

Melhore o desempenho de seu banco de dados Oracle

De vento em popa

Com uma configuração esperta e seleções SQL com velocidade otimizada é possível melhorar o desempenho do aplicativo de dados sem altos gastos com hardware.

POR BADRAN FARWATI

O Oracle V2, primeiro banco de dados relacional comercial tendo o SQL como linguagem de consulta, foi lançado no mercado em 1979. Um ano antes, nascia o avô de uma dinastia de CPUs, o 8086. Este chip continha, na época, 29.000 transistores e operava com uma velocidade de 8 MHz. Seu descendente atual, o Pentium IV, traz mais de 125 milhões de transistores e sua velocidade de processamento aumentou em mais de 3 GHz. Mesmo que o número de transistores não se converta diretamente em desempenho, pode-se chegar a um ganho de velocidade de alguns milhares por cento. Com uma CPU dessas, o desempenho para o usuário da décima e atual geração do Oracle não deveria sequer entrar em discussão.

Entretanto, a discussão ainda existe. Embora o número de transistores se duplique a cada ano e meio (lei de Moore), em velocidade semelhante cresce também a complexidade das tarefas executadas nos computadores, amplia-se a necessidade de recursos dos aplicativos e aumentam as expectativas dos usuários. Como resultado, o tema ajuste é cada vez mais atual para desenvolvedores, programadores e administradores de bancos de dados.

O ajuste nosso de cada dia

Para melhorar a velocidade do banco de dados, existe toda uma linha de estratégias promissoras. Uma discussão exaustiva daria para preencher livros e mais livros. Este artigo se concentrará nos aspectos facilmente aplicáveis no dia-a-dia e que não exijam grande conhecimento especializado, hardware ou software complementar. Apresentaremos dicas e fundamentos que podem ajudar qualquer administrador de banco de dados a:

- ➔ Criar índices de tabelas com desempenho melhorado;
- ➔ Formular instruções em SQL de maneira mais eficiente;
- ➔ Trabalhar com grandes tabelas com maior facilidade; e
- ➔ Administrar as conexões de usuários economizando recursos.

O desempenho de um banco de dados depende de inúmeros outros fatores. Claro que, entre eles, estão o hardware e o sistema operacional utilizados no servidor do banco de dados. A primeira regra de ouro no caso do Linux é, portanto, concentrar-se no essencial, desativar serviços supérfluos e não carregar módulos do kernel desnecessários.

Velocidade Máxima ou Freio de Mão Puxado?

O índice, elemento do banco de dados que foi desenvolvido especialmente para fins de ajuste, representa um papel fundamental no quesito desempenho. Entretanto, ele não reduzirá muito o tempo de resposta se usado de maneira errônea, podendo até mesmo transformar-se em um “freio de mão puxado”. Além das regras gerais para a construção de índices, deve-se também atentar a uma série de diretrizes específicas do Oracle.

Em primeiro lugar, pode-se mencionar os diferentes tipos de índice que o Oracle oferece. Além dos índices *Btree*, há também os índices de domínio (adequados para busca por texto integral; ver [1]). Os índices bitmap oferecem vantagens para tabelas cujas colunas tragam somente poucos valores que se relacionem a nenhuma ou a poucas colunas, e para consultas, cujas cláusulas *WHERE* contenham funções agregadas como *COUNT* ou *SUM* [1]. Nesses casos, o índice *Btree* utilizado por padrão não é ideal. Os índices bitmap não são adequados para tabelas ou colunas constantemente atualizadas. Nesse caso, eles podem até mesmo levar a aumento de custos e a perda de desempenho. Por

consequente, também não servem para OLTP (*Online Transaction Processing* – Processamento de Transações Online).

Dicas Rápidas

O exemplo a seguir ilustra o ganho que se pode obter com a escolha correta do tipo de índice: uma tabela de clientes com dois milhões de linhas contém uma coluna *Sexo*. Por sua natureza, essa coluna raramente é atualizada e somente pode aceitar dois valores. Nesse caso, um índice Btree utilizaria em média 29.696 kB e um índice bitmap somente 753 kB. Em um sistema de teste, o comando `COUNT` precisou de 1,01 segundos com o índice Btree nessa coluna, contra somente 0,03 milissegundos do índice bitmap.

Um índice deve evitar que um banco de dados compare cada linha com as consultas feitas. Contudo, quando mais de 10% das linhas da tabela forem devolvidos, um rastreamento completo da tabela pode ser mais rápido do que uma consulta com ajuda de índice. Quando possível, deve-se excluir o índice para esse tipo de coluna. Porém, se o índice for absolutamente necessário, pode-se forçar um rastreamento completo da tabela em uma seleção que não utilizará o índice com a instrução `FULL`. Exemplo: `SELECT /*+ FULL(nome_tabelas) */...` Entretanto, o Oracle somente entenderá a instrução caso o otimizador correspondente esteja configurado.

As tabelas que possuem poucas linhas também não devem ser indexadas, pois também nesse caso um rastreamento da tabela pode ser mais rápido que um acesso por meio de índice (porém, caso o usuário queira obter clareza com uma chave primária, deve se preparar para um desempenho provavelmente pior).

Se os tipos de dados se diferenciarem em uma coluna indexada e na cláusula `WHERE` da consulta relacionada à coluna, o índice não será utilizado pelo Oracle. Caso a tabela de clientes conte-

na uma coluna tipo `Tipo de Cliente VARCHAR2`, em que se utilize a consulta `WHERE tipo_de_cliente = 324`, então o tipo `NUMBER` não corresponderá ao tipo `VARCHAR2` na tabela. Dessa forma, o Oracle converte o tipo de coluna com `WHERE TO_NUMBER(tipo_de_cliente) = 324` e lê a tabela seqüencialmente, sem precisar da ajuda do índice.

Veículos de Função

Caso uma cláusula `WHERE` utilize funções como `SUBSTR`, `!=` (desigual), `NOT`, `TRUNC`, `LOWER`, `UPPER` ou um dos operadores aritméticos, um índice de Oracle comum será inútil. O banco de dados poderia, entretanto, consultar um índice formado com base na função utilizada na consulta. Para tal índice baseado em função é necessário aplicar o parâmetro `QUERY_REWRITE_ENABLED` em `init.ora` sobre `TRUE` e utilizar o otimizador baseado em custos (veja mais sobre isso neste artigo).

A [figura 1](#) demonstra o desempenho de uma consulta SQL, que utiliza a função `UPPER` na cláusula `WHERE`. No primeiro teste, a coluna na qual `UPPER` foi utilizada chegou a ser indexada, mas não baseada em função. O otimizador ignora totalmente esse índice (`TABLE ACCESS (FULL)`). A segunda consulta foi executada após o

```

SQL> create index laender_idx on laender(laender_name);
Index created.
SQL> select * from laender where upper(laender_name) = 'BRAZIL';
Execution Plan
-----
  0  SELECT STATEMENT Optimizer=ALL_ROWS (Cost=46 Card=2 Bytes=78)
  1  0  TABLE ACCESS (FULL) OF 'LAENDER' (TABLE) (Cost=46 Card=2 Bytes=78)

SQL> drop index laender_idx;
Index dropped.
SQL> create index laender_idx on laender(UPPER(laender_name));
Index created.
SQL> select * from laender where upper(laender_name) = 'BRAZIL';
Execution Plan
-----
  0  SELECT STATEMENT Optimizer=ALL_ROWS (Cost=8 Card=2 Bytes=78)
  1  0  TABLE ACCESS (BY INDEX ROWID) OF 'LAENDER' (TABLE) (Cost=8 Card=2 Bytes=78)
  2  1  INDEX (RANGE SCAN) OF 'LAENDER_IDX' (INDEX) (Cost=4 Card=2)

```

Figura 1: As consultas que contêm funções beneficiam-se somente de índices construídos com base na mesma função.

índice dessa coluna ter sido novamente construído com base na função `UPPER`. Agora o índice é aceito pelo otimizador (`INDEX (RANGE SCAN)`).

Economia com índices

O otimizador do Oracle utiliza somente índices aproveitáveis. Porém, o motor do SQL mantém todos os índices definidos para uma tabela, independentemente de serem utilizados ou não na execução de uma declaração de SQL. Isso custa recursos de I/O e sobrecarrega não somente a memória, mas também a CPU, pois o índice precisa ser organizado a cada inserção, atualização ou exclusão. Por isso é recomendável apagar índices

Listagem 1: startmi.sql

```

01 SET PAGESIZE 0
02 SET FEEDBACK OFF
03 SET VERIFY OFF
04 ACCEPT v_value CHAR PROMPT 'digitar nome_indice ou "todos" para executar o monitoramento: '
05 SPOOL start_monitoring_idx.sql
06 SELECT 'ALTER INDEX ' || lower(nome_indice) || ' MONITORING USAGE;'
07 FROM user_indexes
08 WHERE index_type = 'NORMAL' AND nome_indice LIKE decode(UPPER('&v_value'), 'ALLE', '%', UPPER('&v_value'));
09 SPOOL OFF
10 @START_MONITORING_IDX.SQL;
11 SET FEEDBACK ON
12 SET VERIFY ON
13 SET PAGESIZE 24

```

que não são utilizados [2]. Com `ALTER INDEX nome_indice MONITORING USAGE;` pode-se verificar se um índice é utilizado ou não. A **listagem 1** fornece um script SQL com o nome `startmi.sql`, com o qual todos os índices de um esquema são colocados sob monitoramento.

Na visualização `V$OBJECT_USAGE` encontram-se, após a execução do script, informações sobre todos os índices colocados sob monitoramento. Na coluna `MON` dessa visualização pode-se verificar se esse índice está momentaneamente monitorado; as colunas `START_MONITORING` e `STOP_MONITORING` informam o período no qual ele esteve sob monitoramento e a coluna `USE` diz se o índice foi utilizado nesse espaço de tempo. Quando um índice não foi utilizado, claro que não se pode automaticamente considerá-lo supérfluo. É também possível que certas consultas (como relatórios trimestrais) somente ocorram em grandes intervalos de tempo. Para interromper o monitoramento, o comando `ALTER INDEX nome_indice NOMONITORING USAGE` deve ser executado.

Controle de fragmentação

Os índices Btree podem ser fragmentados com o tempo se houver muitas atualizações ou inserções. Sabendo disso, pode-se informar o parâmetro `PCTFREE` durante a construção de um índice (e somente nesse momento, não posteriormente), para reservar espaço livre adicional para atualizações futuras. Por padrão, ele ocupa 10% do tamanho de um bloco. O grau de fragmentação é determinado com a seguinte instrução: `ANALYZE INDEX nome_indice VALIDATE STRUCTURE;`. Esse comando reúne para a seção atual informações sobre o índice informado e o salva na tabela `INDEX_STATS`. Após o fim da seção essa informação é excluída novamente. Caso se queira mantê-la, ela deve ser copiada para uma outra tabela. Os resultados encontram-se na seguinte instrução usando o comando `SELECT`:

```
SELECT nome, del_lf_rows, lf_rows -- del_lf_rows lf_rows_used, 2
TO_CHAR(del_lf_rows / (lf_rows)*100, '999.99999') ratio FROM 2
index_stats WHERE nome = UPPER('nome_indice');
```

Caso o valor na coluna `DEL_LF_ROWS` tenha mais de 15% a 20% do valor na coluna `LF_ROWS_USED`, será necessária a reconstrução do índice.

Dica: Caso uma grande quantidade de conjuntos de dados de uma tabela precise ser excluída em um sistema de produção, recomenda-se apagar o índice, iniciar o processo de exclusão e finalmente reconstruir o índice. Se o índice não for apagado, será necessário atualizá-lo após cada exclusão de um conjunto de dados.

Acelere a construção

Os índices do Oracle sempre são influenciados por parâmetros. Muitos deles podem fazer uma grande diferença no desempenho do servidor. Esses parâmetros podem ser alterados posteriormente com o comando `ALTER INDEX nome_indice REBUILD ONLINE;`. Desse modo o índice é mantido até que se faça uma cópia. A vantagem é que a tabela base pode ser continuamente consultada ou atualizada. Entretanto, a cópia exige muito em termos de poder de processamento e não é recomendada quando for necessário carregar, de uma só vez, dados que representem aproximadamente de 20% a 30% do total de informações da tabela.

Os parâmetros de índice que influenciam o desempenho do banco de dados são:

➤ **PARALLEL n** Este parâmetro permite acessos paralelos a uma tabela. Quando um índice é criado, a tabela relacionada é verificada por completo para compilar os ROWIDs de todos os conjuntos de dados. Assim, o parâmetro `n` deve corresponder ao número de CPUs menos 1. Com isso, uma máquina com quatro CPUs produziria a seguinte declaração `CREATE INDEX`:

```
CREATE INDEX nome_indice ON nome_tabela(nome_coluna) PARALLEL 3;
```

➤ **NOLOGGING** Este parâmetro evita a geração no *redo log* e aumenta a velocidade em até 30%. Entretanto, caso haja uma falha do servidor durante a geração do índice, o processo deve ser reiniciado. Exemplo:

```
CREATE INDEX nome_indice ON nome_tabela(nome_coluna) PARALLEL 3 2
NOLOGGING;
```

Ajuste em partes

Desde as versões 8 e 8i do Oracle é possível segmentar uma tabela em vários blocos de linhas. As unidades menores que surgem da segmentação são denominadas partições da tabela. O índice também é segmentado de maneira correspondente. Mesmo em tabelas grandes, isso melhora em muito o desempenho, pois a partir de então não será mais necessário buscar em toda a tabela para uma consulta, mas somente nas partições relacionadas.

Há quatro métodos diferentes de particionamento. A primeira possibilidade é o particionamento por blocos. Nesse modo, escolhe-se uma coluna da tabela como chave de partição. Os valores dessa coluna serão divididos em blocos e classificadas pelas partições individuais. A **listagem 2** exhibe um exemplo de como uma tabela particionada é criada e indexada de acordo com esse método.

Esse tipo de partição é razoável quando a faixa de valores da chave é estática, se for possível dividir os dados de maneira lógica e se pudermos distribuí-los igualmente (de forma que as partições sejam também mais ou menos do mesmo tamanho).

Quando não há uma chave adequada para uma partição por blocos, um particionamento por dispersão (*hash*) pode ser escolhido. Nesse caso, os valores seqüenciais não serão necessariamente armazenados na mesma partição. As linhas das tabelas serão distribuídas igualmente nas partições por uma função de dispersão (*hash*).

O terceiro e relativamente novo método por *listagem* (a partir do Oracle 9i) permite ao administrador controlar exatamente os dados que serão armazenados e para quais partições eles irão. Esse método é adequado para chaves descontínuas, como nomes de produtos ou nomes de países. Para esse tipo de particionamento, deveria ser reservada uma partição padrão que incluísse a chave indeterminada (como um novo produto). Exemplo:

```
CREATE TABLE teste_para_pais(
...
pais VARCHAR2(100),
...)
PARTITION BY LIST (pais) (
PARTITION EU VALUES ('ALEMANHA', 'FRANCA'),
PARTITION ORI_MED VALUES ('PALESTINA', 'LIBANO'),
PARTITION ORIENTE VALUES ('JAPAO', 'CHINA'),
PARTITION outros VALUES (DEFAULT));
```

Quando as quantidades de dados forem muito grandes, as partições por blocos podem ser divididas em subpartições com o método *COMPOSITE*. Nesse caso, cada partição por bloco será mais uma vez dividida (seja com ajuda do método *por blocos* ou pelo método *por listagem*).

SQL com Nitro

Toda as consulta SQL realizadas são armazenadas em cache no chamado *shared pool*. A cada nova consulta, o Oracle examina primeiro se uma seleção idêntica já não está disponível em cache; nesse caso, o banco de dados poderia evitar a análise repetida (*hard parse*). Entretanto, o programa precisa analisar também a nova declaração quando ela se diferencia somente em parte da expressão de uma outra já armazenada.

Por exemplo, muitas consultas que seguem o modelo `SELECT * FROM colaborador WHERE nome = "Nome"`, que se diferenciam somente no nome, sempre seriam processadas e armazenadas separadamente. Dessa forma, o cache obviamente estaria sendo utilizado com menor eficiência, desperdiçando também espaço em disco.

Esses problemas podem ser resolvidos de forma simples, com mais economia de recursos e melhor desempenho, bastando utilizar na consulta uma variável para a parte alterada (no exemplo acima, no lugar do nome). Com essa técnica (vinculação SQL), a seleção do exemplo seria assim: `SELECT * FROM colaborador WHERE nome =:VAL;` A seleção seria então armazenada em cache e utilizada para todas as consultas de acordo com esse modelo.

Caso se queira executar essa consulta no SQLPlus, deve-se primeiro definir a variável e então atribuir-lhe um valor:

```
SQL> VAR nome VARCHAR2(20)
SQL> Exec :nome := 'Farwati'
PL/SQL procedure successfully completed.
SQL> SELECT * FROM colaborador WHERE nome = :nome;
```

Muitas linguagens de script implementaram funções para a interface do Oracle, na qual se pode definir uma variável de vinculação. O PHP, por exemplo, disponibiliza a função `OCIBindByName()`. [3] Exemplo:

```
<?
$db = OCILogin('userid', 'password', 'mydb');
$stmt = OCIParse($db, 'SELECT * FROM colaborador WHERE nome = :nome');
$nome = 'Farwati';
OCIBindByName($stmt, ':nome', $nome, -1);
OCIExecute($stmt);
?>
```

No PL/SQL isso tudo é muito fácil. Ele utiliza basicamente apenas variáveis de vinculação. Como:

```
CREATE OR REPLACE PROCEDURE hole_colaborador(p_nome in VARCHAR2()) AS
BEGIN
    SELECT * FROM colaborador WHERE nome = p_nome;
END;
```

Listagem 2: Aplicando uma partição por blocos.

```
01 CREATE TABLE particao_teste ( name VARCHAR2(80), alt NUMBER(3) ) PARTITION BY RANGE(nome) (PARTITION part1 VALUES LESS THAN('M') ↗
    TABLESPACE ts0, PARTITION part2 VALUES LESS THAN('Z') TABLESPACE ts1);
02 CREATE INDEX particao_teste_idx ON particao_teste(nome) LOCAL ( PARTITION part1 TABLESPACE ts0, PARTITION part2 TABLESPACE ts1);
```

Isso funciona, uma vez que cada referência no PL/SQL já é uma variável de vinculação. Mesmo quando se utilizam valores literais (como, por exemplo, a consulta `SELECT * FROM colaborador WHERE nome = "Farwati";`) em vez de variáveis no código PL/SQL, uma variável de vinculação sempre será automaticamente usada.

A propósito, o desempenho também é melhorado quando o programador limita o tipo de dados de uma variável à capacidade ou exatidão real necessária; do contrário, o Oracle utilizará a capacidade máxima do tipo (por exemplo, quase 38 casas decimais em `NUMBER`) e com isso desperdiça, de certo modo, espaço e tempo de execução.

Baixe os custos

Toda declaração SQL é primeiramente analisada pelo interpretador SQL e em seguida definida pelo otimizador. Este transmite um plano ao gerador e a declaração é finalmente executada pelo mecanismo SQL (Figura 2).

O otimizador tem à disposição (até o Oracle 9i) basicamente dois métodos: O método baseado em regras (RBO – *Rule Based Optimizer* – Otimizador Baseado em Regras) e o método baseado em custo (CBO – *Cost Based Optimizer* – Otimizador Baseado em Custo). O otimizador baseado em regras utiliza 15 regras de prioridade pré-definidas (veja mais em [4]). Em contrapartida, o método base-

ado em custo não segue nenhuma regra pré-estabelecida, mas informa o melhor caminho de acesso possível baseado em estatísticas que o usuário deve criar explicitamente. Caso nenhuma estatística esteja disponível e o otimizador não tenha boas bases de decisão, o desempenho de uma consulta provavelmente será muito ruim.

Exemplo: Uma tabela `pedidos` contém uma coluna indexada chamada `produção`, que pode somente armazenar dois valores, `completo` e `em_produção`. A tabela armazena os pedidos de um mês e tem mais de um milhão de linhas. Os pedidos são finalizados geralmente em poucas horas, por isso nunca há, em média, mais do que uma centena de pedidos abertos. Para uma consulta que possa encontrar todos os pedidos abertos, o CBO escolhe um entre dois caminhos de execução possíveis, conforme exista ou não uma estatística para o índice da coluna `produção`.

O CBO sabe que há somente dois valores para essa coluna. Quando não há estatística à disposição, ele considera que os valores são divididos mais ou menos igualmente e, portanto, opta por utilizar o índice, embora no caso fosse muito mais rápido realizar uma varredura completa da tabela. Caso o CBO conheça a divisão de valores com base em uma estatística anteriormente calculada (1:10.000), estará bem ciente de que o índice será a opção mais lenta.

Escolha suas armas

O usuário do Oracle (até a versão 9i) pode informar no arquivo `init.ora` qual dos dois métodos deverá ser utilizado pelo otimizador; o Oracle utiliza o RBO quando for inserido no arquivo `OPTIMIZER_MODE = RULE`. Entretanto, se a variável for aplicada ao valor `CHOOSE`, a escolha do método dependerá de diversos fatores: se nenhuma estatística tiver sido criada, o Oracle utilizará o RBO. Caso sejam incluídas estatísticas, o Oracle utilizará o CBO. Além disso, o usuário pode definir entre as instruções, além dos níveis de aplicação, o método do otimizador (por exemplo, `SELECT /*+ RULE */`). A versão 10g não fornece mais suporte para o RBO, de forma que a variável `OPTIMIZER_MODE` não é mais necessária. As aplicações que executam a otimização baseada em regra precisarão ser reescritas.

Até antes do Oracle 9i o CBO conhecia somente as variantes de otimização `ALL_ROWS` e `FIRST_ROWS`. Na `ALL_ROWS`, o otimizador tenta encontrar, com base em custo e com recursos mínimos, a melhor taxa de transferência para uma declaração integral de SQL. Com a `FIRST_ROWS`, o otimizador utiliza uma mescla de método baseado em custo e heurística, com o objetivo de encontrar o primeiro par de linhas o mais rápido possível. Entretanto, em muitos casos o método heurístico pode fazer com que o plano de execução se torne fundamentalmente mais pesado com relação ao custo.

Dessa forma, desde a versão 9i o Oracle traz a possibilidade de dar ao otimizador a instrução `FIRST_ROWS_n`, pela qual `n` pode aceitar os valores 1, 10, 100 ou 1.000. Com esse parâmetro, sem levar em conta as estatísticas, é utilizado um método baseado em custo que otimiza a resposta de 1, 10, 100 ou 1.000 primeiras linhas.

Listagem 3: Seleções Longas

```
SELECT sql_text , executions , disk_reads / decode(executions, 0, 1, executions) / 300 "resposta"
FROM v$sql
WHERE disk_reads / decode(executions,0,1,executions)/300 >15
AND executions > 0
ORDER BY hash_value, child_number;
SELECT sql_text , executions , buffer_gets / decode(executions, 0, 1, executions) / 4000 "resposta"
FROM v$sql
WHERE buffer_gets / decode(executions,0,1,executions)/4000 >15
AND executions > 0
ORDER BY hash_value, child_number;
```

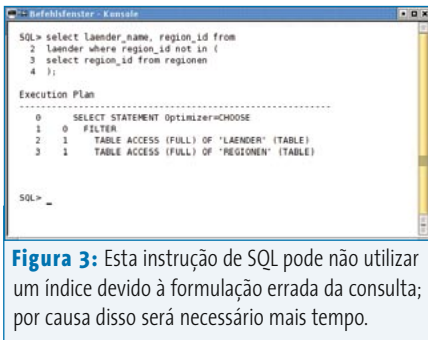


Figura 3: Esta instrução de SQL pode não utilizar um índice devido à formulação errada da consulta; por causa disso será necessário mais tempo.

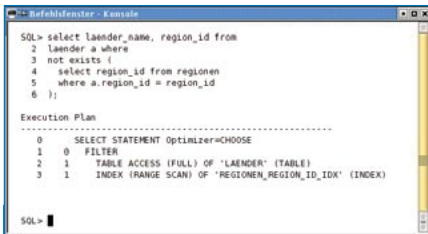


Figura 4: A mesma seleção de SQL, pode utilizar primeiramente o índice formulado dessa maneira e ganhar muito em velocidade.

Com o comando: `ANALYZE TABLE nome_tabela ESTIMATE COMPUTE STATISTICS` são reunidas estatísticas sobre a tabela e todos os seus índices. A palavra-chave `COMPUTE`, no comando acima, faz com que o Oracle crie estatísticas para todas as linhas da tabela. Caso a tabela seja grande, a análise pode ser bastante demorada. Por isso é possível definir, com a palavra-chave `ESTIMATE`, que a consulta se limite a apenas uma parte da tabela.

As estatísticas são calculadas regularmente. Entretanto e acima de tudo, logo após um novo índice ter sido construído, o volume de dados ou a estrutura de um objeto logo é alterada. O pacote `DBMS_STATS` pode fazer uma cópia de segurança da última estatística. Dessa forma, ao menos os dados antigos ficam à disposição do CBO no caso de algo errado ocorrer em um novo cálculo. A consulta SQL mostrada a seguir fornece a última data de análise para todas as tabelas do esquema:

```
SQL> SELECT nome_tabela, num_rows, 2
last_analyzed FROM user_tables;
```

Pegue um atalho

O SQL é uma linguagem muito flexível. Diversas declarações compostas podem levar aos mesmos resultados. Entretanto, o tempo necessário para isso pode variar de maneira drástica.

```
SELECT titulo, autor, editora FROM 2
titulo, editora WHERE autor = 'XXX' AND 2
editora = 'YYY';
```

No caso desta simples consulta, obriga-se o banco de dados a atribuir automaticamente as colunas às tabelas corretas. Entretanto, o tempo necessário para essa consulta pode ser reduzido caso se utilize “apelidos” explicitamente:

```
SELECT t.titulo, t.autor, e.editora FROM 2
titulo t, editoras e WHERE t.autor = 'XXX' 2
AND e.editora = 'YYY';
```

Na **figura 3** um comando `TABLE ACCESS (FULL)` é executado em ambas as tabelas com base na declaração de SQL. Em contrapartida, a **figura 4** mostra o mesmo resultado como produto de uma outra consulta. Nesse caso, o índice `REGIONEN_REGION_ID_IDX` pode ser utilizado.

Com o comando `EXPLAIN PLAN` o usuário pode exibir o caminho de execução do otimizador do SQL. No `Sqlplus`, o parâmetro `AUTOTRACE` está disponível para criar um relatório desse tipo automaticamente: `SET AUTOTRACE ON /OFF/ TRACEONLY`. Com `TRACEONLY` o relatório é ativado. Contudo, o resultado e o plano de execução não serão emitidos. Para tanto, o administrador precisa criar o papel `PLUSTRACE` e atribuí-lo ao usuário em operação:

```
Conn / as sysdba
GRANT PLUSTRACE TO nome_usuario;
```

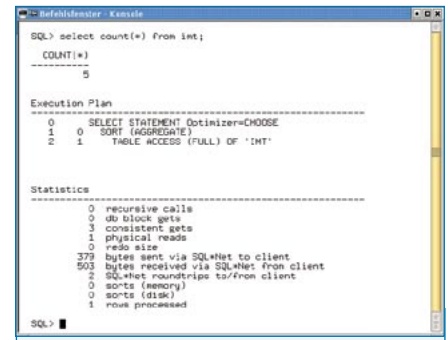


Figura 5: Estatística de uma instrução de SQL.

Por fim, ele criará a tabela `PLAN` com auxílio do script `utlpxplan.sql`, que se encontra no diretório `$ORACLE_HOME/rdbms/admin`. Dessa maneira, torna-se possível visualizar as estatísticas de usuário e o caminho de execução para determinadas consultas.

Com a declaração `SET AUTOTRACE TRACEONLY STATISTICS` no `Sqlplus` são criadas estatísticas de dois importantes aspectos do desempenho do banco de dados: `disk reads`, que indica a velocidade de leitura do disco, e `buffer gets`, que mostra aproximadamente o quanto se usou de recursos da CPU. Caso os custos para a declaração planejada se mostrem relativamente altos, é necessário executar uma otimização (Como esses valores dependem de muitos parâmetros, não podemos indicar aqui um valor comum.

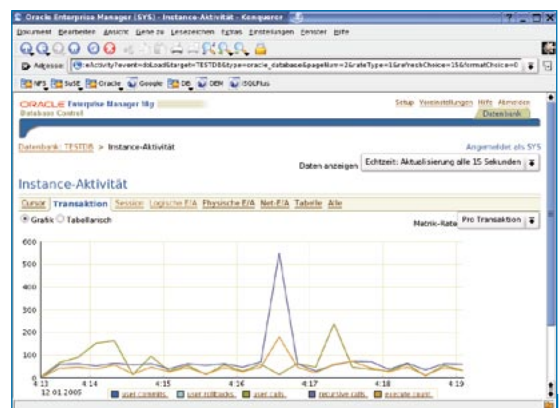


Figura 6: O console de administração (*Enterprise Manager*) da última versão do Oracle oferece diferentes diagramas nos quais se pode verificar a utilização de recursos, o tempo de resposta, a taxa de transferência e outras medidas relevantes de desempenho.

Por exemplo, seriam relativamente elevados os custos computacionais quando uma declaração retornasse 1.000 linhas e precisasse executar cerca de 1.000.000 *consistent gets* e 10.000 *physical reads* (veja a [figura 4](#)).

Dessa forma, pode-se pesquisar diferentes variantes de uma consulta e decidir pela que lhe oferece a melhor relação custo-benefício. A [listagem 3](#) ajuda a identificar instruções SQL ruins, que levam mais de 15 segundos para executar (isso em nosso exemplo, num servidor que pode executar 300 operações de I/O de disco e 4.000 leituras de buffer (*buffer reads*) por segundo).

Conexões de primeira

O Oracle inicia um processo exclusivo para cada nova conexão de usuário. Quando, por exemplo, 30 funcionários que trabalham com o banco de dados estão logados e outros 300 usuários acessam o banco de dados indiretamente através da aplicação web que utilizam, são iniciados 330 processos do Oracle – isso quando se trata de conexões conservadoras, já que o quadro pode ser muito pior nos casos em que o sistema abre mais de uma conexão para cada usuário, e isso não é incomum. Qualquer que seja a justificativa, isto é um baita desperdício de recursos.

Por essa razão o Oracle disponibiliza uma solução de *shared server* (*servidor compartilhado* – sucessor do servidor *multi-threaded*), que mantém a quantidade de processos bastante reduzida e administra as conexões e consultas como um grande *pool*. Isso economiza muito em recursos, especialmente para sistemas OLTP.

Ajuste a memória

Vale como regra prática: quanto maior a SGA (*System Global Area* – área global do sistema), melhor o desempenho, contanto que se mantenha espaço suficiente na memória para outros processos e que a SGA nunca precise ser trocada.

Listagem 4

```

01 SET SERVEROUTPUT ON
02 SET LINESIZE 1000
03 SET FEEDBACK OFF
04 SELECT name, created, log_mode, open_mode
05 FROM v$database;
06 prompt
07 prompt #####
08 prompt
09 DECLARE
10     v_value NUMBER;
11
12 BEGIN
13
14     -- Dictionary Cache Hit Ratio
15
16     SELECT round(sum(getmisses) / (sum(gets)+0.00000000001) * 100,2)
17     INTO v_value
18     FROM v$rowcache;
19
20     DBMS_Output.Put('Dictionary Cache Hit Ratio: ' || v_value || '% ');
21     IF v_value > 10 THEN
22         DBMS_Output.Put_Line('aumento dos parâmetros do SHARED_
23 POOL_SIZE para alcançar um hit ratio inferior a 10% ');
24     ELSE
25         DBMS_Output.Put_Line('Wert ist OK. ');
26     END IF;
27
28     -- Dictionary Cache Hit Ratio
29
30     SELECT round(sum(RELOADS)/sum(pins) *100,2)
31     -- (1 -(Sum(reloads)/(Sum(pins) + Sum(reloads)))) * 100
32     --SELECT sum(pinhits) / sum(pins)
33     INTO v_value
34     FROM v$librarycache;
35
36
37     DBMS_Output.Put('Dictionary Cache Hit Ratio: ' || v_value || '% ');
38     IF v_value > 1 THEN
39         DBMS_Output.Put_Line('aumento do SHARED_POOL_SIZE para
40 alcançar um hit ratio inferior a 1% ');
41     ELSE
42         DBMS_Output.Put_Line('Wert ist OK. ');
43     END IF;
44
45
46     -- Dispatcher Workload
47
48     SELECT (Sum(busy) / (Sum(busy) + Sum(idle))) * 100
49     INTO v_value
50     FROM v$dispatcher;
51
52     DBMS_Output.Put('Dispatcher Workload : ' || round(v_value) || '% ');
53     IF v_value > 50 THEN
54         DBMS_Output.Put_Line('aumento do número de MTS_
55 DISPATCHER traz Performa');
56     ELSE
57         DBMS_Output.Put_Line('Wert ist OK. ');
58     END IF;
59
60 END;
61 /

```

A SGA abriga o *shared pool*, um espaço de memória utilizado por todos os usuários do Oracle ao mesmo tempo. Ele se compõe de três partes: a *library cache*, que armazena temporariamente todas as consultas para uma eventual recuperação, o *dictionary cache* e o *large pool*, que é utilizado pelo servidor compartilhado, pelas consultas paralelas e pelo gerenciador de recuperação.

Por meio de um dimensionamento otimizado desses espaços de memória, o administrador pode conseguir a permanência das declarações de SQL freqüentemente executadas na *library cache* sem utilizar muito espaço na memória. De grande ajuda no ajuste de tamanho da memória é a visualização em `V$LIBRARYCACHE`.

A coluna `PINS` dessa visualização exibe a quantidade de solicitações por namespace. Os `PINHITS` são as consultas encontradas na *library cache*. Na coluna `RELOADS` pode-se consultar a freqüência na qual uma declaração de SQL já retirada do cache precisa ser recarregada. Em um tamanho ótimo de *shared pool*, o número de recarregamentos será por volta de zero. A coluna `INVALIDATIONS` mostra a freqüência de invalidações dos dados na *library cache* quando eles precisaram ser re-analisados. Em um ambiente ideal, o tamanho aqui também deve ser zero.

```
SELECT namespace, pins, pinhits, reloads,
invalidations FROM V$LIBRARYCACHE ORDER
BY namespace;
```

A **figura 5** mostra o resultado dessa consulta. Havia 230 recarregamentos e duas declarações de SQL inválidas. A relação entre `RELOADS` e `PINS`, a `HIT RATIO`, é uma boa medida do desempenho da *library cache*. Os recarregamentos não devem ultrapassar 1% dos `PINS`. A **listagem 4** traz um script que pesquisa a taxa de *hit ratio* (freqüência na qual

encontramos os dados) de uma instância no *dictionary cache*, na *library cache* e no *dispatcher workload*.

Uma outra chave é o espaço de memória livre no *shared pool* durante o pico de carga. Este pode ser consultado por `V$SGASTAT`. A situação ótima é quando o mínimo de memória possível permanece livre (ou seja, ela está sendo usada ao máximo) e, ainda assim, não há sobrecarga.

```
SELECT * FROM V$SGASTAT WHERE name =
'free memory' AND pool = 'shared pool';
```

Com o parâmetro `SHARED_POOL_SIZE` no arquivo `init.ora`, seu tamanho (em bytes) é controlado. Esse valor pode ser alterado somente após um período adequado de monitoramento.

O *large pool*, utilizado para consultas paralelas, para `RMAN` (backup e gerenciador de recuperação) e para o servidor compartilhado, também precisa ser configurado quando um desses três serviços é utilizado. É possível configurá-lo por meio do parâmetro `LARGE_POOL_SIZE` no arquivo `init.ora`. A fórmula para a medição do tamanho do *large pool* é a seguinte: `UGA * número de usuários = Tamanho do Large Pool`.

A `UGA` (*User Global Area* – área global de usuário) é calculada automaticamente através da análise de utilização da memória e do cursor. O cálculo pode ser avaliado com a seguinte consulta:

```
SELECT SUM(value) tamanho , name From
V$SESSTAT, V$STATNAME WHERE NAME LIKE
'session uga memory%' AND V$SESSTAT
statistic# = V$STATNAME.statistic#
group by name;
```

A média da `session uga memory` e o `session uga memory max` para todas as sessões durante um pico de carga constitui o tamanho das variáveis `LARGE_POOL_SIZE`.

Resumo da ópera

Muitos fatores influenciam o desempenho do banco de dados. Além de um dimensionamento ótimo da SGA, pode-se aumentar a velocidade, sobretudo por meio de índices customizados e pela construção conscienciosa de uma declaração de SQL. ■

Pequena história do Oracle

A primeira versão do *Oracle Relational Database Management System* (Oracle RDBMS) foi lançada em 1979 pela então *Relational Software Inc.* (RSI). O chamado Oracle V2 (nunca houve uma versão 1) não tinha suporte à transações, mas implementava recursos básicos da linguagem SQL, como *queries* e *joins*. A versão 3 foi lançada em 1983, após a empresa mudar seu nome para *Oracle Corporation*, e já suportava *commits* e *rollbacks*, além de ser a primeira a rodar em sistemas UNIX. A primeira versão para Linux foi a 8i, em 1999. Atualmente, o *Oracle 10g*, roda em 13 plataformas diferentes.

INFORMAÇÕES

- [1] B. Farwati, *Busca Completa de Texto com Oracle*, LINUX Magazine alemã, Ed. 10/04 p.70
- [2] Guia de Ajuste de Desempenho de Banco de Dados e Referência - Segunda Edição (9.2 - em inglês), out. 2002. *Part number: A96533-02* ;
- [3] Referência da função `ocibindbyname` (artigo em alemão): at.php.net/manual/en/function.ocibindbyname.php
- [4] Otimizador Baseado em Regra (em inglês): download-west.oracle.com/docs/cd/B10501_01/server.920/a96533/rbo.htm
- [5] Processos de Gerenciamento em Oracle (página em inglês) www.cs.umb.edu/cs634/ora9idocs/server.920/a96521/manproc.htm

SOBRE O AUTOR

Badran Farwati trabalha como Administrador e Programador de Banco de Dados na Biblioteca Nacional da Áustria. Ele pode ser contatado (em inglês) através do email b.farwati@kolkhos.net.