**Dica 1: Analise o plano de execução das instruções SQL**

Este é o passo inicial do trabalho de tuning. Antes de tunar uma instrução SQL é necessário avaliar o seu desempenho e verificar se ela realmente precisa ser otimizada. O método principal para realizar este trabalho é analisar o plano de execução.

Um Plano de Execução é uma sequência de operações que o banco de dados realiza para executar uma instrução SQL. Ele é exibido em forma de uma árvore de linhas, que representam os passos e que contém as seguintes informações:

· Ordenação das tabelas referenciadas pela instrução;

· Método de acesso para cada tabela mencionada na instrução;

· Método join para as tabelas afetadas pelas operações de join da instrução;

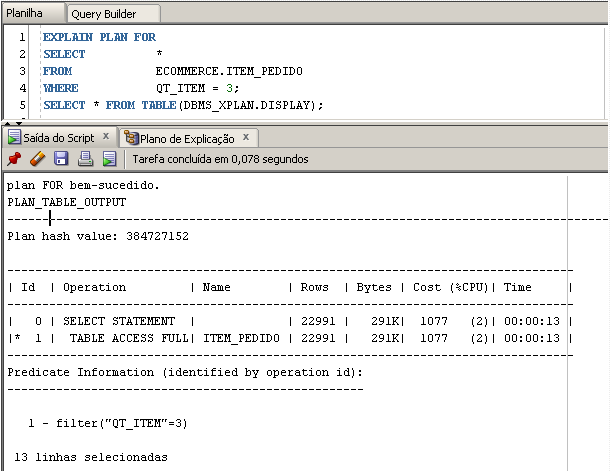
· Operações de dados tais como filter, sort ou agregação;

· Custo e cardinalidade de cada operação;

· Conjunto de partições (tabelas particionadas) acessadas;

· Se ocorreu processamento paralelo.

Para gerar o plano de execução de uma instrução SQL, devemos utilizar o comando EXPLAIN PLAN seguido pela instrução SQL desejada. Isso fará com que um componente do banco, chamado otimizador de query, após escolher o melhor plano de execução da query, insira os dados descrevendo este plano em uma tabela chamada PLAN\_TABLE. Para analisá-lo, é necessário escrever uma query para pesquisar nesta tabela. Veja na **Figura 2** um exemplo (executado no SQL Developer 3.0) de como utilizar estes comandos para gerar e analisar um plano de execução.



**Figura 2.** Exemplo de um Plano de Execução de uma instrução SQL.

Os passos do plano de execução podem variar de acordo com o ambiente em que for executado, e o otimizador poderá escolher planos de execução diferentes de acordo com as configurações do ambiente do banco de dados, tais como: hardware e estatísticas de objetos.

Em um plano de execução, a linha inicial é aquela que está identada mais para a direita. Na **Figura 2,**a linha com Id = 1 será a primeira a ser executada. Os próximos passos a serem realizados são as linhas pai das linhas citadas anteriormente, ou seja, a linha que está identada no nível à esquerda mais próximo acima delas. Se duas linhas estão identadas igualmente (no mesmo nível), a linha mais acima é executada primeiro.

Ao analisar um plano de execução, avalie:

· As operações que estão sendo executadas (coluna **Operation**), a quantidade de bytes (coluna **Bytes**) e o custo (coluna **Cost**) de cada passo ou o tempo de resposta das operações (coluna **Time**);

· Se a tabela driving (tabela condutora da expressão, onde os dados são primeiramente selecionados) tem o melhor filtro;

· Se as ligações (joins) em cada passo retornam o menor número de linhas para o próximo passo;

· Se o método de acesso ou ligação (join) é apropriado para o número de linhas que serão retornadas. Por exemplo, nested loops join através de índices pode não ser tão bom quando muitas linhas estão sendo retornadas;

· Se visões estão sendo usadas eficientemente;

· Se existem produtos cartesianos não intencionais (pode ocorrer quando por esquecimento ou desconhecimento, colunas de tabelas relacionadas não estão ligadas);

· Se cada tabela está sendo acessada eficientemente;

· Se os predicados na cláusula WHERE estão sendo usados eficientemente e se o número de linhas retornadas de uma tabela não é muito grande. Se um full table scan está ocorrendo em uma tabela, verifique se é possível alterar algum predicado da cláusula WHERE para que o otimizador empregue uma condição mais seletiva e possa utilizar um índice.

Para tunar uma instrução SQL, altere-a inúmeras vezes, analise o plano de execução de cada versão gerada e opte por implementar aquela que consumir menos recursos (colunas, Bytes e Cost (%CPU)) ou a que apresentar o menor tempo de resposta (coluna Time). Em bancos OLTP, implemente a instrução SQL que tiver o menor tempo de resposta. Verifique também se as operações que estão sendo executadas em cada passo do plano de execução são adequadas para a quantidade de dados a ser retornada.

Quando a quantidade de dados a ser retornada normalmente é menor ou igual a 4% dos dados da(s) tabela(s) consultada(s), o tempo de resposta é menor se o otimizador utilizar índices (Index Search), caso contrário, o otimizador deverá realizar uma varredura de dados completa na(s) tabela(s) – isto é, um Full Table Scan. Se um caminho de acesso indesejado for identificado no plano de execução, é possível forçar uma determinada operação (possivelmente mais performática) através do uso de hints. Hints são dicas que podem ser fornecidas ao otimizador com o objetivo de ajudá-lo a montar um plano de execução melhor. O uso de hints, por se tratar de um assunto bastante extenso, não será abordado neste artigo.

Se ao seguir e aplicar os procedimentos descritos nesta dica e a eficiência da instrução SQL não melhorar, considere reescrevê-la (refactoring) ou criar novos índices nas tabelas envolvidas, o que será abordado a seguir.

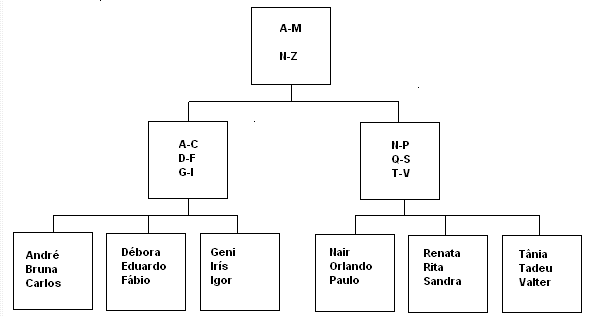
Para mais informações sobre essa dica, consulte o artigo “Analisando o Plano de Execução para tunar instruções SQL“ na seção **Links**.

**Dica 2: Crie índices cuidadosamente**

Índices permitem agilizar as consultas (SELECT), porém aumentam em até três vezes o tempo das atualizações (INSERT, UPDATE, DELETE e MERGE) e consomem espaço em disco.

No Oracle existem quatro grupos de índices que podem ser criados para otimizar consultas de acordo com as características dos dados e das consultas:

**1. B-tree (normal):** Crie este índice quando for necessário efetuar consultas em colunas que possuem alta cardinalidade (grande variação de valores), como por exemplo, colunas que representam a chave primária da tabela, em que não há valores repetidos. Veja na **Figura 3** uma representação gráfica da estrutura deste tipo de índice;



**Figura 3**. Estrutura interna de um índice B-tree.

**2. Bitmap:** Crie este índice quando for necessário efetuar consultas em colunas que possuem baixa cardinalidade (pouca variação de valores), como por exemplo, colunas que representam o sexo do cliente (só há duas variações: feminino e masculino). Veja na Figura 4 uma representação gráfica da estrutura deste tipo de índice;

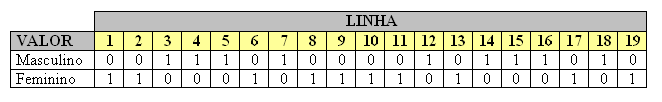
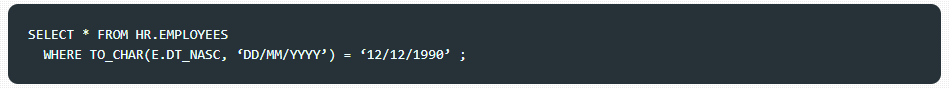


Figura 4. Estrutura interna de um índice Bitmap.

**3. Function Based**: Estes tipos de índices podem ter a estrutura de índices B-tree ou Bitmap. Utilize-os quando for necessário efetuar consultas aplicando funções nas colunas contidas em condições seletivas. Exemplo:



**4. Domain Indexes**: São tipos de índices projetados para tarefas especializadas ou mais complexas, tais como processamento de imagens ou pesquisas textuais. Como exemplos de índices desse grupo podemos citar: Oracle Spatial e Oracle Text. O índice Oracle Text é muito bom para otimizar consultas textuais em colunas do tipo VARCHAR2 ou CLOB.

Para criar índices, considere as seguintes recomendações:

a) Crie índices para agilizar somente as consultas mais frequentes. Índices não utilizados ou subutilizados só ajudam a piorar a performance das atualizações e consomem espaço em disco que poderia ser utilizado para outros fins;

b) Crie índices em uma ou mais colunas. Crie índices de acordo com as consultas que são realizadas. Se uma consulta é realizada, por exemplo, filtrando três colunas de uma tabela, crie um índice que contemple as três colunas (e não um índice para cada coluna);

c) Não crie índices em colunas de tabelas pequenas. Índices em tabelas pequenas são desnecessários, pois nestas tabelas o custo de realizar um Full Table Scan (FTS) é baixo, inviabilizando a necessidade de índices;

d) Crie índices nas colunas de uma tabela que possuem chaves estrangeiras (FKs). Isso irá otimizar os métodos de ligação (joins) e consequentemente a performance geral das queries;

e) Escolha o tipo de índice correto. De acordo com a condição do filtro (cláusula WHERE) ou a cardinalidade dos dados, escolha o índice mais apropriado para a situação. Exemplo: ao filtrar dados de uma tabela de clientes pelo nome dos mesmos, crie um índice b-tree na coluna que representa o nome, pois os valores dessa normalmente devem possuir alta cardinalidade.

**Dica 3: Utilize paralelismo**

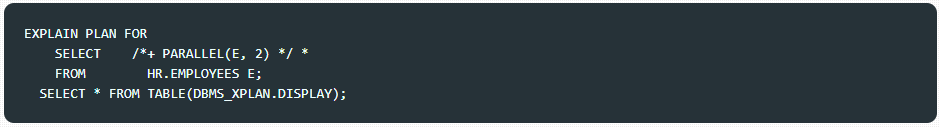
Paralelismo é o nome dado ao recurso que permite explorar o uso de múltiplos processadores ou múltiplos núcleos de processadores para executar uma única tarefa, com o objetivo de reduzir o tempo total de execução da mesma. Empregando paralelismo, uma tarefa é dividida em pequenas partes, e cada uma poderá ser executada em um núcleo ou processador diferente.

Como atualmente temos muitos servidores de banco de dados com múltiplos processadores e múltiplos núcleos, devemos aprender a explorar estes recursos no banco de dados para otimizar a performance de nossas queries.

Para utilizar paralelismo em uma instrução SQL em bases de dados Oracle, podemos incluir o hint PARALLEL na instrução SQL. Este hint precisa de dois parâmetros para ser executado: o nome ou apelido (álias) da tabela e a quantidade de threads (processos) em que a tarefa principal será dividida. No banco de dados Oracle, este hint só executa paralelismo quando é realizada uma varredura completa dos dados (Full Table Scan) na tabela. Paralelismo com Index Scan é um assunto mais extenso e complexo, e por isso não será abordado neste artigo.

Para demonstrar a utilização do hint PARALLEL, a Listagem 1 exemplifica uma query que retorna uma lista de empregados da tabela EMPLOYEES do schema HR do Oracle.

Listagem 1. Código SQL com o hint PARALLEL.



Paralelismo é um recurso poderoso para otimizar queries, porém devemos ter muito cuidado para não usá-lo em excesso. Se passarmos a adotar paralelismo em instruções que não necessitam de otimização ou em todas as instruções que são executadas no banco, a sobrecarga de gerenciamento dos processos paralelos poderá ocasionar problemas sérios de performance devido ao “peso” do gerenciamento dos mesmos. Deste modo, use paralelismo apenas nas queries mais críticas e normalmente quando outros métodos de tuning já foram avaliados e não obtiveram os resultados esperados.

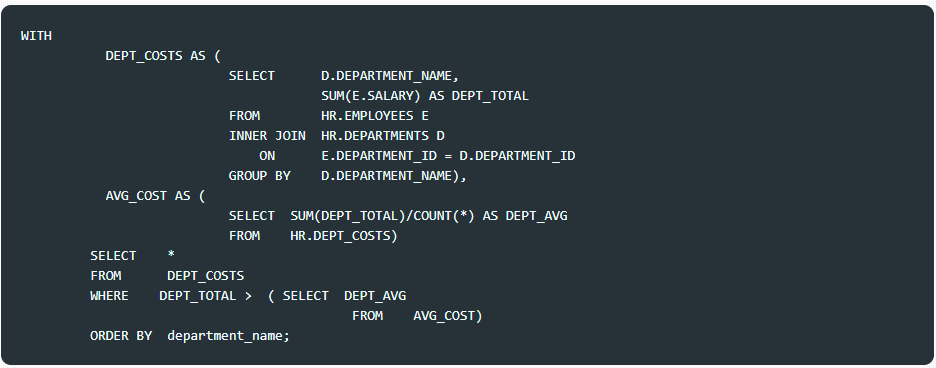
**Dica 4: Utilize a cláusula WITH**

A cláusula WITH, existente no Oracle Database desde a versão 9i release 2, pode ser utilizada para otimizar a performance de consultas SQL, possibilitando a reutilização de blocos de subquery, criando uma espécie de tabela temporária, que existe somente no escopo da instrução SQL em que ela está contida.

A cláusula WITH permite a redução do uso de recursos do banco de dados e pode ser usada para executar queries complexas ou remotas. Por exemplo, quando uma ou mais tabelas são referenciadas múltiplas vezes em uma query, para evitar acessar a(s) mesma(s) tabela(s) várias vezes, esta cláusula cria uma tabela temporária que armazena o resultado da consulta e que possibilita acessarmos somente a tabela temporária. Dessa forma, acessar os dados da tabela temporária torna-se mais rápido do que acessar os dados da(s) tabela(s).

Para demonstrar como utilizar este recurso, a Listagem 2 exemplifica uma query que consulta dados no schema HR do Oracle.

Listagem 2. Código SQL com a cláusula WITH.



Observe nesta listagem que as subqueries que poderiam ser redundantes foram eliminadas através da criação da tabela temporária DEPT\_COSTS. Em testes gerais, conseguimos otimizar 30% do custo médio de execução dessa query utilizando a cláusula WITH.

Para mais informações sobre essa dica, consulte o artigo “Cláusula WITH (para tunar queries)” na seção Links.

**Dica 5: Utilize functions, stored procedures e packages**

Crie functions e stored procedures no banco de dados e utilize estes objetos nas aplicações para acessar e atualizar dados no banco, ao invés de criar instruções SQL ad hoc.

Esse item gera muita polêmica. Ao empregar essa dica muitas regras de negócio da aplicação acabam sendo transferidas para as functions e stored procedures, o que muitos profissionais não aceitam, pois aprendemos estudando Engenharia de Software que devemos separar uma aplicação em camadas, onde cada camada tem um papel específico e não deve misturar código de acesso/atualização de dados com código de regras de negócio. Não queremos defender ou criticar essa prática, porém, se temos como prioridade a performance da aplicação, empregar essa dica poderá lhe render um ótimo desempenho.

A utilização de functions e stored procedures pelas aplicações pode oferecer os seguintes benefícios:

a) Facilitam a reutilização de instruções SQL pela própria aplicação e por outras aplicações. Normalmente é mais fácil compartilhar objetos PL/SQL do que objetos desenvolvidos, por exemplo, em Java ou .Net. Quando você desenvolve um objeto em uma destas linguagens e deseja compartilhá-lo, é necessário compilar algumas classes em um componente e compartilhar esse componente em um servidor de aplicação. O componente desenvolvido em Java não se comunica com .Net e vice-versa, mas ambos acessam o banco Oracle. Deste modo, se o código das classes estivesse em um objeto no banco de dados (seja ele uma function ou uma procedure), seria mais fácil compartilhá-lo;

b) Aplicam o uso de variáveis bind. Variáveis bind são variáveis externas a uma instrução SQL. O uso de stored procedures para retornar ou atualizar dados emprega naturalmente o conceito de variáveis bind, que permite agilizar e melhorar a segurança das instruções SQL. Aplicações que submetem instruções SQL com variáveis bind ao banco de dados garantem que o otimizador de queries do Oracle reutilize planos de execução, economize memória e agilize dessa forma a execução das instruções SQL. O uso de variáveis bind também ajuda a evitar ataques de SQL Injection;

c) Diminuem o tráfego de dados (round trips) na rede. Ao passo que misturamos regras de negócio com dados, podemos evitar que muitos dados sejam enviados pela rede para a aplicação. Pense no caso de uma transferência bancária que é constituída de três passos: 1) verificação de saldo na conta corrente; 2) retirada de valor da conta origem; 3) aplicação do valor na conta destino. Para verificar o saldo da conta corrente, podemos aplicar algumas regras de negócio dentro de uma stored procedure e não retornar nenhum dado para a aplicação neste passo, o que não poderia ocorrer sem o uso de stored procedures.

Para mais informações sobre essa dica, consulte o artigo “Otimizando a performance de aplicações com stored procedures“ na seção Links