

# AVOIDING A DATABASE REORGANIZATION

Understanding, detecting, and eliminating harmful database fragmentation

Craig A. Shallahamer

*Oracle Services - System Performance Group*

*02-NOV-94 V2.2*

## Abstract

Experience indicates most databases are reorganized to eliminate database fragmentation thus increasing performance. However, reorganizing a production database is very expensive (labor and downtime) and induces risk. These obstacles vividly manifest themselves when working with very large databases (VLDBs). A VLDB site cannot afford the downtime and risk associated with reorganizing their entire database. While performance usually improves, focusing on a few select objects yields similar performance gains.

This paper presents the real and perceived risks and benefits of performing a complete database reorganization, by addressing database fragmentation types, how they are created, how performance is impacted, how to detect them, and how to resolve the fragmentation.

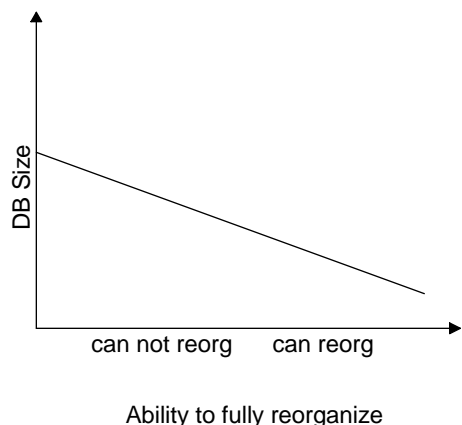
## Introduction

“Your database is slow because it’s fragmented. Defragment the database and performance will increase.” Database Administrators (DBAs) have heard this statement many times from “reliable” sources. To solve their performance problems most DBAs will reorganize their entire database. Unfortunately, many DBAs do not consider and do not understand their reorganization options.

Understanding the above situation leads one to ask questions such as: What type of fragmentation is my database suffering from? Which objects are suffering? Can I obtain the same performance improvement by defragmenting specific objects rather than the entire database? The answers to these questions are complex. However, by understanding how different types of fragmentation manifest themselves in a database, detecting the fragmentation, understanding if the fragmentation is a performance detriment, and precisely resolving the fragmentation, one can develop a strategy to avoid a costly and risky full database reorganization.

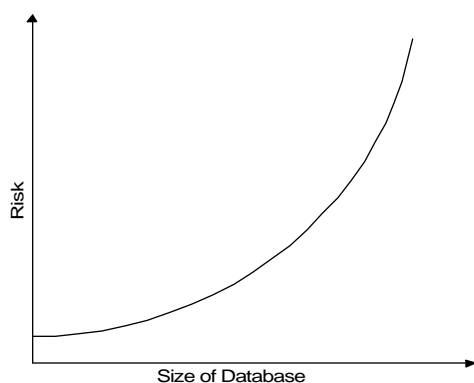
This paper addresses five different database fragmentation types.

1. Tablespace freespace bubble fragmentation
2. Tablespace freespace honeycomb fragmentation
3. Segment fragmentation
4. Table row fragmentation
5. Table block fragmentation



**Figure 1: As the size of a database grows, the opportunity to fully reorganize diminishes**

As Oracle databases continue to grow and applications become increasingly complex, there will come a time when it is impossible to perform full database reorganizations. The combined risk and down-time will nullify the opportunity to fully reorganize. Therefore, a solution must be developed. The information presented in this paper will naturally lead one towards very straightforward solutions. My strategy is to first explain the different types of database fragmentation followed by solutions to avoid a full database reorganization.



**Figure 2: As database size increases so does the risk associated with administering the database system.**

## Reorganization and Risk

This paper frequently mentions the word “risk.” For the remainder of this paper, references to risk refer to a combination of human error, hardware failure, and the up-time requirements versus reorganizing a very large database (VLDB).

The first type of risk is human error. Anyone who has rebuilt an Oracle database understands how easy the rebuild effort can become complicated and lengthened by making a single typographical error. Human error will always be with us. Avoiding a database reorganization reduces human intervention and reduces the risk of human error.

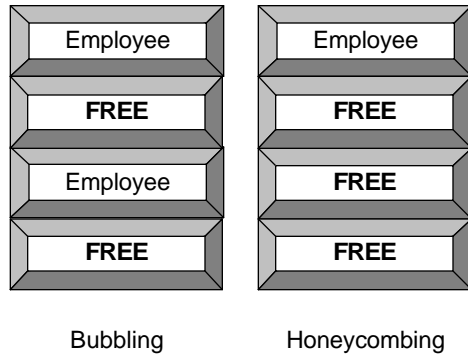
The second type of risk is hardware failure. It has happened to me, so it can happen to you. During a database reorganization reload phase, a disk drive failed. Thankfully my tape backup was good. The larger the database the more hardware required to support the database. The more hardware involved, the larger the risk of a single hardware failure.

The third type of risk involves the effect of human error and hardware error with very large databases (VLDBs). There is a direct relationship between database size and the time it takes to reorganize the database. Recovery time significantly increases as database size increases. Often up-time system requirements and the risk associated with rebuilding a VLDB is too great to justify rebuilding the database.

Because there are risks associated with reorganizing a database, it behooves the database administrator to design and

configure their database to avoid a database reorganization.

### Tablespace Freespace Fragmentation



**Figure 3: Tablespace freespace fragmentation manifests itself with single chunks of freespace, known as bubbling, or contiguous chunks of freespace, known as honeycombing.**

As the name implies, tablespace freespace fragmentation consists of freespace chunks residing within a tablespace. Swiss cheese is a good tablespace freespace fragmentation model. The cheese represents the data, and each hole represents a freespace chunk.

Tablespace freespace fragmentation is the result of segments being created and dropped. If a segment is dropped, its extents become free. For example, if a fifty extent table is dropped, fifty freespace chunks are created. A common misunderstanding is to believe deleted rows produce tablespace freespace fragmentation. While deleted rows result in table block fragmentation, deleted rows do not result in tablespace freespace fragmentation.

The type of tablespace freespace fragmentation that results when an object is dropped depends on the physical location<sup>1</sup> of

<sup>1</sup> While queries based around Oracle's data dictionary view may give the appearance that segment extents are physically contiguous, the operating system may have the data scattered throughout a disk(s). Neither Oracle nor any vendor will guarantee data to be physically contiguous. Data block "scatter" can be eliminated by physically reordering a table's rows.

the segment's extents. Fifty bubbles will be created when a segment, consisting of fifty non-adjacent extents, is dropped. If all extents are adjacent, a honeycomb will be created. This paper will address both types of tablespace freespace fragmentation.

### Bubble Fragmentation

The first tablespace freespace fragmentation type is commonly known as a "bubble." Figure 3, Table 1, and Table 2 show a bubble can be any size and must be surrounded by other objects. For a bubble to be filled with data, an entire extent must fit within a bubble. Bubbling ruins freespace utility.

To understand the effect of bubbling, let's suppose a segment must extend. The segment's storage parameters require a ten block extent be created. The server process<sup>2</sup> scans the free-extent table (Table 2) looking for a ten-block or larger bubble. Since the entire extent must fit within a bubble, a nine block bubble cannot be used. Tablespace storage is less efficient and performance suffers because a tablespace containing many small bubbles wastes space, and a server process wastes time trying to find suitable freespace.

The freespace acquisition algorithm allows one or two free-extent table (**fet\$**) scans. The first **fet\$** table pass, as mentioned above, is performed in hopes of finding a single bubble large enough to contain the entire extent. If this fails, a second pass is performed looking for adjacent bubbles that can be coalesced to form a single large chunk of space in which the new extent will be placed. If suitable freespace is not found, the message, "ORA-1540 Unable to allocate *x* bytes in tablespace *y*" is returned.

<sup>2</sup> An Oracle server process is synonymous to the two-task shadow process. The Oracle V6 two-task shadow process is synonymous to an Oracle7 dedicated server process.

Is tablespace freespace fragmentation a performance detriment? The obvious answer is yes, but the real answer is yes and no. Most objects infrequently extend. Therefore, two **fet\$** table scans are not noticeable. Suppose an application creates an interim table and inserts rows causing the segment to extend many times. In this case, performance will be affected. In an extreme case, such as an Oracle import, performance may increase over 50% by eliminating tablespace freespace fragmentation.[2] To summarize, in most cases tablespace freespace fragmentation is not a performance problem, but rather an

indication of bad segment size selection or unpredictable segment growth.

Oracle7 helps reduce tablespace freespace fragmentation. Suppose a ten block extent must be created and an eleven-block bubble is being considered. Oracle7 will create an eleven block extent. The Oracle7 designers realized small bubbles waste space and decrease future segment extension performance.

Tablespace freespace fragmentation can be examined by constructing a simple query based around the **dba\_extents** data dictionary view (Table 1) or the **fet\$** table (Table 2). Figures 2 and 3 show two bubbles and one honeycomb.

file id	block id	no. of blks	segment name	
1	1195	5	SYS.OBJ\$	
1	1100	1750	<b>free</b>	A 1750 block bubble
1	2850	310	SYS.COL\$	
1	3160	3760	<b>free</b>	A 3760 block bubble
1	6920	26	SYS.VIEW\$	
...	...	...	...	
1	9307	8	SYS.COM\$	
1	9315	3700	<b>free</b>	
1	13015	3	<b>free</b>	A 5453 block honeycomb
1	13018	1750	<b>free</b>	

**Table 1: A map of the SYSTEM tablespace.**

file id	tablespace id	block id	length	size (blks)	
1	1	1100	1750		A 1750 block bubble
1	1	3160	3760		A 3760 block bubble
...	...	...	...		
1	1	9315	3700		
1	1	13015	3		A 5453 block honeycomb
1	1	13018	1750		

**Table 2: The free extent table, fet\$.**

While Oracle7 has taken a large step in minimizing bubbling, a poorly configured application or database will foster bubbles. This is unfortunate, because a tablespace must be rebuilt to eliminate bubbles.

Finally, a solid database configuration is paramount. Nearly fifty percent of all database tuning engagements are solved by addressing configuration issues, and not by tuning SQL statements. The Optimal Flexible Architecture (OFA)[4] presents a configuration standard is currently in place at hundreds of production database installations around the world. Minimally fragmented tablespaces are one of the OFA guideline benefits.

### Honeycomb Fragmentation

A honeycomb is the second tablespace freespace fragmentation type. A natural honeycomb contains honey chunks surrounded by other honey chunks. The database world is similar. A honeycomb contains multiple freespace chunks (bubbles) surrounded by other freespace chunks. Figure 3, Table 1, and Table 2 show the difference between bubble and honeycomb fragmentation.

Honeycomb fragmentation is created like bubble fragmentation. The **fts** table snapshot (Table 2) shows two bubbles and one honeycomb. The freespace acquisition algorithm encompasses both bubbles and honeycombs. So just as with bubbling, the performance impact of honeycombs is manifested only when a segment is frequently created, dropped, or extended.

Bubbling wastes freespace and diminishes freespace usability. Honeycombs retard space acquisition efficiency. A honeycomb may only be coalesced during the second

**fts** scan. If a honeycomb has been coalesced into a bubble (manually or automatically), it could possibly be used during the first **fts** scan. The honeycomb coalescing process is commonly called "gluing." The Oracle7 background process **smn** periodically<sup>3</sup> coalesces honeycombs. Coalescing Oracle V6 honeycombs can be done manually[2] or by an automated script.[6] Because Oracle7 periodically coalesces freespace, an application that produces honeycombs is less likely to impact performance.

Folklore and myths have lead to the belief that tablespace freespace fragmentation is a serious performance problem. As the previous paragraphs have shown, tablespace freespace fragmentation only affects application performance when a segment is frequently created, dropped, or extended. These operations are application specific, and should be isolated and controlled by proper segment placement and sizing, and proper application processing.

### Segment Fragmentation

The performance impact of multiple segment extents is a topic that has gained much attention from within Oracle and its customers. Not only has Oracle's message been mixed and confused but third party published books and articles are making conclusions based upon folklore and conjecture. This section will provide information critical to making an informed decision about the performance impact of multiple segment extents.[7]

---

<sup>3</sup> A large honeycomb or multiple honeycombs are coalesced a few bubbles at a time. For example, smn may take 30 minutes to completely coalesce a fifty bubble honeycomb.

Segment fragmentation occurs at the extent level, and not the database block or table row level. When a segment grows it naturally extends. Once extension has occurred, the segment is technically fragmented.

Segment Name	Segment Type	Number of Fragments	Segment Fragmentation
benefit	table	2	yes
benefit_n1	index	12	yes
benefit_u1	index	56	yes
bonus	table	50	yes
bonus_u1	index	1	no
dept	table	75	yes
dept_n1	index	25	yes
emp	table	2	yes
emp_u1	index	1	no

Table 3: A view of fragmented segments.

Occasionally a multiple extent segment is initially created to eliminate future extension time or to pre-allocate space. However, intentional segment fragmentation is usually done to stripe a segment across multiple disk drives. A striped segment usually increases both query and update performance. Figure 4 shows the striped *employee* table residing on three physical devices.

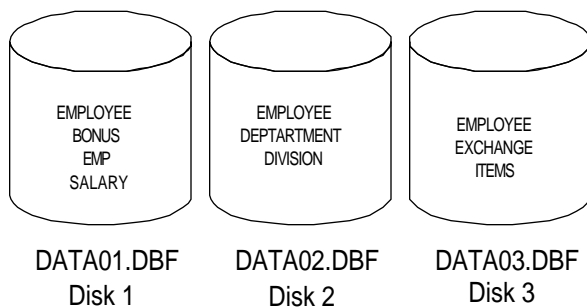


Figure 4: Segments are many times intentionally fragmented to increase read performance. This figure shows the EMPLOYEE table (consisting of three extents) striped across three database files.

The segment fragmentation performance threat has been debated, and misinformation

has been spread. Research clearly shows segment fragmentation does not impact real-life production system performance. As the system load increases, the performance impact of multiple segment extents decreases.

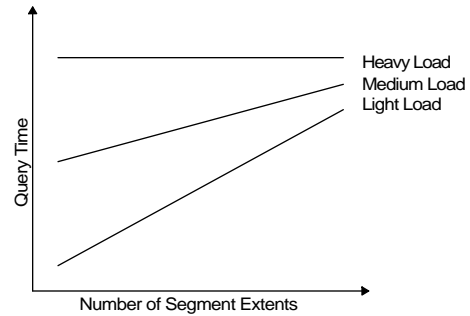


Figure 5: The performance of multiple segment extents. Field experience, thought experiments, and scientific experiments clearly demonstrate as system load increases, the impact of multiple segment extents decreases.

The below facts and their consequences are graphically shown in Figure 5.

#### The Facts:

- Oracle retrieves and writes data by the block, not by the extent.
- Oracle's I/O requests (usually less than 32 kilobytes) are queued with other user requests, and do not have a higher priority than other user requests.
- A maximum of  $n$  additional I/O request could be made, where  $n$  is the number of segment extents.
- As the number of I/O requests increases, the probability increases that a request which decreases query performance for user  $x$  will increase query performance for user  $y$ .

- Neither Oracle nor most operating systems guarantee that segment data will be physically contiguous.
- Only full-table scans can be affected by multiple extents.
- Indexed scans typically reference scattered data<sup>4</sup>.
- Recursive SQL is used when working with multiple extents.
- The data dictionary cache containing used extents (**dc\_used\_extents**) and free extents (**dc\_free\_extents**) must be larger to accommodate more used extent information (**uet\$**) and free extent information (**fet\$**).

### The Consequences:

- In a production system, it is extremely unlikely a disk will service full-table scan requests in a single continuous stream without interruption.
- In a production system, disk heads move all over servicing competing requests. A few extra movements, which are noticeable on a single user system (E.g., PC), are not noticeable to the user.
- In a production system, the net effect of possible (yet improbable) additional I/Os is negligible.

Segment fragmentation can be detected by querying the **user\_segments** and the **user\_extents** data dictionary views (Table 3).

---

<sup>4</sup> Data blocks referenced by a single index leaf block are usually scattered throughout many data blocks.

As mentioned above, eliminating segment fragmentation typically does not improve performance, but induces risk and forces system down time. There are only a handful of valid cases for rebuilding a segment.

Rebuild a segment when;

1. The segment is approaching the physical maximum number of extents<sup>5</sup>.
2. The segment has reached the physical maximum number of extents.
3. Disk striping will increase performance.
4. Defragmenting the segment will increase performance. For example, when a table is used in a time sensitive, I/O intensive, single-threaded operation, placing all data inside a single extent **may** increase performance.

### Row Fragmentation

While tablespace fragmentation has been blamed for many performance problems, most DBAs overlook row fragmentation. A row will become fragmented if it enlarges and the entire row can not fit in its current data block. Experts tend to agree that many performance problems are the result of row fragmentation.

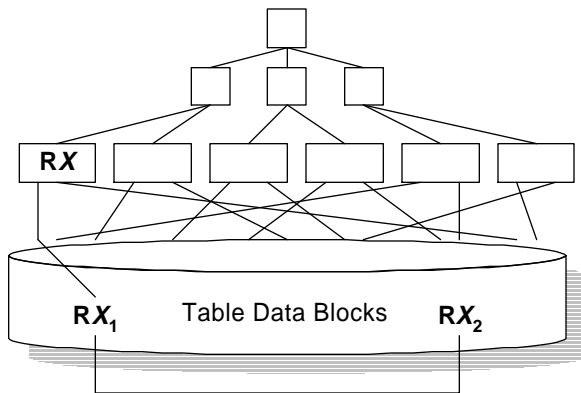
Row fragmentation, more commonly known as row chaining, manifests itself in two forms. The most common form occurs when a row enlarges during an update. If an enlarged row will not fit into its block, the entire row is migrated to another block. A forwarding address is left at the original row's block. Unfortunately, all indexes

---

<sup>5</sup> Each segment header block can hold a limited number of extent pointers. A 2 kilobyte database block can hold 121 extent pointers. A 4 kilobyte database block can hold 249 extent pointers.

referencing the row are not updated with the new row address.

When a server process retrieves a row via a full-table scan or an indexed-scan, both the row's original block and its current block must be read. This can be a serious performance detriment. Figure 6 shows when fragmented row *X* is retrieved, additional block(s) must be referenced.



**Figure 6: Chained row *X* decreases read and write performance during both index scans and full-table scans. During an index scan the server process reads index blocks and then reads both row pieces *RX*<sub>1</sub> and *RX*<sub>2</sub>. During a full-table scan, when the block containing row piece *RX*<sub>1</sub> is read, the server process “jumps” and reads the block containing *RX*<sub>2</sub> and then returns to reading blocks after the block containing *RX*<sub>1</sub>.**

If a row is too large to fit into a single empty database block the row is broken up into "row pieces" and placed in multiple blocks. Row migration can be eliminated but row pieces can never be placed into a single database block. For the remainder of this paper, references to row fragmentation only refer to row migration.

It's really quite simple why row fragmentation has been overlooked for so long. Many DBAs have concluded that tablespace freespace fragmentation and

segment fragmentation are causing performance problems. As a result, the classic solution is to rebuild the entire database. While this eliminates tablespace freespace fragmentation and segment fragmentation, it also eliminates row fragmentation. Because performance increased after a rebuild, the incorrect conclusion is made that tablespace freespace fragmentation and segment fragmentation were responsible. Performance increases because row fragmentation is eliminated<sup>6</sup>, rows are tightly packed into each database block, and the high-water mark is reset. As a result, many DBAs feel rebuilding a database is the best solution to their performance problems.

Have you ever packed for a camping trip using an itemized list? Imagine if you looked up an item on your list and spent time looking for the item. But instead of finding the item, you discover a piece of paper directing you to another location. Your packing speed would be dramatically reduced.

The same situation occurs with row fragmentation. The server process expects to find a particular row in a data block only to discover another block must be accessed.

While row fragmentation does impact performance, it occurs in very characteristic situations. Since row fragmentation only occurs when a row enlarges, an application that only inserts rows will never be affected.

Batch applications (manually or automatically) or information repository applications will not typically suffer from row fragmentation. Applications that frequently update information (E.g., free-

<sup>6</sup> Row fragmentation is eliminated regardless of whether segments are exported with "compress=y".

form text entry applications, warehouse application) may suffer from row fragmentation.

It is important to check for row fragmentation because it can sneak up on you when you do not suspect it. For example, a general ledger journal entry application generally only inserts rows causing no row fragmentation. But in this actual case, because users did not want to forget to enter journal entries at month end, at the beginning of each month each journal entry was entered with a zero amount. At month end the actual figures were entered and each row's length increased resulting in severe row fragmentation and poor financial posting and report performance.

Many new Oracle DBAs are appalled Oracle does *not* supply an automatic database rebuild utility. On the bases of their experience with *other* "database" products, they incorrectly assume rows will naturally fragment resulting in horrendous performance. What these neophytes do not recognize is the RDBMS provides a very powerful weapon to thwart row fragmentation.

This weapon is the proper use of segment storage parameters **pctfree** and **pctused**. [3] The **pctfree** parameter restricts the amount of data a block holds during inserts. A larger **pctfree** setting reserves more space in each block for row growth. While space is wasted during inserts, space can be used when rows enlarge. The **pctused** parameter kicks in after rows have been inserted up to the **pctfree** level. The **pctused** setting halts row insertion (by keeping a block off a free list) until **pctused** percentage data block space exists. Therefore, to reduce the likelihood of row fragmentation, set **pctfree** high and **pctused** low.

A forceful way to prevent row fragmentation is to define a fix length column (char) instead of the more common variable length column (varchar).

Variable length columns provide space savings while reducing row fragmentation. Setting the correct column format and the correct storage parameters minimizes both row fragmentation and space wastage. The result is efficient database block space usage and speedy data retrieval.

Identifying row fragmentation in Oracle V6 is centered on the **v\$waitstat** virtual data dictionary view. Oracle7 has made the task of identifying row chains very simple. Simply *analyzing* a table and querying the **user\_tables** view will reveal the number of row chains.

If a segment containing fragmented rows has been detected there are two basic ways to solve the problem. If rows are not likely to continue growing, rebuild the table. For example, suppose during a project implementation help text was constantly being updated causing row fragmentation. Once the project was in production and the help text was no longer being modified, row fragmentation no longer occurred. To solve this problem, the help text table must only be rebuilt. Suppose the help text was modifiable by users who sometimes typed in freeform notes. In this case, it may be wise to increase **pctfree** to reduce the likelihood of future row fragmentation.

To summarize, row fragmentation significantly affects database performance during index scans and full-table scans. Performance decreases because two data blocks must be retrieved to access a single row. By understanding column format options, segment characteristics, selecting appropriate storage parameters, identifying

offending segments, and fixing them, the likelihood of row fragmentation can be virtually eliminated.

## Table Block Fragmentation

Although table block fragmentation is common, it is difficult to detect. As with other forms of database fragmentation, because table block fragmentation is difficult to detect, it is often overlooked. This is unfortunate. Fragmented table blocks reduce both full table and index scan performance, and waste space.

A fragmented table block looks like a pile of "pick up sticks" near the end of the game. There are plenty of sticks but 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 deletes a row from a table it fragments a table's block. Obviously space is wasted unless another row is inserted into the table's block.

Picture yourself scanning the cheese department at your favorite grocery store. If maximum cheese poundage per block is your goal, you would grab cheese blocks with no holes. If you wanted to retrieve ten pounds of cheese (assuming the cheese blocks have the same physical dimensions), the job would be completed quicker and with less effort if you grabbed cheddar cheese instead of Swiss cheese<sup>7</sup>. Picking up the Swiss cheese forces you to perform more work to retrieve 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.

Just as with row fragmentation, table blocks do not become fragmented by applications that only insert rows. Unlike row fragmentation, only when a row is deleted does a table block become fragmented. A deleted chained row compounds the problem since two data blocks will become fragmented when a single fragmented row is deleted.

The application types that suffer from table block fragmentation may involve process flow (where data moves from one table to another) or time life data. "Interim" segments commonly store information for a limited duration. If an interim object is not truncated (dropped in V6) before or after each run, table rows are usually deleted. Deleting interim table rows results in severe table block fragmentation.

Row deletion is under application control. The block fragmentation effect can be minimized by carefully setting **ptfree** and **ptused**. By setting **ptused** high, each block will be placed on the free list after a few rows are deleted. Because free list manipulation causes recursive SQL, a balance must be maintained between a high **ptused** and minimal recursive SQL.

Oracle's data dictionary, performance utilities, and sophisticated tool sets[6] provide the raw tools to detect table block fragmentation. Table block fragmentation can be measured by finding out how much space a table can possibly hold and comparing this to how much data is currently being held. Once the offending

---

<sup>7</sup> In addition to retrieval efficiency, cheddar tastes better than Swiss cheese.

tables have been detected they can be individually rebuilt.

## The Solution

Once one has a thorough grasp of database fragmentation, a straightforward method to eliminate and prevent future harmful fragmentation becomes evident. Some of the steps, outlined below, have been discussed in detail, while some steps require architecting or purchasing powerful tools.

- 1. Understand your risks.** Chapter five discusses how risk can manifest itself in a growing complex database system.
- 2. Understand fragmentation.** Chapters six through nine detail five different database fragmentation types.
- 3. Object level detection.** The absolute key to avoiding a full database reorganization is detecting which specific objects are suffering from harmful fragmentation and specifically addressing them.
- 4. Prevent reoccurrence.** Once an object is discovered to be suffering from harmful database fragmentation, changes must be made to prevent the object from fragmenting after the object is rebuilt. If changes are not made, performance will taper off after the object is rebuilt<sup>8</sup>.
- 5. Rebuild object.** To eliminate fragmentation, the object must be rebuilt. There are a number of ways to rebuild a table. For example, the export and import utilities, the **create table as select** SQL command, or the **copy** SQL command can be used.

## Developing Powerful Tools

Any DBA worth his salt understands once the items discussed in this paper are understood, an automated process can be engineered to periodically detect and resolve fragmentation. However, writing these tools is not as simple as one might first expect.

There are two challenges a worthwhile toolset must conquer. A trustworthy toolset will use multiple detection algorithms since each detection method has strengths and weaknesses. A DBA needs fragmentation severity information from a variety of viewpoints clearly presented. Only then can one begin to assess the benefits of rebuilding an object.

Once an algorithm has been developed, the second and equally difficult hurdle must be met. The various flavors of fragmentation discussed impact nearly all database systems. The fragmentation and the defragmentation process effect are intensified in a VLDB environment. Therefore, the second challenge for any toolset requires problem identification within acceptable time constraints. This is a serious obstacle when working with millions of rows tables in a twenty-four by seven shop.

---

<sup>8</sup> When an table is rebuilt performance may increase because data blocks are fully packed (block fragmentation is eliminated), row fragmentation is eliminated, and the high-water mark is set the lowest possible value.

The goal of this paper is to help DBAs understand database fragmentation issues and their impact. Only then can one hope to make intelligent decisions regarding defragmenting database objects. Put another way, the next time someone suggests you rebuild a database because of fragmentation problems, I hope you can ask them specifically what type of fragmentation they are referring to. Then if they are able to intelligently respond, you can proudly claim that based upon your analysis and automated tools the different fragmentation types are constantly being monitored and efficiently administered.

## References

1. Millsap, Cary V., Shallahamer, Craig A., Adler, Micah (1993). Predicting the Utility of the Nonunique Index. *Oracle Magazine*, Spring 1993, Volume VII Number 2
2. "Blowing Out The Walls." Oracle Corporation White Paper 1991, Revision: February 1994
3. Oracle Corporation (1993). *Oracle7 Server Concepts Manual*. Oracle Part No. A12713-1, August 10, 1993
4. Millsap, Cary V. (1994). *The OFA Standard, Oracle7 for Open Systems*. Oracle Part No. A19308-1, May 1994
5. "Core Clinic" Class Notes (1994). Oracle Consulting Core Technologies Team, Revision: 3.1-7.0.15 January 2, 1994
6. Oracle Consulting Core Technologies Team (1994). *Administrative and Performance Suite (APS)*. Curator: Craig A. Shallahamer  
Email: cshallah@us.oracle.com
7. Shallahamer, Craig A. (1994). *The Performance Impact of Segment Fragmentation*. Oracle Corporation White Paper 1994
8. Shallahamer, Craig A. (1994). Total Performance Management. Originally entitled, The Art and Science of Performance Management. *Proceedings of the 1994 Oracle Applications User Group (OAUG) Conference*, Paper #49
9. Millsap, Cary V. (1994). *Storage Management for Oracle Version 7*. Cary V. Millsap, Oracle Corporation White Paper 1995

## Acknowledgments

This work was supported by Oracle Corporation and was conducted while the author was working for (and still is) Oracle Corporation. A special thanks to Ravi Balwada, Mike Brouillette, Steve Herrerias, Brian Kush, Cary Millsap, Rick Minutella, and Bill Slocumb for their continual criticism and stimulating discussions.

## About the Author

Craig Shallahamer is a Practice Manager of Oracle's System Performance Group, a part of the Oracle Services. The team is responsible for building new tools and capabilities like the ones described in this paper for Oracle and its customers. The System Performance Group provides system design, capacity planning, and performance management services to customers worldwide.

Since joining Oracle in 1989, Mr. Shallahamer has worked at hundreds of clients around the world. His specialization is performance management and capacity planning related research and on-site engagements that have resulted in a number of published papers at EOUG, OAUG, IOUW, Asia Pacific OUG, and *Oracle Magazine*. Craig is based in Portland, Oregon and can be reached at 503/220-5122 or on the internet at [cshallah@us.oracle.com](mailto:cshallah@us.oracle.com). Craig's papers are available via the Word-Wide Web at <http://www.europa.com/~orapub>.