ESTUDO DA APLICAÇÃO DE TÉCNICAS DE *TUNING* EM SISTEMAS COMPUTACIONAIS

Trabalho de Conclusão de Curso

Engenharia da Computação

Alberis Garcês de Castro
Orientador: Prof.ª Maria Lencastre Pinheiro de Menezes Cruz

Universidade de Pernambuco
Escola Politécnica de Pernambuco
Graduação em Engenharia de Computação

ALBERIS GARCÊS DE CASTRO

ESTUDO DA APLICAÇÃO DE TÉCNICAS DE *TUNING* EM SISTEMAS COMPUTACIONAIS

Monografia apresentada como requisito parcial para obtenção do diploma de Bacharel em Engenharia de Computação pela Escola Politécnica de Pernambuco – Universidade de Pernambuco.

Recife, dezembro de 2011.

POLI
ESCOLA POLITÉCNICA
DE PERNAMBUCO

MONOGRAFIA DE FINAL DE CURSO

Avaliação de defesa (cópia do aluno)

No dia 21 de Dezembro de 2011, às 10:00 horas, reuniu-se para deliberar a defesa da monografia de conclusão de curso do discente ALBERIS GARCES DE CASTRO, orientado pelo professor Maria Lencastre Pinheiro de Menezes Cruz, sob título Estudo de aplicação de técnicas de tunig em sistemas computacionais, a banca composta pelos professores:

Sérgio Murilo Maciel Fernandes

Maria Lencastre Pinheiro de Menezes Cruz

Após a apresentação da monografia e discussão entre os membros da Banca, a mesma foi considerada:

☒ Aprovada ☐ Aprovada com Restrições* ☐ Reprovada

e foi-lhe atribuída nota: 8,8 (oito, oito)

*(Obrigatório o preenchimento do campo abaixo com comentários para o autor)

O discente terá 07 dias para entrega da versão final da monografia a contar da data deste documento.

Comentários da banca

EXECUTAR MODIFICAÇÕES CONFORME DESCRITO.

Sérgio Murilo Maciel Fernandes

MARIA LENCASTRE PINHEIRO DE MENEZES CRUZ

Este trabalho é dedicado a
Deus que é aquele que nos dá toda a capacidade,
meus pais Agnes e Maurício que me fizeram chegar até aqui com muito esforço,
minha avó Silvia que contribuiu muito com minha educação e formação,
minha esposa muito amada Shyrlene que está sempre ao meu lado e
minhas três filhas queridas Kamylla, Ana Júlia e Maria Alice.

Resumo

Melhoria de desempenho em sistemas e bancos de dados é um assunto que exige atenção nos sistemas computacionais modernos que tendem a ser cada vez mais complexos a nível de funcionalidades e utilização de recursos de *hardware* e *software*. *Tuning,* ou ajuste de desempenho, é uma área de grande importância com diversas técnicas e metodologias disponíveis para se alcançar a otimização. É possível executar consultas *SQL* (Structured Query Language) de maneira mais eficiente com a aplicação de técnicas de *tuning.* Este trabalho aborda algumas das principais técnicas que visam melhorar o desempenho do sistema computacional como um todo e em especial o banco de dados. As metodologias de *tuning* foram aplicadas em um sistema de rastreamento de veículos que ao longo do seu desenvolvimento apresentou vários problemas de desempenho. Estes, foram analisados e os fatores causadores identificados e corrigidos utilizando-se de algumas técnicas de otimização de hardware, software e modificação da estrutura da base de dados e suas consultas *SQL*. Os resultados obtidos após estas alterações apresentam uma melhoria significativa de desempenho de várias funcionalidades da aplicação. Concluiu-se que o processo de otimização é uma tarefa importante e contínua que se inicia no projeto de design da aplicação e abrange várias áreas de um sistema, tais como: *hardware*, so*f*tware, banco de dados e arquitetura do projeto.

Abstract

Performance improvement in systems and databases is an issue requiring attention in modern computer systems that tend to be increasingly complex in terms of functionality and use of hardware and software. Tuning, or performance tuning, is an area of great importance with various techniques and methodologies available to achieve optimization. You can run SQL (Structured Query Language) more efficiently by applying techniques of tuning. This work discusses some key techniques to improve the performance of the computer system as a whole and especially the database. The tuning methods were applied to a system for tracking vehicles that along its development presented several performance problems. These were analyzed and the causative factors identified and corrected using some optimization techniques for hardware, software and modifying the structure of the database and your SQL statements. The results obtained after these changes have a significant performance improvement of several features of the application. It was concluded that the optimization process is an important and ongoing task that begins in the design of application design in several areas of a system, such as hardware, software, database design and architecture.

Sumário

Capítulo 1 Introdução **1**

1.1 Objetivos 3

1.2 Metodologia 3

1.2.1 Revisão geral sobre o tema abordado 4

1.2.2 *Tuning* de consultas *SQL* 4

1.2.3 Definição do estudo de caso 4

1.2.4 Aplicação das técnicas estudadas 4

1.2.5 Resultados da aplicação das técnicas de otimização 4

1.3 Estrutura do Documento 5

Capítulo 2 Fundamentação Teórica **7**

2.1 Visão Geral de Otimização 7

2.1.1 Quando Realizar *Tuning* 8

2.2 Projetando e Desenvolvendo para o Desempenho 10

2.2.1 Opções de Investimento de *Hardware* 10

2.2.2 *Tuning* de *Hardware* 11

2.2.3 *Tuning* de *Software* 13

2.3 *Tuning* em SGBDs 15

2.4 Estrutura do SGBD Oracle 10g 17

2.4.1 A Instância Oracle 19

2.4.2 PGA 20

2.4.3 Componentes de Memória da SGA 20

2.4.4 Processos de Segundo Plano 22

Capítulo 3 *Tuning* de Declarações *SQL* 25

3.1 O Otimizador Oracle 25

3.2 Índices 27

3.3 *Hints* 30

3.3.1 Classificação dos *Hints* 31

3.3.2 Sintaxe dos *Hints* 32

3.3.3 Tipos de *Hints* 33

3.3.4 Usando *Hints* 35

Capítulo 4 Estudo de Caso 44

4.1 O Sistema 44

4.2 Realizando *Tuning* do *Hardware* 46

4.3 Realizando *Tuning* do *Software* 48

4.4 Realizando *Tuning* da Instância 52

4.5 Realizando *Tuning* de Declarações *SQL* 53

Capítulo 5 Conclusão 63

5.1 Dificuldades Encontradas 64

5.2 Trabalhos Futuros 64

Bibliografia 66

Índice de Figuras

Figura 1. SGBA como interface entre aplicação e a base de dados.**Erro! Indicador não definido.**

Figura 2. Arquitetura básica de um sistema de banco de dados [10]. 17

Figura 3. Arquitetura do Oracle 10g [6].....19

Figura 4. Componentes do Otimizador Oracle [11]. 26

Figura 5. Painel de visualização dos veículos do sistema de rastreamento. 45

Figura 6. Cerca de visualização do sistema de rastreamento. 51

Figura 7. Visão da rota percorrida por um veículo. 57

Figura 8. Visão do gráfico de velocidade veicular. 59

Índice de Tabelas

Tabela 1. Resultados obtidos no estudo de caso.....60

Tabela de Símbolos e Siglas

DBA – Data Base Administrator

I/O – *Input / Output*

SGBD – Sistema Gerenciador de Banco de Dados

SQL – Structured Query Language

RAM – Random Access Memory

RAC – Real Application Clusters

Capítulo 1

Introdução

Na última década, a sociedade tem vivenciado um grande avanço tecnológico. Neste contexto, sistemas de informações cada vez mais complexos têm sido desenvolvidos para atender às necessidades crescentes das empresas com relação a organização e tratamento das informações. É natural que neste processo evolutivo os dados se tornem cada vez mais valiosos e críticos. Grande parte das empresas depende cada vez mais de seus sistemas e suas bases de dados para funcionar de forma eficiente, e para produzir resultados organizados e selecionados de forma confiável. Conseqüentemente, empreendimentos que manipulam grande volume de informações requerem que estas sejam armazenadas em sistemas de gerenciamento de bancos de dados com características compatíveis com esta atual realidade. O armazenamento da informação também evoluiu do simples processamento de transações isoladas, ou sistemas de arquivos do próprio sistema operacional, para o gerenciamento de banco de dados cada vez maiores e com níveis de complexidade e controle mais elevados.

A motivação para o desenvolvimento deste trabalho originou-se a partir da experiência com aplicações que exigem tempo de resposta muito curto e possuem tabelas de histórico de informações em sua base que crescem rapidamente, chegando a causar lentidão no sistema. Pode-se citar como exemplo um sistema de rastreamento veicular, responsável por monitorar milhares de veículos de um país. Este sistema armazena as informações de posicionamento geográfico, velocidade, movimentação e direção para todos os veículos rastreados. Estas informações são, para cada veículo, verificadas a cada minuto e gravadas no banco de dados do sistema. Aplicações como esta possuem tabelas que crescem dezenas de milhares de registros mensalmente. A realização de consultas nestas tabelas, com volume de informação muito elevado, gera tempos de resposta muito acima do esperado pelo *software* da aplicação cliente. Este atraso é um gargalo inaceitável para uma aplicação que requer uma resposta em tempo real, ou no máximo em alguns segundos.

O administrador de banco de dados – o *DBA* – precisa acompanhar o desempenho do SGBD (Sistema Gerenciador de Banco de Dados). Nesse monitoramento o administrador utiliza geralmente técnicas de otimização que possibilitam a realização de ajustes na forma de funcionamento do banco, afim de que o mesmo possa manter-se operando de forma satisfatória. A busca pela melhoria do desempenho envolve várias disciplinas, desde o ajuste de configuração de cada SGBD, a escolha de um plano de execução para uma consulta, chegando até mesmo à escolha de uma infraestrutura adequada para que a otimização possa acontecer.

O conjunto de técnicas e práticas que visam melhorar o desempenho do SGBD é denominado de *tuning*. De acordo com [2], a tradução literal da palavra *tuning* seria sintonia ou ajuste fino de alguma coisa para que funcione melhor. Conforme [3]: "*Tuning* diz respeito ao ajuste do SGBD para melhor utilização dos recursos, provendo um uso eficaz e eficiente do SGBD".

Existem muitos recursos e técnicas de *tuning* disponíveis para serem usadas em aplicações de bancos de dados. Porém, a escolha do mais indicado, para a solução do problema em estudo, é tarefa difícil. Determinadas técnicas resolvem problemas pontuais do sistema, mas acabam por gerar novos problemas em outras partes do mesmo. Além disso, existe pouco material disponível na língua portuguesa sobre esse tema. Este trabalho terá o foco em técnicas de *tuning*, disponibilizando conteúdo cuja intenção é servir como fonte de consulta e apoio à tomada de decisões e geração de relatórios analíticos, colaborando assim para aprimorar e agilizar os processos internos de uma organização.

A seguir serão apresentados os objetivos deste trabalho e a metodologia utilizada para alcançar esses objetivos. Por fim será apresentada a estrutura de todo o documento.

1.1 Objetivos

Este trabalho se propõe a desenvolver um estudo das técnicas de *tuning* mais utilizadas e realizar um estudo de caso, com embasamento teórico, que auxilie na melhor compreensão e uso das técnicas de *tuning*. A otimização e *tuning* é uma disciplina muito abrangente e com pouco material específico, disponível, relacionado a estudos comparativos. Neste trabalho, a partir da experiência do autor com um sistema real, busca-se contribuir para geração e disponibilização de resultados que possam servir de fonte de consulta e aprendizado para pessoas que desejem ter maior conhecimento na área de *tuning*.

As metas específicas realizadas para alcançar o objetivo proposto foram:

- Estudo detalhado das principais técnicas de *tuning* de aplicações e banco de dados, utilizadas tanto para o monitoramento proativo quanto para a eliminação de gargalos que diminuem o desempenho da aplicação.
- Abordagem de forma ampla do *tuning* automático de *SQL*, aprofundando o funcionamento do otimizador do SGBD e o gerenciamento das estatísticas que o SGBD Oracle [4] pode proporcionar.
- Abordagem de técnicas de otimização de declarações *SQL*, detalhando o uso de ferramentas para otimizar a execução de *queries*.
- Realização de um estudo de caso aplicando as técnicas estudadas em um sistema real.

1.2 Metodologia

A metodologia empregada para a execução do projeto foi dividida em cinco pontos detalhados a seguir.

1.2.1 Revisão geral sobre o tema abordado

- Revisão bibliográfica da literatura sobre as diferentes técnicas de *tuning*.

1.2.2 *Tuning* de consultas *SQL*

- Estudo sobre diferentes técnicas de otimização de *queries,* com o aprofundamento do funcionamento das principais técnicas: indexação e *hints*. A sintaxe de criação e utilização de índices e dos principais tipos de *hints* será exibida, bem como exemplos de uso destes recursos.

1.2.3 Definição do estudo de caso

- O sistema estudado tomou como base de estudo um sistema real que trata de uma solução para rastreamento de veículos de uma empresa privada. Os dados são referentes ao posicionamento dos veículos dos clientes, e foram fornecidos apenas em caráter de estudo. Por uma questão de sigilo, nada será exibido além dos resultados obtidos com a aplicação das técnicas e parte da estrutura de tabelas da base de dados.

1.2.4 Aplicação das técnicas estudadas

- De acordo com o estudo realizado foram selecionadas as técnicas que se adéquam a realidade do sistema em estudo.

- Aplicação das técnicas estudadas.

1.2.5 Resultados da aplicação das técnicas de otimização

- Análise dos dados resultantes obtidos pela aplicação das técnicas.

- Apresentação dos resultados.

O SGBD utilizado neste projeto foi o *Oracle 10g Express Edition*, que é de propriedade da Oracle. Este foi escolhido por que possui um conjunto de recursos de otimização amplo.

A ferramenta de interface com a base de dados escolhida foi o *Toad for Oracle*, que é de propriedade da Quest *Software*. A mesma foi escolhida por ser de fácil utilização e interface amigável, além de possuir o *SQL Monitor* como aplicativo adicional que permitirá observar, com precisão, o tempo de execução de cada *query* que será estudada.

1.3 Estrutura do Documento

O presente trabalho foi dividido em seis capítulos conforme a seguir:

- **Capítulo 1: Introdução**

  Contém todo o texto introdutório sobre a monografia, abordando a motivação, objetivos e metas.

- **Capítulo 2: Fundamentação teórica**

  Reúne os principais conceitos necessários para a compreensão do trabalho proposto. Apresenta os fundamentos da otimização de sistemas computacionais e seus diferentes tipos.

- **Capítulo 3: *Tuning* de declarações *SQL***

  Apresenta as técnicas de *tuning* de declarações *SQL*, baseadas no conjunto de ferramentas que o *Oracle 10g Express Edition* provê. O capítulo foca na utilização das técnicas de indexação e *hints*, discorrendo sobre sua forma de utilização e quando o uso de cada tipo de técnica é indicado.

- **Capítulo 4: Estudo de caso**

Apresenta os resultados obtidos a partir da utilização das técnicas de *tuning* no sistema em estudo.

- **Capítulo 5: Conclusão**

  Finaliza o trabalho, apresentando algumas conclusões, dificuldades encontradas e possíveis trabalhos futuros.

Capítulo 2

Fundamentação Teórica

Este capítulo aborda as características gerais da otimização de sistemas computacionais e provê informações acerca de *tuning* de base de dados Oracle.

2.1 Visão Geral de Otimização

Tuning de desempenho de uma aplicação é um tema bastante abrangente que envolve desde a escolha de um *hardware* apropriado até a modelagem do *software*. O ajuste do desempenho requer uma modificação na configuração inicial do sistema. Conforme [4], esta nova configuração do sistema envolve alocar recursos de forma a manter as funcionalidades iniciais disponíveis, operando corretamente e com velocidade de resposta satisfatória para o propósito ao qual o sistema se destina.

O fator que impulsiona a realização de *tuning* é a identificação do gargalo mais significativo para que sejam realizadas as mudanças apropriadas, visando reduzir ou até mesmo eliminar o feito provocado pelo gargalo. Em geral este ajuste é executado de forma reativa, ou seja, quando o problema surge trabalha-se para encontrar a melhor solução para eliminá-lo, ou enquanto o sistema ainda está na fase de desenvolvimento ou após sua implantação.

Uma forma eficaz para saber quando é necessário realizar *tuning* é ter um parâmetro de desempenho estabelecido, que possa ser usado para comparação caso surja um problema de desempenho. A maioria dos administradores (*DBA*s) conhece bem o seu sistema e isso torna possível a identificação de períodos de pico de utilização. Por exemplo, um *DBA* pode identificar que os períodos de pico do seu sistema ocorrem nos intervalos de 09h00min às 10h00min e de 14h30min às 15h00min. Além disso, pode verificar que o período de menor atividade é de 00h00min as 06h00min.

É importante identificar esses momentos de alta carga e se possível instalar uma ferramenta de monitoramento que reúna dados de desempenho para aqueles intervalos. O ideal é que esta coleta de informação seja configurada na fase inicial da produção do sistema, e que sempre sejam feitos ajustes de acordo com os dados obtidos. Os dados recolhidos podem incluir as seguintes informações:

- Estatísticas da aplicação (volumes de transação e tempos de resposta);
- Estatísticas do sistema operacional;
- Estatísticas de *hardware* (discos rígidos, memória e dispositivos de I/O);
- Estatísticas de rede.

2.1.1 Quando Realizar *Tuning*

Existem dois tipos distintos de *tuning*:

- **Monitoramento pró-ativo**

  Geralmente ocorre em intervalos regulares e pré-definidos, onde os resultados das estatísticas de desempenho são examinados e comparados com padrões estabelecidos para avaliar se o comportamento do sistema foi modificado. Monitoramento pró-ativo também pode ser chamado de ajuste pró-ativo.

  Normalmente este monitoramente não resulta em modificações na configuração do sistema e sim expõe um problema grave que está se desenvolvendo. Em algumas situações, um profissional especializado pode analisar e identificar problemas potenciais sozinho, porém a utilização de ferramentas de monitoramento é essencial.

  Realizar ajustes pró-ativos no sistema quando aparentemente não há degradação do desempenho pode ser uma atividade perigosa, resultando em modificações desnecessárias que podem surtir um

efeito contrário ao esperado. Para realizar *tuning* algumas etapas devem ser seguidas, conforme apresentado em [4].

Monitoramento é normalmente parte de um exercício de planejamento de capacidade, onde o uso de recursos é examinado para observar a forma como o aplicativo está sendo utilizado e a forma como o aplicativo usa o banco de dados e os recursos do *host*.

- **Eliminação de gargalos**

De acordo com [4], geralmente um gargalo implica em um problema de desempenho da aplicação. O ajuste deve fazer parte do ciclo de vida de uma aplicação, através da análise, projeto, codificação, produção e estágios de manutenção planejada. Porém, algumas vezes o ajuste é deixado para o momento após o sistema ser posto em produção. Nesta condição, o ajuste torna-se um exercício de combate reativo ao surgimento de problemas, onde o gargalo mais importante é identificado e todos os esforços são direcionados para sua eliminação.

A finalidade do *tuning* é diminuir o consumo de recursos ou reduzir o tempo de processamento de uma operação. Em geral, os problemas de desempenho são geralmente o resultado da disputa para, ou o esgotamento de, alguns recursos do sistema causados pelo uso excessivo de um recurso específico. Este recurso se torna o gargalo do sistema. Quando um recurso está esgotado, o sistema é incapaz de se redimensionar para níveis mais altos de desempenho, podendo até parar de funcionar corretamente.

Muitas vezes, a forma mais eficaz de solucionar um problema de gargalo é modificar a aplicação ou a maneira como o aplicativo é usado. Mudanças no SGBD e nas configurações de *hardware* do *host* também podem ser necessárias.

2.2 Projetando e Desenvolvendo para o Desempenho

Para que o sistema tenha um bom desempenho, é necessário iniciar a análise durante a fase de *design* e continuar ao longo da vida do aplicativo. Para que seja mais fácil ajustar o sistema durante a produção é preciso considerar com cuidado os problemas de desempenho durante a fase inicial do projeto. Caso algum problema de desempenho seja detectado provavelmente mudanças no projeto de *design* da aplicação serão necessárias com a finalidade de aperfeiçoar o sistema.

2.2.1 Opções de Investimento de *Hardware*

Atualmente, existe a disponibilidade de processamento de alta potência, e unidades de disco e memórias de custo relativamente baixo. Com isso, existe uma tendência a se comprar mais recursos de *hardware* para melhorar o desempenho imediato, assim que algum problema relacionado a este assunto surge.

No entanto, qualquer aumento de desempenho alcançado pelo *upgrade* do *hardware* deve ser considerado como um alívio em curto prazo para um problema imediato. Se a demanda e as taxas de carga na aplicação continuarem a crescer, é muito provável que o mesmo problema surja em um momento futuro quando a melhoria do recurso de *hardware* não mais atender a demanda do sistema.

Existem ainda situações em que o *hardware* adicional não melhora o desempenho do sistema. Sistemas mal projetados executam mal, não importa quanta memória, disco rígido ou processador adicional sejam alocados para suprir a necessidade da aplicação. Para que isto não ocorra, a aplicação precisa ter a capacidade de processar mais carga de trabalho com o aumento proporcional dos recursos.

De acordo com [9], antes de se investir em compra de *hardware* complementar, é necessário se certificar, por exemplo, que não exista serialização ou *threading* única, que é um único fluxo de controle seqüencial dentro de um programa, acontecendo dentro do aplicativo. Este tipo de ajuste, a longo prazo, tornasse mais valioso para aumentar a eficiência da aplicação.

2.2.2 *Tuning* de *Hardware*

Uma vez detectada a real necessidade de se fazer *tuning* de *hardware*, deve-se levar em consideração o dimensionamento do *hardware*, planejamento da capacidade e *design* do *hardware*, bem como sua configuração.

Escolher o recurso correto é uma decisão importante que se pode tomar para aperfeiçoar o desempenho do sistema. Na maioria dos casos, o sistema de baixo processamento vai exigir a adição de novos recursos mais tarde quando, por exemplo, a quantidade de usuários que o acessam simultaneamente crescer consideravelmente.

A seguir, de acordo com [4], são abordados os principais componentes de *hardware* considerados em um projeto de otimização:

- **CPU (*Cental Pocessament Unit)***

  Pode haver uma ou mais CPUs, cada uma com um ou mais núcleos, e podem variar em poder de processamento de simples unidades de processamento encontradas em dispositivos móveis (caso o sistema seja construído para executar em aparelhos celulares) até CPUs de alta potência (encontradas em servidores).

  O dimensionamento dos outros componentes de *hardware* é proporcional ao poder de processamento e freqüência de operação das CPUs do servidor.

- **Memória**

  Servidores de banco de dados e aplicação requerem uma quantidade de memória considerável para armazenar dados e evitar demoras no acesso ao disco rígido.

  O Oracle armazena informações em *caches* de memória e no disco rígido. Acessar a memória é muito mais rápido do que o acesso ao disco, que leva uma quantidade significativa de tempo. Tipicamente esta diferença de tempo de acesso é da ordem de dez milissegundos.

- **Subsistema de I/O**

  O subsistema de I/O é composto por vários componentes, incluindo o barramento de I/O no servidor, o controlador de I/O, o barramento de I/O no sistema de armazenamento e em última análise, a unidade de disco. Podem variar entre o disco rígido de um computador pessoal cliente até *arrays* de discos de elevado desempenho.

  Arrays de discos podem executar milhares de solicitações de I/O a cada segundo e fornecer disponibilidade através de redundância em termos de múltiplos caminhos e discos espelhados redundantes.

- **Rede**

  A grande parte dos computadores em um sistema está conectada a uma rede, que pode variar de uma simples linha de modem a uma *Local Architecture Network* - LAN - de alta velocidade interna.

  As principais preocupações com as especificações de rede são largura de banda (volume) e a latência (retardo na velocidade).

De acordo c om [5], algumas perguntas podem ser feitas para ajudar na escolha do *hardware* correto:

- Que tipo de aplicativo será executado?
- Quanta processamento será exigido da *Cental Processament Unit* - CPU?
- Que tipo e quantas CPUs / núcleos são necessários para suportar a carga do sistema esperado?
- Qual a quantidade de memória física necessária?
- Será que os servidores devem ser configurados com um *Real Application Cluster* (RAC)? Se sim, quantos nós devem haver?
- Qual a velocidade dos componentes de rede?

- Qual o tamanho ideal das unidades de armazenamento?

As respostas a estas e outras perguntas ajudam o administrador do sistema a decidir qual a configuração ideal de *hardware* para a aplicação.

2.2.3 *Tuning* de *Software*

Da mesma forma que os computadores possuem componentes de *hardware* comuns, os aplicativos possuem componentes funcionais. Esta divisão do *software* em componentes funcionais nos permite compreender melhor a aplicação e sua arquitetura.

Alguns componentes do sistema são adquiridos para acelerar a implementação do aplicativo ou para dar suporte a reusabilidade de componentes comuns em outras partes da aplicação.

A principal diferença entre componentes de *software* e de *hardware* é que enquanto os componentes de *hardware* apenas executam uma tarefa, uma parte do *software* pode realizar o papel de vários componentes de *software*. Por exemplo, uma unidade de disco rígido apenas armazena e recupera dados gravados, mas um programa cliente pode gerenciar a interface do usuário e executar lógica de negócios definida na implementação [4].

Para que uma aplicação execute de forma satisfatória e com eficácia, os componentes de *software* devem estar bem definidos e interligados de forma a aperfeiçoar o desempenho da aplicação.

De acordo com [4] a maioria das aplicações possui os seguintes componentes de *software*:

- **Gerenciador da interface do usuário**

  Este componente é o mais visível para o usuário do aplicativo que interage diretamente com o sistema através deste componente.

  A coleta de dados do usuário que serão transferidos para a lógica de negócios é realizada através da interface do usuário. Outras

funcionalidades importantes como a validação dos dados de entrada e a navegação através de telas, níveis ou estados da aplicação também são características particulares.

- **Implementação de lógica de negócios**

  Este componente implementa as regras que regem toda a funcionalidade intrínseca do sistema. Erros cometidos neste nível podem ser muito caros.

  São funcionalidades comuns deste componente: mover um modelo de dados para uma estrutura de tabelas relacionais, definir restrições na estrutura de tabelas relacionais e codificação da lógica processual para implementar as regras de negócio do sistema.

- **Gerenciador da requisição do usuário e alocação de recursos**

  Este componente é implementado em todas as partes do *software*. No entanto, existem algumas requisições e recursos que podem ser influenciados pelo *design* da aplicação e outros que não podem.

  Em um aplicativo multiusuário, a maioria das alocações de recursos por requisição de usuários são tratadas pelo servidor de banco de dados ou o sistema operacional. No entanto, em uma aplicação de grande porte, onde o número de usuários cresce rapidamente; o administrador do *software* deve tomar medidas pró-ativas, a fim de assegurar que nenhum componente de *software* se tornará sobrecarregado e instável.

  Dentre as funcionalidades deste componente pode-se destacar o gerenciamento de conexões com o banco de dados, execução de comandos *SQL – Structured Query Language* - de forma eficiente, informações de estado e gestão de clientes e balancear a carga de solicitações de usuários através de recursos de *hardware* [4].

- **Gerenciador de dados e transações**

Este componente está intimamente ligado ao servidor de banco de dados e ao sistema operacional. Entre suas principais funcionalidades pode-se destacar o fornecimento do acesso simultâneo aos dados de forma otimizada e aplicação das regras definidas para os dados.

2.3 *Tuning* em SGBDs

Os Sistemas de Gerenciamento de Banco de Dados (SGBDs) são programas com a finalidade de manipulação das informações de um banco de dados [10]. Observando a Figura 1 pode-se notar que o SGBD serve como interface que gerencia como a aplicação acessará os dados.

Figura 1. SGBA como interface entre aplicação e a base de dados.

Dentre as funções de um SQBD pode-se destacar o armazenamento dos dados e meta-dados, a eficiência na recuperação de dados armazenados, garantia que as restrições impostas sobre os dados serão satisfeitas, tratamento correto e eficiente de acesso simultâneo aos dados armazenados, recuperação contra falhas e gerenciamento de *backups.*

Em um SGBD existem alguns recursos que colaboram para a melhoria de desempenho do sistema. Entre eles pode-se destacar:

- ***Stored Procedure*** é um conjunto de instruções implementadas, que, uma vez armazenadas ou salvas, ficam dentro do servidor pré-compiladas, aguardando que o usuário do banco de dados invoque sua execução.

  As *stored procedures*, após salvas no servidor, ficam em uma posição da memória *cache* somente aguardando serem chamados para executarem uma operação. As ações também já ficam pré-carregadas,

dependendo somente dos valores dos parâmetros. Após a primeira execução, elas se tornam ainda mais rápidas [12].

- ***Function*** é bem semelhante a uma *stored procedure* e sua principal diferença que a primeira retorna um valor e não pode invocar a execução da segunda.

- ***Trigger*** é um gatilho disparado quando um determinado evento de inclusão, alteração ou exclusão acontece. Uma *trigger* pode chamar uma *procedure* ou *function* para executar uma tarefa quando um evento acontece.

Os elementos citados podem colaborar para o desempenho do sistema quando substituem uma parte da regra de negócios que, no caso de sistemas *web*, muitas vezes é executada pelo próprio *browser* do *host*. Assim a velocidade de exibição de dados no *browser* aumenta. Em toda tarefa de *tuning* é necessário realizar uma análise detalhada de ganho real que a aplicação pode ter, pois a criação de *triggers*, *procedures* e *functions* desnecessárias podem comprometer o desempenho do servidor de banco de dados. Caso haja sobrecarga no servidor, deve-se considerar uma melhoria no *hardware* para este fim.

De acordo com [4], existem outros recursos de tuning em um SGBDs como o otimizador do SGBD que avalia o melhor plano de execução para a declaração SQL, índices que são estruturas associadas as tabelas que melhoram a execução das consultas e hints que são dicas de execução de *queries* que o desenvolvedor pode dar ao SGBD ignorando a atuação do otimizador. Estes elementos serão estudados com detalhes no Capítulo 3 que diz respeito a *tuning* de declarações *SQL*.

De um modo geral todos os recursos citados acima colaboram diretamente para o melhor desempenho de declarações SQL e podem contribuir para eliminação de gargalos do sistema quando estas ferramentas são utilizados de forma planejada. Um SGBD precisa ser bem administrado para que o sistema de banco de dados funcione de forma eficiente e eficaz. O *DBA* (*Data Base Administrator*) é o profissional responsável pela administração da base de dados [4].

A Figura 2 exibe a arquitetura básica de um SGBD.

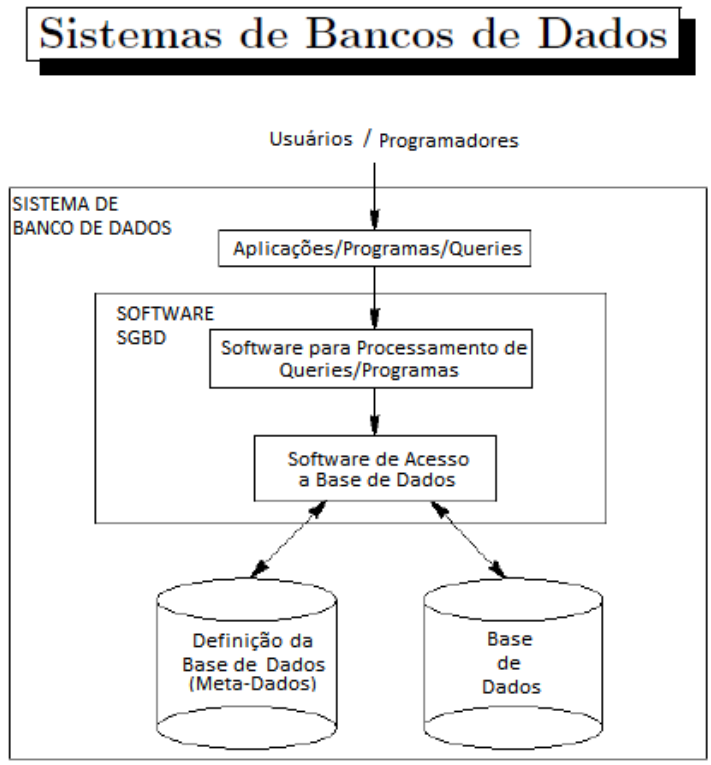

Figura 2. Arquitetura básica de um sistema de banco de dados [10].

2.4 Estrutura do SGBD Oracle 10g

O *Oracle Server Database* 10g é o SGBD que utilizaremos para estudo no decorrer desta monografia. Ele permite criar, armazenar, gerenciar e recuperar os dados no banco de dados. O conhecimento da arquitetura interna do Oracle é de grande importancia para um bom entendimento das técnicas de otimização e *tuning* do SGBD. Para isto, um administrador de banco de dados deve ter uma compreensão aprofundada da arquitetura de banco de dados.

Uma analogia poderia ser um mecânico tentando consertar um carro que está com um problema. O técnico não saberia por onde começar, a menos que ele conheça muito bem quais são os componentes do carro, o que eles fazem, como eles funcionam e como eles interagem entre si. Isso é muito importante antes que ele possa começar a procurar uma solução para o problema. O mesmo vale para um

banco de dados; se um usuário apresenta um problema a um *DBA*, este não poderá chegar a uma solução correta se não conhecer tudo, dentro do possível, sobre o banco de dados.

O servidor Oracle consiste em arquivos físicos e componentes de memória. O *Oracle Database 10g* é um produto composto por três componentes principais [6]:

- **O servidor Oracle**

  Este é o SGBD Oracle que é capaz de armazenar, gerenciar e manipular dados. Ele consiste de todos os arquivos, estruturas e processos que formam o *Oracle Database 10g*. O servidor Oracle é composto de uma instância Oracle e um banco de dados Oracle.

- **A instância Oracle**

  Consiste dos componentes de memória do Oracle e os processos de segundo plano (*background process)*.

- **O banco de dados Oracle**

  Este é o repositório centralizado onde os dados são armazenados. Possui uma estrutura física que é visível para o sistema operacional composto por arquivos do sistema operacional e uma estrutura lógica que é reconhecida apenas pelo servidor Oracle.

A Figura 3 apresenta a arquitetura do banco de dados Oracle 10g, conforme descrita em [6]. Ele é dividido em componentes de memória que formam a instância do Oracle e os componentes do banco de dados físico, onde diferentes tipos de dados são armazenados.

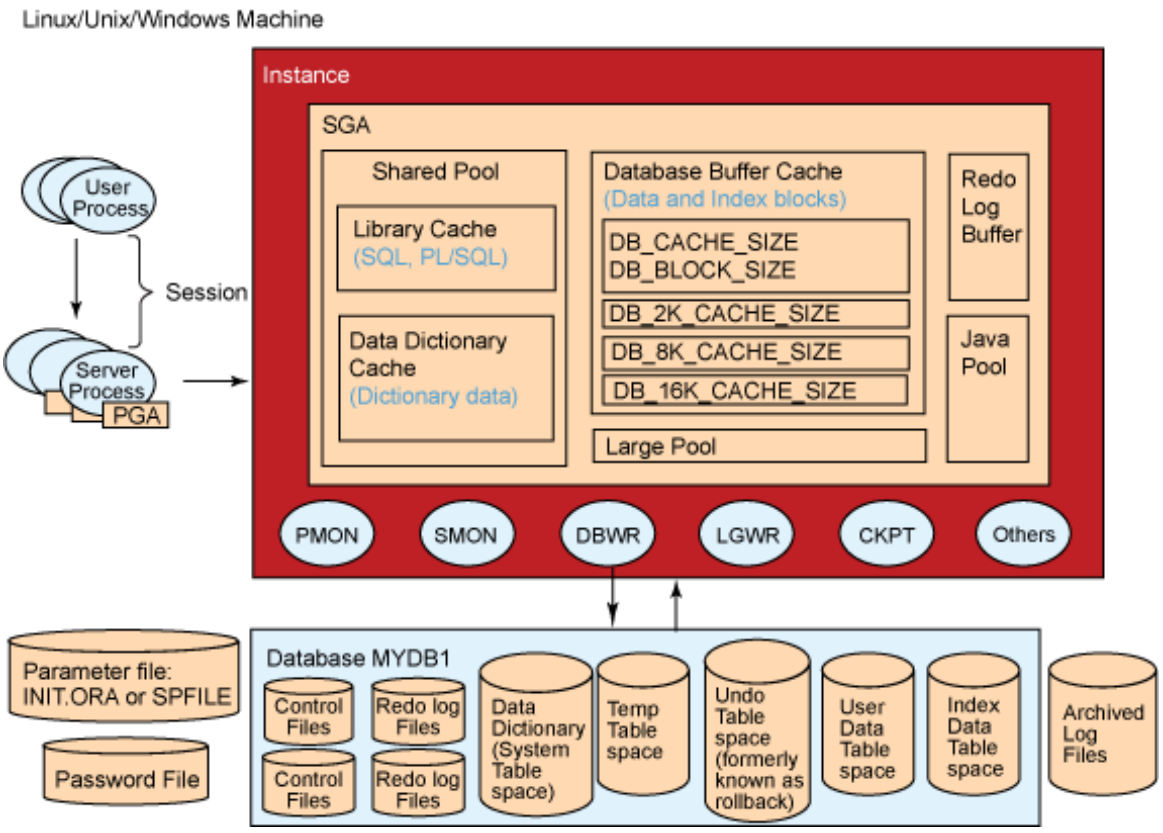

Figura 3. Arquitetura do Oracle 10g [6].

2.4.1 A Instância Oracle

A instância Oracle é composta de alguns componentes de memória e processos de segundo plano. A instância é criada na memória todas as vezes que o banco de dados é iniciado, e é associada a apenas um único banco de dados. Os componentes de memória compartilhados do banco de dados Oracle 10g também são conhecidos como SGA (*Sistem Global Area)*.

A instância e seus componentes são configurados através de um arquivo conhecido como o arquivo de parâmetros. No Oracle 10g, existem dois tipos de arquivos de inicialização, ou seja, os arquivos de parametro spfile e pfile. Nestes arquivos são definidos os tamanhos de vários *buffers* e *pools* do SGA. Também é possível especificar o nome da instância, o nome do banco de dados e outros parâmetros relacionados com o tamanho necessário para a instância Oracle [7].

2.4.2 PGA

A PGA (*Program Global Area)* é um *buffer* de memória que contém os dados e informações de controle de uma sessão de um usuário. A PGA é criada e alocada quando um novo processo é inicializado no servidor. As suas informações dependem da configuração do Oracle.

Existe uma área de memória PGA para cada usuário que está executando seus trabalhos no SGBD. Dentro da PGA existem três estruturas: uma contendo um espaço para a pilha (para armazenar as variáveis e matrizes), outra contendo dados sobre a sessão do usuário e uma terceira com as informações dos cursores usados. A PGA não é compartilhada entre os usuários. Ela é única para cada sessão [7].

2.4.3 Componentes de Memória da SGA

A instância do Oracle é composta da área do Sistema Global (SGA) e os processos de segundo plano. Conforme [4] O SGA é composto pelos seguintes componentes de memória:

- ***Redo Log Buffer***

  É um *buffer* circular que armazena todas as alterações feitas no banco de dados. Este conteúdo é transferido periodicamente para a memória para os arquivos de *redo log on-line* pelo processo de segundo plano *LGWR* (*Log Writer*).

  O conteúdo do *redo log buffer* é essencial para a recuperação da instância no caso de falha. O tamanho do *redo log buffer* é determinado pelo parâmetro de inicialização *log_buffer*.

- ***Database Buffer Cache***

  É uma área na memória que armazena todos os blocos lidos em disco para consulta ou modificação. Os blocos que precisam ser modificados são modificados ainda carregados na memória e são gravados no disco periodicamente.

O conteúdo do *database buffer cache* é compartilhado por processos de múltiplos usuários e é gravado em arquivo de dados no disco pelo processo de segundo plano *DBWR* (*Database Writer*). O tamanho do *database buffer cache* padrão é definido pelo parâmetro de inicialização DB_BLOCK_SIZE e o tamanho dos blocos de informação pelo DB_BLOCK_SIZE.

- ***Shared Pool***

  O conteúdo desta área de memória é compartilhado por vários usuários e o seu tamanho pode ser definido através do parâmetro de inicialização SHARED_POOL_SIZE. Dois caches específicos compõem a *shared pool*.

  O primeiro deles é a *lybrary cache* que é responsável por armazenar e reutilizar, sempre que possível, instruções *SQL*, os planos de execução das *queries* e procedimentos armazenados. Este *cache* impede que instruções *SQL* sejam analisadas repetidamente. Isto colabora para melhorar o desempenho do sistema durante a execução de comandos *SQL*.

  O último *cache* da *shared pool* é o *data dictionary cache* que é composto por blocos que mantêm em arquivos de dados informações lidas do dicionário de dados. As informações normalmente armazenadas são referentes à conta do usuário, índice de tabela e outras informações de objetos, privilégios e conteúdo relevante que é acessado com frequência.

- ***Large Pool***

  O parâmetro de inicialização LARGE_POOL_SIZE delimita o tamanho desta área de memória. É utilizada para tratamento de requisição de I/O muito grandes feitas pelos processos do servidor.

- ***Java Pool***

Esta é uma área de memória usada para toda a sessão de código Java específico e pelos dados dentro da máquina virtual Java. Seu tamanho é configurado através do parâmetro de inicialização JAVA_POOL_SIZE.

- ***Streams Pool***

  É uma área de memória usada pelo produto *Oracle Streams* para o seu funcionamento.

O estudo dos componentes da *SGA* é importante, pois uma aplicação pode ter problemas de desempenho devido ao mau dimensionamento das áreas de memória estudadas acima. O *DBA* precisa conhecer bem os parâmetros de inicialização para poder corrigir o dimensionamento de cada região de memória visando eliminar um possível gargalo no sistema determinado por fatores de dimensionamento que afetam o desempenho da execução da instância do SGBD.

2.4.4 Processos de Segundo Plano

Os processos de segundo plano, também conhecidos como *background process* de uma instância do Oracle, são responsáveis por executar funções de I/O assíncronas entre a instância do Oracle e arquivos físicos do banco de dados que existem no disco rígido. Há cinco processos de segundo plano principais no Oracle 10g [7]. São eles:

- **DBWR (Database Writer)**

  Este é um processo de segundo plano tem como função transferir blocos modificados, a partir do *database buffer cache*, para os arquivos de dados. O *DBWR* escreve os blocos modificados do *database buffer cache* para os arquivos de dados físicos. O *DBWR* é otimizado para reduzir os eventos de I/O, por esta causa ele não escreve os dados a cada comando COMMIT. Geralmente o *DBWR* escreve os dados para o disco se muitos dados estão armazenados no *database buffer cache* na SGA e não existe espaço livre para novos dados. Os dados menos

recentemente usados são escritos para os arquivos de dados em primeiro lugar.

- **LGWR (*Log Writer*)**

  O processo *LGWR* grava todas as entradas de *redo log* para o disco. Os dados de *redo log* são armazenados em memória no *redo log buffer cache*, na *SGA*. No momento em que uma transação é realizada e efetivada com o comando *commit* e o *redo log buffer* estiver preenchido, o *LGWR* escreve as entradas de redo log nos arquivos apropriados.

- **CKPT (*Checkpoint*)**

  Em um momento específico, todos os dados do *database buffer cache* modificados são escritos em disco pelo processo *DBWR*; este evento é chamado de *checkpoint*. O processo *CKPT* é responsável por informar ao processo *DBWR* o momento de gravar os dados em disco. O *DBWR* também atualiza os arquivos de controle do banco de dados para indicar o *checkpoint* mais recente. O processo *CKPT* é opcional, se ele não estiver presente, o *LGWR* assume sua responsabilidade.

- **SMON (*System Monitor*)**

  O processo *SMON* efetua a recuperação da instância em caso de falhas, durante a sua inicialização. Em um sistema com múltiplas instâncias (como na configuração Oracle *Parallel Server*, por exemplo), o processo *SMON* de uma instância também pode executar a recuperação de outras instâncias que podem ter falhado. Ele também limpa os segmentos temporários que não estão sendo usados, liberando memória, e recupera qualquer transação pendente no caso de uma falha em arquivos físicos ou mesmo no disco. O processo de recuperação dessas transações é executado pelo processo *SMON* quando a *tablespace* afetada volta a ficar disponível.

- **PMON (*Process Monitor*)**

O *PMON* executa a recuperação do processo de um usuário quando esse processo falha. Limpa a área de memória e libera os recursos que o processo do usuário estava usando. O *PMON* também verifica os processos servidores e os reinicializa se tiver acontecido qualquer falha ou travamento.

- **ARCH (*Archiver*)**

  Este é um processo de segundo plano opcional que copia os arquivos *redo log* para fita ou mesmo outro disco, no momento em que um deles torna-se completo. Esse processo geralmente está presente quando o banco de dados está sendo utilizado no modo *archivelog*. Os arquivos *redo log* são usados somente para a recuperação de um banco de dados.

Capítulo 3

Tuning de Declarações *SQL*

Este capítulo aborda as características gerais da otimização de declarações *SQL* para se alcançar um desempenho ótimo das *queries*.

O conhecimento de técnicas de *tuning* de declarações *SQL* é necessário para se realizar a otimização da base de dados. O *DBA* precisa ter uma visão global das possibilidades de gargalos que geram tempos de resposta muito acima do esperado, pois, em alguns casos, esta latência é decorrente de consultas *SQL* mal elaboradas e que podem ser otimizadas utilizando-se de algumas técnicas que serão comentadas no decorrer deste capítulo.

3.1 O Otimizador Oracle

A função do otimizador Oracle é aperfeiçoar a execução de consultas *SQL* visando obter melhor desempenho. Uma declaração *SQL* pode ser executada, dependendo da situação, de várias formas diferenciadas, tais como varredura de índices, *full table scan* (que é uma varredura completa em todos os registros da tabela), *loops* aninhados e *joins*.

O otimizador Oracle determina qual plano de execução é mais eficiente. Ele considera os caminhos de acesso disponíveis e a fragmentação da informação baseado em estatísticas para os objetos do esquema do banco de dados acessados pela consulta *SQL*. Ele considera também os *hints* que são sugestões de otimização colocadas na instrução *SQL* [11].

O otimizador de consultas executa as seguintes etapas:

- Gera um conjunto de planos de execução da instrução com base em caminhos de acesso disponíveis e *hits*.

- Estima o custo (tempo e uso dos recursos) de cada plano de execução, baseado em estatísticas do dicionário de dados e na característica de distribuição e armazenamento das tabelas e índices acessados pela consulta.

- O otimizador compara o custo dos planos de execução e escolhe um com o menor custo.

Os componentes do otimizador de *queries* são exibidos na Figura 4:

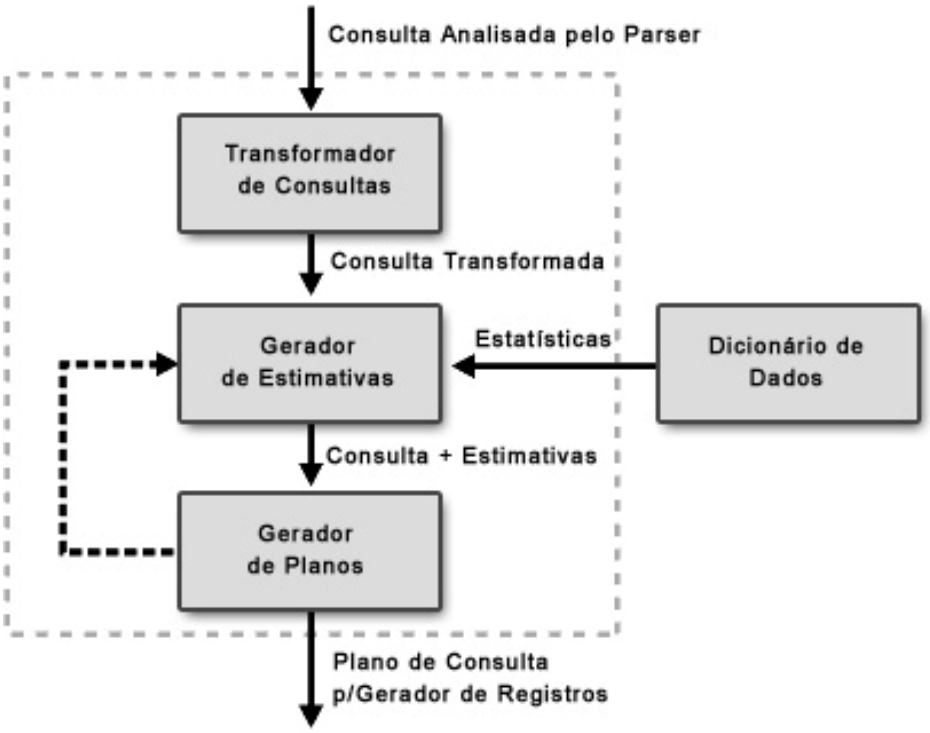

Figura 4. Componentes do Otimizador Oracle [11].

O transformador de consultas é o primeiro componente do Otimizador Oracle. Ele recebe como entrada uma declaração *SQL* analisada pelo analisador gramatical do Oracle e o divide em blocos relacionados entre si.

O transformador de consultas tem como objetivo principal realizar a verificação a respeito da existência de uma maneira ótima para se obter o mesmo resultado. Se isto existir, ele gera um melhor plano de execução para a *query*.

Este processo utiliza três técnicas, que podem ser utilizadas combinadamente ou individualmente. São elas: agrupamento das visões (*view merging*), eliminarção de sub-consultas e reescrita da consulta utilizando visões materializadas [4].

O gerador de estimativas pode ser considerado o principal componente do otimizador, pois é o responsável por indicar o esforço estimado para cada plano apresentado pelo transformador de consultas. Para gerar a estimativa, o gerador de estimativas baseia-se em seletividade, cardinalidade e custo da execução.

Caso o banco de dados possua algumas estatísticas sobre a tabela ou índice, isto será utilizado para melhorar a precisão da estimativa.

O gerador de planos é o componente que testa os diferentes planos de execução para uma consulta e escolhe aquele que apresenta o menor custo de execução e conseqüentemente um melhor desempenho.

3.2 Índices

Conforme descrito em [8], "*Os índices são estruturas opcionais associadas a tabelas que permitem que as consultas SQL sejam executadas mais rapidamente*.".

Semelhantemente a um índice de um livro, um índice é um objeto logicamente e fisicamente independente do dado associado à tabela.

A criação dos índices no banco de dados deve ser estudada de acordo com a real necessidade e a freqüência de utilização dos dados da tabela em estudo pela aplicação. Os índices podem reduzir bastante o acesso aos recursos de I/O, contribuindo assim para melhorar o desempenho da aplicação como um todo.

Os índices são objetos importantes para realização de *tuning* das consultas e podem ser usados sem a necessidade de reescrever as declarações *SQL*. No entanto, podem interferir negativamente no desempenho geral da aplicação se forem usados de forma incorreta. Não é recomendado o uso de índices em dados que sofrem constantes modificações. Uma vez criado o índice para uma tabela específica, o Oracle precisará mantê-lo atualizado, ou seja, operações de

atualização, inserções e exclusões de linhas na tabela, geram conseqüentemente atualizações no índice associado a ela [8].

Existem vários tipos de índices disponíveis no Oracle, e cada um deles possui benefícios para determinadas situações. A lista a seguir dá idéias de desempenho associadas a cada tipo de índice de acordo com [4]:

- **Índice B-Tree**

  Este é o tipo de índice padrão. Ele apresenta um ótimo resultado quando utilizado para indexar a chave primária da tabela, valores únicos ou próximos de únicos, ou linhas que estão dentro de uma faixa de valores como um intervalo de datas por exemplo. São usados como índices concatenados. Índices B-tree podem ser utilizados para recuperar dados classificados pelas colunas indexadas.

- **Índice Concatenado ou Composto**

  Um índice concatenado é um índice que é criado sobre múltiplas colunas em uma tabela. As colunas em um índice concatenado podem aparecer em qualquer ordem, e através desta ordem é definido se o índice completo é usado ou não na consulta. A vantagem de uma chave concatenada é que freqüentemente ela é mais precisa na busca de um grupo de registros que uma chave única. Ela melhora o desempenho do índice, pois a combinação de colunas irá apontar para um menor número de linhas que índices compostos de colunas individuais.

- **Índice Agrupado**

  Os índices agrupados são mecanismos para armazenar linhas relacionadas de uma ou mais tabelas no mesmo segmento. Linhas que têm o mesmo valor chave para o agrupamento são armazenadas juntas. Na teoria isto irá aumentar a velocidade de junções de tabelas, porque as linhas a serem unidas estarão armazenadas no mesmo bloco. Na prática, índices agrupados são bastante limitados e apenas

devem ser usados quando as tabelas são “sempre” referenciadas juntas.

Algumas das desvantagens de índices agrupados são: *full table scan* em apenas uma das tabelas do agrupamento será bastante lento, pois os blocos para outras tabelas no agrupamento também terão que ser percorridos; as inserções podem ser lentas por causa do esforço adicional de manter o agrupamento; os benefícios de desempenho para as junções podem ser mínimas.

- **Índice *Bitmap***

  Este tipo é apropriado para os dados de baixa cardinalidade, ou seja, colunas com poucos valores distintos, caso contrário muitos bitmaps precisam ser criados e mantidos. Através de técnicas de compressão, podem gerar um grande número de rowids com eventos de I/O mínimos.

  Combinar índices bitmap sobre colunas não seletivas permite que operações de AND e OR sejam executadas de forma eficiente com eventos de I/O minimos. Índices bitmap são particularmente eficiente em consultas que utilizam COUNT(), porque a consulta pode ser satisfeita dentro do índice.

  Em um índice *bitmap*, o *Oracle* cria um *bitmap* para cada valor único da coluna em análise. Cada *bitmap* contém um único *bit* (zero ou um) para cada linha na tabela. Um *bit* “1” indica que a linha tem o valor especificado pelo *bitmap* e um “0” indica que não tem.

  O Oracle pode rapidamente percorrer estes *bitmaps* para encontrar linhas que satisfazem um critério específico, também pode rapidamente comparar múltiplos bitmaps para encontrar todas as linhas que satisfazem aos múltiplos critérios. Podem ser usados em qualquer ordem e em qualquer combinação, assim são mais flexíveis que um índice concatenado correspondente, o qual requer o uso de pelo

menos a primeira coluna do índice. Se usado apropriadamente é bastante compacto, bem mais que o índice concatenado.

O conhimento dos diferentes tipos de índices possíveis do Oracle é importante para a realização de *tuning* quando necessário. É importante destacar que o uso desta técnica, sem uma análise aprofundada e com embasamento teórico, não garante melhora no desempenho. Realizar *tuning* com eficiência é saber qual técnica ele deve usar, em que momento e em quais casos específicos.

Abaixo a sintaxe geral de criação de índices oficial da Oracle é exibida [8]:

```
CREATE [UNIQUE | BITMAP] INDEX [schema.]index

ON { cluster_index_clause

     | table_index_clause

     | bitmap_join_index_clause

     };
```

3.3 *Hints*

Apesar do otimizador de consultas do Oracle realizar um bom trabalho na escolha da melhor forma de executar uma declaração *SQL* e no uso de índices para milhares de consultas realizadas pelo sistema, ele não é perfeito.

O *DBA* pode saber informações sobre os seus dados que o otimizador não reconhece. Por exemplo, o administrador do banco de dados pode saber que um determinado índice é mais seletivo para determinadas consultas. Com base nessas informações, o *DBA* pode ser capaz de escolher um plano de execução mais eficiente que o plano escolhido automaticamente pelo otimizador. Nestes casos o uso de *hints* (sugestões) é indicado para forçar o otimizador a usar um plano de execução ótimo.

O Oracle possui *hints* que se pode usar para determinadas consultas, de modo que o otimizador seja desconsiderado, na esperança de conseguir melhor desempenho para determinada consulta.

Os *hints* modificam o caminho de execução quando um otimizador processa uma instrução específica. O parâmetro OPTIMIZER_MODE do arquivo de parâmetros “init.ora” pode ser usado para modificar todas as instruções no banco de dados para que sigam um caminho de execução específico, mas um *hint* para um caminho de execução diferente substitui qualquer coisa que esteja especificada no init.ora.

Como em toda a tarefa de análise para realização de *tuning*, os *hints* devem ser utilizados apenas quando houver necessidade. A escolha do plano de execução feita pelo otimizador, em geral, atende as necessidades de realização de consultas *SQL* com um menor custo para a aplicação em geral.

Os *hints* disponíveis variam de acordo com a versão do banco de dados instalado. Embora este trabalho focalize apenas os *hints* que são usados com maior freqüência, muitos dos *hints* que não são abordados com detalhes podem oferecer grandes ganhos de desempenho com um sistema específico.

Hints aplicam-se apenas para otimizações do bloco de uma declaração em que aparecem. Um bloco de declaração é uma instrução SELECT, UPDATE ou DELETE simples, uma declaração pai ou subconsulta de uma declaração complexa, ou uma parte de uma consulta composta.

Por exemplo, uma consulta composta formada por duas consultas combinadas pelo operador UNION tem dois blocos, sendo um para cada um das consultas componentes. Por esta razão, os *hints* na primeira consulta componente se aplicam apenas a sua otimização, e não para a otimização da segunda consulta componente.

3.3.1 Classificação dos *Hints*

Os *hints* possuem a seguinte classificação geral:

- ***Single-table***

  São especificados em uma tabela ou *view*. INDEX e USE_NL são exemplos de *hints single-table*.

- ***Multi-table***

  São semelhantes aos *hints* single-table, exceto que os *hints* multi-tables podem especificar uma ou mais tabelas ou *views*. LEADING é um exemplo de *hint multi-table*.

- ***Query block***

  Operam em blocos de *query* únicos. STAR_TRANSFORMATION e UNNEST são exemplos de *hints query block*.

- ***Statement***

  São aplicados a todas as declarações *SQL*. ALL_ROWS é um exemplo de um *hint statement*.

3.3.2 Sintaxe dos *Hints*

Os *hints* enviam sugestões em uma declaração *SQL* para o otimizador em forma de comentário dentro da própria declaração. Um bloco de uma declaração pode conter apenas um comentário contendo *hints* após as palavras-chave SELECT, UPDATE, MERGE ou DELETE.

A seguir são exibidos os dois estilos de comentário que o Oracle suporta em um bloco de declaração *SQL* [4].

```
{DELETE | INSERT | MERGE | SELECT | UPDATE} /*+ hint
[text] [hint[text]]... */
```

ou

```
{DELETE | INSERT | MERGE | SELECT | UPDATE} --+ hint
[text] [hint[text]]...
```

Onde:

- DELETE, INSERT, SELECT, MERGE e UPDATE são palavras-chave que iniciam um bloco de declaração.
- O símbolo + faz o Oracle interpretar o comentário como uma lista de *hints*. O sinal de mais deve vir imediatamente após o delimitador de comentário.
- *hint* é um dos *hints* discutidos nesta seção. Se o comentário possuir múltiplos *hints*, então cada um deve ser separado do outro pelo menos por um espaço em branco.

O Oracle ignora *hints* que sejam criados de forma incorreta, mas considera outros *hints* que sejam definidos corretamente dentro do mesmo comentário.

3.3.3 Tipos de *Hints*

- ***Hints* de métodos de acesso**

  Os *hints* que são agrupados em métodos de acesso permitem que o otimizador varie o modo como a consulta real é acessada. Esse grupo de *hints* é usado freqüentemente, especialmente o *hint* INDEX. Ele oferece orientação a respeito de se e como os índices são usados, e como os índices correspondentes serão mesclados para chegar à resposta final. Os *hints* de método de acesso são os seguintes [4]:

AND_EQUAL	CLUSTER	FULL
HASH	INDEX	INDEX_ASC
INDEX_COMBINE	INDEX_DESC	INDEX_FFS
INDEX_JOIN	NO_INDEX	RPWID

- ***Hints* de transformação de consulta**

Este tipo de *hint* é útil especialmente em data warehouse onde são utilizadas tabelas de fato e dimensão. O *hint* FACT pode forçar determinada tabela a ser tabela FACT ou principal para uma consulta. O *hint* NO_FACT realiza o oposto. O *hint* STAR é usado apenas para acessar de modo eficaz a tabela FACT na junção de várias tabelas. Os *hints* de transformação da consulta são os seguintes [4]:

FACT	MERGE	NO_EXPAND
NO_FACT	NO_MERGE	NOREWRITE
REWRITE	STAR	USE_CONCAT
STAR_TRANSFORMATION		

- ***Hints* de operação de junção**

  O agrupamento das operações de junção mostra como as tabelas mesclam dados. Uma operação de junção, como USE_MERGE ou USE_HASH, pode ser melhor para apanhar todas as linhas para uma consulta (vazão), enquanto USE_NL pode ser melhor para apanhar a primeira linha (tempo de resposta). Os *hints* de operação e junção são os seguintes [4]:

DRIVING_SITE	HASH_AJ	HASH_SJ
LEADING	MERGE_AJ	MERGE_SL
NL_AJ	NL_SJ	ORDERED
PUSH_SUBQ	USE_HASH	USE_MERGE
USE_NL		

- **Usando a execução paralela**

  O agrupamento de execução paralela aplica-se a bancos de dados usando a opção paralela. Estes *hints* redefinem a especificação da

tabela para o grau de paralelismo. Os *hints* de execução paralela são os seguintes:

NOPARALLEL	NOPARALLEL_INDEX	PARALLEL
PARALLEL_INDEX	PQ_DISTRIBUTE	

- **Outros *Hints***

  Os *hints* APPEND e NOAPPEND podem ser usados sem a opção paralela, mas constantemente são usados com ela. O agrupamento de cachê diz respeito aos *hints* que colocarão itens como usados mais recentemente (CACHE) e usados menos recentemente (NOCACHE). Os *hints* são os seguintes [4]:

APPEND	CACHE	CURSOR_SHARING_EXACT
NOAPPEND	NO_UNNEST	NO_PUSH_PRED
NOCACHE	PUSH_PRED	ORDERED_PREDICATES
UNNEST		

3.3.4 Usando *Hints*

Nesta seção serão exibidos exemplos de utilização de alguns dos *hints* mais utilizados no dia-a-dia dos *DBA*s. Detalhes de outros *hints* podem ser encontrados na documentação da Oracle [4] ou em livros específicos deste assunto.

- **Especificando múltiplos *Hints***

  Pode-se usar mais de um *hint* de cada vez, embora isso possa fazer com que algum ou todos os *hints* sejam ignorados caso estejam com a sintaxe errada ou em confronto de lógica de utilização.

  Sintaxe:

```
SELECT /*+ FULL(tabela)  CACHE(tabela) */ coluna1,...
```

A tabela na sintaxe acima corresponde à tabela na qual será realizado um *full table scan* e o *cache*.

Exemplo:

```
SELECT /*+ FULL(empregado) CACHE (empregado) */
emp_numero, nome, departamento
From empregado
Where departamento = 1;
```

A especificação de vários *hints* que entram em conflito entre si faz com que a consulta não use esses *hints*.

- **Usando um *alias***

Quando *aliases* são usados sobre determinada tabela que deseja aplicar o uso de *hint*, é necessário especificar o *alias* e não o nome da tabela no *hint*. Se o nome da tabela no *hint* for especificado quando um *alias* for usado, o *hint* será desconsiderado.

Sintaxe:

```
SELECT /*+ FULL(A) */ coluna1, from tabela1 as A
```

- **Usando o *hint* INDEX**

Este *hint* é utilizado para forçar que um ou mais índices sejam executados para determinada consulta. Pode-se especificar um ou mais índices com este *hint*, e o Oracle escolherá um ou mais índices especificados com base no melhor plano de execução. Caso se queira especificar apenas um, o otimizador considerará apenas um índice.

Sintaxe:

```
SELECT /*+ INDEX (tabela indice1, indice2,...) */
coluna1, . . .
```

Exemplo:

```
SELECT /*+ INDEX (empregado depart_idx) */
emp_numero, nome, departamento
From empregado
Where departamento = 1;
```

Neste exemplo o índice depart_idx será utilizado.

- **Usando o *hint* ORDERED**

  Este *hint* faz com que as tabelas sejam acessadas em uma ordem específica, com base na ordem das tabelas na cláusula *FROM* da consulta. Ele pode ser usado para a otimização baseada em custo.

  Sintaxe:

  ```
  SELECT /*+ ORDERED (tabela indice1, indice2,...) */
  coluna1, . . .
  ```

  Exemplo:

  ```
  SELECT /*+ INDEX (empregado deptno_idx) */
  emp_nemero, nome, departamento
  From empregado
  Where departamento = 1;
  ```

  Neste exemplo o índice deptno_idx será utilizado.

- **Usando o *hint* PARALLEL**

  Este *hint* faz com que consultas *full table scan* seja divididas em partes (o grau de paralelismo) e proceda cada parte com um processo diferente do sistema operacional. O Grau de paralelismo é aplicado a cada operação de uma instrução *SQL*.

  É possível especificar o número desejado de servidores simultâneos que podem ser usados para uma operação paralela. Pode-se especificar este *hint* às partes INSERT, UPDATE e DELETE de uma

instrução. É necessário que na criação das tabelas a cláusula PARALLEL tenha sido utilizada.

Sintaxe:

```
SELECT /*+ PARALLEL  (tabela, DEGREE, INSTANCES) */ ,
. . .
```

O grau é o número de partes em que a consulta é dividida. A instância (o segundo número especificado após o grau) representa o número de instâncias usadas.

Exemplo:

```
SELECT  /*+  INDEX  (empregado  deptno_idx)  */
emp_numero,          nome,          departamentob
From empregado
Where departamento = 1;
```

Neste exemplo o índice deptno_idx será utilizado.

- **Usando o *hint* FIRST_ROWS**

  Este índice direciona uma consulta para ser otimizada com base na recuperação mais rápida da primeira linha. Este tipo de índice é muito útil quando o desenvolvedor faz uma interface para o usuário que apanha um único registro no banco de dados. E seria uma péssima opção para sistemas de relatórios, onde um número muito grande de registros é selecionado.

  Este índice é ignorado quando utilizamos as instruções UPDATE e DELETE, pois todas as linhas recuperadas serão atualizadas ou excluídas. Também é ignorado quando utilizamos qualquer função de agrupamento (GROUP BY, DISTINCT, INTERSECT, MINUS, UNION), pois todas as linhas para o agrupamento precisam ser recuperadas.

  Sintaxe:

```
SELECT /*+ FIRST_ROWS (n) */ coluna1, . . .
```

Exemplo:

```
SELECT /*+ FIRST_ROWS (10) */ empno, ename, deptno
From emp
Where deptno = 1;
```

- **Usando o *hint* RULE**

  Cada *hint* que é emitido causa o uso do otimizador baseado em custo, exceto o *hint* ROLE que faz com que o otimizador use a otimização baseada em regra. Isso quer dizer que a distribuição dos dados na tabela e nos índices não é considerada. Ao invés disto, o otimizador executa a consulta com base em um conjunto de regras predefinidas pelo Oracle. Com exceção dos *hints* DRIVING_SITE e ORDERED, todos os outros *hints* serão ignorados quando utilizado o *hint* RULE.

  Sintaxe:

```
SELECT /*+ RULE */ coluna1, . . .
```

  Exemplo:

```
SELECT /*+ RULE */ empno, ename, deptno

From emp

Where deptno = 1;
```

- **Usando o *hint* FULL**

  Este *hint* instrui a consulta a desconsiderar o otimizador e realizar uma varredura completa da tabela. O *hint* FULL possui uma funcionalidade diferente, com base na consulta que esta sendo ajustanda. Ele pode ser usado para forçar uma varredura completa quando uma grande parte da tabela estiver sendo consultada. O custo da leitura do índice e das linhas pode ser maior do que simplesmente ler a tabela inteira.

Este *hint* pode causar um resultado inesperado. Causar uma varredura de tabela completa pode fazer com que as tabelas sejam acessadas em uma ordem diferente, pois uma tabela principal diferente é usada. Isso pode gerar um desempenho melhor, fazendo-o pensar que a varredura de tabela completa foi o benefício principal, quando a mudança da ordem da tabela principal foi a causa real do melhor desempenho.

Sintaxe:

```
Select /*+ FULL(tabela) */ coluna1, . . .
```

Exemplo:

```
SELECT /*+ FULL(emp) */ empno, ename, deptno
From emp
Where deptno = 1;
```

- **Usando o *hint* LEADING**

  À medida que a complexidade das consultas aumenta, torna-se mais difícil descobrir a ordem de todas as tabelas usando o *hint* ORDERED. Normalmente pode-se descobrir qual tabela deve ser acessada primeiro (tabela principal), mas é possível não saber qual tabela acessar depois dessa. A *hint* LEADING permite que uma tabela para controlar a consulta seja especificada. O otimizador descobre qual tabela usar depois dela. Caso mais de uma tabela seja especificada com esse *hint*, ela será ignorada. O *hint* ORDERED cancela o hint LEADING.

  Sintaxe:

```
SELECT /*+ LEADING (tabela1)  */ coluna1, . . .
```

  Exemplo:

```
SELECT /*+  LEADING (DEPT) */ emp.empno, ename,
dept.deptno, itemno From emp, dept, orders

Where emp.deptno = dept.deptno and emp.empno =
orders.empno

and dept.deptno = 1  and emp.empno = 7747  and
orders.ordno = 45;
```

- **Usando o *hint* USE_NL**

  Este *hint* normalmente é o modo mais rápido em termos de tempo de resposta para retornar uma única linha. Contudo, é conseqüentemente mais lento no retorno de todas as linhas de uma tabela. Este *hint* faz com que uma instrução seja processada usando *loops* aninhados, que retorna a primeira linha de uma tabela, com base no resultado de outra tabela. Isso é o oposto de uma junção por mesclagem, que apanha linhas que correspondem às condições de cada tabela e depois realiza a mesclagem e isso normalmente leva mais tempo para obter a primeira linha.

  Sintaxe:

```
Select /*+ USE_NL(tabela indice1, índice2) */
coluna1, . . .
```

  Exemplo:

```
Select /*+ USE_NL(dept) */ empno, ename, dept.deptno

From emp, dept

Where emp.deptno = dept.deptno and dept.deptno = 1

and emp.empno = 7747;
```

- **Usando o *hint* APPEND**

Este *hint* é ótimo para ser utilizado quando houver bastante disponibilidade de espaço no disco rígido. Ele não verifica se existe espaço dentro dos blocos atualmente usados para instruções, mas, em vez disso, anexa os dados aos novos blocos. Potencialmente poder-se-ia desperdiçar espaço, mas a velocidade da busca seria melhorada.

Além disso, se um INSERT tiver paralelismo com o *hint* PARALLEL, APPEND será utilizado como padrão. Porém, é possível usar o *hint* NOAPPEND para cancelar este comportamento.

Sintaxe:

```
INSERT /*+ APPEND*/ …
```

Exemplo:

```
insert /*+ APPEND */ into emp (empno, deprno) values
(7747,10);
```

- **Usando o *hint* USE_HASH**

Este *hint* normalmente é o modo mais rápido de unir muitas linhas de várias tabelas, se existir memória suficiente para esta operação. USE_HASH é semelhante aos *loops* aninhados, onde o resultado de uma tabela é percorrido através do resultado da tabela unida. A diferença aqui é que a segunda tabela, aquela sendo percorrida, é colocada na memória. É preciso ter um HASH_AREA_SIZE e PGA_AGGREGATE_TARGET grande o bastante para que isto funcione corretamente, caso contrário, a operação ocorrera no disco e pode não apresentar um bom desempenho.

Sintaxe:

```
Select /*+ USE_HASH(tabela1) */ coluna1, . . .
```

Exemplo:

```
Select /*+ USE_HASH(dept) */ empno, ename,
dept.deptno

From emp, dept Where emp.deptno = dept.deptno

  and emp.empno = 7747;
```

Capítulo 4

Estudo de Caso

Este capítulo apresenta a aplicação usada na monografia e aborda as características do resultado da aplicação das técnicas de *tuning* visando obter um melhor desempenho.

4.1 O Sistema

O sistema usado como estudo de caso é um aplicativo de rastreamento e gerenciamento de veículos, máquinas e equipamentos de grande porte. É responsável pelo rastreamento e gerenciamento de veículos automotores e máquinas.

Atualmente existem cerca de mil e quinhentos veículos com rastreador instalado. O sistema utiliza dados enviados pelos rastreadores dos veículos monitorados. Entre eles, tem-se informações de posicionamento geográfico, velocidade, aceleração, sensores de movimento, violação, pânico, entre outros. Os dados são gerados a partir do módulo GPS instalado e das entradas e saídas do próprio rastreador.

Esses dados são enviados via GPRS como arquivos XML para um servidor centralizado. As informações chegam ao servidor por um *gateway* de entrada e os arquivos são armazenados no servidor.

O banco de dados Oracle utiliza *jobs*, *triggers*, *procedures* e *functions* para ler os arquivos XML, processar as informações e armazenar as informações tratadas nas tabelas do banco de dados.

Cada rastreador envia informação em média a cada trinta segundos. A maior tabela da base de dados é a tabela que armazena o histórico de posições. Esta tabela cresce rapidamente, possuindo uma quantidade de registros

muito grande. Diariamente, mil e quinhentos rastreadores geram quatro milhões e trezentos mil registros na tabela de histórico.

A Figura 5 mostra a tela de visualização da posição atual dos veículos monitorados na cidade de João Pessoa, na Paraiba. O balão de informações, ver detalhe na figura, apresenta várias informações sobre o estado atual do veículo. A API do Google Maps é utilizada nesta interface de visualização:

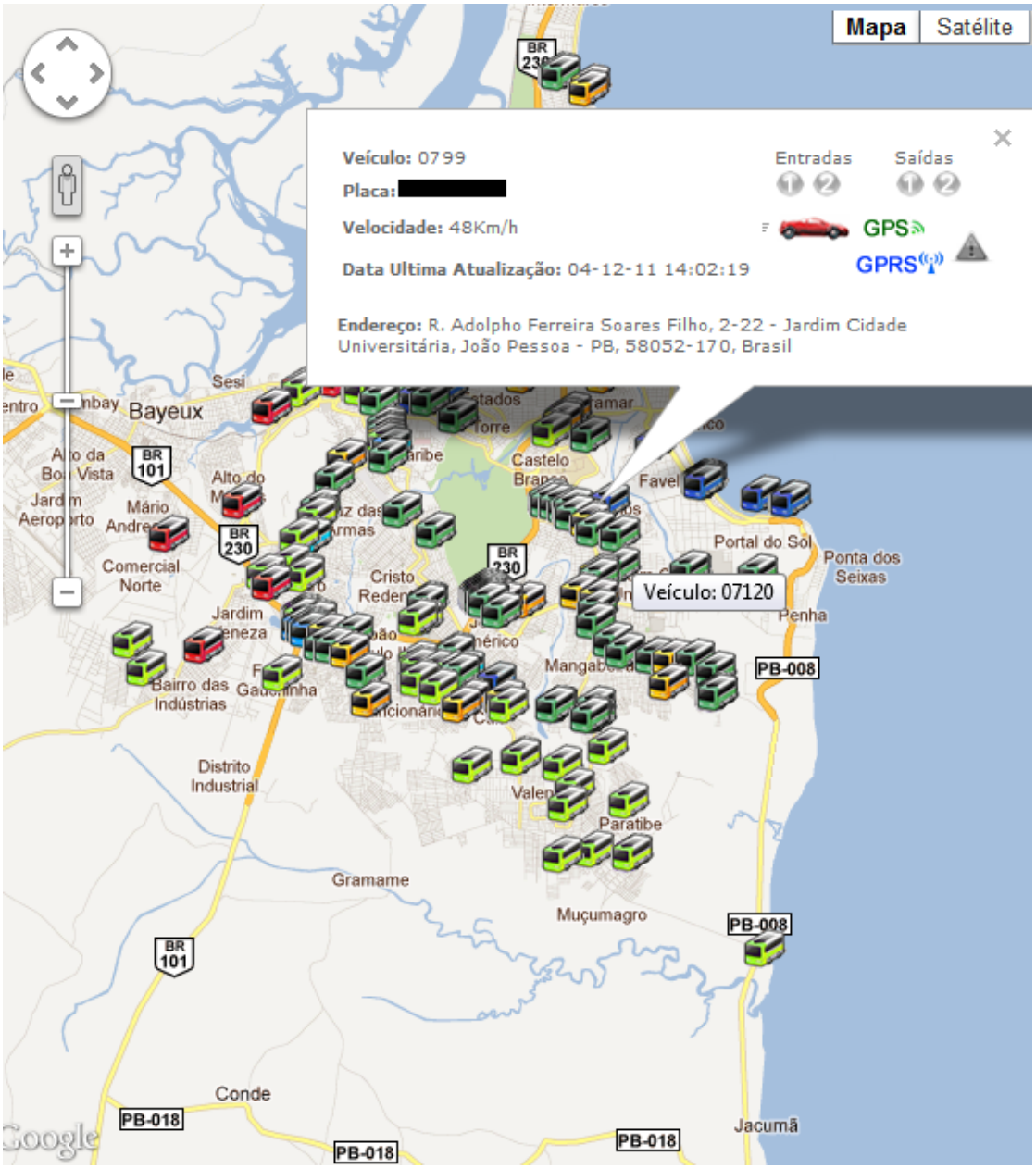

Figura 5. Painel de visualização dos veículos do sistema de rastreamento.

O sistema possui recursos como traçado de percurso, relatórios de velocidade por período, relatórios de histórico de posição, cálculos da velocidade média (realizados dinamicamente por *procedures* para identificar a média da velocidade veicular em um dia e horário da semana específico). Para que as informações necessárias à disponibilização destes recursos sejam geradas, faz-se necessário consultar com muita freqüência as tabelas de histórico, que possuem um volume de dados muito grande.

A execução de consultas que utilizam tabelas como esta podem gerar uma latência inaceitável no sistema *web,* que se propõe a exibir os dados dos usuários em tempo real para fins de segurança e de um monitoramento eficaz.

Técnicas de *tuning* são essenciais para manter o sistema operando de maneira satisfatória. A seguir serão destacadas algumas técnicas de *tuning* que foram utilizadas ao longo do desenvolvimento do sistema estudado para manter um bom desempenho do sistema apresentado.

4.2 Realizando *Tuning* do *Hardware*

No capítulo 2 deste trabalho o assunto *tuning* de *hardware* foi abordado, mostrando que, antes de realizar *tuning* de *hardware,* deve-se realizar uma análise geral do sistema para identificar a real necessidade do investimento em recurso de *hardware* adicional.

Para o sistema de rastreamento identificou-se que a capacidade do recurso de I/O disponível estava insuficiente. Os arquivos XML contendo as informações de rastreamento eram acumulados no servidor *gateway* de dados e chegavam a ocupar toda a capacidade do disco rígido que era de duzentos e cinqüenta gigabytes.

Este gargalo fazia com que a aplicação parasse de gerar novos arquivos XML com as informações atualizadas de cada rastreador. Este é um exemplo claro de um gargalo que uma simples otimização de consultas *SQL* ou uma reestruturação do *software web* não poderia resolver.

Foi realizado *tuning* de *hardware* com a aquisição de um novo servidor *gateway* de dados com capacidade para utilizar os discos rígidos em *Raid* 0 e com um *terabyte* de capacidade cada um. Verificou-se que para este servidor de dados uma capacidade de memória RAM de oito gigabytes era suficiente, pois a função deste servidor era basicamente de recebimento das informações dos rastreadores, criação dos arquivos com a nomenclatura ddmmyyyyhhiiss.xml com as informações recebidas a cada trinta segundos. O tamanho de cada arquivo gerado fica em torno de meio megabyte e a memória RAM utilizada é liberada logo após o arquivo ser criado.

Mesmo com essa nova configuração de *hardware* do servidor *gateway* de dados, a quantidade de arquivos XML gerados crescia linearmente a taxa de um megabyte por minuto. Considerando-se que por uma questão de desempenho e segurança da manutenção da disponibilidade do sistema, os discos rígidos não poderiam ultrapassar setenta e cinco por cento da sua capacidade, ou seja, setecentos e cinqüenta megabytes, esta capacidade seria ultrapassada dentro de menos de dezoito meses. O seguinte cálculo foi utilizado para se obter este resultado:

1 megabyte * 60 * 24 * 30 = 43.200 (megabytes de informação por mês)

750.000 megabytes disponíveis / 43.200 = 17,36 meses

Como estes dados precisam ser mantidos por pelo menos três anos por questões de auditoria, fez-se necessária a criação de um aplicativo auxiliar que é executado a cada hora por um arquivo “.bat”. Este aplicativo verifica a data de criação, que está contida no próprio nome do arquivo. Caso esta data seja superior a seis meses, o arquivo é removido para outro diretório. Diariamente as informações deste diretório são compactadas, copiadas para uma fita *dat* e excluídas do diretório.

Este procedimento garante que o espaço ocupado no dispositivo de I/O do servidor de dados seja mantido em torno de vinte e cinco por cento ou duzentos e cinqüenta megabytes.

Uma medida de *tuning* do tipo monitoramento proativo foi tomada para evitar a lentidão especificamente do servidor de banco de dados Oracle. Verificou-se que,

com o crescimento contínuo do tamanho da base de dados, os dispositivos de interface de I/O utilizados supririam a demanda do SGBD por cerca de dois anos sem a necessidade de adquisição de *hardware* adicional.

Entretanto, como o número de clientes da empresa detentora do *software* está previsto para crescer bastante nos próximos anos, outras duas unidades de disco rígido foram adquiridas para garantir uma maior segurança frente à demanda que está por vir. A capacidade dos componentes de I/O passou de quinhentos gigabytes para um *terabyte*.

Outra melhoria de *hardware* realizada no servidor de banco de dados foi a modificação de memória RAM de quatro gigabytes para oito gigabytes. Este investimento em *hardware* foi importante, pois muitas procedures, triggers e *functions* do banco de dados são responsáveis por parte da regra de negócios do sistema.

4.3 Realizando *Tuning* do *Software*

Para se realizar *tuning* do *software* é necessário considerar o projeto da aplicação. A otimização do aplicativo é um processo contínuo, pois alguns gargalos são identificados apenas quando o sistema já está na fase de produção. Nestes casos as modificações precisam ser feitas sem interromper o serviço, pois isso geraria uma insatisfação muito grande do usuário cliente.

O projeto do sistema estudado foi feito na linguagem de programação orientada a objetos PHP seguindo-se o design de aplicação em camadas bem definidas, entre elas:

- Camada de Interface com o usuário;
- Camada responsável pela regra de negócios da aplicação; e
- Camada de acesso aos dados.

Esta divisão em camadas torna possível ao programador realizar codificações no código fonte da aplicação com maior rapidez. Sistemas estruturados em uma

única camada tornam a manutenção do código fonte uma tarefa árdua, principalmente para aplicações de complexidade moderada e alta; a regra de negócios fica incorporada aos arquivos de *interface* com o usuário, podendo se tornar trechos de código muito longos, dificultando a leitura do programador.

Esta preocupação é relevante para a realização de *tuning* no sentido de que o desenvolvedor precisa corrigir os gargalos que surgem devido a um problema referente ao *software* de forma rápida.

Após a fase de implantação do sistema de rastreamento de veículos, foi encontrado um problema de gargalo que comprometeu bastante o seu desempenho. O mapa de visualização dos veículos exibido, na Figura 5, é atualizado a cada trinta segundos. No momento desta atualização, uma série de procedimentos e cálculos são executados para confirmar se a posição do veículo sofreu alteração e se os dados enviados pelos rastreadores são válidos. Esta verificação é necessária porque existe um percentual de erro na informação do rastreador, que pode ser decorrente de alterações bruscas de temperatura, variações elevadas na tensão de alimentação e até mesmo trepidação, no caso do veículo estar transitando em um terreno acidentado.

Os cálculos e procedimentos que realizam as verificações descritas acima eram feitos no próprio *software* da aplicação. Isto exigia muito da aplicação em PHP e o processamento dependia diretamente dos recursos de *hardware* da máquina cliente, pois muitas validações eram feitas pelo browser em *Javascrip*. Este processamento local geralmente causava latência no momento de atualizar a tela de visualização do usuário.

Realizar *tuning* de *software* foi necessário para corrigir o problema descrito nos parágrafos anteriores. A solução foi remover o processamento do *browser* do usuário e criar *triggers*, *procedures* e *functions* no próprio banco de dados Oracle. Desta forma, o processamento mais crítico foi passado para o servidor de banco de dados Oracle da aplicação. A execução destas *procedures* e *functions* pelo próprio SGBD melhorou consideravelmente a velocidade de resposta; o problema da limitação de *hardware* de cliente passou a não impactar mais a atualização da tela de visualização de veículos do sistema.

Após esta alteração caso surgisse algum gargalo relativo ao *hardware* este estaria no domínio de controle do próprio *DBA*, que poderia realizar *tuning* de *hardware* no servidor de banco de dados para suprir a deficiência.

Constata-se através das medidas tomadas descritas acima, para a realização de *tuning*, que otimizar o *software* não diz respeito necessariamente a alterar dentro do próprio *software* as regras de negócios e forma como os dados são exibidos. No caso estudado para se otimizar o *software* o processamento foi transferido para o SGBD.

Outro caso de otimização do *software* de rastreamento foi identificado após a criação do recurso de cercas de visualização. Como podemos observar na Figura 6, as cercas de visualização são regiões circulares geograficamente definidas pelo usuário do sistema. O sistema monitora a entrada e/ou saída da região delimitada pela cerca de um veículo do usuário. O usuário pode parametrizar o sistema para que um *e-mail* ou uma mensagem SMS seja enviada para ele, caso ocorra algum evento de movimentação de entrada e/ou saída na região da cerca.

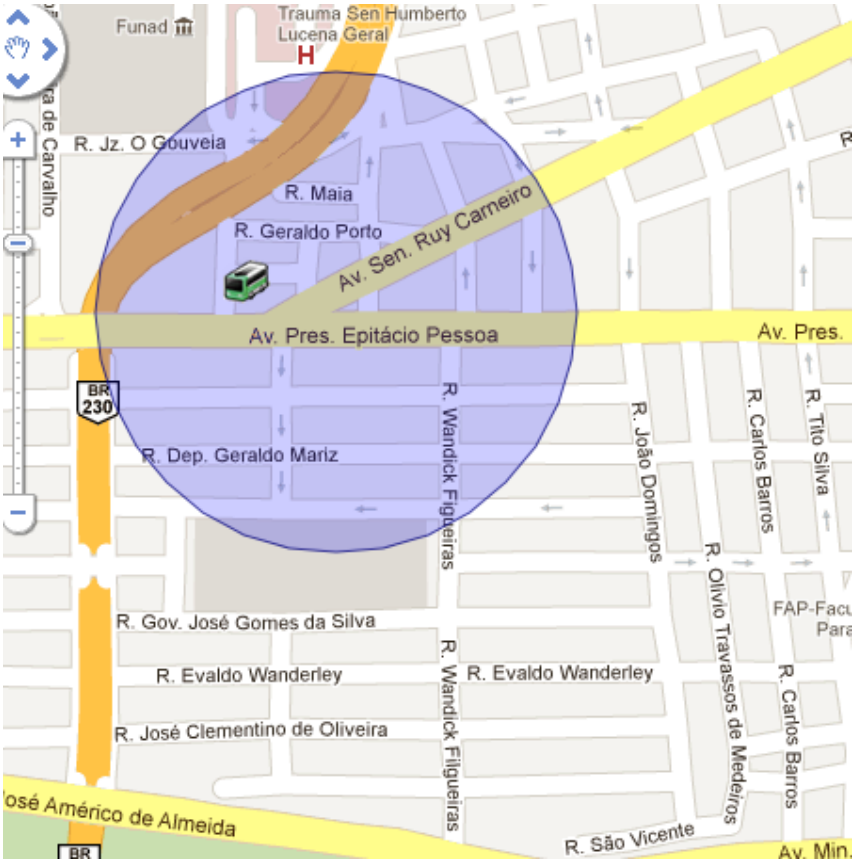

Figura 6. Cerca de visualização do sistema de rastreamento.

Um aplicativo auxiliar foi desenvolvido na linguagem de programação em Delphi. A função deste aplicativo é verificar de acordo com os últimos registros de informação de um veículo, se o mesmo entrou ou saiu na região geográfica delimitada pela cerca. Na ocorrência de um dos dois eventos o aplicativo envia uma mensagem de *e-mail* ou SMS, conforme definido no cadastro do usuário para o *e-mail* de usuário cadastrado.

Entretanto, o aplicativo precisava aguardar a confirmação de recebimento do *e-mail* para finalizar o processo. Isso gerou certa lentidão da ordem de alguns minutos no envio dos *e-mails*. Como não era um gargalo tão crítico, a sua eliminação foi deixada para outro momento, pois outras implementações precisavam ser desenvolvidas.

Contudo, uma nova funcionalidade foi incorporada ao envio de *e-mails*. Agora o usuário recebe notificações via *e-mail* e/ou SMS de ocorrência de velocidade limite

excedida e de violação dos diversos sensores do veículo, que na prática são as entradas e saídas do rastreador. Este novo recurso sobrecarregou significativamente o aplicativo responsável pelo envio de *e-mails* aumentando o gargalo, e fazendo o atraso passar da ordem de segundos para horas.

O ideal é que o *tuning* em geral seja aplicado no momento do monitoramento pró-ativo, contudo na prática a demanda das empresas faz com que a prioridade seja entregar as solicitações dos usuários em detrimento da otimização do sistema.

O caso especificado anteriormente serve como exemplo do que acontece na prática no dia-a-dia empresarial. Neste ponto, o *tuning* pró-ativo não se aplicava mais. Uma situação crítica de gargalo precisava ser solucionada.

A solução de otimização foi eliminar o uso do aplicativo de envio de *e-mails* e foram criadas novas tabelas no banco de dados que armazenavam os dados dos emails criados. Quando um novo registro é inserido nesta tabele uma *trigger* é disparada e chama uma *procedure* que envia o email para o destinatário correspondente.

A diferença no desempenho se dá pelo fato de que para cada registro inserido na tabela *a trigger* chama uma nova instância da *procedure*, e os *e-mails* são enviados em paralelo. Neste caso o paralelismo garantiu uma melhoria significativa de desempenho, e os *e-mails* são enviados na ordem em segundos.

4.4 Realizando *Tuning* da Instância

No caso do sistema de rastreamento desenvolvido, boa parte da regra de negócio foi transferida para o SGBD, como se pode notar no texto da seção anterior. Dessa forma, faz-se necessário o constante monitoramento do desempenho do SGBD Oracle para prevenir possíveis gargalos em outras partes do sistema.

Quando novos procedimentos são delegados à instância, novos gargalos podem surgir. O *DBA* precisa saber avaliar até que ponto é vantajoso, em termos de desempenho, usar o SGBD para realizar procedimentos em cima dos próprios dados. A experiência pessoal com o sistema em estudo produziu ótimos resultados

em termos de melhoramento de desempenho na utilização do SGBD da forma descrita anteriormente.

Conforme visto no capítulo 2, realizar *tuning* da instância da base de dados Oracle pode ser necessário, dependendo do sistema. Otimizar a instância resume-se basicamente em definir os parâmetros adequados para os seus componentes de memória e processos de segundo plano. É importante que os *buffers* e *pools* da *SGA* sejam bem definidos nos arquivos de parâmetros *pfile* e *spfile* para garantir um bom desempenho da execução da instância.

Para o sistema em estudo, a configuração padrão de memória para os seus componentes de memória e processos de segundo plano foi adotada. Na configuração padrão o próprio SGBD gerencia o tamanho dos seus componentes de memória e os redimensionam se for necessário.

4.5 Realizando *Tuning* de Declarações *SQL*

A seguir exibiremos o *script* de criação das principais tabelas da base de dados de nosso estudo. Por questões de sigilo, a empresa detentora do *software* não nos autorizou a exibição da estrutura completa. Porém, para o nosso estudo, as tabelas abaixo representam bem, e de forma adequada, a lógica da base de dados do sistema de rastreamento:

```
CREATE TABLE TB_RASTREADORES
(
  ID                          INTEGER,
  DATA_ULTIMA_ATUALIZACAO     DATE,
  SITUACAO                    VARCHAR2(255 BYTE),
  ATIVO                       INTEGER
)
TABLESPACE USERS
PCTUSED    0
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            INITIAL           64K
            MINEXTENTS        1
            MAXEXTENTS        2147483645
            PCTINCREASE       0
```

```
            BUFFER_POOL      DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE
NOPARALLEL
MONITORING;


CREATE TABLE TB_VEICULO
(
  ID           INTEGER,
  ID_EMPRESA   INTEGER,
  FROTA        VARCHAR2(255 BYTE),
  PLACA        VARCHAR2(255 BYTE),
  MODELO       VARCHAR2(255 BYTE),
  CHASSI       VARCHAR2(255 BYTE),
  RENAVAM      VARCHAR2(255 BYTE),
  SITUACAO     VARCHAR2(255 BYTE),
  ATIVO        INTEGER
)
TABLESPACE USERS
PCTUSED    0
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE
NOPARALLEL
MONITORING;


CREATE TABLE TB_HISTORICO_POSICAO
(
  ID              INTEGER,
  ID_RASTREADOR   INTEGER,
  DATA_POSICAO    DATE,
  LATITUDE        INTEGER,
  LONGITUDE       INTEGER,
  DIRECAO         VARCHAR2(255 BYTE),
  SATELITE        INTEGER,
  CEP             INTEGER,
```

```
  ENDERECO        VARCHAR2(255 BYTE),
  NUMERO          INTEGER,
  BAIRRO          VARCHAR2(255 BYTE),
  CIDADE          VARCHAR2(255 BYTE),
  UF              VARCHAR2(2 BYTE),
  ATIVO           INTEGER
)
TABLESPACE USERS
PCTUSED    0
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE
NOPARALLEL
MONITORING;
```

As tabelas de veículo, rastreador e histórico das posições dos rastreadores são umas das principais tabelas do sistema de rastreamento estudado. Para um melhor entendimento os relacionamentos serão descritos a seguir.

Um veículo pode possuir um ou mais rastreadores instalados. As informações dos rastreadores são armazenadas na tabela de histórico de posição. Esta última tabela cresce continuamente e em média quatro milhões e duzentos mil registros diariamente, considerando-se mil e quinhentos veículos monitorados.

A realização de uma simples consulta de informações entre intervalo de datas para um rastreador na tabela de histórico de posição chega a durar trinta segundos. Um atraso como este para a exibição dos dados em um relatório já é um gargalo que gera transtorno.

Porém, a situação torna-se mais crítica quando o mapa de visualização gerencial precisa executar consultas para obter as informações necessárias para exibir o veículo na tela.

Este gargalo foi resolvido com a criação de uma nova tabela cujo *script* de criação é exibido a seguir:

```
CREATE TABLE TB_ULTIMA_POSICAO_VEICULO
(
  ID                    INTEGER,
  ID_RASTREADOR         INTEGER,
  DATA_ULTIMA_POSICAO   DATE,
  VELOCIDADE            INTEGER,
  MODELO                VARCHAR2(255 BYTE),
  FROTA                 VARCHAR2(255 BYTE),
  SATELITE              INTEGER,
  LATITUDE              INTEGER,
  LONGITUDE             INTEGER,
  NUMERO                INTEGER,
  UF                    VARCHAR2(2 BYTE),
  CIDADE                VARCHAR2(255 BYTE),
  BAIRRO                VARCHAR2(255 BYTE),
  DIRECAO               VARCHAR2(255 BYTE),
  IGNICAO               VARCHAR2(255 BYTE),
  ENDERECO              VARCHAR2(255 BYTE),
  PLACA                 VARCHAR2(255 BYTE),
  CEP                   INTEGER,
  ATIVO                 INTEGER
)
TABLESPACE USERS
PCTUSED    0
PCTFREE    10
INITRANS   1
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            MINEXTENTS       1
            MAXEXTENTS       2147483645
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )
LOGGING
NOCOMPRESS
NOCACHE
NOPARALLEL
MONITORING;
```

Esta tabela guarda um registro para cada rastreador. Esses registros são atualizados no mesmo momento em que o novo registro é inserido na tabela de histórico de posição dos veículos.

Desta forma evita-se que seja realizado um *full table scan* para retornar as posições onde a data de última posição for máxima para cada rastreador. O custo computacional de uma atualização em um registro e uma busca na tabela de últimas posições dos veículos é milhares de vezes menor do que o custo computacional para realizar a consulta na tabela de histórico de posições com centenas de milhares de registros.

Neste caso foi realizado *tuning* na estrutura da base de dados visando um melhor desempenho da aplicação como um todo.

Contudo, a tabela de histórico de posição dos veículos é essencial para a geração de relatórios e desenho no mapa do traçado de rota dos veículos como exibido na Figura 7.

Figura 7. Visão da rota percorrida por um veículo.

Para otimizar a execução das *queries* que utilizam a tabela de histórico de posição de veículos foram criados alguns índices visando diminuir o tempo de execução das *queries*.

Abaixo o *script* de criação de um índice composto é exibido para as colunas id_rastreador e data_posicao da tabela de histórico de posição:

```
CREATE INDEX IND_HISTORICO_POSICAO
 ON TB_HISTORICO_POSICAO
 USING BTREE (ID_RASTREADOR, DATA_POSICAO);
```

Foi criado um índice composto do tipo B-Tree e foi escolhido pois apresenta um ótimo resultado quando utilizado para indexar a chave primária da tabela, valores únicos ou próximos de únicos, ou linhas que estão dentro de um intervalo de datas ou outro tipo de faixa de valores [4].

Em conjunto com o índice *hints* serão utilizados *hints* para garantir que o otimizador do Oracle fará uso do índice criado. Abaixo uma declaração *SQL* que realiza uma consulta na tabela de histórico de posição dos veículos é exibida.

```
SELECT /*+ INDEX (TB_HISTORICO_POSICAO
IND_HISTORICO_POSICAO)*/
 ID_RASTREADOR,
 LATITUDE, LONGITUDE,
 DATA_POSICAO, VELOCIDADE
FROM TB_HISTORICO_POSICAO
WHERE ID_RASTREADOR = 4321 AND DATA_POSICAO
 BETWEEN
 TO_DATE('dd/mm/yyyy hh24:mi:ss','30/11/2011 12:30:00')
 AND TO_DATE('dd/mm/yyyy hh24:mi:ss','01/12/2011 00:30:00');
```

A utilização de índices e *hints* nas consultas críticas do sistema melhoraram o desempenho das declarações *SQL*. A consulta acima sem o uso desas técnicas levava em torno de trinta a cinquenta segundos para ser executada na base de dados do sistema. Já com o uso das técnicas de indexação e *hints* este tempo foi reduzido para cerca de dez segundos.

Este foi um ganho considerável de desempenho principalmente na tela de exibição da traçado da rota dos veículos e nos relatórios, como por exemplo, os relatórios de histórico de velocidade do veículo.

Outra funcionalidade do sistema que melhorou com o uso de índices e *hints* foi o gráfico de velocidade veicular que exibe o desenvolvimento da velocidade do

veículo ao longo de um dia escolhido. Este gráfico chegava a demorar um minuto para poder ser visualizado, após a aplicação da técnica de *tuning* ele pode ser visualizado torno de doze segundos. O gráfico citado pode ser visto na Figura 8.

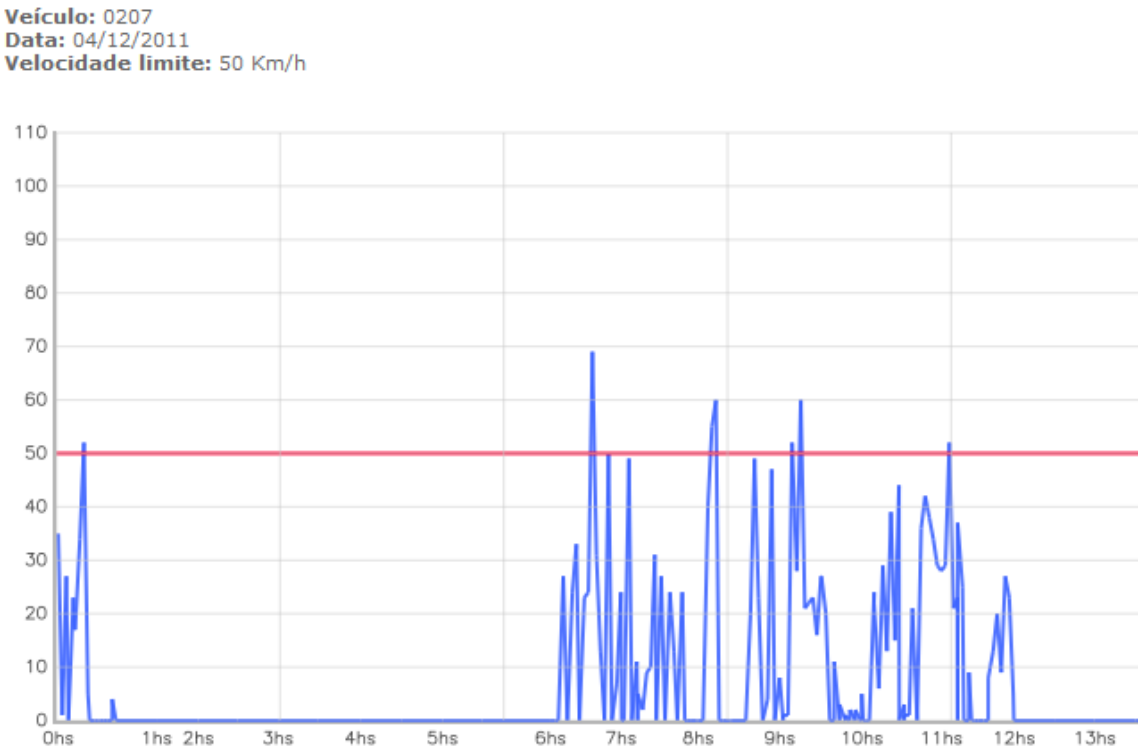

Figura 8. Visão do gráfico de velocidade veicular.

A seguir é exibida a Tabela 1 onde são mostrados os resultados obtidos a partir da aplicação de técnicas de otimização de desempenho para solucionar problemas comentados nesta seção.

Tabela 1. Resultados obtidos no estudo de caso.

Problema	***Tuning***	**Solução**	**Analise do Resultado**
Capacidade de I/O insuficiente. Arquivos XML ocupavam todo o espaço do disco rígido no servidor *gateway*.	Problema de *Hardware* Solução em *Hardware*	Aquisição de um novo servidor *gateway* capacidade para utilização discos rígidos em *Raid* 0 e com um *terabyte* de capacidade cada um.	O sistema voltou a operar normalmente. A solução garantiu que a curto prazo o servidor não pararia mais por questões de capacidade.
Apesar da solução anterior a quantidade de arquivos continuava crescendo linearmente. O espaço ocupado nos discos rígidos não poderia ultrapassar 75% da sua capacidade. Isto ocorreria em menos de 18 meses. Dados deviam ser mantidos por pelo menos três anos.	Problema de *Hardware* Solução em *Software*	Criação de um aplicativo executado a cada hora, que verifica a data de criação, caso ela seja > 6 meses, o arquivo é removido para outro diretório. Diariamente o *backup* deste diretório é feito em fita e as informações antigas excluídas.	Procedimento garante que o espaço ocupado no dispositivo de I/O do servidor de dados seja mantido em torno de 25%.
Número de clientes da empresa tem previsão de crescer bastante nos próximos anos. A capacidade dos dispositivos de I/O do servidor de banco de dados não seria suficiente.	Problema de *Hardware* Solução em *Hardware*	Duas unidades de disco rígido foram adquiridas para garantir uma maior segurança na manutenção da disponibilidade do sistema.	A capacidade dos componentes de I/O passou de quinhentos *gigabytes* para um *terabyte*, permitindo que o sistema comporte o dobro de novos clientes.
Os cálculos e procedimentos que realizam as verificações necessárias para a exibição do mapa de visualização eram feitos no próprio *browser* do *host*. O desempenho dependia do *hardware* do cliente.	Problema de *Software* Solução em SGBD	Remover o processamento do browser do usuário e criar *triggers*, *procedures* e *functions* no próprio banco de dados Oracle.	A execução destas *procedures* e *functions* pelo SGBD reduziu o tempo de resposta; a configuração de *hardware* de cliente passou a ter um impacto mínimo na visualização do mapa.

Aplicativo auxiliar feito em Delphi com a função de enviar email estava sobrecarregado por conta da quantidade de emails para serem enviados seqüencialmente.	Problema de *Software* Solução em SGBD	Aplicativo eliminado. Criadas novas tabelas que armazenam os dados dos emails. O paralelismo no envio de emails foi incorporado com a criação de uma *trigger* que dispara *procedure* o envio do email no momento da inserção.	A solução reduziu o atraso do envio de emails de cerca de uma hora para alguns segundos.
Consultas simples na tabela de histórico podem durar mais de meio minuto. A situação era crítica, pois o mapa de visualização precisava consultar na tabela para exibir as informações na tela.	Problema de *SGDB* Solução em SGBD	Criação de uma nova tabela que guarda a informação da última posição de cada veículo. Estes registros são apenas atualizados.	O custo da atualização em um registro e uma busca na tabela de últimas posições é milhares de vezes menor do que o custo para consultar na tabela de histórico com milhões de registros.
Apesar da solução anterior, pesquisar na tabela de histórico de posições ainda é necessário para a exibição de vários relatórios.	Problema de *SGDB* Solução em SGBD	Criação de índices para a tabela de histórico de posição e *hints* para uso dos índices.	Consultas simples na tabela de histórico reduziram em média 75% do tempo de execução.

Observa-se que, para o tipo de sistema estudado, a distribuição de problemas de *hardware* (43%), *software* (28%) e SGDB (28%) mostra uma predominância dos problemas de *hardware* e mais especificamente de capacidade de armazenamento das informações. Isto se deve ao fato da quantidade de dados manipulados pela aplicação em estudo ser grande.

Para outros tipos de sistemas, posivelmente os problemas de *hardware* não sejam tão numerosos. Isto vai depender do *hardware* inicial escolhido. No caso estudado o número de clientes cresceu acima do esperado e o dimensionamento de hardware não atendeu a demanda dos usuários. Este comportamento não é comum a todos os tipos de sistemas. Consequentemente, espera-se que as otimizações do

SGBD possuam maior percentual de resolução na maioria dos sistemas computacionais desenvolvidos..

Já nas observações dos recursos utilizados para solucionar os problemas podemos observar a seguinte distribuição: *hardware* (28%), *software* (14%) e SGBD (57%). Todos os problemas de desempenho relacionados a *software* e SGBD foram solucionados realizando-se *tuning* no SGBD. Já os problemas de *hardware* foram solucionados com técnicas de *tuning* de *software* ou do próprio *hardware*.

Nota-se que a utilização dos recursos do próprio SGBD resolveu a maior parte (57%) dos problemas de otimização. Posivelmente em outros tipos de sistema a predominância do tuning de SGBD seja mais expressiva.

As técnicas de *tuning* de SGBD utilizadas se distribuíram da seguinte forma:

- Utilização de *procedures*, *triggers* e *functions* em conjunto (50%)
- Utilização de índices e *hints* (25%)
- Alteração na estrutura da base de dados (25%)

Ao longo deste projeto esparava-se uma maior predominância de otimizações de SGBD na utilização de índices e *hints*. Portanto, as características da base de dados estudada não permitia a aplicação de uma maior diversidade deles. Possivelmente, para bases com estruturas diferentes, com maior número de tabelas, relacionamentos e junções, o uso de índices e hints possa ser bem mais explorado.

Em contrapartida, a utilização de *triggers*, *procedures* e *functions* para transferir parte da regra de negócio do sistema para o SGBD foi bastante explorada. Espera-se que seja também para sistemas *web* em geral onde o custo do processamento do *browser* do cliente possa trazer lentidão demasiada.

Capítulo 5

Conclusão

Este trabalho abordou a temática de *tuning* de sistemas computacionais com ênfase em sua aplicação em banco de dados. Foi feito um levantamento de algumas técnicas de *tuning* onde os seguintes tópicos principais foram abordados:

- *tuning* de *hardware;*
- *tuning* de *software,*
- *tuning* da instância Oracle e
- *tuning* de declarações *SQL*.

Sobre estes itens, discutiu-se acerca de suas principais características e procurou-se focar em aspectos de utilização de *tuning* e em quais situações a otimização deve ser aplicada buscando-se eliminar, ou no mínimo reduzir, gargalos que surgem ao longo do desenvolvimento de um sistema.

No que se refere ao tuning da instância, explorou-se detalhes como o funcionamento do otimizador Oracle na escolha do plano de execução de uma declaração *SQL*. A arquitetura do SGBD Oracle 10g foi detalhada e recursos como componentes de memória e processos de segundo plano relevantes na temática de desempenho da instância foram abordados.

Características específicas de melhoria de desempenho de declarações *SQL,* como utilização de índices e *hints,* foram detalhados. Para facilitar o entendimento destes recursos, mostrou-se a sintaxe específica de cada um dos tipos mais usados de índices e *hints* do Oracle, bem como exemplos práticos de sua utilização.

Um estudo de caso mostrou como as técnicas abordadas foram aplicadas na prática ao longo do projeto de desenvolvimento de um sistema computacional. O sistema estudado é uma aplicação *web* de rastreamento e monitoramento de

veículos em tempo real que utiliza o banco de dados Oracle e, como é comum em sistema de complexidade mediana, apresentou vários problemas de desempenho e necessidades de redimensionamento de recursos. Ao longo do projeto diversas medidas foram tomadas visando garantir um monitoramento de veículos eficiente e em tempo real.

Na aplicação estudada utilizaram-se técnicas de otimização de *hardware*, *software*, banco de dados e declarações *SQL*. Este fato reforça a nossa conclusão de que realizar *tuning* é uma tarefa que abrange o sistema em todas as suas partes.

Não basta ao *DBA* focar seus esforços em otimização de consultas *SQL* em detrimento das outras áreas como estrutura da aplicação, dimensionamento de recursos de *hardware* e *software*. O ideal para se realizar um estudo da melhoria de desempenho e eliminação de gargalos de um sistema é analisar o sistema em todas as suas partes componentes e saber como elas se inter-relacionam. Desta forma é possível chegar a soluções mais eficientes.

5.1 Dificuldades Encontradas

Uma das barreiras encontrada ao longo do projeto foi a dificuldade de mostrar na prática, de forma mais detalhada, a aplicação de uma maior diversidade de *hints* para a aplicação em estudo. Este fato foi decorrente da característica estrutural da base de dados estudada, pois cada tipo de *hint* é aplicável a consultas com características específicas como foi descrito no texto deste trabalho.

Apenas alguns tipos de *hints* são aplicáveis às consultas abordadas neste projeto e isso foi um fator de impedimento para que o seu uso fosse abordado em nosso estudo de caso com maior aprofundamento.

5.2 Trabalhos Futuros

Uma proposta de trabalho futura seria a utilização de técnicas de tuning, especificamente na otimização de *queries* de forma mais aprofundada. Para isto

sugere-se utilizar uma base de dados com maior complexidade e diversidade em seus relacionamentos.

Outros trabalhos podem proseguir na obtenção de estatísticas de otimização de *hardware*, e SGBD em outros tipos de sistemas computacionais. Produzindo assim material comparativo em relação a este trabalho.

Bibliografia

[1] NORBERTO,E. UNIVERSIDADE FEDERAL DO MATO GROSSO DO SUL. Introdução a Banco de Dados. Disponível em http://www.dct.ufms.br/~edson/bd1/db1.pdf (PDF) pp. 1. Acessado em 10 de novembro de 2011.

[2] Dicionário Bab la. Tradução. Disponível em http://pt.bab.la/dicionario/ingles-portugues/*tuning*. Acessado em 17 de novembro de 2011.

[3] BAPTISTA, C. de S. Administração de Sistemas de Gestão de Banco de Dados, 2008. Disponível em: www.dsc.ufcg.edu.br/~baptista/cursos/ABD/ADM1.ppt. Acessado em 22 de setembro de 2011.

[4] Oracle® Database Performance Tuning Guide, 10g Release 1 (10.1). Part No. B10752-01. December 2003.

[5] WHALEN, E. *Performance Tuning* Oracle *RAC on Linux*. Disponível em http://www.perf*tuning*.com/files/pdf/Performance%20*Tuning*%20Oracle%20RAC%20on%20Linux.pdf. Acessado em 19 de novembro de 2011.

[6] VILLALBA, C. Disponível em http://carlos.syr.edu/oracle-database-architecture. Acessado em 20 de novembro de 2011.

[7] SAUDINO FILHO, G. A arquitetura do Oracle. Disponível em http://www.linhadecodigo.com.br/Artigo.aspx?id=99. Acessado em 20 de novembro de 2011.

[8] VALIATI, P. Índices no Oracle – Parte 1. Artigo da *SQL Magazine*, edição 36.

[9] UNIVERSIDADE FEDERAL DE CAMPINA GRANDE. Disponível em http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/threads/threads1.html. Acessado em 06 de dezembro de 2011.

[10] UNIVERSIDADE DE SÃO PAULO. Disponível em http://www.ime.usp.br/~andrers/aulas/bd2005-1/aula5.html. Acessado em 06 de novembro de 2011.

[11] RONCONI, V. O otimizador Oracle para desenvolvedores – Parte II – Otimizador baseado em custos. Disponível em http://www.linhadecodigo.com.br/Artigo.aspx?id=737. Acesado em 06 de dezembro de 2011.

[12] BIANCHI, W. Introdução às Stored Procedures com SQL Server 2000/2005. Disponível em http://www.devmedia.com.br/post-2213-Introducao-as-Stored-Procedure-com-SQL-Server-2000-2005.html. Acessado em 07 de dezembro de 2011.