Mitos, meias-verdades, enganos e Fraudes da Fragmentação

Por : José Laurindo Chiappa

DBA BRASIL 2.0

IBTA
Instituto Brasileiro de Tecnologia Avancada

Nossos Patrocinadores

STROHL
Brasil

Apresentação Pessoal

- Atuante há 28 anos em TI em geral, sendo 21 anos dedicados à tecnologia Oracle ®.
- Qualificações de Database Specialist/DBA Sênior/Analista obtidas via Atuação em Empresas nas áreas de Finanças, Produção Industrial, Comércio e outras. Atuação eventual (desde Out/95) como Instrutor.
- Programação geral em shell scripting, C, Oracle Forms, Reports, Graphics.

FRAGMENTAÇÃO é um conceito extremamente difundido na área de TI, e mesmo para os usuários finais. Ainda assim, sendo algo arraigado ao imaginário de muitos técnicos e usuários na área, é um assunto sobre o qual muito se diz e pouco se explica, e que acaba servindo de justificativa para os mais diversos problemas de performance, e gerando os mais diversos MITOS e FOLCLORES.... Pretendemos definir e delimitar o assunto, para então permitir Ações mais exatas...

Inicialmente , vamos especificar uma e limitar o Conceito.

DEFINIÇÃO : a definição básica no Dicionário é :

fragmentação
frag·men·ta·ção
sf

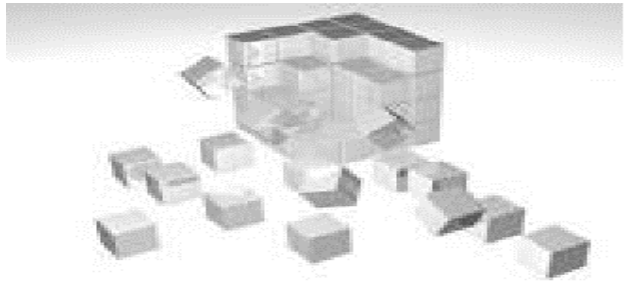
1 Ato ou efeito de fragmentar(-se); divisão.

ETIMOLOGIA : *der* de *fragmentar+ção*

Na área de TI, a utilização do termo popularizou-se na época dos Sistemas Operacionais mais primitivos (como o MS-DOS), onde :

- a leitura dos diversos fragmentos que compõem um arquivo (dado que não é viável um único conjunto físico contendo todo o arquivo) era Diretamente influenciada negativamente na performance se os diversos fragmentos não estivessem o mais próximos possível entre si (por exigir movimentação extra da cabeça de leitura do disco E/Ou devido ao Sistema não contar com um dicionário de dados especificando o ponto físico de início de um fragmento)

- dada a capacidade máxima de I/O simultâneo limitada de tais ambientes, tamanhos maiores do que o máximo usável pelo sistema poderiam ser também causa de solicitação de I/O única particionada em múltiplos I/Os físicos



Relação com databases relacionais : desde quase a introdução dos RDBMSs modernos, os conceitos originais de FRAGMENTAÇÃO não se aplicam, pois todos os principais RDBMSs deixaram de usar a API básica de manipulação de arquivo do Sistema Operacional, passando a controlar o conteúdo e a gravação de seus arquivos internamente, controlar o tamanho de pedaço/fragmento diretamente (no caso da Oracle introduzindo a figura da tablespace system-allocated, constituída por arquivos compostos de fragmentos físicos de tamanhos controlados pelo RDBMS), e usar funções como FSEEK , o que (combinado com o fato de criar os fragmentos/pedaços de um arquivo em áreas contíguas do disco) passaram a permitir simplesmente posicionar a cabeça de leitura no início de cada pedaço/fragmento e solicitar um I/O multiblock único.

Apesar disso, muito por hábito os fabricantes de RDBMS (inclusive e em especial a Oracle) continuaram a denominar as questões/issues de performance derivadas do armazenamento físico de FRAGMENTAÇÃO, mesmo que esse termo tenha deixado de fazer sentido nos RDBMSs modernos.

Pretendemos DETALHAR as issues principais possíveis (ao invés de as varrer para debaixo do tapete com a Etiqueta geral e genérica de FRAGMENTAÇÃO) e expor aqui a causa REAL das issues : com esse conhecimento é absolutamente Possível usarmos um database controlado pelo RDBMS por ANOS A FIO sem presenciar nenhuma "FRAGMENTAÇÃO"....

Observações e Detalhamentos :

1. só falaremos de TABELAS aqui, e especificamente tabelas HEAP-ORGANIZED (sem nenhum tipo de organização Física), via de regra o tipo default num RDBMS e especialmente no RDBMS Oracle : pela fato de Índices Obrigatoriamente serem gravados de maneira Ordenada em disco, possuem outras características que podem causam issues normalmente também IMPROPRIAMENTE chamadas de Fragmentação, mas não serão discutidas aqui

2. exceto no caso do chained e de migrated rows (que será discutido mais abaixo), as issues se relacionam com FULL TABLE SCAN, caso em que os EXTENTs todos que compõem a tabela a ser lida na íntegra vão ser lidos do disco um a um, usando I/O multiblock : tendo em mente a exceção indicada, via de regra NENHUMA das issues que discutiremos deve influenciar em I/O single-block

3. via de regra a Gravação de dados em disco é feita nos blocos lidos do disco presentes no CACHE : outras issues de performance podem ocorrer com qtdade anormal de blocos a gravar em disco a partir do cache, mas não os discutiremos aqui

4. não discutiremos o caso dos LOBs, cujo espaço é controlado de maneira diversa aos controles dos HEAP SEGMENTs que discutiremos, Além do fato de poderem ser alocados em tablespaces próprias

5. além das issues que discutiremos serem todas (de maneira IMPRÓPRIA e PREGUIÇOSA), arquivados no mesmo label genérico de "FRAGMENTAÇÃO" o MITO principal é que todas elas (ou a maioria delas) aconteceriam AUTOMATICAMENTE e INEVITAVELMENTE com o uso NORMAL do database em situação NORMAL, dado o tempo de uso e crescimento dos dados....

Conceitos Básicos de I/O e Armazenamento

(Os conceitos a expor são genéricos na maior parte mas são Fundamentados no RDBMS Oracle)

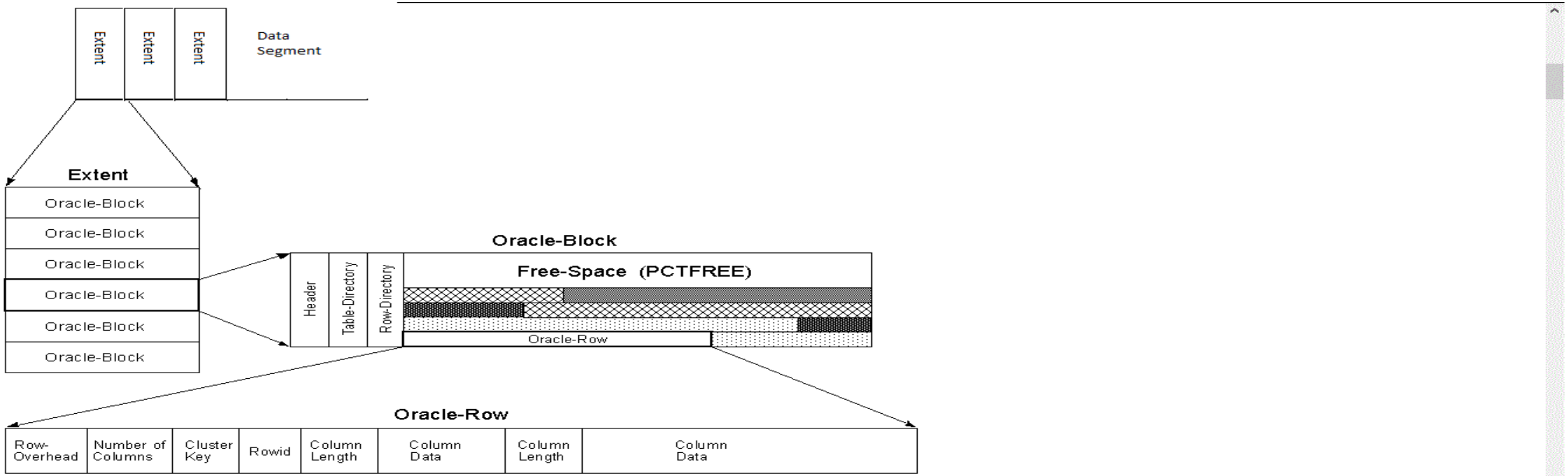
- Os dados são organizados em registros (ou linhas, ou tuplas) e gravados em blocos (ou Páginas : uma unidade mínima, normalmente na ordem de kilobytes) - nunca um só bloco é alocado no disco para futuras gravações mas sim EXTENTs (um pedaço contíguo de blocos, em quantidade varíavel) : a soma dos extents compõe o SEGMENTO DE DADOS, a tabela no nosso caso.

- Caso o primeiro setor/posição livre do disco seja inferior ao tamanho do extent, o RDBMS vai procurar o próximo, para que Fisicamente os blocos que compõem um extent SEJAM contíguos/sequenciais.

- A leitura de dados em disco pode ser feita OU lendo diretamente apenas o bloco onde o dado desejado reside (SE for possível identificar a posição física do bloco, normalmente via Índice) OU lendo todos o(s) EXTENT(s) possível/possíveis de conterem o dado desejado.

- Em especial no RDBMS Oracle, o controle de utilização de espaço em disco é otimizada para re-uso : assim, após DELETEs ou UPDATEs os eventuais blocos e/ou extents sem dados CONTINUAM reservados para o mesmo segmento que os usava, de modo que quando novos dados forem introduzidos o RDBMS não seja solicitado a formatar/procurar em disco por espaço adicional, causando melhor performance para o caso ‘normal’ de uso, onde um segmento via de regra após ter seus dados eliminados , novos dados serão introduzidos em breve. EVIDENTEMENTE, para os casos Pontuais onde há uma regra de negócio Garantindo que não haverá novos dados após um INSERT, esta característica causará Issues.

Figura 1 – Estruturas Internas



Combinando-se os conceitos acima é evidenciado que a Fragmentação propriamente dita não é mais possível de se acontecer num RDBMS moderno, mas como dito a expressão ainda é usada,
mesmo que de forma inconsistente.
IMPORTANTE : é crítico Frisarmos que NENHUM dos pontos acima significa que os problemas derivados de Armazenamento em disco não são mais passíveis de acontecer e causar queda de
performance.
Pretendemos indicar e detalhar abaixo os mais comuns.

Causas e Origens das Issues de Armazenamento de dados e Procedimentos de Solução

A. extents livres não sendo reutilizados por causa de tamanhos diferentes/não-múltiplos entre si

Causa : esta é a situação originalmente proposta como Fragmentação na documentação Oracle, mas como indicado, isso só pode acontecer em situações de tablespaces compostas por arquivos não gerenciados automaticamente, o que permite Extents de tamanhos incompatíveis - digamos, numa tablespace com segment management MANUAL que assim o permite, os extents livres em discos são de 300kbytes e a o tabela foi criada para solicitar extents de 1 Mb : não há como vc juntar vários extents de 300 Kbytes para formarem exatamente 1 Mbyte sem sobra, assim os extents livres de 300 Kbytes serão IGNORADOS e o RDBMS Oracle vai criar novos extents de 1Mbyte... Este cenário era a situação mais comum de ser apontada como “FRAGMENTAÇÃO”.

Com a introdução das tablespaces com alocação UNIFORM (onde todos os extents são do mesmo tamanho) e das tablespaces com alocação AUTO-ALLOCATED (onde o tamanho dos extents a serem criados é primeiro de 64 Kbytes, de pois que vários de 64 Kbytes foram criados o RDBMS passa a criar extents de 1 Mb, etc , este cenário ficou IMPOSSÍVEL : com este setup, se vc tiver múltiplos extents de 64 Kbytes contíguos eles Vão poder sem combinados num de 1 Mb naturalmente, sem ‘sobras’ ou movimentação de extents requerida...

IMPORTANTE : essa operação de combinar múltiplos extents contíguos para se obter um extent de tamanho maior se chama COALESCE, e ocorre AUTOMATICAMENTE em segmentos residindo em tablespaces gerenciadas localmente com extent size gerenciado pelo sistema (OBVIAMENTE, extents de tamanho UNIFORM não precisam disso) – uma das muitas vantagens de tablespaces gerenciados pelo RDBMS localmente, sem dicionário de dados, via bitmap.

EVIDENTEMENTE, extents não contíguos entre si não podem ser sofrer COALESCE (seja pelo procedimento Automático nativo de tablespaces gerenciadas localmente, seja por um comando COALESCE) : esse cenário é ALTAMENTE improvável com tablespaces de tamanhos gerenciados pelo sistema (pois antes de passar a criar extents de tamanho maior (1 Mbyte, normalmente) NECESSARIAMENTE o sistema já criou diversos segmentos menores (de 64 kbytes, normalmente), assim a chance de vc ter diversos segmentos de tamanho menor não-contíguos é pequeníssima, mas pode existir numa situação do tipo :

Suponha que cada quadrado representa um extent de 64 kbytes, e cada cor representa um segmento (A azul, B vermelho e C amarelo) ; ou seja, criou-se um segmento A, que cfrme sofre data input alocou dois extents de 64 Kbytes (0 e 1, digamos), um segmento B que precisou alocar os extents 2 e 3, e um segmento C que foi alocando os extents 4, 5, 5, assim por diante. Suponha que o segmento B liberou os extents 2 e 3 após uma remoção de dados (TRUNCATE, digamos), chegando nesta situação :

Se o segmento C foi crescendo e chegar ao ponto de passar a alocar extents de 1 Mbyte, pelo fato de não estarem contíguos em disco com os demais extents livres de 64 Kbytes, os extents 2 e 3 Não Serão reaproveitados pelo segmento C, que vai ter que combinar os próximos extents de 64 kbytes eventualmente livres e contíguos mais à frente para somarem o 1 Mbyte desejado, OU vai formatar/criar novos extents de 1 Mbyte...
REPITO, esses extents 2 e 3 *** Não São Desperdício *** - ok, não puderam ser usados pelo segmento C MAS serão naturalmente usados pelos Diversos outros segmentos que necessitarem de extents de 64 kbytes... É ISTO que desqualifica esta situação como FRAGMENTAÇÃO propriamente dita, pois (como referido anteriormente) no mito original o espaço 'fragmentado' em princípio não seria usado nunca mais enquanto não ocorresse a 'desfragmentação', o que não é o caso, é puro MITO e IMPOSSIBILIDADE espaço que rigorosamente Nunca Vai ser reusado neste cenário...

Consequências : a quantidade de extents a ser lida/acessada num Full Table Scan vai subir, já que o espaço pré-existente contido nos extents de tamanhos incompatíveis com o assignado para a tabela foi ignorado e extents Adicionais foram criados, E mais espaço em disco do que o minimamente necessário vai ser alocado.

Prevenção : utilizar o gerenciamento pelo sistema OU extents de mesmo tamanho na criação de tablespaces

Correção : recriação do segmento (via export+drop+import ou MOVE ou package DBMS_REDEFINITION), preferencialmente numa nova tablespace gerenciada Automaticamente ou com extents de tamanho UNIFORM

OBS : as demais possibilidades de "Fragmentação" daqui em diante elencadas nos próximos Itens são muito mais específicas do RDBMS Oracle, não terão correspondente Exato em outros RDBMSs.

B. acesso a um registro único requerendo leitura de mais de um bloco

Causas : pode ser causado por ROW MIGRATION (ie, um UPDATE aumentando o dado fisicamente fez com que o registro/linha que antes cabia não coubesse mais no bloco, portanto esse registro/essa linha vai ser Migrada/Movida para outro bloco) ou por ROW CHAINING (ie, registro lógico com tamanho maior do que o tamanho livre máximo no bloco – por exemplo, registro com a soma de suas colunas/campos de mais de 8k num bloco de 4k)

Consequências : o dado que normalmente requereria um só block I/O passou a requerer múltiplos block I/Os

Prevenção : criar registros lógicos com comprimentos compatíveis com o block size, não abusar de strings longas, não ter grande quantidade de colunas/campos no registro lógico para evitar a possibilidade de ultrapassar o tamanho do bloco

Correção : recriação do segmento (via export+drop+import ou MOVE ou package DBMS_REDEFINITION), preferencialmente diminuindo o registro lógico E se possível aumentando o espaço usável no bloco.

IMPORTANTE : por mais várias das diversas técnicas indicadas permita operação ONLINE, nenhum dos procedimentos é GRÁTIS em termos de custo para sua realização, portanto devemos AVALIAR se o montante de linhas migradas/continuadas Justifica isto.

C. grande número de blocos não totalmente preenchidos e/ou com espaço disponível não sendo reutilizado

Causas :

_- operações em APPEND/DIRECT mode (vide próximo Item)

- DELETEs ou UPDATEs liberaram o espaço mas ele continua sendo alocado para o segmento original, segmento esse que Excepcionalmente não vai mais sofrer INSERTs ou UPDATEs que o reutilizem

- em segmentos de dados com espaço interno controlados manualmente (ie, que não usam o recurso de ASSM, Automatic Segment Space management) , OU o espaço normalmente reservado para futuros UPDATEs (PCTFREE) está muito além do que os UPDATEs efetivamente usaram OU o limite porcentual máximo permitido num bloco (PCTUSED) está muito abaixo

Consequência : a quantidade de extents a ser lida/acessada num Full Table Scan vai subir, pois para a mesma informação menos registros caberão num bloco, portanto mais blocos vão ser alocados e portanto mais extents

Prevenção : utilizar o gerenciamento automático de espaço em blocos dos segmentos de dados, OU quando isso não for possível gerenciar Adequadamente os parâmetros de STORAGE das tabelas

Correção : recriação do segmento (via export+drop+import ou MOVE ou package DBMS_REDEFINITION) numa tablespace que permita ASSM ou se não for possível, recriação do segmento Alterando os parâmetros de controle de espaço interno nos blocos, evitar ou restringir o uso de APPEND/DIRECT mode Storage

D. novos extents criados ignorando extents existentes com espaço disponível

Causas : afora a possibilidade de Extents livres em disco de tamanho diferente do tamanho que o segmento em crescimento solicita (impossível em tablespace LMT), a segunda Causa possível é o uso de INSERT em Append-Mode (também chamado de Direct Load em algumas fontes - é uma feature opcional que pode ser Ativada com um HINT na sintaxe /*+ APPEND */ , por parâmetros na API nativa do Oracle OCI ou por comandos em alguns utilitários Oracle, como o sql*loader) : este tipo de Operação provoca o RDBMS a alocar novos extents no espaço livre da tablespace e depois appendar estes extents ao final do segmento sofrendo a operação – para performance, entre outros pontos essa operação apresenta Vantagens decorrentes do fato que extents 'virgens' obviamente não contém dados, sendo ZERO a chance de outra sessão necessitar os acessar/alterar portanto sendo desnecessários locks/latches.... PORÉM, é Inevitável que o espaço livre já pré-existente nos extents já criados (e que contém dados) seja Ignorado...

IMPORTANTE : é CRUCIAL indicar que esse espaço não usado pelo DML em APPEND/DIRECT mode absolutamente *** NÃO ESTÁ PERDIDO ***, ela vai sim ser NATURALMENTE usado pelos próximos DMLs em modo normal...

Consequências : a quantidade de extents a ser lida/acessada num Full Table Scan vai subir, já que o espaço pré-existente foi ignorado e extents Adicionais foram criados.

Prevenção : usar tablespaces com segmentos de mesmo tamanho OU com tamanho Automaticamente criado pelo sistema, e restringir APPEND/DIRECT mode aos casos efetivamente necessários

Correção : recriação do segmento (via export+drop+import ou MOVE ou package DBMS_REDEFINITION) : aplicar APENAS e TÃO SOMENTE se Positivamente não vão haver DMLs em modo não-append/direct

E. extents em branco/sem dados presentes (normalmente no final do segmento mas não Obrigatoriamente)

Causas : a principal Causa é espaço retido após DELETEs (ou UPDATEs diminuindo os dados) que liberaram todos os blocos de extents – como já indicado no item sobre espaço em branco nos blocos, Novamente frisamos que o espaço sem dados mas ainda alocado para seu segmento original está Reservado para uso futuro , e vai SIM ser AUTOMATICAMENTE REUSADO nos próximos INSERTs/UPDATEs que o segmento sofrer, Não Sendo de forma alguma espaço "desperdiçado", "fragmentado" ou seja qual termo inapropriadamente se aplique.

APENAS e TÃO SOMENTE nas raras Ocasiões em que há uma regra de negócio 100% garantindo que o segmento/tabela não sofrerá mais DMLs é que se deve aplicar a Correção abaixo indicada.

Consequências : enquanto não for reutilizado esse espaço contido nos extents em branco VAI ser lido também em caso de full table scan, que devido à esse I/O adicional em princípio não necessário vai ser mais custoso, além de causar que o segmento aloque mais espaço do que o minimamente necessário

Prevenção : onde/quando viável, ao invés de DELETE usar o TRUNCATE (que zera o segmento de dados completamente, liberando assim todo o espaço em disco previamente alocado) – se necessário, em caso de remoção Parcial dos dados pode-se copiar para um outro destino temporário os dados que se quer manter, executar o TRUNCATE e depois re-inserir de volta os dados salvos.

Correção : recriação do segmento (via export+drop+import ou MOVE ou package DBMS_REDEFINITION) **OU** SHRINK (da tabela ou da tablespace toda).

OBS : o SHRINK simplesmente tenta identificar espaços sem uso que podem ser desalocados, enquanto o MOVE e ou DBMS_REDEFINITION realmente reconstroem o segmento de dados.

Detecção de Issues no Armazenamento / Espaço Alocado para Tabelas

Os principais meios que podemos utilizar para auxiliar na análise e efetuar a detecção dos tópicos anteriores são :

1. utilização da package DBMS_SPACE : essa built-in possui rotinas que reportam para um dado segmento o total geral de blocos , o total de blocos livres (ie, que já contiveram dados mas não mais) o total de blocos não-usados (ie, que nunca foram formatados), E indicações de preenchimento dos blocos, ie : blocos de 0 a 25% livres (condição Freespace 1 na rotina SPACE_USAGE da package), blocos de 25 a 50% livres (condição Freespace 2), blocos de 50 a 75% livres , e blocos acima de 75% livres (condições 3 e 4).
   Essa informação pode ser útil para para identificar taxa de alocação x uso efetivo de blocos, blocos não totalmente preenchidos não sendo reutilizados, etc.

   Além da documentação Oracle, exemplos de utilização podem ser encontrado em https://asktom.oracle.com/pls/apex/f?p=100:11:0::::P11_QUESTION_ID:231414051079 e em https://asktom.oracle.com/pls/apex/f?p=100:11:0::::P11_QUESTION_ID:4685088047969 .

2. Analyze e/ou Coleta de Estatísticas com DBMS_STATS : ao se coletar estatísticas para o CBO diversas informações físicas do segmento (como NUM_ROWS, AVG_ROW_LENGTH, EMPTY_BLOCKS) são coletadas, e com elas podemos ter uma noção/média do comprimento da linha/registro lógico (para detectarmos eventual ROW MIGRATION).

Além disso, no caso do ANALYZE especificamente possuímos opções para analisar itens físicos como ROW CHAINING (via ANALYZE nomedatabela LIST CHAINED ROWS, com o script fornecido pela Oracle UTLCHAIN.SQL listando as informações coletadas pelo ANALYZE), ou mesmo (o que não é de interesse direto no âmbito da "Fragmentação", mas é possível) validar a estrutura interna procurando por corrupção estrutural, via ANALYZE TABLE nomedatabela VALIDATE STRUCTURE;

Os scripts administrativos fornecidos pela Oracle residem em %ORACLE_HOME%/rdbms/admin.

OBS : além disto, é possível identificar se o sistema está acessando múltiplos blocos para recuperar uma só linha via estatísticas de sistema, que podem ser consultadas com uma query como essa :

```
select * from v$sysstat where name like 'table fetch continued row%';
```

3. Análise de conteúdo de bloco via DBMS_ROWID : o ROWID registra informações fisicas sobre o armazenamento de uma linha/registro da tabela, tais como o bloco, número do datafile, posição da linha, etc – extraindo a informação de blocos, podemos encontrar os blocos com contendo menos linhas do que o esperado...

A consulta básica seria algo similar a :

```
SELECT count(*) as "Rows per Block"
  FROM  &&v_tabname
 WHERE dbms_rowid.rowid_block_number(rowid) =
     (
     SELECT min(dbms_rowid.rowid_block_number(rowid))
       FROM &&v_tabname
     )
/
UNDEFINE v_tabname
```

Comprovado que existe uma Significativa quantidade de blocos contendo menos linhas do que o esperado dado o comprimento do registro lógico, devemos Analisar os itens expsots para encontrar a Causa...

4. análise das views internas (principalmente DBA_SEGMENTS e DBA_EXTENTS) : como indicado anteriormente, além de consultar a utilização dos blocos/extents já alocados E de analizarmos onde os dados estão residindo/identificarmos o porcentual de uso, queremos localizar os casos de extents não contíguos... A consulta básica seria tipo :

```
SELECT FILE_ID, EXTENT_ID, BLOCK_ID, BLOCKS, BLOCK_ID+BLOCKS INICIO_PROX_EXTENT_PREVISTO
  FROM DBA_EXTENTS
 WHERE OWNER='&V_OWNER' AND SEGMENT_TYPE='TABLE' AND SEGMENT_NAME='&V_NOME_TABELA'
 ORDER BY FILE_ID, EXTENT_ID
/
UNDEFINE V_OWNER
UNDEFINE V_NOME_TABELA
```

Pelo resultado anterior podemos encontrar casos em que o próximo EXTENT não começou no bloco previsto, E portanto algo impediu de iniciar o EXTENT nesse bloco... Pode ser um simples caso de outro segmento tendo alocado antes extent iniciando nesse bloco (o que por si só NÃO ACARRETA ISSUE ALGUMA em tablespaces LMT, ou no máximo pode causar ** mínima ** alocação não-reusável em casos extremos (vide caso A) , ou pode ser indicação de espaço não sendo reusado, talvez por INSERT em APPEND/DIRECT mode , devemos Analisar...

5. Outra consulta derivada da acima : podemos também utilizar a metodologia de identificar datafiles que possuem ‘buracos’ no meio, ie, blocos ou mesmo extents que não foram re-usados – INSISTO, não necessariamente isso é um problema, pois podem ter sido simplesmente resultado de DELETEs que liberaram esse espaço, que *** NÃO ESTÁ ‘Fragmentado’, não está WASTED/DESPERDIÇADO, e vai ser Naturalmente reusado – cabe Análise...

A consulta básica seria :

```
column sb   heading "Start Block"
column eb   heading "End Block"
column nsb  heading "NextExt Start Block"
column FILE_ID format 99999
accept v_tablespace prompt "Nome da Tablespace, assumirß %% e Maiusc:"
select * from
  (
   with x as (select file_id, block_id sb, block_id+blocks-1 eb,
                     lead(block_id,1) over(partition by file_id
                                           order by block_id) nsb
                from dba_extents
               where tablespace_name like '%' || upper('&v_tablespace') || '%'
               order by 1,2,3
             )
   select file_id, sb, eb, nsb, case when (nsb-eb) > 1 then 'HOLE - ' ||
      (nsb-eb) || ' blocks' else 'OK' end stat
     from x
  )
where stat <> 'OK';
/
```