DeepL

Subscribe to DeepL Pro to translate larger documents.
Visit www.DeepL.com/pro for more information.

GlobalLogic®

Afinação do

Shalabh Mehrotra,
Arquitecto de Soluções
Sénior

O desempenho da base de dados é um dos aspectos mais desafiantes das operações da base de dados de uma organização, e a sintonia SQL pode ajudar a melhorar de forma significativa a saúde e o desempenho de um sistema. Este livro branco demonstra como melhorar o desempenho de uma linguagem de

Tabela de Conteúdos

Introdução 3
Orientações Gerais de Sintonia SQL 3
- Escrever Perguntas Sensatas 3
- Subconsultas de afinação 3
- Limitar o número de mesas de uma adesão 3
- As chamadas recursivas devem ser mantidas a um nível mínimo 3
- Melhorar a Velocidade Parcial 4
- Utilização do pseudónimo 4
- Tabelas de Condução 4
- Tabelas de Caching 5
- Evitar o uso de Seleccionar * Cláusulas 5
- Existente vs. Em 5
- Não Existente vs. Não In 5
- In com Menos vs. Não In para Colunas Não-Indexas 6
- Subconsultas relacionadas vs. Vistas em linha 6
- Vistas Utilização 6
- Utilizar a descodificação para reduzir o processamento 6
- Desigualdades 7
- Utilização da União no lugar de Ou 7
- Usando a União Tudo em vez da União 7
- Influenciando o Optimizer Usando Dicas 7
- Verificação de presença 7
- Utilização de índices para melhorar o desempenho 8
- Porque não são utilizados índices 8

Conclusão 9
Referências 9

Introdução

O desempenho da base de dados é um dos aspectos mais desafiantes das operações da base de dados de uma organização. Uma aplicação bem concebida pode ainda ter problemas de desempenho se a SQL que utiliza for mal construída. É muito mais difícil escrever SQL eficiente do que escrever SQL funcionalmente correcta. Como tal, a afinação de SQL pode ajudar a melhorar significativamente a saúde e o desempenho de um sistema. A chave para sintonizar SQL é minimizar o caminho de pesquisa que a base de dados utiliza para encontrar os dados.

O público-alvo deste whitepaper inclui programadores e administradores de bases de dados que queiram melhorar o desempenho das suas consultas SQL.

Sintonia SQL Geral Directrizes

Os objectivos de escrever qualquer declaração SQL incluem a entrega de tempos de resposta rápidos, utilizando o mínimo de recursos CPU, e alcançar o menor número de operações de E/S. O conteúdo seguinte fornece as melhores práticas para optimizar o desempenho SQL.

Escreva Sensible Consultas

Identificar instruções SQL que estão a demorar muito tempo a executar. Identificar também instruções SQL que envolvem a junção de um grande número de grandes tabelas e junções externas. A forma mais simples de o fazer envolve normalmente a execução das instruções individuais usando SQLPlus e a sua calendarização (SET TIMING ON). Use EXPLAIN para ver o plano de execução da declaração. Procure qualquer acesso completo à tabela que pareça duvidoso. Lembre-se, um scan completo de uma pequena tabela é frequentemente mais eficiente do que o acesso por rowid.

Verificar se existem quaisquer índices que possam ajudar o desempenho. Uma forma rápida de o fazer é executar a declaração utilizando o Optimizador Baseado em Regras (RBO) (SELECT /*+ RULE */). Sob o RBO, se um índice estiver presente, ele será utilizado. O plano de execução resultante pode dar-lhe algumas ideias sobre quais os índices com que deve brincar. Poderá então remover a dica RULE e substituí-la pelas dicas específicas de índice que desejar. Desta forma, o Optimizador Baseado em Custos (CBO) continuará a ser utilizado para acessos a tabelas onde as dicas não estejam presentes. Lembre-se, se os volumes de dados mudarem ao longo do tempo, a dica que ajudou pode

tornar-se um estorvo! Por esta razão, as dicas devem ser evitadas, se possível, especialmente a /*+ RULA */ dica.

Tente adicionar novos índices ao sistema para reduzir o excesso de digitalizações de tabelas completas. Tipicamente, as colunas chave estrangeiras devem ser indexadas, uma vez que estas são regularmente utilizadas em condições de junção. Ocasionalmente pode ser necessário adicionar índices compostos (concatenados) que apenas ajudarão as consultas individuais. Lembre-se, uma indexação excessiva pode reduzir o desempenho INSERT, UPDATE e DELETE.

Afinação Subconsultas

Se o SQL contém subconsultas, afine-as. De facto, sintonize-as primeiro. A consulta principal não funcionará bem se as subconsultas não puderem funcionar bem por si próprias. Se um join lhe proporcionar a funcionalidade da subconsulta, tente primeiro o método join antes de tentar o método de subconsulta. Preste atenção às subconsultas correlacionadas, pois tendem a ser muito dispendiosas e insentivas em termos de CPU.

Limite o Número de Tabelas de uma adesão em

Há vários casos em que o tempo de processamento pode ser reduzido várias vezes, quebrando a instrução SQL em instruções menores e escrevendo um bloco PL/ SQL para reduzir as chamadas à base de dados. Além disso, os pacotes reduzem a E/S, uma vez que todas as funções e procedimentos relacionados são armazenados em cache em conjunto. Utilize o pacote DBMS_SHARED_POOL para localizar uma área SQL ou PL/SQL. Para fixar um conjunto de pacotes à área SQL, iniciar a base de dados e fazer uma referência aos objectos que os fazem ser carregados. Use DBMS_SHARED_POOL.KEEP para o pinar.
O descaroçamento evita a fragmentação da memória. Também ajuda a reservar a memória para programas específicos.

As chamadas recursivas devem ser mantidas a um mínimo de

As chamadas recursivas são instruções SQL que são desencadeadas pela própria Oracle. Uma grande

quantidade de SQL recursiva executada pelo SYS poderia indicar actividades de gestão de espaço, tais como a extensão das atribuições que ocorrem. Isto não é
escalável e com impacto no tempo de resposta do utilizador. SQL recursiva executada sob outro ID de utilizador é provavelmente SQL e PL/ SQL, e isto não é um problema.

O Oracle trace utility tkprof fornece informação sobre chamadas recursivas. Este valor deve ser tido em consideração ao calcular a necessidade de recursos para um processo. O tkprof também fornece o cache da biblioteca e fornece o nome de utilizador do indivíduo que executou a instrução SQL. As estatísticas geradas pelo Tkprof podem ser armazenadas numa tabela tkprof_table para serem consultadas mais tarde.

Melhorar o Parse Velocidade

Os planos de execução das declarações SELECT são armazenados em cache pelo servidor, mas a menos que a mesma declaração seja repetida, os detalhes do plano de execução armazenados não serão reutilizados. Mesmo espaços diferentes na declaração farão com que esta procura falhe. A utilização de variáveis de vinculação permite utilizar repetidamente as mesmas declarações enquanto se altera o critério da cláusula WHERE. Assumindo que a declaração não tem um plano de execução em cache, deve ser analisada antes da execução. A fase de parse para declarações pode ser diminuída pela utilização eficiente do aliasing.

Alias Utilização

Se um pseudónimo não estiver presente, o motor deve resolver quais as tabelas que possuem as colunas especificadas. Um pseudónimo curto é analisado mais rapidamente do que um nome de tabela longa ou um pseudónimo. Se possível, reduzir o pseudónimo a uma única letra. O seguinte é um exemplo:

Má declaração
```
SELECCIONAR primeiro_nome, último_nome, país DE
empregado, países
ONDE o country_id = id E
último_nome = 'HALL';
```

Boa declaração
```
SELECT e.first_name, e.last_name, c.country FROM
employee e, countries c
ONDE e.country_id = c.id
E e.last_name = 'HALL';
```

Condução Tabelas

A estrutura das cláusulas FROM e WHERE das declarações DML pode ser adaptada para melhorar o desempenho da declaração. As regras variam dependendo se o motor da base de dados está a utilizar o optimizador de regras ou o optimizador baseado no custo. A situação é ainda mais complicada pelo facto de

que o motor pode realizar uma união por fusão ou uma união por loop aninhado para recuperar os dados. Apesar deste desafio, existem algumas regras que pode utilizar para melhorar o desempenho do seu SQL.

O resultado dos processos Oracle define uma tabela de cada vez. Começa pela recuperação de todos os dados para a primeira tabela (de condução). Uma vez recuperados estes dados, estes são utilizados para limitar o número de linhas processadas para as tabelas subsequentes (conduzidas). No caso de múltiplas entradas de tabelas, a tabela de condução limita as linhas processadas para a primeira tabela conduzida.

Uma vez processado, este conjunto combinado de dados é o conjunto de condução para a segunda tabela conduzida, etc. Grosseiramente traduzido, isto significa que é melhor processar tabelas que irão recuperar um pequeno número de linhas primeiro. O optimizador fará isto o melhor que puder, independentemente da estrutura do DML, mas os seguintes factores podem ajudar.

Tanto os optimizadores baseados em regras como em custos seleccionam uma tabela de condução para cada declaração de DML. Se uma decisão não puder ser tomada, a ordem de processamento é desde o fim da cláusula FROM até ao início. Por conseguinte, deve sempre colocar a sua mesa de condução no fim da cláusula FROM. Escolha sempre a tabela com menor número de registos como tabela de condução. Se três tabelas estiverem a ser unidas, seleccione as tabelas de intersecção como a tabela de condução.

A mesa de intersecção é a mesa que tem muitas mesas dependentes dela. As mesas de interseção subsequentes devem ser colocadas em ordem de modo a que aqueles que recuperarem a maior parte das filas estejam mais perto do início da cláusula FROM. Contudo, a cláusula WHERE deve ser escrita na ordem oposta, com as condições das mesas de condução em primeiro lugar e a mesa de condução final em último lugar (exemplo abaixo).

```
DE d, c, b, a
ONDE a.join_column = 12345
E a.join_column = b.join_column AND b.join_column =
c.join_column AND c.join_column = d.join_column;
```

Se agora queremos limitar as linhas trazidas de volta da tabela "D", podemos escrever o seguinte:

```
DE d, c, b, a
ONDE a.join_column = 12345 AND a.join_column =
b.join_column AND b.join_column = c.join_column AND
c.join_column = d.join_column AND d.name = 'JONES';
```

Dependendo do número de filas e da presença de índices, a Oracle pode agora escolher "D" como a mesa de condução. Uma vez que "D" tem agora dois factores limitantes (join_column e name), pode ser um melhor candidato como mesa de condução. A declaração pode ser melhor escrita como:

DE c, b, a, d
ONDE d.name = 'JONES
E d.join_column = 12345
AND d.join_column = a.join_column AND a.join_column = b.join_column AND b.join_column = c.join_column

Este agrupamento de factores limitantes orientará o optimizador de forma mais eficiente, fazendo com que a tabela "D" retorne relativamente poucas filas, e tornando-a assim uma mesa de condução mais eficiente. Lembre-se, a ordem dos itens tanto na cláusula DE como ONDE, não obrigará o optimizador a escolher uma mesa específica como mesa motriz, mas pode influenciar a decisão do optimizador. O agrupamento de condições restritivas numa única mesa reduzirá o número de filas devolvidas dessa mesa, o que a tornará uma candidata mais forte para se tornar a mesa de condução. Além disso,
pode ter o controlo sobre qual tabela conduzirá a consulta através da utilização da dica ORDERED. Não importa de que ordem é o optimizador, essa ordem pode ser anulada pela dica ORDERED. A chave é usar a dica ORDERED e variar a ordem das tabelas para obter a ordem correcta do ponto de vista do desempenho.

Caching Tabelas

As consultas serão executadas muito mais rapidamente se os dados a que se referem já estiverem armazenados em cache. Para tabelas pequenas e frequentemente utilizadas, o desempenho pode ser melhorado através da colocação em cache de tabelas. Normalmente, quando ocorrem digitalizações de tabelas completas, os dados em cache são colocados no final da cache de memória intermédia (LRU), o que é menos utilizado recentemente. Isto significa que são os primeiros dados a serem colocados em cache quando é necessário mais espaço de cache.

Se a tabela estiver em cache (ALTER TABELA de funcionários CACHE;), os dados são colocados na extremidade Mais Recentemente Utilizada (MRU) do buffer, e por isso é menos provável que seja paginada antes de ser novamente consultada. As tabelas de cache podem alterar o caminho do CBO através dos dados e não devem ser utilizadas sem uma cuidadosa consideração.

Evite Utilizar as Cláusulas Select *

A referência dinâmica da coluna SQL (*) dá-lhe uma forma de se referir a todas as colunas de uma tabela. Não utilizar a
* porque é muito ineficiente -- o * tem de ser convertido para cada coluna por sua vez. O analisador SQL
trata todas as referências de campo, obtendo os nomes das colunas válidas do dicionário de dados e substitui-as na linha de comando, o que consome tempo.

Existente vs. Em

A função EXISTS procura a presença de uma única fila que satisfaça os critérios declarados, por oposição à declaração IN que procura todas as ocorrências. Por exemplo:

PRODUTO - 1000 filas
ITEMS - 1000 filas

```
(A)
SELECCIONAR
p.product_id FROM
produtos p
ONDE p.item_no IN (SELECT i.item_no
                   FROM itens i);

(B)
SELECCIONAR
p.product_id FROM
produtos p
ONDE EXISTE (SELECCIONAR '1')
                   DE itens i
                   ONDE i.item_no = p.item_no)
```

Para a consulta A, todas as linhas do ITEMS serão lidas para cada linha do PRODUTOS. O efeito será de 1.000.000 linhas lidas no ITEMS. No caso da pergunta B, será lido um máximo de 1 linha do ITEMS para cada linha de PRODUTOS, reduzindo assim as despesas de processamento da declaração.

Não existe vs. Não In

Em declarações de subconsulta como as que se seguem, a cláusula NOT IN provoca uma espécie/fusão interna.

```
SELECCIONAR * DE ALUNO
ONDE o aluno_num NOT IN (SELECT student_num da turma)
```

Em vez disso, utilizar:
SELECCIONAR * DE estudante c
ONDE NÃO EXISTE
(SELECCIONAR 1 DA Aula a.ONDE a.student_num = c.student_num)

In with Minus vs. Not In for Non-Indexed Colunas

Em declarações de subconsulta como as que se seguem, a cláusula NOT IN provoca uma espécie/fusão interna.

SELECT * FROM system_user
WHERE su_user_id NOT IN
(SELECT ac_user FROM account)

Em vez disso, utilizar:
SELECT * FROM system_user
WHERE su_user_id IN
(SELECCIONAR o
su_id_utilizador_de_sistema_utilizador
MINUS
SELECT ac_user FROM conta)

Subconsultas relacionadas com Correspondência vs. Inline Vistas

Não utilize subconsultas correlatas de código nas suas aplicações, pois estas terão um impacto adverso no desempenho do sistema. Em vez disso, utilize visualizações em linha (subconsultas na cláusula das suas declarações seleccionadas), que executam ordens de magnitude mais rapidamente e são muito mais escaláveis. A consulta abaixo mostra todos os funcionários que ganham mais do que o salário médio do departamento em que trabalham.

Antes:

```
SELECCIONAR exterior.*
        DE emp emp.exterior
        onde.exterior.salário >
            (SELECCIONAR
                avg(salário) de emp
                emp interior
                WHERE inner.dept_id = outer.dept_id);
```

A consulta anterior contém uma subconsulta correlacionada, que é extremamente ineficiente e muito intensiva em CPU.
A subconsulta será executada para cada registo de empregado na tabela EMP. À medida que o número de registos no PEM aumenta, o desempenho pode degradar-se exponencialmente. Quando reescrito com vistas em linha, a consulta não é

funcionalmente equivalente, mas é também significativamente mais escalável e tem a garantia de ter um desempenho superior ao do seu antecessor.

Depois:

```
SELECCIONAR e1.*
        DE e1, (SELECT e2.dept_id dept_id, avg(e2.
salário) avg_sal
                DE emp e2
                GROUP BY dept_id)
dept_avg_sal WHERE e1.dept_id =
dept_avg_sal.dept_id
       E e1.salary > dept_avg_sal.avg_sal;
```

Vistas Utilização

Cuidado com as instruções SQL com vistas nelas. Por estranho que pareça, a Oracle não executa necessariamente uma visão da mesma forma por si só que o faz num complexo Declaração SQL contendo tabelas. Considerar incluir a definição da vista na consulta principal, incluindo o seu código sem o nome real da vista. As visualizações podem causar potenciais problemas de desempenho quando têm junções externas (a CBO passa a estar em mau estado, mesmo em 9I) ou são consideradas visualizações não fusíveis pela Oracle.

Use Decodificar para Reduzir Processamento

Utilize DECODE quando quiser digitalizar as mesmas linhas repetidamente ou juntar-se à mesma mesa repetidamente.

```
    SELECT count(*) , sum(sal) FROM emp
    WHERE deptno = 10
    E ename LIKE 'MILLER';

    SELECT count(*) , sum(sal)
    FROM emp
    ONDE deptno = 20
    E ename LIKE 'MILLER';
```

O mesmo resultado pode ser alcançado utilizando uma única consulta, como se segue:

```
SELECT count(decode(deptno,20,'x')) dept20_count,
    count(decode(deptno,10,'x')) dept10_count,
    sum(decode(deptno,20,sal)) dept20_sal,
    sum(decode(deptno,10,sal)) dept10_sal
DE emp
ONDE ELABORAR 'MILLER' ;
```

Desigualdades

Se uma consulta utiliza desigualdades (item_no > 100), o optimizador deve estimar o número de filas devolvidas antes
pode decidir a melhor maneira de recuperar os dados. Esta estimativa é propensa a erros. Se estiver ciente dos dados e da sua distribuição, então pode usar dicas de optimização para encorajar ou desencorajar digitalizações de tabelas completas para melhorar o desempenho.

Se estiver a ser utilizado um índice para uma varredura de gama na coluna em questão, o desempenho pode ser melhorado substituindo >= por >=. Neste caso, item_no > 100 torna-se item_no >= 101. No primeiro caso, ocorrerá uma varredura completa do índice. No segundo caso, o Oracle salta
directamente para a primeira entrada de índice com um item_no de 101 e varrimentos de intervalo a partir deste ponto. Para grandes índices, isto pode reduzir significativamente o número de blocos lidos.

Utilização da União no lugar de Ou

Em geral, considerar sempre o verbo UNION em vez do verbo OR nas cláusulas WHERE. A utilização do OU numa coluna indexada faz com que o optimizador efectue uma varredura de mesa completa em vez de uma recuperação indexada.

Usando Union All em vez de Union

A operação SORT é muito cara em termos de consumo de CPU. A operação UNION ordena o resultado definido para eliminar quaisquer filas que se encontrem dentro das sub-perguntas.
UNION ALL inclui filas duplicadas e não requer uma espécie. A menos que exija que estas linhas duplicadas sejam eliminadas, use UNION ALL.

Influenciando o Optimizer Usando Dicas

As dicas são instruções especiais para o optimizador. É possível alterar o objectivo de optimização de uma declaração individual utilizando a Dica. Algumas Dicas comummente utilizadas
são CHOOSE, RULE, FULL(table_name), INDEX(table_ name index_name), USE_NL, USE_HASH(table_ name), PARALLEL(table_name parallelism), etc.

SELECCIONAR /*+regra*/ nome,
acct_allocation_percentage
DE contas WHERE account_id = 1200

A declaração SQL acima será processada utilizando o optimizador baseado em RULE.

```
SELECT /*+ index(a, acct_id_ind) */ nome, acct_
allocation_percentage
A partir de contas a
WHERE account_id = :acct_id E client_id= :client_id
```

Na declaração SQL acima, foi utilizada uma dica de índice para forçar a utilização de um determinado índice.

Presença Verificação

Se o processamento estiver condicionado à presença de certos registos numa tabela, poderá utilizar códigos como, por exemplo

```
SELECT count(*)
INTO v_count DE
artigos
ONDE o item_tamanho = 'PEQUENO';

SE v_conta = 0 ENTÃO
-- Fazer o processamento relacionado com não
pequenos itens presentes FIM SE;
```

Se houver muitos artigos pequenos, perde-se tempo e processamento na recuperação de múltiplos registos que não são necessários. Isto seria melhor escrito como um dos seguintes:

```
SELECT count(*)
INTO v_count DE
artigos
ONDE o item_tamanho =
'PEQUENO' E o rownum =
1;

SE v_conta = 0 ENTÃO
  -- Fazer o processamento relacionado com não
pequenos itens presentes FIM SE;
OU
INÍCIO
  SELECCIONAR '1
  INTO v_dummy DE
  artigos
  ONDE item_tamanho =
'PEQUENO' E rownum = 1;
EXCEPÇÃO
  QUANDO NÃO_DADOS_FUNDADOS ENTÃO
    -- Fazer o processamento relacionado com não
pequenos itens presentes FIM;
```

Nestes exemplos, apenas um único registo é recuperado na verificação de presença/ausência.

Utilização de índices para melhorar o desempenho

Os índices existem principalmente para melhorar o desempenho. Mas eles não vêm sem um custo. Os índices devem ser actualizados durante as operações INSERT, UPDATE e DELETE, o que pode abrandar o desempenho. Alguns factores a considerar na utilização de índices incluem:

- Escolher e utilizar os índices de forma apropriada. Os índices devem ter uma elevada selectividade. Os índices bitmapped melhoram o desempenho quando o índice tem menos valores distintos como Macho ou Fêmea.

- Evitar utilizar funções como "UPPER" ou "LOWER" na coluna que tem um índice. Caso não haja forma de evitar a função, utilizar índices funcionais.

- A partição do índice deve ser considerada se a tabela em que o índice se baseia for particionada. Além disso, todas as chaves estrangeiras devem ter índices ou devem constituir a parte principal da Chave Primária.

- Ocasionalmente pode querer utilizar um índice concatenado com a coluna SELECT. Esta é a solução mais favorecida quando o índice não só tem todas as colunas da cláusula WHERE, mas também a
colunas da cláusula SELECT. Neste caso, não há necessidade de aceder à tabela. Também pode querer utilizar um índice concatenado quando todas as colunas da cláusula WHERE formam as colunas principais do índice.

- Ao utilizar o 9i, pode tirar partido dos exames de saltar. As varreduras de saltar índice eliminam a limitação imposta pelo posicionamento da coluna, uma vez que a ordem da coluna não restringe a utilização do índice.

- Os grandes índices devem ser reconstruídos a intervalos regulares para evitar a fragmentação dos dados. A frequência da reconstrução depende das extensões das inserções de tabelas.

Porque os índices não são utilizados

A presença de um índice numa coluna não garante a sua utilização. Segue-se uma lista de factores que podem impedir a utilização de um índice:

- O optimizador decide que seria mais eficiente não utilizar o índice. Como regra geral, um índice será utilizado em dados distribuídos uniformemente se restringir o número de linhas devolvidas a 5% ou menos do número total de filas. No caso de dados distribuídos aleatoriamente, um índice será utilizado se restringir o número de linhas devolvidas a 25% ou menos do número total de linhas.

- Realiza operações matemáticas na coluna indexada, ou seja, ONDE o salário + 1 = 10001

- Concatena uma coluna, ou seja, ONDE o primeiro nome || ' '
  || último nome = 'JOHN JONES

- Não inclui a primeira coluna de um índice concatenado na cláusula WHERE da sua declaração. Para que o índice seja utilizado numa correspondência parcial, deve ser utilizada a primeira coluna (margem de chumbo).

- A utilização de declarações OR confunde o CBO. Raramente optará por utilizar um índice numa coluna
referenciados utilizando uma declaração OR. Ignorará até dicas de optimizador nesta situação. A única forma de garantir a utilização de índices nestas situações é usar a /*+ RULE */ dica.

- Utiliza-se o operador nulo numa coluna que é indexada. Nesta situação, o optimizador irá ignorar o índice.

- Misturam-se e combinam-se valores com tipos de dados de colunas. Esta prática levará o optimizador a ignorar o índice. Por exemplo, se o tipo de dados da coluna for um número, não utilizar aspas simples em torno do valor na cláusula WHERE. Da mesma forma, não deixe de utilizar

citações únicas em torno de um valor quando este é definido como uma coluna alfanumérica. Por exemplo, se uma coluna for definida como um varchar2(10), e se existir um índice construído sobre essa coluna, referencie os valores da coluna dentro de aspas simples. Mesmo que apenas armazene números nela, ainda assim precisa de utilizar aspas simples em torno dos seus valores na cláusula WHERE, pois não o fazer resultará numa verificação completa da tabela.

- Quando a Oracle encontra um NÃO, optará por não utilizar um índice e realizará, em vez disso, um scan de mesa completa. Lembre-se, os índices são construídos sobre o que está numa mesa, não sobre o que não está numa mesa.

Conclusão

Oitenta por cento dos problemas de desempenho da sua base de dados surgem devido a SQL incorrecta. A concepção e desenvolvimento de SQL óptimo é quintessencial para alcançar um desempenho escalável do sistema e tempos de resposta consistentes. A chave para sintonizar muitas vezes resume-se a quão eficazmente se pode sintonizar essas consultas de problema único.

Para sintonizar eficazmente, é necessário conhecer os seus dados. O seu sistema é único, pelo que deve ajustar os seus métodos para se adequar ao seu sistema. Um único índice ou uma única consulta pode levar um sistema inteiro a uma quase paralisação. Obtenha esses maus SQL e conserte-os. Torne-o um hábito...e mantenha-se fiel a ele.

Referências

1. Documentação Oracle 9I

2. Oracle Performance Tuning 101 de Gaja Krishna Vaidyanatha, Kirtikumar Deshpande e John Kostelac

3. http://www.dba-village.com/dba/village/dvp_papers.Main?CatA=45 por Sumit Popli e Puneet Goenka.

Sobre a GlobalLogic Inc.

A GlobalLogic é um líder de serviços de desenvolvimento de produtos de ciclo de vida completo que combina conhecimentos profundos de domínio e experiência inter-indústria para ligar os fabricantes aos mercados de todo o mundo. Utilizando o conhecimento adquirido com o trabalho em produtos inovadores e tecnologias disruptivas, colaboramos com os clientes para lhes mostrar como a investigação e desenvolvimento estratégicos podem tornar-se uma ferramenta para gerir o seu futuro. Construímos parcerias com líderes empresariais e tecnológicos que definem o mercado e que querem fazer produtos surpreendentes, descobrir novas oportunidades de receitas e acelerar o tempo de chegada ao mercado.

Para mais informações, visite www.globallogic.com

GlobalLogic®

Contacto

Emily Younger
+1.512.394.7745
emily.younger@globallogic.com