.4 Administration Impact

Which would you rather park, twenty small cars or one two-hundred meter long truck? Have you ever seen a train with one or two one-thousand meter cars? These examples are absurd, but then so is placing all of a segment's data into a single extent. It is much easier to administer multiple smaller objects or smaller object extents, than huge objects and huge extents. The key issue is when a segment extends, it needs a *single* chunk of contiguous space. *The chances of finding a contiguous chunk of freespace decreases as the requested size increases.* So, the smaller the segment extent, which will force more segment extents, the smaller the administrative impact.

4.5 Resolution

Let's say, in a highly unlikely and career challenging case, SF was causing performance and administration issues. As with most solutions, there is the quick and radical solution and a natural solution. Remember, one is not just interested in solving current problems, but in preventing the problems from reoccurring in the future. Therefore, when an object is recreated, reset the storage parameters to reduce the likelihood of future SF. If the issues are so intense they override downtime issues and rebuild risk, then the object can be rebuilt. Another less "in your face" solution is to reset the segment storage parameters to reduce the increase of future segment fragmentation. Most DBAs will choose the storage parameter resetting option.

4.6 Case Study

Sara, the Senior DBA, understands that while unlikely, segment fragmentation can become a real performance issue during batch processing row insertion. What occurs is called, *rapid dynamic extension*. This is, when rows are inserted, the segment extends, rows are inserted, the segment extends, and repeat. The database freespace time involved as a percentage of the total operation can become significant.

One of the batch processes Sara was responsible for took 30 minutes to complete. By **tracing** and **tkprof**ing the batch processes, she discovered Oracle freespace manipulation required 10 minutes and row insertion required 20 minutes. So 30% of the processing time was purely database overhead.

³ Don't I wish this was the case. This is related to the Oracle *instance parameter* FAST. Setting FAST to TRUE can provide significant performance improvements.

Because Sara new about the application specifics, she was able to modify the *interim segment*⁴ storage parameters. (For example, the Oracle Applications allow one to update a table, which contains the storage parameter values for interim objects. When the objects are created, the application references this table and uses the values in the actual *create statement*. If a custom application is involved, simply find the *create SQL* and modify the code. The objective is to minimize rapid dynamic extension, by having larger extents, while keeping the extents small enough to reduce the chance of not finding a contiguous chunk of freespace and not wasting space.)

Sara searched through the source code and found the create statements. To her disdain, the segment extent sizes where set to 16KB. After some quick math, she reset the **initial** and the **next** storage parameters to 50MB and the **pctincrease** to 100.

After the changes were tested and placed into production, Sara observed the performance results and reported them to her management. The batch processing time was reduced by nearly 30%. Sara was rewarded for her outstanding work by being given the additional responsibility of five more databases to manage...she quit six months later after gaining 120 lbs. by eating pizza and fast food for 157 out of 174 days⁵.

5. Data Block Fragmentation

The result of *data block fragmentation* (DBF) looks like a pile of "pick up sticks" near the end of the game. There are plenty of sticks but there are also plenty of space where the sticks used to reside. Just as when a player removes a stick, which fragments the pile, when a server process removes a row from a table, it fragments the row's block. Obviously, space is wasted unless another row is inserted into the row's block.

When an Oracle table data block becomes fragmented, performance is negatively impacted, especially if the database system processes terabytes of data each day. Unfortunately, DBF is typically overlooked or simply ignored because it is relatively difficult to detect and even more difficult to quantify. This is a shame because while system performance is decreased, the fragmentation resolution is very simple and typically does not distrust application usage.

5.1 The Cause

Data block fragmentation occurs whenever a row is deleted from a table and therefore removed from a data block. Some objects are, by their nature, very susceptible to DBF, while other segments are not very susceptible. Think for a moment about the application objects you are administering. Which objects frequently have rows deleted? Which objects never have rows deleted?

5.2 Detection Methods

Detecting DBF is relatively difficult because it is difficult to determine, in each block, exactly how much space is free, how much space is occupied by rows, and how much space is overhead. Fortunately, one does not need an exact number and there are ways to *estimate* DBF.

Data block fragmentation can be quantified at three basic granularity levels. The first is at the block level, the second is at the byte level, and the third is also at the byte level but focuses on *each* block instead of the overall average. Keep in mind that it is not necessary to quantify block fragmentation for each table. Only quantify DBF for tables you suspect may have DBF issues.

Regardless of how one quantifies DBF, the same basic formula remains. That is, actual row space occupied divided by the empty block row space. For example, if there is 8192 bytes of row space (8KB data block size), but only 800 bytes were being used, 90% (1-800/8192) of the space is unused. This represents, on average, very fragmented blocks.

⁴ Temporary segments are special segments Oracle stores sorting related information. Production segments store production data. Interim segments exist somewhere in between. They are niether production segment-like or temporary segment-like. So we give them an appropriate category, *interim*, and have them stored in a corresponding tablespace, *interim*.

⁵ If you are interested, that's 90.2% of the time.

All About Oracle Database Fragmentation

DBF detection at the *block level* is relatively simple and gathering the information does not place a substantial burden on the system. Basically, one counts the number of distinct blocks that contain one or more rows and divides this by the number of blocks below the high-water mark. For example, if a table contains 5000 blocks below the high-water mark, yet only 2500 of those blocks contain one or more rows, the DBF is 50%.

If one thinks about this detection method for a while, they will realize that there is a flaw with this method. A block is deemed *full* regardless of whether it contains 1 row or 100 rows. So this measurement of DBF will tend to produce a more favorable DBF value. That is, the result will show that DBF is not nearly as bad (i.e., there is less empty space in each block) as it really is. So the block method will return a best case scenario.

Another DBF detection method works at the *byte level*. However, this method requires the table be *analyzed* and then full table scanned. The strategy focuses on the average row size. Typically, one takes the average row size times the number of table rows, and then divides this by the number of blocks multiplied by the database block size. The returned value is a fairly good estimate of average block fragmentation.

But for a real low level look, which would produce a very nice graphic, one could calculate the DBF for each individual block and systematically rebuild each block (discussed below). Just query a table summing each row's rowsize for each block and divide this number by the empty block space.

5.3 Performance Impact

As mentioned above, DBF definitely reduces both query performance and to a lesser extent DML performance. Performance is impacted because the higher the DBF, the less row data retrieved per block access. That is the overhead per row retrieved is higher. Keep in mind there is overhead regardless of whether the block resides on disk or in the data block buffer cache.

Picture yourself scanning the cheese department at your favorite food store. If maximizing the cheese poundage per block is your goal, you would grab cheese without holes. Assuming the cheese blocks have the same physical dimensions, if you wanted to eat ten pounds of cheese⁶, the job would be completed quicker and with less effort if you grabbed and ate cheddar cheese instead of Swiss cheese⁷. Grabbing and eating Swiss cheese forces you to perform more work for the same amount of cheese.

An Oracle server process works much the same way. It makes more sense to have a server process retrieve 100 data blocks containing 600 rows instead of 150 data blocks containing the same 600 rows. Since Oracle retrieves data by the data block and not by the row, the more *tightly packed* a data block, the more "bang for the I/O" you receive during retrieval.

5.4 Administration Impact

From an administration perspective, DBF wastes space. Disk space is relatively cheap these days, so this typically is not an issue. However, with large DW applications this can be a real issue. Saving an average of 50 bytes in a 900,000,000 row table is significant. Quantifying DBF can help determine how much space is being wasted and add valuable information in determining any necessary defragmentation steps.

5.5 Resolution

While the performance impact of DBF can be significant, resolving DBF can be relatively simple. Before one can make a sound judgement about resolving DBF, one needs to detect and quantify its intensity. Then one must combine this information with up-time requirements, complexity, the intensity of the problem, and the benefits gained, to determine what resolution strategy is the most appropriate.

⁶ If you know anyone who can eat ten pounds of cheese *in one sitting*, please email a digital picture of the actual act of eating and I'll place it on OraPub's web site.

⁷ In addition to retrieval efficiency, cheddar tastes better than Swiss cheese.

There are two basic resolution methods. The first method is to alter the table's storage parameters allowing the blocks to naturally "fill up the holes" in each block. The second method is to rebuild the object. Rebuilding the object is rarely the best solution, especially with large production objects (which are the objects we are typically looking at anyway).

One could write a procedure, which would make a list of all the substantially fragmented blocks and defragment just the highly fragmented blocks. Block by block, the procedure would copy the fragmented block's rows somewhere else, then the rows would be deleted (emptying out the block), then re-insert the rows. This type of procedure could be written to *slowly* and deliberately scan all database blocks that "need" to be rebuilt.

By decreasing the **pctfree** parameter, each table block will accept more rows than in the past. For example, instead of the average number of rows per block being 20, by decreasing **pctfree**, the average number of rows per block may increase to 25. By increasing the **pctused** parameter, a block will become *more attractive* and remain on the free list longer thereby increasing the chance of attracting rows for insertion.

5.6 Case Study

Carl loved math, but somehow gained employment with the responsibility of managing *the* company database. Carl had just learned the truth about database fragmentation and suspected *the key* application table was suffering from (gasp here) data block fragmentation. He discovered the large 1.7GB table (106,497 16KB blocks) gets both full table scanned and index scanned throughout the day, and during evening batch processing additional rows are inserted. Carl new full well that if this table had significant DBF, because of its size and processing requirements, general system performance would be impacted.

Carl's block level analysis began by determining the total space allocated *below* the high-water mark. After *analyzing* the table, and querying the **blocks** column, he determined there were 97,292 blocks below the high-water mark. By running a basic query counting the distinct blocks that contain at least one row, he discovered 96,474 blocks contain at least one row. Therefore, 99% of the blocks contained at least one row. So from a block level perspective, DBF was not significant.

After *analyzing* the table, Oracle showed (via **dba_tables**) the average row size for each of the 18,858,065 rows is 71 bytes. This meant there was an average of 1,338,922,615 bytes of data in the table. Since there were 1,594,032,128 bytes (97,292 blocks * 16KB) below the high water mark, an average of 84% of the available space (i.e., space below the high-water mark) was being used for rows.

This is how Carl summarized this experience to management. "My block level analysis indicated 99% of the available blocks contained at least one row and my byte level analysis indicated that 84% of available block space is being used. Taking into consideration block overhead combined with the reserved 10% for row growth (i.e., **pctfree** equals 10%), these values are very good, that is, byte level block fragmentation is not significant. This is what I expected, because this table, to my knowledge, only has rows inserted. The rows are never deleted and never updated." Carl was immediately promoted to senior management...nine months later the company filed bankruptcy.

6. Index Leaf Block Fragmentation

Most people don't think about *index leaf block fragmentation* (ILBF), but it can have a significant affect on performance in the same way table block fragmentation does. And because it is *very* difficult to accurately quantify, most people just ignore ILBF and hope it is not causing problems. Or the other extreme occurs when people over react and schedule periodic index rebuilds. To have a solid and balanced understanding of ILBF, one must know what causes it, how to detect it, what the performance and administration issues are, and how to resolve the fragmentation.

6.1 The Cause

Oracle's standard index structure is the B*-Tree. While the detailed structure is out of scope for this paper, how ILBF manifests and fundamental B*-tree structural knowledge is important to understand.

- The B*-tree structure is different from the binary tree structure and the B-tree structure.
- A B*-tree is made up of three types of blocks or nodes:

- **Root node.** There is one root node. All index access starts at the root node. The root node has many child nodes.
- **Branch blocks.** There can be many branch blocks. A single branch block can have many children (typically more than 100) and branch blocks are arranged in multiple levels (usually less than four). Regardless of deleted rows, branch blocks are not removed. Therefore, it is possible to have empty branch blocks.
- **Leaf blocks.** The bottom layer of a B*-tree index is made entirely of leaf blocks. Each leaf block contains the actual column index entries and their associated ROWIDs (i.e., file number, block number, and row number). When a table row is deleted, all associated index entries are also deleted. This will fragmented the index leaf blocks. If a table row is updated the leaf block entry is not changed *unless* the indexed column(s) changes. If a row is inserted, the appropriate index leaf block insertion is performed. Only row deletes causes index leaf block fragmentation. Regardless of deleted rows, leaf blocks are not removed. Therefore, it is possible to have partially or completely empty leaf blocks...index leaf block fragmentation.
- The same number of "hops," that is, moving down through the index tree, is always the same. By definition and structure, it is impossible for the number of hops to be different when hopping from the root node down to any leaf block.

6.2 Detection Methods

Accurately detecting and quantifying ILBF is *extremely* difficult because of the tools and statistics Oracle makes available. The best we can do is calculate the average number of leaf block entries per row and the average size of leaf block entries, compare this over time, and observe the change. If the average leaf block entry size increases (it doesn't actually increase, but the statistics will lead us to believe this) or the average number of entries per leaf block drops, we know ILBF exists and is increasing.

We can gather many the index statistics by simply performing an **analyze index <owner.index_name> validate structure**. The index structure and some other related details are available by querying the **index_stats** view. For our purposes, the pertinent column is the number of leaf blocks, **lf_blks**. Since we know the number of table rows (based upon *analyzing* the table and querying **dba_tables**), we can easily calculate the rows per leaf block and the average leaf block entry size. An example of this is presented in the *Case Study* below.

Unlike the other fragmentation detection methods, our ILBF calculation does not actually quantify fragmentation. However, when the calculations are periodically performed, they can be compared over time. If the numbers change, we then know the characteristics of the leaf blocks have changed. Based upon the *value change*, we can determine the extent and direction (i.e., increasing or decreasing) of ILBF.

6.3 Performance Impact

The performance impact of ILBF only affects index scans and can be as severe as table block fragmentation. How performance is affected, is just like table block fragmentation. So for details, please refer to the *Table Block Fragmentation, Performance Impact* subsection.

6.4 Administration Impact

Other than wasted space, there is no administrative impact with ILBF.

6.5 Resolution

Oracle has made resolving ILBF much easier and less obtrusive than in the past. ILBF can be naturally resolved⁸ by simply changing the storage parameters or the index can be physically rebuilt while the application is available.

⁸ When naturally resolved, ILBF resolution is typically not as thorough as table block fragmentation because a row's index leaf block entry is, in a sense, "assigned" to a leaf block based upon the indexed columns.

Indexes can be built in an *offline* mode and then placed *online*. Unlike tables (unless you have a super-slick script or vendor product), this allows fragmented indexes to be rebuilt without any downtime. However, while there is no downtime, there is an overall system performance cost while the index is being rebuilt. Rebuilding an index takes a substantial amount of CPU and disk I/O. And if the index is rather large, the physical space used when rebuilding the index (which is in the user's default tablespace) can also be rather substantial. So don't be fooled by "on-line reorg" type tools or facilities, there is always a cost.

Just as with tables blocks, index leaf blocks utilize the **pctfree** storage parameter. This can be *decreased* to allow more index entries to be placed into each leaf block, decreasing overall ILBF.

6.6 Case Study

Bill was a sidewalk painter who went back to school to "learn about computers and make lots of money." After interviewing with a company's human resources interview expert, Bill was hired as a junior DBA. After he attended an *Introduction to Oracle* course (not to be confused with OraPub's *Advanced Performance Management* or *Advanced Database Administration* courses), he was given responsibility for a web based stock trading application. This was Bill's dream come true. Not only did he "learn about computers," but he got to watch others "makes lots of money." He was sure his turn was next.

After learning about index leaf block fragmentation (ILBF), Bill suspected one of his key table indexes might be suffering. The related table had 18,619,345 rows and Bill discovered (purely by chance) it was being both full table scanned (during nightly batch processing) and index scanned (during daily OLTP). He determined the index consumed 131,690 16KB blocks, of which there is one root block, 432 branch blocks, and 131,257 leaf blocks. Based upon this information, he calculated there was an average of 142 rows per leaf block (18,619,345 rows / 131,258 leaf blocks) and each leaf block entry (16KB database block) was on average 115 bytes ((131,257 leaf blocks * 16KB) / 18,619,345 rows).

Bill new this information by itself was not fragmentation related useful. What was useful, is two weeks later the data was gathered again and compared. He discovered that while the number of index leaf blocks remained at 131,257 the number of table rows decreased to 15,000,462. He did the math (took him a while though) and found there was an average of 114 rows per block (15,000,462 rows / 131,257 leaf blocks) and each leaf block entry was on average 143 bytes ((131,257 leaf blocks * 16KB) / 15,000,462 rows). The drop in rows per leaf block and an increase in the average leaf block entry size indicated there was more unused space in each leaf block, that is, on average each leaf block is more fragmented. If the trend continued, Bill knew the index would have to be rebuilt. In the mean time, to reduce the likelihood of continued ILBF, Bill decreased the **pctfree** storage parameter.

Bill convinced his management an index rebuild was required. Unfortunately, Bill was not aware of Oracle's on-line index rebuild. When Bill dropped the index at 9am, I/O activity became so severe, all work virtually came to a stand-still. Determined to complete the job, Bill started the index rebuild. Being that the Oracle DBA job market is so hot, just about the time the index rebuild completed, Bill had secured new employment. What Bill couldn't understand was that his human resources department actually helped him get the new job. Bill also didn't catch that he was working for his old company's top competitor.

7. Row Fragmentation

Row fragmentation (RF) or more commonly known as *row chaining*, was virtually ignored just a few years ago. RF usually manifests in a few well-known application tables. With early detection, any performance impact, which can be significant, can be averted. RF is easy to detect, easy to prevent, and somewhat more difficult to resolve. And in many cases, the current RF can be eliminated while the application is available.

7.1 The Cause

Row fragmentation can only be caused by an *update* statement that increases row length. Inserts, deletes, and queries can not cause row fragmentation.⁹ When an updated row grows resulting in an increase in its row size and if

⁹ There is one exception. If a row during insert is larger than a block, Oracle will break up the row into "row pieces" and store them in different blocks. The solution, which is usually not justified, is to increase the database block size.

All About Oracle Database Fragmentation

the entire row can not be stored in its current data block, Oracle moves or *migrates* the *entire* row into a new block. However, all references to the row continue to point to its original home.¹⁰ As a result, each time the block is physically read from disk, an additional physical I/O is required. And each time the block is referenced in the block buffer cache, an additional I/O (logical or physical) is also required.

7.2 Detection Methods

Oracle makes detailed row fragmentation detection very easy. Simply **analyze** the table and query the **chain_cnt** column from the **user_tables** view. There is also an option, when analyzing a table, to store information about the chained rows (e.g., the ROWID) for later analysis. For a RF system wide perspective and impact analysis, as shown below, simply query the **v\$sysstat** table. The output below shows the number of fragmented rows the Oracle instance has touched since startup.

```
SQL> select value
  2  from v$sysstat
  3  where name = 'table fetch continued row';

      VALUE
-----
      8889
1 row selected.
```

The script below demonstrates how to gather table specific, row fragmentation details. The output below shows the GL_BALANCES table currently contains 10 fragmented rows.

```
SQL> analyze table gl_balances compute statistics;
Table analyzed.
SQL> Select chain_cnt
  2  From user_tables
  3  Where table_name = 'GL_BALANCES';

CHAIN_CNT
-----
       10
1 row selected.
```

To gather RF details for a specific operation run the following query *just before* and *just after* the operation. Then simply subtract the values. The result will be the number of fragmented rows touched during the operation.

```
SQL> Select my.value
  2  From v$mystat my,
  3  v$sysstat sys
  4  Where my.statistic# = sys.statistic#
  5  And sys.name = 'table fetch continued row';
```

¹⁰ Otherwise, all index references would have to be adjusted. Oracle decided it was better to leave a "forwarding address" than to locate and change all row references.

```
-----
1
```

```
1 row selected.
```

7.3 Performance Impact

The performance affect of RF is arguably the worst of any Oracle database fragmentation type. This is because for each fragmented row, twice as many physical or logical I/O's are required. In a large and active OLTP or DW application, this can add up to millions or billions of extra I/O's each day!

Keep in mind, RF will affect both the I/O subsystem and the CPU subsystem. When additional physical I/O's are read, the Oracle server process must ask the I/O subsystem for additional I/O. And when the buffers are already in Oracle's buffer cache, the Oracle server process must touch two blocks instead of just one. Block buffer manipulation takes CPU time and forces latches to be held longer than necessary. As a result, RF can impact a busy OLTP or DW production application in a variety of ways.

Because RF details are available from a variety of sources, one can *quantify* the performance impact. As mentioned above, for each fragmented row access, there is an additional I/O. So when the Oracle statistics show a fragmented row was touched, you can be sure either an additional physical or logical I/O occurred. Just add up the numbers and you can quantify the affect.

It is important to quantify the performance impact, because eliminating RF *may* require the *cost* of application downtime. As a result, one needs to understand the *benefit* of resolving RF and must determine if it is greater than the *cost*. The *Case Study* below shows how one DBA quantified RF cost.

7.4 Administration Impact

From an administration perspective, RF has virtually no affect. Significant additional space is not required because even if a row is fragmented, there remains only a single copy of the row. There is extra space required for the original row pointer, but that is relatively insignificant.

7.5 Resolution

Before a RF resolution strategy is discussed, make sure both the performance impact cost and the resolution cost is well understood. If the row rebuild process is been automated, the resolution cost, even in a large production system, can be minimal.

With this said, other than commercial products, there are two basic RF resolution methods or tools. The preventative method is to increase the **pctfree** storage parameter. This will force Oracle to not insert as many rows into each block, thereby allowing more room for rows to expand during an update. The other method, which is usually used in combination with resetting **pctfree**, is to either rebuild only the fragmented rows or rebuild the entire table.

Regardless of the rebuild option, properly setting **pctfree** is key to preventing RF from manifesting. When designing application objects, be aware of tables which will be updated and have row growth potential. If this is the case, set **pctfree** to an abnormally high value. You can always lower the value if necessary. And lowering **pctfree** is less impactful in a system, which is what you want, than rebuilding fragmented rows and later increasing **pctfree**.

Rebuilding anything in a large production database is harsh. Therefore, to reduce downtime and risk, always rebuild at the lowest possible level of granularity. For this discussion, the lowest level is a fragmented row. Remember, the **analyze** command has an option to store information about the fragmented rows. This gives one the opportunity to copy the rows to an interim table, delete the original rows, and re-insert the copied and un-fragmented rows. If automated, the entire operation can take just a few seconds. The complications arise when database constraints are involved. But, by locking the table and disabling the constraints before the actual row rebuild, this problem can be overcome.

Application design is also important in preventing RF. For example, breaking basic relational database design techniques such as using *repeating groups* is a recipe for intense RF. Think about it. What if, when the row is inserted, only the first repeating group is filled. During the next batch process, the next repeating group is filled. If this continues, the row will grow to a point where there is not enough room for the row to remain in its *home* block.

It will have to moved, resulting in a fragmented row. One way to minimize this occurrence is to set **pctfree** extremely high.

7.6 Case Study

Joe is a DBA and has been monitoring RF activity via the **v\$sysstat** performance view. He noticed the **v\$sysstat** view indicated during daily OLTP processing, an average of 20 fragmented rows where being touched each second. Because of the multitude of huge production tables, Joe could not simply **analyze** all production tables to find which table(s) contained fragmented rows. Joe had to think. By understanding that RF is created by updating rows, Joe narrowed the search to only two tables: A 5 million row table and a 2 million row table. By monitoring which tables were being accessed (e.g, **v\$session**, **v\$sqlrea**, **v\$sqltext**) combined with **v\$sysstat** statistics (e.g., **where name = 'table fetch continued row'**), Joe was able to determine the CUSTOMER table was responsible for 95% of all fragmented row accesses.

Joe did some quick checks and found the CUSTOMER table contains 5,000,000 rows, has **pctfree** set to 20, is frequently queried during daily OLTP processing and during evening batch processing, the table is updated and new rows are inserted. When Joe **described** the CUSTOMER table, he found a slew of repeating groups, which store historical information about quarterly customer activity. Instead of having a CUSTOMER_ACTIVITY table with a foreign key from the CUSTOMER table, customer activity is stored as repeating groups in the CUSTOMER table. Every month, each customer row is updated. If the customer has activity, the row will grow. Over time, this has manifested into RF.

By interviewing some application designers, Joe discovered the **pctfree** value of 20 was set five years earlier during the initial data load to maximize "wasted space." Joe immediately changed **pctfree** to 60 to allow for more room in each block thereby allowing for row growth. He also set **pctused** to 40 percent to keep blocks off the free list thereby discouraging row insertion. If these settings were found to be extreme, they could easily be changed at a later date. Immediately, Joe noticed RF stopped increasing, but he still had to deal with the existing fragmentation.

Joe developed a two phase plan to justify to himself and his management the need to **analyze** the CUSTOMER table and then de-fragment the rows. The first phase was to quantify the fragmented rows performance affect, estimate how long it would take to defragment any fragmented rows, and show how the de-fragmentation process would not possess a significant risk to the database system. The second phase was to physically de-fragment the fragmented rows.

Quantifying the RF impact was rather simple. Joe new during OLTP processing around 20 fragmented rows were touched a second and around 95% of the those fragmented rows where from the CUSTOMER table. He also knew OLTP processing begins at 0830 and ends at 1730, Monday through Friday (a 9 hour OLTP processing window). Performing some simple math¹¹ convinced Joe RF caused an additional 615,600 physical I/O's and 615,600 logical I/O's during OLTP processing each day. By querying **v\$sysstat** Joe new around 400,000 blocks are touched in the block buffer cache each hour. He therefore concluded that each day around 1.5 hours (615,600/400,000) of unnecessary block buffer activity was occurring within the 9 hour OLTP processing window. Therefore, by eliminating the fragmented rows, an additional 17% (1.5 hours / 9 hours) of real work could be processed. And since the system had 10 CPUs in it, Joe concluded that eliminating the fragmented rows would have a similar affect of purchasing nearly two additional CPUs.

A similar analysis was done on the I/O subsystem and presented to management. "Getting two additional CPUs" by simply de-fragmenting the table was enough to convince Joe and his management the benefits were greater then the row de-fragmentation cost.

Joe wrote a script to automatically de-fragment the table. It basically **analyzed** the table with the option to store fragmented row details. It then locked the table in exclusive mode, disabled the constraints, inserted the fragmented rows into another table, deleted the fragmented rows, re-inserted the fragmented rows, enabled the constraints, and released the exclusive lock. It turns out the **analyze** took nearly two hours, but the actual de-fragmentation, when the CUSTOMER table was exclusively locked, took only 5 minutes.

¹¹ How many I/O's = 9 OLTP hours * 3600 sec/hour * (95% * 20 I/O's / sec) = 615,600 I/O's

Management was so impressed with the Joe's analysis, the de-fragmentation execution, and the resulting performance increase, he was given the latest full-featured cell phone. This of course, only further encroached work into Joe's basically pathetic personal life. Six months later he crushed the cell phone between his bare (and shaking) hands.

8. Conclusion

This paper was written for those who really want to understand what Oracle database fragmentation is all about. My hope is that the next time someone suggests you rebuild your one tera-byte table because it has 50 extents, you ask them why. And in their feeble attempt to justify shutting down the entire application for two days which will result in no performance increase, you will be able to bring them, however painful it may be, into a better understanding of the real issues and how to properly deal with them. And if you want a more hands-on experience, check out *OraPub's Internet Video Seminar*, specifically about Oracle database fragmentation. Thank you for your time.

9. References

- Jain, R. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 1991. ISBN 0-471-50336-3
- Levin, R.; Kirkpatrick C.; Rubin, D. *Quantitative Approaches to Management*. McGraw-Hill Book Company, 1982. ISBN 0-07-037436-8
- Menascé, D.; Almeida, V.; Dowdy, L. *Capacity Planning and Performance Modeling*. PTR Prentice Hall, Englewood Cliffs NJ, 1994. ISBN 0-13-035494-5
- Shallahamer, C. *Avoiding A Database Reorganization*. Oracle Corporation White Paper, 1995. <http://www.orapub.com>
- Shallahamer, C. *Predicting Computing System Throughput and Capacity*. Oracle Corporation White Paper, 1995. <http://www.orapub.com>
- Shallahamer, C. *Total Performance Management*. Oracle Corporation White Paper, 1994. <http://www.orapub.com>

10. Related References and Resources

- Chatterjee, S.; Price, B. *Regression Analysis by Example*. John Wiley & Sons, 1991. ISBN 0-471-88479-0
- Cook, D., Dudar, M., Shallahamer, C. *The Ratio Modeling Technique*. Oracle Corporation White Paper, 1997. <http://www.orapub.com>
- Millsap, C. *Designing Your System To Meet Your Requirements*. Oracle Corporation White Paper, 1995. <http://www.orapub.com>
- Michalko, M. *Thinkertoys*. Ten Speed Press, 1991. ISBN 0-89815-408-1
- Rubinstein, M., Firstenberg, I. *Patterns Of Problem Solving*. Prentice Hall, 1995. ISBN 0-13-122706-8
- Saksena, Virag. *Identifying Resource Intensive SQL In A Production Environment*. Oracle Corporation White Paper, 1996. <http://www.orapub.com>
- Saksena, Virag. *Tuning The Oracle Server - Identifying Internal Contention*. Oracle Corporation White Paper, 1996. <http://www.orapub.com>
- Shallahamer, C. *Direct Contention Identification Using Oracle's Session Wait Views*. OraPub White Paper, 1999. <http://www.orapub.com>
- Shallahamer, C. *Optimizing Oracle Server Performance In A Web/Three-Tier Environment*. OraPub White Paper, 1999. <http://www.orapub.com>
- Shallahamer, C. *All About Oracle Database Fragmentation*. **OraPub Internet Video Seminar**, 2000 (est.). <http://www.orapub.com>
- Shallahamer, C. *Direct Contention Identification Using Oracle's Session Wait Views*. **OraPub Internet Video Seminar**, 1999. <http://www.orapub.com>

All About Oracle Database Fragmentation

Shallahamer, C. *Holistic Problem Isolation Method*. **OraPub Internet Video Seminar**, 2000
<http://www.orapub.com>

Shallahamer, C. *UNIX Operating System Bottleneck Detection*. **OraPub Internet Video Seminar**, 1999.
<http://www.orapub.com>

Shallahamer, C. *Optimizing FSG Performance*. **OraPub Internet Video Seminar**, 1999. <http://www.orapub.com>

Shallahamer, C. *Optimizing Quick Pick Performance*. **OraPub Internet Video Seminar**, 1999.
<http://www.orapub.com>

11. About the Author

Mr. Shallahamer's thirteen-plus years of experience in the IT marketplace brings a unique balance of controlled creativity to any person, team, or classroom. As the President of OraPub, Inc., his objective is to empower Oracle performance specialists and capacity planners. Recently, Mr. Shallahamer directed the global technical training efforts for Oracle Consulting's technical consultants. Previously he managed the Western area of Oracle Services System Performance Group. Since joining Oracle in 1989, Mr. Shallahamer has co-founded three highly respected technical consulting groups (National Product Specialist Team, Core Technologies, System Performance Group) and has worked at hundreds of client sites around the world. His specializations include business management, training/education, performance optimization and management, performance prediction/modeling/planning, and system/technical architecture design related research, training, coaching, and consulting. As a result, Mr. Shallahamer has had the pleasure to publish and present a number of papers at the EOUG-E/ME, OAUG-A/E, IOUG-A, OpenWorld, and in *Oracle Magazine*.

12. Fragmentation Scripts

The following two scripts come with absolutely no warranty and you may use them at your own risk. The scripts along with many others, are available for free at OraPub's web-site, <http://www.orapub.com>. The first script, **fgtbl.sql**, gathers and displays *table* fragmentation details, while the second script, **fgidx.sql**, gathers and displays *index* fragmentation details.

```
-- *****
-- * Copyright Notice   : (c)1999 OraPub, Inc.
-- * Filename          : fgtbl.sql
-- * Author            : Craig A. Shallahamer
-- * Original          : 10-nov-99
-- * Last Update       : 15-nov-99
-- * Description       : Get TABLE fragmentation details for a given object.
-- * Usage             : start fgtbl.sql <owner> <table name>
-- * Details           : There needs to be an index with the leading
-- *                   column a VARCHAR2 for this script to work.
-- *****

def owner=&&1
def tnm=&&2

set echo off verify off heading off

set termout on
prompt Analyzing table &owner..&tnm COMPUTING statistics...
```

```
set termout off
--prompt analyze currently turned OFF!!!!!!!!!!!!!!
analyze table &owner..&tnm compute statistics;

set termout on
prompt Gathering raw data 1 of 4. Should be very quick...
set termout off

--set termout on
--set verify on feedback on echo on

col val1 new_val cnm noprint format a20
col val2 new_val inm noprint format a20
col x noprint
select distinct(a.index_name) val2, a.column_position x, a.column_name val1
from dba_ind_columns a, dba_tab_columns b
where b.owner = a.table_owner
      and b.table_name = a.table_name
      and b.data_type = 'VARCHAR2'
      and a.table_owner = upper('&owner')
      and a.table_name = upper('&tnm')
      and a.column_position = 1;

select 'Index: &inm Column: &cnm' "." from dual;

set termout on
prompt Gathering raw data 2 of 4. Will take a while for big table...
set termout off
set verify off feedback off echo off

--set termout on
--set verify on feedback on echo on

col val3 new_val blks_w_data noprint
select /* +index(&inm) */
       count(distinct(dbms_rowid.rowid_relative_fno(rowid)||'.'||
                      dbms_rowid.rowid_block_number(rowid))) val3
from &owner..&tnm
where &cnm > '-9999999';

set termout on
prompt Gathering raw data 3 of 4. Should take less than 30 seconds...
set termout off
```

All About Oracle Database Fragmentation

```
--set termout on
--set echo on feedback on verify on

col val4 new_val hwm noprint
col val5 new_val above_hwm noprint
col val6 new_val row_chains noprint
col val7 new_val row_size noprint
col val7a new_val pct_used noprint
col val7b new_val pct_free noprint
col val8 new_val num_rows noprint

select  num_rows      val8,
        blocks        val4,
        empty_blocks  val5,
        chain_cnt     val6,
        avg_row_len   val7,
        pct_used      val7a,
        pct_free      val7b
from    dba_tables
where   table_name = upper('&tnm')
and     owner       = upper('&owner');

set termout on
prompt Gathering raw data 4 of 4...
set termout off

col val9 new_val block_size noprint
select value val9
from   v$parameter
where  name = 'db_block_size';

set termout on
prompt Calculating fragmentation statistics
set termout off

--set termout on
--set verify on feedback on echo on

col val10a new_val raw_blocks noprint
col val10b new_val bytes noprint
col val10c new_val blks_wo_data noprint
col val10d new_val blks_wo_data_hwm noprint
col val10e new_val bytes_hwm noprint
col val10f new_val bytes_used_hwm noprint
```

```

select (&hwm+&above_hwm) val10a,
       (&hwm*&above_hwm)*&block_size val10b,
       (&hwm-&blks_w_data+&above_hwm) val10c,
       (&hwm-&blks_w_data) val10d,
       (&hwm*&block_size) val10e,
       (&num_rows*row_size) val10f
from   dual;

col val10a new_val blocks_pct_used noprint
col val10b new_val bytes_pct_used noprint
col val10c new_val blocks_pct_used_hwm noprint
col val10d new_val bytes_pct_used_hwm noprint
col val10e new_val bytes_not_used noprint
col val10f new_val bytes_not_used_hwm noprint
select (&blks_w_data)/(&hwm+&above_hwm) val11a,
       (&num_rows*row_size)/((&hwm+&above_hwm)*&block_size) val11b,
       (&blks_w_data)/(&hwm) val11c,
       (&num_rows*row_size)/((&hwm)*&block_size) val11d,
       (&bytes-&bytes_used_hwm) val11e,
       (&bytes_hwm-&bytes_used_hwm) val11f
from   dual;

col val12 new_val sf noprint
select count(*) val12
from   dba_extents
where  segment_name= upper('&tnm')
       and owner      = upper('&owner');

set termout on
set echo off feedback off verify off

prompt Generating report...

col own          format a20          fold_after
col tnm          format a30          fold_after
col pctfreeex    format 999,999,999 justify right fold_after
col pctusedx     format 999,999,999 justify right fold_after
col nr           format 999,999,999 justify right fold_after
col blks         format 999,999,999 justify right fold_after
col bytess       format 999,999,999 justify right fold_after
col bytesshwm    format 999,999,999 justify right fold_after
col bytessusedhwm format 999,999,999 justify right fold_after
col hwmx         format 999,999,999 justify right fold_after
col bwd         format 999,999,999 justify right fold_after
col bwod        format 999,999,999 justify right fold_after

```

All About Oracle Database Fragmentation

```
col bwdhwm      format 999,999,999 justify right fold_after
col bwodhwm    format 999,999,999 justify right fold_after
col sfx        format 999,999,999 justify right fold_after
col rfx        format 999,999,999 justify right fold_after
col a          format a60          justify right fold_after
col bytpctus   format 999 justify right fold_after
col blkpctus   format 999 justify right fold_after
col bytpctushwm format 999 justify right fold_after
col bnu        format 999,999,999 justify right fold_after
col bnuhwm     format 999,999,999 justify right fold_after
col blkpctushwm format 999 justify right fold_after

select 'Owner          : '||'&owner' own,
       'Table name     : '||'&tnm' tbnm,
       '-- Storage Parameters -----' a,
       'pct_free       : '||&pct_free pctfreex,
       'pct_used       : '||&pct_used pctusedx,
       '-- Segment Fragmentation -----' a,
       'Number of extents : '||&sf sfx,
       '-- Row Fragmentation -----' a,
       'Rows           : '||&num_rows nr,
       'Rows frag      : '||&row_chains rfx,
       '-- Space Allocated -----' a,
       'Blocks alloc    : '||&raw_blocks blks,
       'Bytes alloc     : '||&bytes bytess,
       '-- All Alloc Blocks -----' a,
       'Blocks alloc    : '||&raw_blocks blks,
       'Blocks w/data   : '||&blks_w_data bwd,
       'Blocks wo/data  : '||&blks_wo_data bwod,
       'Blocks pct used(BF): '||&blocks_pct_used blkpctus,
       'Bytes alloc     : '||&bytes bytess,
       'Bytes used      : '||&bytes_used_hwm bytessusedhwm,
       'Bytes not used   : '||&bytes_not_used bnu,
       'Bytes pct used(BF) : '||&bytes_pct_used bytpctus,
       '-- Below The HWM Blocks -----' a,
       'Block HWM       : '||&hwm hwmx,
       'Blocks w/data   : '||&blks_w_data bwdhwm,
       'Blocks wo/data  : '||&blks_wo_data_hwm bwodhwm,
       'Blocks pct used(BF): '||&blocks_pct_used_hwm blkpctushwm,
       'Bytes alloc     : '||&bytes_hwm bytesshwm,
       'Bytes used      : '||&bytes_used_hwm bytessusedhwm,
       'Bytes not used   : '||&bytes_not_used_hwm bnuhwm,
       'Bytes pct used(BF) : '||&bytes_pct_used_hwm bytpctushwm
from dual;
```

```
prompt **** You may need to ANALYZE TABLE &owner..&tnm DELETE STATISTICS

set feedback on
```

```
-- *****
-- * Copyright Notice   : (c)1999 OraPub, Inc.
-- * Filename           : fgidx.sql
-- * Author              : Craig A. Shallahamer
-- * Original            : 15-nov-99
-- * Last Update        : 16-nov-99
-- * Description        : Get INDEX fragmentation details for a given object.
-- * Usage               : start fgidx.sql <owner> <index name>
-- *****

def owner=&&1
def inm=&&2

--set echo off verify off heading off
set echo on verify on heading on

set termout on
analyze index &owner..&inm validate structure;

set termout on
prompt Gathering raw data 1 of X.  Could take some time...
set termout off

set termout on
set echo on verify on heading on

col val0A new_val block_size
select value val0A
from   v$parameter
where  name = 'db_block_size';

col val1A new_val tnm
select table_name val1A
from   dba_ind_columns
where  index_owner = upper('&owner')
       and index_name = upper('&inm');
```

All About Oracle Database Fragmentation

```
col val1AA new_val pct_free
select pct_free val1AA
from dba_indexes
where table_owner = upper('&owner')
      and index_name = upper('&inm');

col valAB new_val sf
select count(*) valAB
from dba_extents
where segment_name= upper('&inm')
      and owner      = upper('&owner');

col val2A new_val no_rows
select /*+parallel(5) */
      count(*) val2A
from &owner..&tnm;

set termout on
set verify on feedback on echo on

set termout on
prompt Gathering raw data 2 of X. Should take less than 30 seconds...
set termout off

set termout on
set echo on feedback on verify on

col val4A      new_val br_blks noprint
col val4B      new_val lf_blks noprint
col val4C      new_val used_blocks noprint
col val4D      new_val empty_blocks noprint
col val4E      new_val alloc_blocks noprint
col val4F      new_val alloc_bytes noprint
col val4G      new_val used_bytes noprint
col val4H      new_val br_blks_rows noprint
col val4I      new_val lf_blks_rows noprint
col val4J      new_val br_entry_size noprint
col val4K      new_val lf_entry_size noprint

select br_blks                val4A,
       lf_blks                val4B,
       1+br_blks+lf_blks      val4C,
       blocks-1-br_blks-lf_blks val4D,
       blocks                 val4E,
       blocks*&block_size      val4F,
```

```

1+br_blks+lf_blks*&block_size      val4G,
&no_rows/br_blks                    val4H,
&no_rows/lf_blks                    val4I,
(&block_size*br_blks)/&no_rows      val4J,
(&block_size*lf_blks)/&no_rows      val4K
from   index_stats;

col val5A new_value no_used_bytes
col val5B new_value pct_blocks_used
select &alloc_bytes-&used_bytes      val5A,
       &used_blocks/&alloc_blocks    val5B
from   dual;

set termout on
prompt Calculating fragmentation statistics
set termout off

set termout on
set echo off feedback off verify off

prompt Generating report...

set heading off

col c format a30                      fold_after justify right
col x format 999,999,999,999         fold_after justify right

prompt INDEX Fragmentation Report

select 'Owner          : '||&owner' c,
       'Table name    : '||&tnm' c,
       'Index name    : '||&inm' c,
       'Block size    : '||&block_size x,
       'pct_free      : '||&pct_free x,
       'Rows          : '||&no_rows x,
       'Number of extents : '||&sf x,
       '-- Space Allocated -----' c,
       'Blocks alloc   : '||&alloc_blocks x,
       'Bytes alloc    : '||&alloc_bytes x,
       '-- All Alloc Blocks -----' c,
       'Blocks alloc   : '||&alloc_blocks x,
       'Blocks w/data  : '||&used_blocks x,
       'Blocks wo/data : '||&empty_blocks x,
       'Percent used   : '||&pct_blocks_used x,
```

All About Oracle Database Fragmentation

```
'Bytes alloc      : '||&alloc_bytes x,  
'Bytes used      : '||&used_bytes x,  
'Bytes not used   : '||&no_used_bytes x,  
'-- All Used Blocks -----' c,  
'Used blocks     : '||&used_blocks x,  
'Used bytes      : '||&used_bytes x,  
'Root blocks     : 1' x,  
'Branch blocks   : '||&br_blks x,  
'  rows per bb   : '||&br_blks_rows x,  
'  entry size(bytes): '||&br_entry_size x,  
'Leaf blocks     : '||&lf_blks x,  
'  rows per lb   : '||&lf_blks_rows x,  
'  entry size(bytes): '||&lf_entry_size x  
from dual;  
  
set feedback on
```